ESCOLA TECNOLÓGICA E PROFISSIONAL DA ZONA DO PINHAL

Sistema de envio de SMS com interface web

LUIGI MATTEO GIRKE

Relatório Final da Prova de Aptidão Profissional do
Curso Profissional de
Técnico de Gestão de Equipamento Informático

Pedrógão Grande | Maio 2025

Cofinanciado pela União Europeia
PORTUGAL 2030
PESSOAS 2030
SELO DE CONFORMIDADE EQAVET

ESCOLA TECNOLÓGICA E PROFISSIONAL DA ZONA DO PINHAL

CURSO PROFISSIONAL DE
TÉCNICO DE GESTÃO DE EQUIPAMENTO INFORMÁTICO

Sistema de envio de SMS com interface web

LUIGI MATTEO GIRKE

ALUNO Nº 2812
TURMA E-22/25

Relatório Final da Prova de Aptidão Profissional

Professor Orientador Engº Rui Veríssimo

Diretor de Curso Engº Vítor Monteiro

Pedrógão Grande | Maio 2025

AGRADECIMENTOS

Em primeiro lugar, gostaria de agradecer ao Rui Veríssimo, orientador do projeto PAP, e ao Vítor Monteiro, diretor do curso de Engenharia. A sua orientação, paciência, apoio e disponibilidade constante foram fundamentais ao longo dos três anos de formação e durante o desenvolvimento deste projeto.

Um agradecimento especial aos meus amigos e família pelo apoio e incentivo incondicionais ao longo do meu percurso académico, especialmente durante este projeto. Por fim, gostaria de agradecer à ETPZP por ter disponibilizado todas as ferramentas essenciais para a realização deste projeto.

ÍNDICE

INTRODUÇÃO

Esta aplicação foi criada para substituir o elevado custo das mensagens de texto para a escola, fornecendo uma solução de comunicação rápida, fácil e económica através de SMS. O facto de estar na Web tornou-a acessível a todos, independentemente do seu sistema operativo.

A aplicação permitia aos utilizadores enviar mensagens para vários destinatários, programar envios para entrega futura, cancelar mensagens programadas e gerir mensagens enviadas através de uma interface de fácil utilização inspirada nos clientes de correio eletrónico. A autenticação foi efetuada localmente utilizando o servidor Active Diretory (AD) local da escola.

Foi construído com Next.js, tirando partido do seu App Router, juntamente com uma base de dados Postgres, componentes ShadCN e outros pacotes. O envio de SMS foi possível através da Interface de Programação de Aplicativos (API) REST (Representational State Transfer) da GatewayAPI. Durante a implementação, a aplicação foi executada num contentor Docker, com o Nginx configurado para encaminhar o tráfego do router para a porta exposta do contentor adequado.

> **Sugestão**: Utilize a **funcionalidade Localizar** para facilitar a navegação neste PDF. Pode aceder-lhe premindo `Ctrl + F` (no Windows), `Command + F` (no macOS), ou `/` atalho na maioria das aplicações.

FERRAMENTAS UTILIZADAS

Aplicações e serviços externos

- **Visual Studio Code (VSCode)**: Ambiente de desenvolvimento integrado (IDE) utilizado para escrever todo o código do projeto. Para além disso, foram utilizados os seguintes plugins:

Suporte a JavaScript EJS: Fornece suporte para modelos EJS (JavaScript incorporado) no Visual Studio Code.

Prettier - Formatador de código: Um formatador de código opinativo para muitas linguagens e integra-se com o VSCode.

ESLint: Uma ferramenta para identificar e relatar padrões encontrados no código ECMAScript/JavaScript, ajudando a manter a qualidade do código.

Snippets ES7 React/Redux/React-Native: Fornece snippets JavaScript e React para um desenvolvimento mais rápido.

Importação automática: Localiza e importa automaticamente componentes, funções e outros módulos React no seu código.

Preservação de maiúsculas e minúsculas com vários cursores: Preserva as maiúsculas e minúsculas do texto quando se utiliza a edição com vários cursores no VSCode.

Erros bonitos do TypeScript: Melhora as mensagens de erro do TypeScript para que sejam mais legíveis e informativas.

Ícones do VSCode: Adiciona ícones a ficheiros e pastas no explorador do VSCode para uma melhor organização visual.

Docker: Fornece suporte para desenvolver e gerenciar contêineres Docker diretamente no VSCode.

Corretor ortográfico de código: Um verificador ortográfico básico para código e comentários, ajudando a detetar erros de digitação.

Tailwind CSS IntelliSense: Fornece sugestões inteligentes e preenchimento automático para as classes CSS do Tailwind no seu código.

- **API REST da Gateway API**: Utilizada para enviar, programar e cancelar mensagens SMS programadas e obter estatísticas sobre SMSs enviados.

PostgreSQL: Sistema de gestão de bases de dados relacionais utilizado para armazenar e gerir dados de aplicações.

No macOS, foi utilizado o Postgres.app

No Windows, o PostgreSQL foi descarregado do sítio Web oficial

- **Figma**: Programa de design utilizado para criar protótipos de design de aplicações, wireframes e interfaces de utilizador de brainstorming.
- **Obsidiana**: Aplicação de anotações baseada em Markdown utilizada para escrever os relatórios e tomar notas ao longo do projeto.
- **Microsoft Word**: Software de processamento de texto utilizado para a formatação e finalização dos relatórios.
- **Git**: Sistema de controlo de versões utilizado para acompanhar as alterações de código ao longo do tempo. Nos sistemas operativos baseados em UNIX, vinha pré-instalado. No Windows, no entanto, precisava de ser instalado separadamente.
- **GitHub**: Plataforma baseada na Web para alojar e colaborar em repositórios Git, utilizada para sincronizar código entre diferentes dispositivos.
- **Bun**: Gerenciador de pacotes usado para instalar as dependências do projeto e os componentes do ShadCN
- **Docker**: Plataforma de contentorização utilizada para criar, implementar e gerir aplicações em ambientes isolados.
- **Nginx**: Servidor Web de elevado desempenho e servidor proxy inverso utilizado para servir aplicações Web e gerir o equilíbrio de carga.
- **dbdiagram.io**: Gerador de diagramas de bases de dados baseado na Web utilizado para visualizar esquemas de bases de dados.
- **ChatGPT**: Modelo de linguagem de IA utilizado numa interface baseada na Web para ajudar a encontrar e corrigir erros de código.

DeepL: Ferramenta de tradução com recurso a IA utilizada na interface baseada na Web para traduzir relatórios para português.
No macOS, as aplicações foram instaladas utilizando o homebrew se o respetivo cask estivesse disponível.

Dependências

Dependências

- `@hookform/resolvers`: Integração do resolvedor para react-hook-form.
- `@radix-ui/react-*@^1`: Componentes de IU acessíveis, personalizáveis e sem estilo.
- `@svgr/webpack`: Transforma SVGs em componentes React.
- `activedirectory2`: biblioteca cliente do Active Diretory.
- `class-variance-authority`: Utilitário para gerir nomes de classes CSS.
- `clsx@^2`: Utilitário para aplicar condicionalmente nomes de classes CSS.
- `cmdk`: Componente de menu de comandos acessível.
- `date-fns@^4`: Biblioteca abrangente de utilitários de data.
- `i18next@^24`: Estrutura de internacionalização para browser e Node.js.
- `i18next-resources-to-backend`: Adaptador de backend para i18next.
- `iron-session@^8`: Gerenciamento seguro de sessões para aplicações Next.js.
- `libphonenumber-js`: Biblioteca JavaScript para análise, formatação e validação de números de telefone.
- `lucide-react`: Biblioteca de ícones React.
- `next-i18n-router`: Roteamento internacionalizado para Next.js.
- `next-themes`: Suporte de temas para Next.js.
- `next@15`: Estrutura React para criar aplicações renderizadas no servidor.
- `nó`: Tempo de execução do JavaScript.
- `pg`: Cliente PostgreSQL para Node.js.
- `react-day-picker`: Componente de seleção de datas acessível.
- `react-hook-form@^7`: Formulários extensíveis e de alto desempenho com validação fácil.
- `react-i18next`: Internacionalização para React.
- `react-loading-skeleton`: Carregadores de esqueleto para React.
- `react-resizable-panels`: Layout de painel redimensionável para React.
- `react@19`: biblioteca JavaScript para a construção de interfaces de utilizador.
- `recharts@^2`: Biblioteca de gráficos compostável construída sobre componentes React.
- `sonner`: Sistema de notificação para React.
- `tailwind-merge`: Utilitário para mesclar classes CSS do Tailwind.
- `tailwindcss-animate`: Utilitário para adicionar animações às classes CSS do Tailwind.
- `zod@^3`: Validação de esquemas com inferência estática de tipos.

Dependências de desenvolvimento

- `typescript@^5`: Superconjunto de JavaScript para tipagem estática opcional.
- `tailwindcss@^3`: Estrutura CSS de primeira utilidade para criar rapidamente designs personalizados.
- `eslint@^8`: Linter JavaScript plugável.
- `postcss@^8`: Ferramenta para transformar CSS com JavaScript.
- `@types/react@19`: Definições TypeScript para React.
- `@types/node@^20`: Definições TypeScript para Node.js.
- `eslint-config-next@15`: Configuração do ESLint para projectos Next.js.
- `@types/react-dom@19`: Definições TypeScript para React DOM.
- `@types/validator@^13`: Definições TypeScript para a biblioteca validator.js.
- `i18nexus-cli@^3`: Ferramenta CLI para gerir recursos i18n.

Uma lista de todas as dependências e suas versões exatas pode ser encontrada no package.json.

Tipografia

O TypeScript era um superconjunto de JavaScript com tipagem estática, que ajudava os programadores a detetar erros precocemente e a melhorar a qualidade do código. Melhorou a experiência de desenvolvimento com funcionalidades como o preenchimento automático e a inferência de tipos, tornando-o ideal para o projeto.

O TypeScript quase não foi alterado, mas algumas regras foram modificadas. As configurações podem ser vistas e modificadas no `tsconfig.json` localizado em `/`.

Durante o tempo de compilação, o comando a seguir foi útil. Ele usou o compilador TypeScript para verificar todo o projeto em busca de erros de tipo, que precisavam ser corrigidos para executar uma compilação.

```
tsc --noEmit
```

Para obter mais informações sobre o TypeScript, foi consultada a documentação oficial. Para a sua utilização no contexto do Next.js, foi consultada esta documentação.

ESTRUTURA DO FICHEIRO

Muita da estrutura de ficheiros existente foi escolhida por ser obrigatória no Next.js ou por ser uma convenção comum. Algumas pessoas colocam `/lib`, `/components`, `/contexts` e `/hooks` dentro do diretório `/app`. No entanto, para manter o diretório app o mais limpo possível, esses diretórios foram colocados fora.

- `/app/` continha todas as páginas e estilos da aplicação, bem como tipos de letra e uma função de internacionalização para carregar traduções do lado do servidor.
- `/components/` contém todos os componentes React.
- `/contexts/` contém todos os contextos React.
- `/hooks/` contém todos os hooks personalizados do React.
- `/lib/` continha todas as funções utilitárias, esquemas `zod` e a maior parte do código do lado do servidor.
- `/locales/` continha todas as traduções i18next.
- `/node_modules/` continha todos os módulos do nó (esta pasta nunca foi tocada).
- `/public/` era uma convenção de ficheiros Next.js para activos estáticos como imagens e ícones.
- `/types/` contém todos os tipos de TypeScript.
- `/` contém todos os ficheiros de configuração.
- `/.next` era uma pasta oculta gerada pelo Next.js sempre que um servidor de compilação ou desenvolvimento era iniciado.
- `/.vscode`: era específico do Visual Studio Code e continha algumas definições de espaço de trabalho para um plug-in de correção ortográfica.

Encaminhamento baseado em ficheiros Next.js

Na secção "`/app` diretory", muitas convenções de ficheiros Next.js foram mencionadas com ligações para a documentação Next.js, mas os princípios básicos e as razões para a estrutura de ficheiros escolhida foram explicados aqui.

Convenções da pasta Next.js:

Os diretórios envoltos em **parênteses rectos** como `/app/[locale]` representavam segmentos de rota dinâmicos, permitindo que as páginas e os componentes no seu interior recuperassem os seus valores (neste caso, o locale atual). Todas as páginas estavam localizadas em `/app/[locale]`, uma vez que toda a aplicação necessitava de acesso ao idioma atual para que a internacionalização funcionasse.

Os diretórios **entre parênteses rectos** como `/app/(root)` serviam como grupos de rotas e eram invisíveis para o utilizador final. Funcionavam como diretórios normais para agrupar páginas diferentes. Neste projeto, foram utilizados para agrupar páginas com o mesmo layout, garantindo que o `layout.tsx` nesse diretório se aplicava a todas as outras páginas **sem criar um segmento de rota real** como `etpzp-sms.com/(root)`.

Os diretórios com **nomes padrão** contendo um `page.tsx` representavam os nomes dos segmentos de rota. Por exemplo, `/app/contacts/page.tsx` estava acessível em `etpzp-sms.com/contacts`.

Os diretórios que começavam por um **sublinhado** indicavam rotas desactivadas (não acessíveis ao utilizador final). Funcionam como comentários de código. Neste projeto, `/_seed/page.tsx` foi utilizado apenas durante o desenvolvimento.

Convenções do ficheiro Next.js:

Os ficheiros `page.tsx` representam páginas acessíveis pelo nome do diretório acima.

Os ficheiros `layout.tsx` servem como esquemas aplicados a todas as páginas no mesmo diretório e diretórios aninhados.

- `loading.tsx` utilizou o React Suspense nos bastidores para exibir uma IU de fallback enquanto a página estava sendo carregada.

Os ficheiros `error.tsx` implementaram limites de erro que capturaram erros inesperados no mesmo diretório, tratando erros inesperados ao fornecer uma IU de recurso.

- `not-found.tsx` era apresentado sempre que ocorria um erro 404.
  A maioria destas caraterísticas foi escolhida por melhorar drasticamente a experiência do utilizador.

Diretório /app

- `/app/favicon.ico` (convenção do ficheiro Next.js): Ficheiro de imagem para definir o ícone da aplicação no separador do browser
- `/app/globals.css`: Ficheiro CSS para variáveis css utilizadas globalmente

- `/app/i18n.js` (convenção de ficheiro i18next): ficheiro de configuração i18next para carregar traduções do lado do servidor. Contém o código deste tutorial
- `/app/layout.tsx` (convenção do ficheiro Next.js): Layout raiz da aplicação
- `/app/not-found.tsx` (convenção do ficheiro Next.js): Página global 404 não encontrada
- `/app/scattered-profiles.module.css`: Módulos CSS utilizados no componente `message-display.tsx`
- `/app/[locale]` (convenção do ficheiro Next.js): Segmento de rota dinâmico para o idioma atual
    - `/app/[locale]/(root)` (convenção do ficheiro Next.js): Grupo de rotas para páginas que utilizam o painel de navegação redimensionável (ver `/app/[locale]/(root)/layout.tsx`).
        - `/app/[locale]/(root)/(message-layout)` (convenção do ficheiro Next.js): Grupo de rotas para páginas semelhantes que usavam as mesmas traduções e precisavam de acesso aos contactos (ver `/app/[locale]/(root)/(message-layout)/layout.tsx`), partilhando a mesma página de erro. Até à data, todas estas páginas utilizavam o **layout** de **três colunas** com painéis redimensionáveis.
        - `/app/[locale]/(root)/(other)` (convenção do ficheiro Next.js): Grupo de rotas para páginas especiais que **não** partilhavam as mesmas caraterísticas. Uma vez que este diretório não tinha um layout, os ficheiros aderiram ao layout acima (ver `/app/[locale]/(root)/layout.tsx`).
- `/app/fonts`: Diretório para fontes locais, importadas no layout raiz (`/app/layout.tsx`). Para obter informações sobre tipos de letra locais, foi consultada a documentação Next.js.
  As diretorias normais não mencionadas aqui são as páginas, que foram explicadas no capítulo "PAGES".

/ `lib` diretório

A pasta `lib` armazenava funções utilitárias reutilizáveis e a maior parte do código do lado do servidor. Isso inclui o arquivo semente do banco de dados e as funções de busca, configuração de autenticação, ações do servidor de autenticação e ações do servidor para alterar dados e fazer chamadas de API, compartilhadas entre diferentes componentes e páginas. Embora o código de autenticação contenha acções do servidor, foi colocado no seu próprio diretório (`/lib/auth`) para separar o tópico e manter o diretório de acções (`/lib/actions`) menos confuso.

- `/lib/`:
  - `form.schemas` para esquemas zod
  - `theme.colors` para a configuração do tema Tailwind e Next.js

- `utils.ts` para funções utilitárias utilizáveis em qualquer lugar

O diretório `/lib/actions` continha acções do servidor e um diretório de teste para efeitos de teste de desenvolvimento para simular chamadas API.

- `/lib/auth` tinha código de autenticação e um diretório chamado `activedirectory`, que incluía funções que envolviam o pacote `activedirectory2`.
- O `/lib/db` tinha um ficheiro para semear a base de dados (`/lib/db/seed.sql`) com o esquema inicial da base de dados, funções de obtenção da base de dados e um Dockerfile para semear a base de dados Postgres executada dentro do contentor Docker durante a produção.

Diretório `/componentes`

Este diretório continha todos os componentes React organizados em subdirectórios e os componentes ShadCN.

- `/components/admin-dashboard`: Contém os componentes utilizados no painel de controlo administrativo e foram colocados numa diretoria separada para separar o tópico
- `/componentes/modais`: Contém os componentes modais/popup e foram colocados numa diretoria separada para separar o tópico
- `/components/shared`: Contém componentes partilhados que foram muito utilizados
- `/components/ui`: Componentes ShadCN mantidos: Os ficheiros no seu interior permaneceram praticamente inalterados, exceto no que diz respeito a pequenos ajustes de cor, que envolveram a substituição de cores codificadas por variáveis CSS.

/ diretório (ficheiros de configuração)

- `components.json` foi usado para a configuração do ShadCN para adicionar componentes no mesmo estilo sempre que um novo fosse adicionado a partir da CLI.
- `tsconfig.json` era o ficheiro de configuração TypeScript.
- `tailwind.config.ts` era para as CSS do Tailwind.
- `.dockerignore` foi utilizado para especificar ficheiros a ignorar durante as implementações do Docker.
- `.env` era o ficheiro da variável de ambiente.
- `.env.docker` era o ficheiro de variáveis de ambiente específico do Docker.
- `.env.example` serviu como um exemplo de variáveis de ambiente.
- `eslintrc.json` era o ficheiro de configuração do ESLint.
- `.gitignore` foi usado para especificar ficheiros a ignorar no Git.
- `.prettierignore` foi utilizado para especificar ficheiros a ignorar no Prettier.
- `.bun.lock` era o ficheiro de bloqueio para o gestor de pacotes Bun.
- `docker-compose.yaml` foi usado para a configuração do Docker Compose.
- `Dockerfile` era o ficheiro para construir imagens Docker.
- `eslint.config.mjs` era o ficheiro de configuração do ESLint em formato de módulo.
- `global.config.ts` foi utilizado para definições de configuração global; foi adicionado para constantes JavaScript utilizadas em vários componentes.
- `118n.config.ts` era o ficheiro de configuração para a internacionalização.
- `middleware.ts` foi utilizado para as funções de middleware.
- `next.config.ts` era o ficheiro de configuração para Next.js com três modificações:
    - `output: standalone` foi utilizado para otimizar o tamanho da compilação, filtrando ficheiros desnecessários.
    - `compress: false` foi definido para as definições de compressão.

Foi adicionada uma configuração do `webpack` para carregar SVGs locais para estilização dinâmica.

- `nginx.conf` era o ficheiro de configuração para o Nginx.
- `package.json` era o ficheiro para gerir as dependências do projeto.
- `README.md` era o ficheiro de documentação.
- `postcss.config.mjs` foi incluído para a configuração do PostCSS e não foi modificado.
- `next-env.d.ts` era o ficheiro de definição TypeScript para Next.js e não foi modificado.

FRONT-END

Estilo

Para além dos componentes ShadCN pré-estilizados, foi utilizado o Tailwind CSS - uma estrutura CSS de utilidade primária que permitiu um desenvolvimento rápido da IU - juntamente com o CSS padrão. Os estilos em linha também foram usados em alguns casos, particularmente porque as classes Tailwind geradas dinamicamente, como `bg-${chosenColor}`, não funcionariam devido ao facto de o Tailwind purgar classes não utilizadas na produção e não reconhecer nomes de classes criados dinamicamente.

CSS padrão

- `globals.css` continha classes CSS e variáveis CSS utilizadas globalmente.
- `scattered-profiles.module.css` eram módulos CSS utilizados no painel de visualização de mensagens nas páginas de visualização de mensagens. Foram fornecidas mais informações no capítulo "PAGES".

O TailwindCSS foi utilizado durante todo o projeto. Serviu como a principal fonte de verdade e foi recomendada a utilização do Tailwind sempre que possível.

O Inline-CSS foi utilizado minimamente nos casos em que não existia outra opção ou em que o estilo era muito específico e seria anulado se fosse utilizado o CSS padrão.

ShadCN com temas dinâmicos

A ShadCN era uma biblioteca de componentes de IU concebida para a criação de aplicações Web modernas com temas personalizáveis e centrada na experiência do utilizador.

Inicializar o projeto:

Foi criado um novo projeto Next.js com o Shad CN UI.

Instalar dependências:

- `next-themes` foi instalado para alternar entre os modos claro e escuro.
- `O Lucide React` foi instalado para os ícones.

Foram adicionados os plugins Tailwind e Prettier VSCode para formatação.

Configurar CSS global (`/app/globals.css`):

As variáveis CSS da página de temas do Shad CN UI foram copiadas para o ficheiro CSS global.

Definir as cores do tema (`theme.colors.ts`):

Foi criada uma interface para cores temáticas e as cores disponíveis foram definidas com base nos temas do Shad CN UI.

Converter variáveis CSS (`/lib/theme.colors.ts`):

As variáveis CSS da cor do tema do Shad CN UI foram convertidas num objeto JavaScript.

Criar função de tema (`/lib/theme.colors.ts`):

Foi desenvolvida uma função para substituir variáveis CSS globais para alterações de cor do tema em tempo real.

Contexto e fornecedor de dados temáticos (`theme-data-provider.tsx`):

Foi implementado um estado para evitar a oscilação entre as cores predefinidas e as cores guardadas no carregamento inicial.

Foi exportada uma função auxiliar para aceder ao contexto do tema em toda a árvore de componentes.
Foi criado um componente Theme Data Provider para gerir o estado do tema e o armazenamento local.
Envolvimento com o fornecedor do tema seguinte (`theme-data-provider.tsx`):
O fornecedor de dados do tema foi encapsulado dentro de um fornecedor do tema seguinte no esquema de nível superior.

Com a configuração básica implementada, foi criado um botão para alternar entre os modos claro e escuro, ligando-o à função `setTheme`. Foi desenvolvido um menu pendente para selecionar as cores do tema, utilizando as cores definidas nos temas.

Para mais informações sobre estes comutadores de ajuste frontal, consultar a página de ajuste no capítulo "PÁGINAS".

Painéis redimensionáveis em React

Esta biblioteca foi criada para componentes React para grupos de painéis redimensionáveis/layouts. Foi utilizada para obter um layout com 2 ou 3 painéis horizontais, e foi escolhida pela sua facilidade de utilização e boa integração com o ShadCN.

Os cálculos para manter os tamanhos das diferentes colunas foram difíceis mas necessários. Foi feito armazenando a contribuição percentual de cada coluna como uma matriz nos cookies e recuperando-a no esquema de raiz para a transmitir aos componentes. O painel mais à esquerda dos 3 foi o mais difícil de configurar, uma vez que também tinha a funcionalidade de colapsar quando era atingida uma determinada largura.

Componentes personalizados relevantes incluídos:

- `resizable-panel-wrapper`, que envolveu todos os componentes `ResizablePanel` ShadCN, só foi utilizado uma vez no componente `app-layout.tsx`.
- `O painel infantil tratava` ele próprio da sua lógica de dimensionamento e era utilizado em quase todas as páginas que utilizavam a disposição baseada em vários painéis.

Havia também a opção de fazer um esquema de 2 colunas sem painéis redimensionáveis, mas a empresa quis aceitar o desafio deste esquema único, que raramente era visto na Web.

PÁGINAS

Iniciar sessão (`/login`)

A página de início de sessão foi fácil e rápida de implementar. Foi colocada fora dos grupos de layout principais, mas dentro de `/app/[locale]/`, o que era necessário para que tivesse o locale atual.

A página continha um formulário simples gerido principalmente por um componente (`/components/login-form.tsx`). Utilizava o componente de cartão ShadCN e incluía 2 campos: um para o e-mail e outro para a palavra-passe. A entrada da palavra-passe continha um botão para alternar o seu tipo entre "texto" e "palavra-passe", fornecendo o comportamento clássico dos botões de mostrar palavra-passe em formulários Web.

Envio de formulário no cliente
Uma vez que o redireccionamento em caso de sucesso do lado do servidor causava problemas, foi decidido utilizar o redireccionamento do router do lado do cliente, levando à implementação do "Cenário 2" encontrado na secção "Configurações de formulários" do capítulo "REGRAS PARA A CONSISTÊNCIA". Primeiro, mostrava os erros através de brindes. Depois, se a resposta do servidor fosse bem sucedida, sincronizava o armazenamento local com as definições da base de dados do utilizador e redireccionava programaticamente para `/`. Também mantinha um estado pendente para desativar elementos durante a submissão.

Submissão do formulário no servidor
O formulário chamou a ação do servidor `de início de sessão` (`/lib/auth/index.ts`), cuja lógica foi explicada na secção "Fluxo de autenticação" do capítulo "AUTENTICAÇÃO E AUTORIZAÇÃO".

Nova mensagem (`/new-message`)

A nova página de mensagens foi, de longe, a página mais complicada de construir. Para navegar até ela, basta clicar no grande botão de cor primária na barra lateral esquerda. O formulário em si na página foi tratado em `/components/new-message-form.tsx` e `/components/recipients-input.tsx`. No entanto, a lógica principal e os estados foram armazenados num contexto dedicado em `/contexts/use-new-message.tsx` para separações.

O formulário de nova mensagem era composto por quatro campos principais visíveis:

Campo do remetente: Campo estático desativado que foi codificado para ser "ETPZP".
Campo Destinatários: componente de entrada personalizada complexa.
Campo Assunto: este campo também altera o título quando este é alterado.
Campo de mensagem: Área de texto utilizada para guardar o conteúdo da mensagem.
A página também continha alguns outros botões:
O botão **Guardar rascunho** foi utilizado para mostrar o estado atual do rascunho (guardado ou não, com os respectivos erros na dica de ferramenta) e também permitiu ao utilizador guardar o rascunho manualmente.

O ecrã completo disponível no ambiente de trabalho permitia ao utilizador ocultar outros elementos da página.

Fechar era uma ligação para `/sent`.

Descartar (no canto inferior esquerdo) era uma ligação para `/enviar` que também apagava o rascunho quando era clicado.

Enviar (no canto inferior direito) mostrava se a mensagem estava agendada ou se devia ser enviada agora e submetia o formulário. Também tinha um pequeno menu lateral que permitia ao utilizador agendar o envio da mensagem.

Apresentação do formulário no cliente

Seguiu o "Cenário 2" encontrado na secção "Configurações de formulários" no capítulo "REGRAS DE CONSISTÊNCIA". Toda a validação do lado do cliente e a exibição de erros do servidor foram tratadas na função `handleSubmit` com brindes para mensagens de erro. O servidor retornou vários sinalizadores, strings de tradução e dados que decidiram como os erros foram exibidos e traduzidos no front-end. No caso de erros `zod`, os erros eram repetidos e exibidos como mensagens de brinde separadas. Cada entrada também tinha certas animações como vermelho a piscar ou apenas um sublinhado vermelho ou um espaço reservado vermelho para mostrar aos utilizadores o que causou o erro. Também mantém um estado pendente para desativar elementos durante a submissão do formulário.

Envio de formulário no servidor

Existia uma ação do servidor para enviar mensagens chamada `sendMessage` localizada em `lib/actions/message.create.ts`. A função começou por efetuar um par de verificações de segurança que, quando falhavam, faziam com que a função saísse mais cedo:

A autenticação do utilizador foi verificada (linha 22 - 30)

A validação do campo foi efectuada com `zod` (linhas 32 - 48)

Foi efectuada uma validação personalizada mais aprofundada para os destinatários (linhas 50 - 60 e linhas 259 - 276)

Em seguida, os dados foram preparados para a chamada da API e a API foi chamada utilizando fetch (linha 62 - 100).

Depois disso, começou a lógica da base de dados.

Foi feita uma verificação principal para ver se a mensagem já tinha sido guardada na base de dados como rascunho, caso em que o rascunho foi atualizado (linha 103 - 157). Caso contrário, era inserida uma nova mensagem (158 - 200). Foi guardado o máximo de informações sobre a mensagem, incluindo os erros da API, caso existissem.

Após a inserção da mensagem, os destinatários foram tratados separadamente (202 - 225). Os destinatários antigos existentes eram primeiro eliminados e, em seguida, eram inseridos novos destinatários.

As respostas eram enviadas de volta ao cliente com cadeias de tradução específicas para cada caso, que eram traduzidas no lado do cliente (linhas 228 - 257).

Entrada de destinatários personalizada

O componente em `/components/recipients-input.tsx` era mais do que um simples input, era um componente personalizado. Como primeira funcionalidade, permitia ao utilizador escrever qualquer cadeia de caracteres e premir enter ou tab para a adicionar

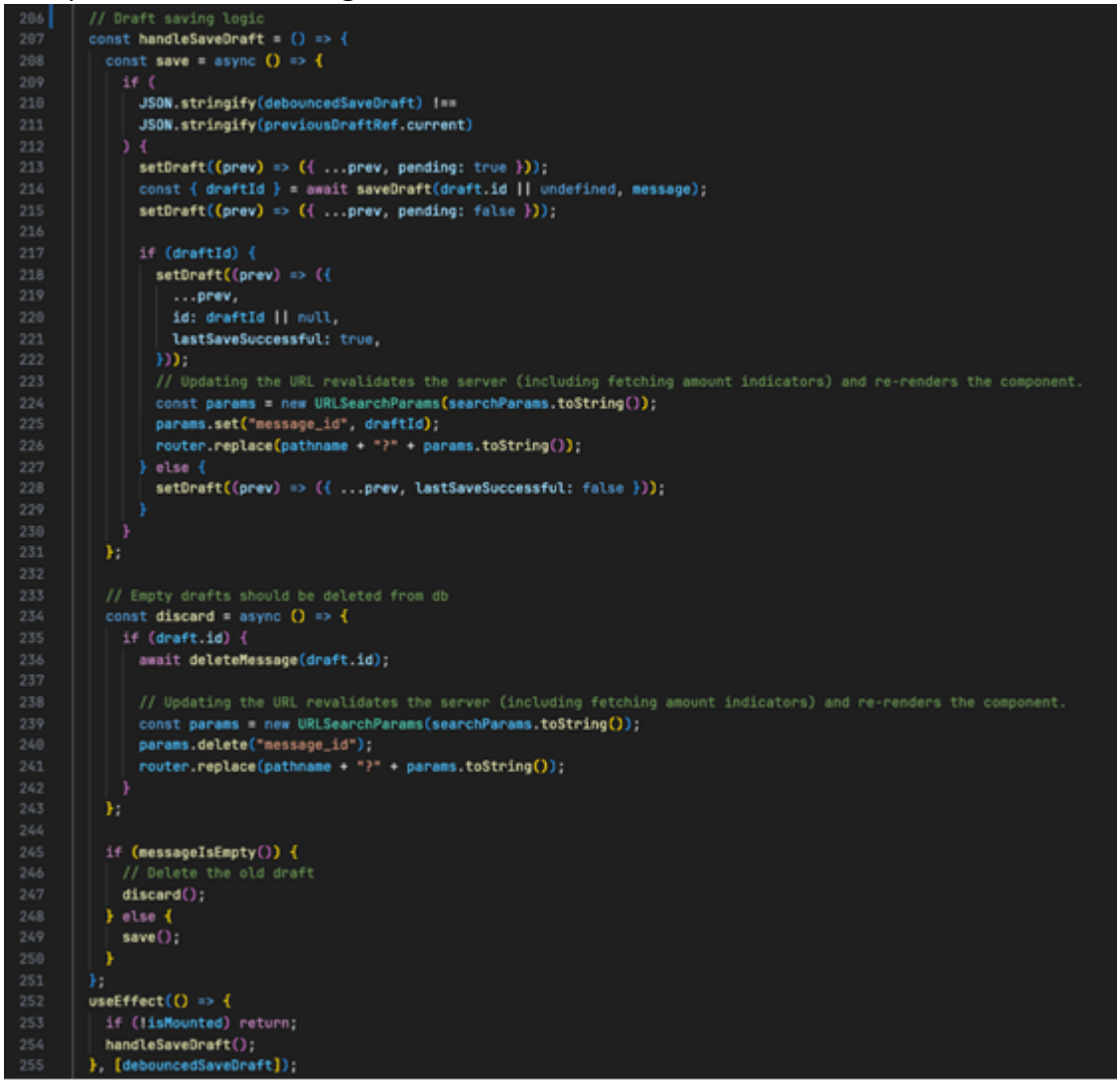
como novo destinatário. O sistema fazia uma validação do lado do cliente para detetar o que poderia estar errado com o número.

Como segunda grande caraterística, aparecia uma janela quando o utilizador começava a escrever, mostrando os destinatários que podia inserir. Este elemento personalizado absolutamente posicionado comportava-se da seguinte forma:

Nem sequer era apresentado se não existissem destinatários ou contactos.
Se a entrada estivesse vazia mas focada, a janela continha "destinatários recomendados" que eram calculados com base na utilização na última semana, bem como se tinham sido guardados como contactos ou não. Se não existirem destinatários suficientes que tenham sido utilizados nas mensagens, o resto foi preenchido com contactos não utilizados, caso existissem.
Se a entrada não estivesse vazia e focada, a janela continha os "resultados da pesquisa", que eram os destinatários filtrados e os contactos baseados no valor que o utilizador colocou na entrada.
O utilizador pode adicionar estes destinatários/contactos a partir do seu teclado, navegando com as setas para cima e para baixo e inserindo-os utilizando enter ou tab. Ou pode simplesmente clicar no destinatário da sua escolha.
Ao adicionar um, este era removido dos resultados de pesquisa ou recomendações simultâneas, uma vez que o utilizador não deveria poder adicionar o mesmo destinatário duas vezes. No entanto, se o utilizador tentasse introduzir um número de telefone que já existisse nos destinatários, este caso também era tratado e era apresentada uma mensagem de erro como brinde.

Sistema de rascunho automático
Optou-se por que, após um período de arrefecimento, o rascunho fosse automaticamente guardado, desde que pelo menos um campo tivesse um valor. Se os campos estivessem todos vazios, o rascunho existente era novamente eliminado da base de dados.
A lógica de gravação do rascunho foi tratada na função `handleSaveDraft` em

/components/new-message-form.tsx



```
206  // Draft saving logic
207  const handleSaveDraft = () => {
208    const save = async () => {
209      if (
210        JSON.stringify(debouncedSaveDraft) !==
211        JSON.stringify(previousDraftRef.current)
212      ) {
213        setDraft((prev) => ({ ...prev, pending: true }));
214        const { draftId } = await saveDraft(draft.id || undefined, message);
215        setDraft((prev) => ({ ...prev, pending: false }));
216
217        if (draftId) {
218          setDraft((prev) => ({
219            ...prev,
220            id: draftId || null,
221            lastSaveSuccessful: true,
222          }));
223          // Updating the URL revalidates the server (including fetching amount indicators) and re-renders the component.
224          const params = new URLSearchParams(searchParams.toString());
225          params.set("message_id", draftId);
226          router.replace(pathname + "?" + params.toString());
227        } else {
228          setDraft((prev) => ({ ...prev, lastSaveSuccessful: false }));
229        }
230      }
231    };
232
233    // Empty drafts should be deleted from db
234    const discard = async () => {
235      if (draft.id) {
236        await deleteMessage(draft.id);
237
238        // Updating the URL revalidates the server (including fetching amount indicators) and re-renders the component.
239        const params = new URLSearchParams(searchParams.toString());
240        params.delete("message_id");
241        router.replace(pathname + "?" + params.toString());
242      }
243    };
244
245    if (messageIsEmpty()) {
246      // Delete the old draft
247      discard();
248    } else {
249      save();
250    }
251  };
252  useEffect(() => {
253    if (!isMounted) return;
254    handleSaveDraft();
255  }, [debouncedSaveDraft]);
```

Se o componente estivesse montado, era chamado a partir de um `useEffect` que era acionado por uma constante que recebia alterações após o debounce sem alterações, accionando o useEffect apenas após esse useDebounce. Foi criado um gancho personalizado para este comportamento em `/hooks/use-debounce.tsx` (linha 252 - 255).

A função verificou então se a mensagem estava vazia e chamou a função correta em conformidade (linhas 245 - 250).

A função de `guardar` verificou se o rascunho atual tinha sido alterado em relação ao rascunho anterior, guardou-o se o tivesse feito, actualizou o ID e o estado do rascunho com base no resultado de guardar e modificou o URL para refletir o novo ID do rascunho enquanto revalidava o servidor (208 - 231).

A função de `eliminação` eliminou o rascunho atual da base de dados, caso tivesse um ID, e actualizou o URL para remover o ID do rascunho, o que revalidou o servidor e voltou a renderizar o componente (234 - 243).

Modais

Esta página utilizou os seguintes modais:

- `schedule-modals.tsx` continha um modal para selecionar uma data de calendário e outro para avisar o utilizador de que a data era inválida.
- `recipient-info.tsx` era mostrado quando um utilizador clicava numa ficha de destinatário, apresentando informações adicionais sobre o destinatário (ou contacto) selecionado.

Desafios
Em primeiro lugar, havia um problema com a gravação do projeto. Sempre que o URL era atualizado (mesmo que apenas com os parâmetros de pesquisa do URL), fazia com que todos os componentes dessa página fossem novamente renderizados, uma vez que o componente do servidor de nível superior recuperava o parâmetro `message_id` do URL para ir buscar os dados do rascunho. Esta nova apresentação levou a que todos os campos perdessem os seus valores, incluindo popups ou menus popover anteriormente abertos, que também ficavam ocultos. Para resolver este problema, foi criado um contexto que mantinha todos os valores durante as novas apresentações.

Além disso, a criação da janela de destinatários sugeridos com todas as suas funcionalidades e a garantia de que não tinha erros demorou muito tempo. Era difícil encontrar uma configuração que tivesse sempre os valores mais actualizados e, à medida que o new-message-context aumentava, tornava-se cada vez mais difícil trabalhar com ele.

Definições (`/settings`)

Determinar a arquitetura do código para as definições foi um desafio devido à falta de orientações claras. Preferindo actualizações automáticas sempre que uma definição era modificada, os botões de guardar foram evitados. A página de definições apresentava uma configuração personalizada em que algumas definições eram geridas por bibliotecas e outras com uma implementação personalizada.

Embora a maioria das configurações tenha sido salva no armazenamento local, os dados do tema e o idioma atual foram armazenados em cookies devido à forma como as bibliotecas os manipularam. No entanto, a atualização direta do armazenamento local não actualizava os componentes React. Para resolver isso, um contexto de configurações foi criado (`/contexts/use-settings.tsx`), que gerenciava o estado das configurações e incluía várias funções auxiliares.

Foi criada uma ação de servidor chamada `updateSetting` que actualizava as definições individuais uma de cada vez (`/lib/actions/user.actions.ts`), o que levou à criação de vários formulários. Esta abordagem, embora resultasse em mais formulários, permitia uma gestão mais fácil através de uma lógica centralizada (`/components/settings-item.tsx`).

Componentes reutilizáveis
Devido à natureza repetitiva das definições, foram desenvolvidos componentes reutilizáveis: um `SectionHeader` para as categorias de definições e um `SettingsItem` para as definições individuais.

O componente `SectionHeader` (`/components/headers.tsx`) era mais simples, uma vez que era necessário passar o título e a legenda a apresentar, juntamente com o nome da etiqueta de ancoragem.

O componente `SettingsItem` (`/components/settings-item.tsx`) era mais complexo, pois continha todo o tratamento de erros e a lógica pendente. Tornou-se personalizável adicionando uma prop `renderInput`, que permitia a passagem de HTML completamente personalizado para entrada, enquanto ainda fornecia acesso aos manipuladores de envio do banco de dados e outros dados importantes. Cada um destes formulários aderiu ao "Cenário 2" encontrado na secção "Configurações de formulários" do capítulo "REGRAS DE CONSISTÊNCIA".

Vale a pena mencionar que o alterador de idioma utilizava a função `updateLanguageCookie`, que não podia ser implementada sem utilizar o router Next.js para substituir e atualizar internamente. Devido a este refrescamento interno, causava um reset, necessitando de um componente próprio devido ao aumento da complexidade da alteração da língua.

Considerações
Inicialmente, considerou-se a possibilidade de utilizar um único formulário para todas as definições, mas este foi rejeitado devido a problemas de desempenho e legibilidade. Um único formulário complicaria o tratamento, exigindo que todo o conjunto de definições fosse enviado para o servidor para cada modificação, o que dificultaria a validação e o tratamento de erros.

Painel de controlo do administrador (`/dashboard`)

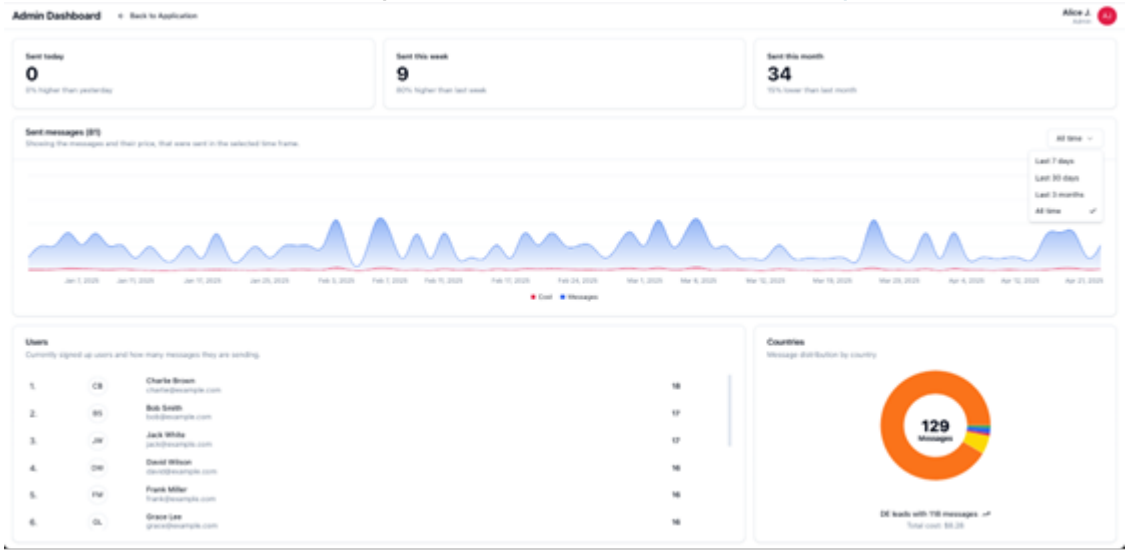
O painel de controlo administrativo foi construído em último lugar e incluía informações estatísticas. Foi colocado fora dos grupos de apresentação principais, mas dentro de `/app/[locale]/`, o que era necessário para que tivesse a localização atual.

A página só estava acessível aos administradores, tal como explicado na secção "Fluxo de autenticação" do capítulo "AUTENTICAÇÃO E AUTORIZAÇÃO". A biblioteca ReCharts foi usada para os gráficos de área e de pizza responsivos. Para colorir o gráfico de área, ela recuperou a cor primária do tema e a cor do perfil. Para colorir o gráfico de pizza, foi usada a cor primária de cada tema. A ordem foi aleatória e as cores foram guardadas num estado para que se alterassem durante as novas renderizações de componentes causadas por utilizadores que modificassem a data.

A página incluía:

3 cartões na parte superior que mostram o número de mensagens enviadas em comparação com o passado.
Um gráfico de área que mostra as mensagens e o custo desde um determinado momento.
Uma opção que alterava a data de início dos outros gráficos.
Uma tabela de utilizadores classifica os utilizadores registados com base nas mensagens enviadas desde a data de início selecionada. O parâmetro de pesquisa `end_date` também poderia ser injetado no URL e a aplicação aplicaria o filtro para uma data final.

Um gráfico circular que apresenta informações sobre os países dos números de telefone dos destinatários, obtidas a partir da API de estatísticas de etiquetas.



Filtragem de datas

A alternância da data de início, encontrada em `/components/admin-dashboard/message-area-chart.tsx`, era um menu pendente Selecionar que substituía o URL atual por um

novo URL com parâmetros de pesquisa actualizados sempre que o seu valor era alterado.

```
<Select
  defaultValue={searchParams.get("start_date") || DEFAULT_START_DATE}
  onValueChange={(value) => {
    const params = new URLSearchParams(searchParams);

    if (value) {
      params.set("start_date", value);
    } else {
      params.delete("start_date");
    }
    if (params.has("end_date")) params.delete("end_date");
    router.replace(`${pathname}?${params.toString()}`, {
      scroll: false, // persist current scroll for better ux
    });
  }}
>
  <SelectTrigger
    className={cn(
      buttonVariants({ variant: "outline" }),
      "w-min appearance-none font-normal justify-between"
    )}
    // className="w-[160px] rounded-lg sm:ml-auto"
    aria-label={t("common:aria_label-select")}
  >
    <SelectValue placeholder={t("area_chart-3_months")} />
  </SelectTrigger>
  <SelectContent align={onMobile ? "center" : "end"}>
    {selectItems.map((item) => (
      <SelectItem key={item.date.getTime()} value={toISO(item.date)}>
        {item.label}
      </SelectItem>
    ))}
    {selectedStartDate.ISO_date &&
      !selectItems.some(
        (item) => toISO(item.date) === selectedStartDate.ISO_date
      ) && (
        <SelectItem value={selectedStartDate.ISO_date} disabled>
          {selectedStartDate.isValid
            ? format(
                new Date(selectedStartDate.ISO_date),
                PT_DATE_FORMAT_NO_TIME
              )
            : selectedStartDate.ISO_date}
        </SelectItem>
      )}
  </SelectContent>
</Select>
```

Obtenção de dados

Uma vez que se esperavam conjuntos de dados maiores após algum tempo de implementação da aplicação, foi utilizado o "Cenário 2" da secção "Obtenção conservadora de dados" em "REGRAS PARA A CONSISTÊNCIA". Isto significava que os dados eram obtidos no componente de servidor de nível superior, onde o parâmetro URL era recuperado e passado para as funções de obtenção de backend. Sempre que os parâmetros do URL eram alterados, era feita uma nova renderização, fazendo com que os

dados fossem actualizados.

```
12  export default async function Dashboard({
13    searchParams,
14  }: {
15    searchParams?: Promise<{
16      // We expect both of these to be in ISO 8601 format (YYYY-MM-DD)
17      start_date?: string;
18      end_date?: string;
19    }>;
20  }) {
21    const s = await searchParams;
22    const dateRange = {
23      startDate: s?.start_date || format(DEFAULT_START_DATE, ISO8601_DATE_FORMAT),
24      endDate: s?.end_date || format(new Date(), ISO8601_DATE_FORMAT),
25    };
26    const messages = await fetchMessagesInDateRange(dateRange);
27    const users = await fetchUsers();
28    const countryData = await fetchCountryStats(dateRange);
29
30    return (
31      <AdminDashboard
32        messages={messages || []}
33        users={users || []}
34        countryStats={countryData}
35      />
36    );
37  }
```

Os dados do componente do servidor de nível superior foram então passados para o componente do cliente `AdminDashboard`, onde foram efectuados cálculos e formatação de dados adicionais.

```
26  export default function AdminDashboard({
27    messages,
28    users,
29    countryStats,
30  }: {
31    messages: LightDBMessage[];
32    users: DBUser[];
33    countryStats: CountryStat[] | undefined;
34  }) {
35    const { t } = useTranslation(["dashboard-page", "errors", "common"]);
36    const messageCounts = countMessages(messages);
37    const { settings } = useSettings();
38    const onMobile = useIsMobile();
39    const onBigScreen = false;
40
41    return (
```

Desafios
Um dos desafios foi conseguir que o gráfico circular funcionasse. Por vezes, não era apresentado. Mais tarde, descobriu-se que isso se devia a uma altura demasiado pequena, pelo que foi adicionada uma altura fixa ao seu contentor principal.

Além disso, teve de ser implementada uma dica de ferramenta personalizada para o gráfico circular, que foi inspirada na dica de ferramenta do gráfico de área para manter a coerência do design.

Outras páginas

Estas páginas eram muito semelhantes:

- `/sent` para mensagens enviadas
- `/scheduled` para mensagens agendadas com uma hora de envio no futuro. Uma vez atingida a hora agendada, ela aparecia em `/sent`
- `/failed` para mensagens de falha em que ocorreu um erro do lado da API ou foi cancelado pelo utilizador
- `/drafts` para mensagens em `rascunho` que tinham sido guardadas mas não enviadas, permitindo aos utilizadores editá-las ou finalizá-las antes de as enviar
- `/trash` para as mensagens no lixo, onde é possível recuperá-las ou apagá-las permanentemente
- `/contacts` para contactos - os contactos continham informações adicionais, como o número de telefone, o nome e uma descrição. Esta página era ligeiramente diferente das outras, mas era suficientemente semelhante para ser colocada no mesmo esquema.

Layout partilhado
As páginas deste capítulo viviam no mesmo grupo de rotas devido à sua semelhança (`/app/[locale]/(root)/(message-layout)/`) que partilhava o mesmo layout e ficheiros de tratamento de erros. O layout envolveu as páginas filhas com um provedor para o contexto de tradução enquanto carregava os namespaces necessários. Uma vez que as páginas necessitavam de acesso aos contactos, as páginas-filhas foram agrupadas com um fornecedor para o contexto de contactos, passando alguns contactos iniciais (linhas 34-

36).

```tsx
import initTranslations from "@/app/i18n";
import TranslationsProvider from "@/contexts/translations-provider";
import { ContactsProvider } from "@/contexts/use-contacts";
import { fetchContacts } from "@/lib/db/contact";

type LayoutProps = Readonly<{
  children: React.ReactNode;
  params: Promise<{ locale: string }>;
}>;

export default async function TranslationLayout({
  children,
  params,
}: LayoutProps) {
  // Internationalization (i18n) stuff
  const i18nNamespaces = [
    "messages-page",
    "contacts-page",
    "modals",
    "common",
    "errors",
  ];
  const { locale } = await params;
  const { resources } = await initTranslations(locale, i18nNamespaces);

  return (
    /* This is a client layout component containing the translation provider for the nav panel */
    <TranslationsProvider
      /* Only wrap what's necessary with the TranslationsProvider */
      resources={resources}
      locale={locale}
      namespaces={i18nNamespaces}
    >
      <ContactsProvider initialContacts={(await fetchContacts()) || []}>
        {children}
      </ContactsProvider>
    </TranslationsProvider>
  );
}

```

Arquitetura da página

- `messages-page.tsx` como suporte para os outros componentes
- `messages-list.tsx` que apresentava os resultados da pesquisa de contactos (coluna do meio)
- `message-display.tsx` que exibia a mensagem em si e também estava envolvida no componente de painel filho. Mais detalhes sobre isso foram fornecidos na seção "Painéis redimensionáveis React" do capítulo "FRONT-END".

Pesquisa/filtragem

O componente `search.tsx` era a interface de utilizador da barra de pesquisa utilizada para pesquisar mensagens e contactos. Chamava a função passada (`onSearch`) após as alterações de entrada e mantinha a consulta do utilizador no URL para poder ser marcada e actualizada acidentalmente. Esta atualização do URL não actualizou os componentes do servidor, uma vez que não foram utilizados os ganchos Next.js.

```tsx
"use client";

import { Input } from "@/components/ui/input";
import { Search as SearchIcon } from "lucide-react";

type SearchProps = React.InputHTMLAttributes<HTMLInputElement> & {
  onSearch: (term: string) => void;
};

export default function Search({ onSearch, ...props }: SearchProps) {
  const url = new URL(window.location.href);
  const params = new URLSearchParams(url.search);
  const handleSearch = (term: string) => {
    // update data by calling parent function
    onSearch(term);

    // Use vanilla javascript to update the url.
    // According to the Next.js docs we should use useSearchParams, usePathname, and useRouter, but that causes the component to re-render and re-fetch data.
    // For optimization purposes, we just fetch once for each page, and then filter that data using client-side javascript.

    // Update the search parameter or delete it if search bar is empty
    if (term) {
      params.set("query", term);
    } else {
      params.delete("query");
    }

    // Update the URL without reloading the page
    url.search = params.toString();
    window.history.pushState({}, "", url);
  };
  return (
    <div className="p-4">
      <div className="relative">
        <SearchIcon className="absolute left-2 top-2.5 h-4 w-4 text-muted-foreground" />
        <Input /** this input is not part of a form, we are just using the input element as it has handy event listeners */
          onChange={(e) => {
            handleSearch(e.target.value);
          }}
          className="focus-visible:ring-1 focus-visible:ring-primary"
          defaultValue={params.get("query")?.toString()}
          {...props}
        />
      </div>
    </div>
  );
}

```

As funções `searchMessages` e `searchContacts` filtraram as respectivas matrizes com base num termo de pesquisa fornecido pelo utilizador, permitindo a pesquisa no lado do cliente. Ambas as funções converteram o termo de pesquisa para minúsculas para comparação sem distinção entre maiúsculas e minúsculas. `searchMessages` procurou correspondências no assunto, corpo ou estado da mensagem, enquanto `searchContacts` procurou o termo no nome ou número de telefone do contacto. Se não for fornecido

nenhum termo de pesquisa em `searchContacts`, é devolvida a lista original de contactos.

```
75  export function searchMessages(
76    messages: DBMessage[],
77    searchTerm: string,
78    currentPage?: number
79  ) {
80    // Convert searchTerm to lowercase for case-insensitive comparison
81    const lowerCaseSearchTerm = searchTerm.toLowerCase();
82
83    // Filter messages based on userId and search term
84    const filteredMessages = messages.filter(
85      (message) =>
86        message.subject?.toLowerCase().includes(lowerCaseSearchTerm) ||
87        message.body.toLowerCase().includes(lowerCaseSearchTerm) ||
88        message.status.toLowerCase() === lowerCaseSearchTerm // Assuming status is also part of the search
89    );
90
91    return filteredMessages;
92  }
93
94  export function searchContacts(
95    contacts: DBContact[],
96    searchTerm: string | null,
97    currentPage?: number
98  ) {
99    if (!searchTerm) return contacts;
100   // Convert searchTerm to lowercase for case-insensitive comparison
101   const lowerCaseSearchTerm = searchTerm.toLowerCase().trim();
102
103   // Filter contacts based on userId and search term
104   const filteredContacts = contacts.filter(
105     (contact) =>
106       (contact.name &&
107         contact.name.toLowerCase().includes(lowerCaseSearchTerm)) ||
108       contact.phone.toLowerCase().includes(lowerCaseSearchTerm)
109   );
110
111   return filteredContacts;
112 }
```

Ecrã de mensagens

Estas páginas deste capítulo, exceto os contactos, tinham estes botões no visor de mensagens:

O botão **Reenviar** servia para pegar em todos os campos de uma mensagem e inseri-los novamente no formulário de nova mensagem. Funcionava criando primeiro um novo rascunho na base de dados e depois passando o ID para o parâmetro message_id na página `/new-message`.

O botão **Mover para o lixo** servia para mover mensagens para o lixo. Na própria página do lixo, a mensagem era eliminada da base de dados.

O botão **Fechar** serve para anular a seleção do item atualmente selecionado, apresentado na coluna da extrema direita. Botões específicos da página:

A **página agendada** também tinha um botão para **cancelar** a mensagem **agendada**, que cancelava o SMS e obtinha um reembolso através da API, movendo a mensagem para falhada. Isto foi útil para testar a aplicação sem custos.

A **página do lixo** também tinha um botão de **recuperar mensagem**, que movia a mensagem de volta para a sua localização original, recuperando-a do lixo.

Foi decidido que cada mensagem apresentaria os seus destinatários em formato de fichas,

que, ao serem clicadas, abririam o modal `recipient-info.tsx` para mostrar mais informações sobre o destinatário (ou contacto). Por defeito, os destinatários estavam colapsados, podendo ser expandidos clicando na pequena seta à direita.
Vale a pena mencionar o esforço para apresentar os perfis dos contactos de forma agradável. Os primeiros cinco destinatários/contactos foram apresentados num pequeno elemento de visão geral com os círculos dos seus perfis. O seu estilo foi tratado no ficheiro `scattered-profiles.module.css` utilizando módulos CSS. Os tamanhos e posições foram codificados, mas as cores foram randomizadas armazenando um array embaralhado em um estado, e cada vez que uma nova mensagem era selecionada, o procedimento era repetido.

Página de contactos
Embora também fosse muito semelhante, tinha os seus próprios componentes porque os dados eram completamente diferentes e o código precisava de ser mantido limpo:

- `contacts-page.tsx` em vez de `messages-page.tsx`.
- `contacts-list.tsx` em vez de `messages-list.tsx`.
- `contact-display.tsx` em vez de `message-display.tsx`.
  Decidiu-se que esta página inclui um botão para criar, editar e apagar contactos na base de dados.

Modais
Decidiu-se que todas as páginas mencionadas envolvem a sua componente de visualização num fornecedor de modais, um fornecedor de um contexto para gerir os modais que estão atualmente abertos.
As páginas de contacto utilizam estes modais:

- `edit-contact.tsx` continha um formulário para editar um contacto com uma configuração `useActionState`
- `create-contact.tsx` continha um formulário para criar um contacto com uma configuração `useActionState`
  As outras páginas utilizam este modal:
- `recipient-info.tsx` apresentou mais informações sobre um destinatário (ou contacto)

REGRAS DE COERÊNCIA

Utilização de acções do servidor

A partir do Next.js 15 com o router de aplicações, foi recomendada a utilização de acções de servidor para obter dados ou efetuar pedidos de API no backend. As acções do servidor simplificaram o desenvolvimento, permitindo aos programadores definir funções do lado do servidor invocadas diretamente a partir de componentes do cliente, fazendo automaticamente pedidos POST no backend quando a ação é chamada.

Anteriormente, os programadores tinham de criar rotas de API separadas para a obtenção de dados, o que era complicado e moroso. Na maioria dos casos, é melhor utilizar acções do servidor em vez de rotas da API. O guia a seguir foi consultado sempre que havia incerteza sobre qual deles usar.

Scenario	API route / Route handler	Server action
Fetching data from a server component		✅
AI data streaming (i.e. openai)	✅	
Fetching data from a client component		✅
Webhooks (i.e. stripe payment flow)	✅	
API request from external app	✅	

Uma vez que o Next.js recomendava tratar as acções de servidor como rotas de API públicas, era altamente recomendável **verificar a autenticação do utilizador em cada ação de servidor** para fins de segurança.

Utilização de contextos React

Introdução: O React Contexts permitiu que os desenvolvedores gerenciassem o estado global e compartilhassem dados entre componentes sem prop drilling. Isso se mostrou útil para aplicativos em que vários componentes exigiam acesso aos mesmos dados, como autenticação de usuário, temas ou configurações, levando a uma abordagem de gerenciamento de estado mais eficiente.

Como funcionou: O React Context criou um objeto de contexto para guardar dados partilhados. Um componente `Provider` envolveu partes da aplicação, tornando o valor do contexto acessível aos componentes aninhados. Os componentes que precisavam do contexto usavam o gancho `useContext` para acessar os dados, garantindo que apenas esses componentes fossem renderizados novamente quando o valor do contexto fosse alterado.

Quando foi aplicado: A regra para usar React Contexts foi estabelecida durante a fase de configuração inicial para criar uma estratégia clara de gerenciamento de estado. Como orientação, os contextos foram criados quando os dados precisavam de ser acedidos por mais de quatro componentes ou quando a perfuração de adereços se estendia para além de três camadas na árvore de componentes.

Configurações de formulários

Havia dois cenários diferentes para os formulários. Com base na complexidade, foi necessário escolher uma das seguintes opções para manter uma base de código consistente:

Cenário 1: Em situações simples, recomendou-se que o formulário fosse enviado diretamente para o servidor sem alterar o processo de envio.
Implementação: Esta configuração envolveu a utilização do gancho React `useActionState`, com a ação passada para a `action` da etiqueta do formulário.
Resposta do servidor: Se a resposta da ação fosse necessária, era necessário criar um `useEffect` com o estado do servidor na matriz de dependências.
Exemplo: Um exemplo deste cenário pode ser encontrado em `/components/modals/create-contact.tsx`.
Cenário 2: Se fosse necessário executar código aquando da submissão do formulário, interrompendo o comportamento natural de submissão, este cenário teria de ser utilizado.
Implementação: Em primeiro lugar, foi necessário criar uma função (normalmente designada `handleSubmit`) para passar para a propriedade `onSubmit` da etiqueta do formulário. Na função `handleSubmit`, a lógica especial poderia ser escrita e a ação do servidor poderia ser chamada.
Resposta do servidor: Se o HTML exigisse a resposta da ação, era necessário criar um `useState` que seria definido na função `handleSubmit`.
Exemplo: Um exemplo deste cenário pode ser encontrado em `/components/login-form.tsx`.
Estes cenários foram desenvolvidos através de experiências, pesquisas e testes exaustivos e provaram ser os mais legíveis, eficazes e eficientes.

Obtenção de componentes do servidor

O Next.js incentivou a obtenção de dados de componentes de servidor de nível superior, uma prática que foi amplamente implementada na aplicação. Ao tirar partido dos componentes do servidor para a obtenção de dados, tirou partido da renderização do lado do servidor, o que melhorou o desempenho e garantiu que todos os componentes aninhados tivessem acesso aos dados necessários sem a necessidade de obter dados adicionais do lado do cliente.

Quando foram necessárias actualizações dos dados, foi utilizada a função `revalidatePath()` da API Next.js. Ao chamar esta função no caminho específico, o componente do servidor é re-renderizado, o que, por sua vez, reenvia os dados mais recentes. Ao utilizar sempre as APIs Next.js, foi possível evitar a atualização da página (apenas a atualização interna), o que fez com que se parecesse mais com uma aplicação.

Obtenção conservadora de dados

Havia muitas formas de obter dados no Next.js. Após extensa pesquisa e testes, 2 métodos foram considerados:

Método 1: Este método envolveu a obtenção de dados do componente do servidor uma vez e, em seguida, a utilização de JavaScript do lado do cliente para efetuar a filtragem. Minimizou a carga do servidor ao executar a consulta à base de dados apenas durante o carregamento inicial da página, resultando em resultados de filtragem instantâneos.

Vantagem: Reduziu a carga do servidor devido a uma única extração de dados no carregamento da página e proporcionou uma filtragem muito rápida para conjuntos de dados de dimensão média ou inferior.

Desvantagem: Pode tornar-se lento se o conjunto de dados for demasiado grande ou se a filtragem for demasiado complexa para o JavaScript do lado do cliente, especialmente se o utilizador tiver um computador antigo.

Método 2: Este método envolveu a passagem de parâmetros de pesquisa actualizados do componente cliente para as consultas dinâmicas da base de dados. Quando os parâmetros de pesquisa eram alterados, o componente do servidor voltava automaticamente a renderizar e a atualizar a base de dados.

Vantagem: não dependia do computador do utilizador, uma vez que a filtragem era feita no servidor na consulta SQL.

Desvantagem: Aumentava a carga do servidor devido às frequentes recolhas de dados (sempre que um filtro era atualizado), bem como ao atraso na consulta da base de dados.

Devido à dimensão média do conjunto de dados (menos de 1000 mensagens por utilizador), **o método 1 foi a abordagem mais adequada** para a maioria dos casos de utilização.

Ficheiros de capa de página

Cada página criada tinha de conter pelo menos um `loading.tsx` e um `error.tsx`, sendo o layout opcional.

- `error.tsx` era um arquivo em Next.js que servia como um apanhado para erros inesperados. Ele criava um limite de erro do React que impedia que o aplicativo falhasse quando ocorriam exceções inesperadas.
- `loading.tsx` era um ficheiro mostrado enquanto a página estava a carregar, contendo todo o esqueleto da IU para essa página.
- `layout.tsx` era um layout em torno da página. Para manter o `page.tsx` mais limpo, era aconselhável colocar tudo o que fosse desnecessário no `layout.tsx`, incluindo o fornecedor de traduções e outros fornecedores, se possível.
- `page.tsx` era a página em si, que deveria ser mantida o mais limpa possível, com um componente de servidor para buscar dados mais tarde, se necessário. Inicialmente, tinha tudo configurado com um limite de suspense `React.Suspense` usado para carregar. Essa abordagem foi benéfica para implementar a pré-renderização parcial, permitindo que algumas partes da página carregassem mais rápido do que outras com um indicador de carregamento separado. No entanto, descobriu-se rapidamente que usar apenas um poderia muito bem seguir a convenção de arquivo Next.js, que mantinha os arquivos mais organizados.

Metadados

Os metadados básicos, como títulos de separadores, ícones e descrições, melhoraram a capacidade de partilha e de marcação. Como resultado, ajudou os utilizadores a identificar rapidamente o sítio Web.

Os metadados foram gerados no lado do servidor utilizando a função `generateMetadata` da API Next.js, que foi essencial para a tradução de metadados.

```
18  export async function generateMetadata({
19    params,
20  }: {
21    params: Promise<{ locale: string }>;
22  }) {
23    const { locale } = await params;
24    const { t } = await initTranslations(locale, ["metadata"]);
25
26    return {
27      title: METADATA_APP_NAME + t("sent-title"),
28      description: t("sent-description"),
29    };
30  }
```

O logótipo da aplicação foi apresentado com o nome `favicon.ico` e colocado em `/app`, que o Next.js reconheceu automaticamente e utilizou para os metadados.

Também era possível exportar os metadados de forma estática, mas, nesse caso, teria sido impossível traduzir para diferentes línguas.

BASE DE DADOS

O PostgreSQL foi escolhido pela sua fiabilidade e capacidades ricas em funcionalidades. Sendo uma base de dados relacional de código aberto, oferecia uma integridade de dados robusta e fortes caraterísticas de segurança. A conexão foi estabelecida usando pg, com o ambiente de produção sendo executado em um contêiner Docker na porta 5432. Mais informações podem ser encontradas no capítulo "DEPLOYMENT".

Inicialmente, considerou-se que **o Prisma**, um conjunto de ferramentas de base de dados e uma camada de mapeamento objeto-relacional (ORM), era utilizado juntamente com a base de dados PostgreSQL. Simplificou o acesso à base de dados, fornecendo uma **API de segurança de tipo**. No entanto, foi rejeitado para manter o projeto leve e minimizar as dependências.

Ligação à base de dados

Foi utilizada a biblioteca `node-postgres`, devido à sua forma eficiente de executar consultas SQL e obter resultados.

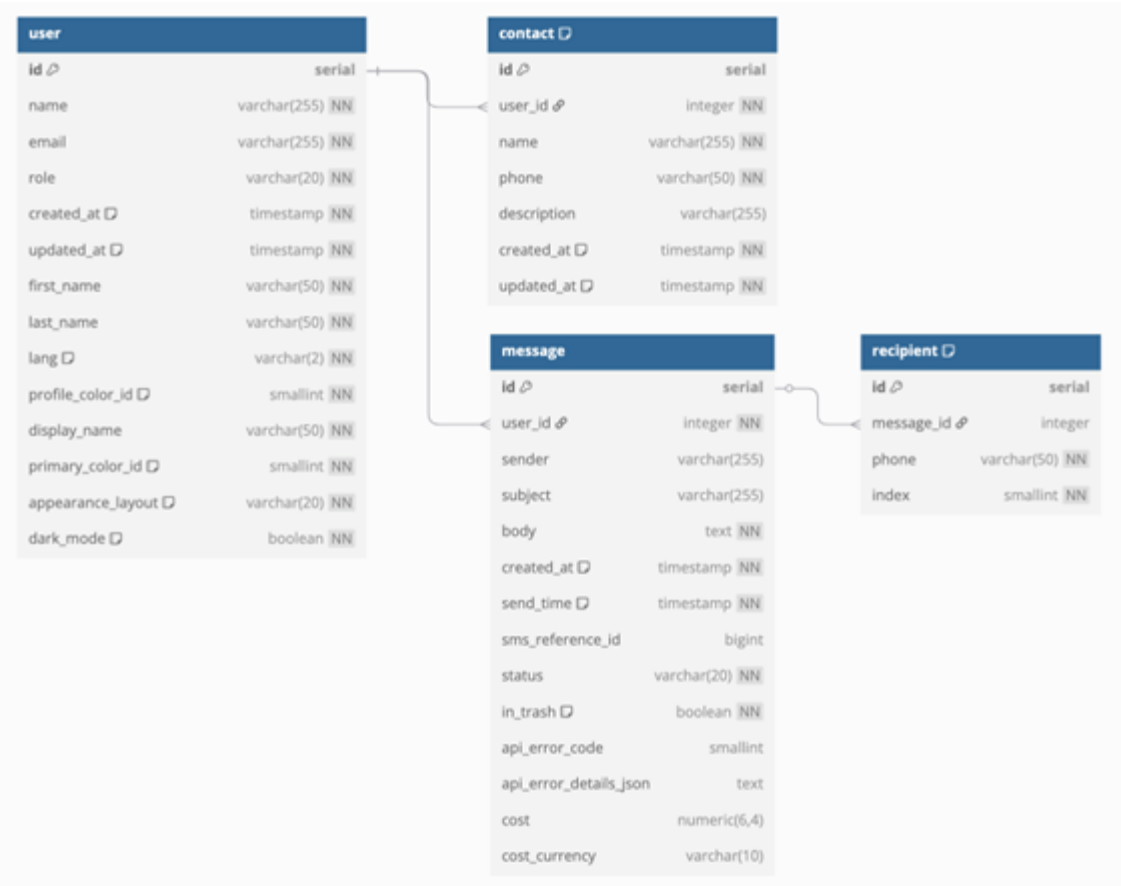
Ao conectar-se ao PostgreSQL usando pg, havia 2 opções: **pool** ou **cliente**. Um **pool** era um grupo de conexões reutilizáveis ideal para consultas concorrentes, que era utilizado devido a múltiplas consultas ao mesmo tempo. Um **cliente**, por outro lado, representava uma única conexão por interação.

Para simplificar a consulta, foi criada uma função auxiliar (`/lib/db/index.ts`) que recebia a consulta SQL e os valores a inserir. Começava por criar uma nova pool, ligava-se a essa pool para criar um novo cliente, e depois consultava-o enquanto detectava erros inesperados, e por fim libertava o cliente de volta para a pool.

```typescript
import { Pool, QueryResult } from "pg";

const pool = new Pool({
  host: process.env.POSTGRES_HOST,
  port: Number(process.env.POSTGRES_PORT),
  user: process.env.POSTGRES_USER,
  password: process.env.POSTGRES_PASSWORD,
  database: process.env.POSTGRES_DB,
});

async function db(query: string, params?: any[]): Promise<QueryResult> {
  const client = await pool.connect();
  try {
    const res = await client.query(query, params);
    return res;
  } catch (err) {
    console.error("Database query error", err);
    throw err; // Rethrow the error for handling in the calling function
  } finally {
    client.release(); // Always release the client back to the pool
  }
}

export default db;
```

Agora era tão fácil quanto importar a função auxiliar db e passar a consulta SQL e os valores. Para obter informações sobre o pg, foi consultada a documentação.

Esquema da base de dados



O esquema da base de dados, definido no ficheiro seed (`/lib/db/seed.sql`), criou quatro tabelas:

1. `O utilizador` detinha todos os dados dos utilizadores, incluindo dados e definições da conta.
2. `O contacto` continha todos os contactos, incluindo os seus dados importantes, como o nome, o número de telefone, a descrição, a data de criação e a data da última atualização.
3. `A` tabela de mensagens contém todas as mensagens, sendo que cada mensagem faz referência à chave primária da tabela do `utilizador`. Além disso, cada mensagem continha dados importantes, como o remetente, o assunto, o corpo (conteúdo do SMS), a data de envio, o estado (enviado, agendado, falhado ou rascunhado) e outros dados devolvidos pela API quando a mensagem foi enviada.
4. `recipient` continha todos os destinatários das mensagens, com cada destinatário a fazer referência à chave primária da tabela de `mensagens`. Além disso, cada destinatário era constituído por um número de telefone único e um índice utilizado para apresentar o destinatário pela mesma ordem definida pelo utilizador na página de novas mensagens durante as rendições de componentes.
   Além disso, todas as tabelas mencionadas utilizavam um campo de chave primária em série denominado `id`, utilizado para distinguir os diferentes itens.

Considerações
Uma consideração no início do projeto era ter tabelas separadas para os tipos de

mensagens (rascunhos, lixo, etc.), mas percebeu-se que era desnecessariamente complexo. Por fim, todas as mensagens foram armazenadas na mesma tabela, com cada mensagem tendo campos como `status` e `in_trash`, que determinavam em qual categoria ela seria mostrada no front-end.

Durante muito tempo, durante o desenvolvimento da aplicação, os contactos foram ligados aos destinatários utilizando a chave primária. No entanto, após três quartos do projeto, foi necessária uma migração devido a problemas de consulta e inserção, bem como a falhas gerais na arquitetura. A nova solução envolvia a verificação de contactos no front-end, passando pelos destinatários para verificar se os seus números de telefone correspondiam aos de um contacto.

Embora funcionasse desta forma, outro aspeto a melhorar era o tratamento diferente das mensagens agendadas. A partir desse momento, as mensagens agendadas permaneciam com o estado "AGENDADO" mesmo quando a data de entrega era atingida, o que não era logicamente exato. Para resolver este problema lógico, poderia ser sugerido mudar o nome do campo para outra coisa ou atualizar o estado para "ENVIADO" quando a data de entrega fosse atingida.

AUTENTICAÇÃO E AUTORIZAÇÃO

A aplicação utilizou uma combinação de Active Diretory (AD) e Iron Session para efeitos de autenticação e autorização.

Diretório Ativo

Uma vez que a escola já utilizava um servidor AD para gerir as contas informáticas dos alunos, este foi integrado na aplicação. Esta combinação tornou a gestão do acesso dos utilizadores muito mais fácil mais tarde, uma vez que as contas dos utilizadores eram geridas num único local.

O AD funcionava de forma semelhante a uma base de dados, armazenando informações sobre todos os utilizadores e respectivos dados. Neste caso, a aplicação tinha 2 grupos específicos configurados no AD: "Utilizadores-SMS" e "Administradores-SMS". Estes grupos foram utilizados para determinar as permissões que cada utilizador tinha na aplicação.

Se um utilizador fizesse parte do grupo "Utilizadores-SMS", era-lhe concedido acesso básico à aplicação, permitindo-lhe enviar mensagens SMS e gerir as suas próprias mensagens. Por outro lado, se um utilizador pertencesse ao grupo "Administradores-SMS", tinha todas as mesmas permissões que o primeiro grupo, juntamente com acesso a um painel de administração que oferecia estatísticas detalhadas sobre todos os utilizadores e mensagens enviadas.

Implementação do Active Diretory

Para ligar ao servidor AD a partir do interior da aplicação, foram considerados 2 pacotes diferentes: `activedirectory` e `activedirectory2`. O pacote `activedirectory2` acabou por ser escolhido por ser o mais atualizado, e o outro não funcionou.

Este pacote utilizava consultas LDAP (Lightweight Diretory Access Protocol) e fornecia um invólucro JavaScript (JS) que permitia a passagem do e-mail e da palavra-passe de uma conta AD válida e já registada, juntamente com alguns outros argumentos. Primeiro, foi criado um objeto de instância do AD, como se mostra abaixo:

```
const ad = new ActiveDirectory({
  url: process.env.AD_URL!,
  baseDN: process.env.AD_BASE_DN!,
  username: process.env.AD_EMAIL!, // what they call username is actually an email
  password: process.env.AD_PASSWORD!,
});
```

Depois disso, os métodos dessa instância podem ser usados como mostrado abaixo:

```
await ad.isUserMemberOf(
  username,
  group,
  (err: object | null, isMember: boolean) => {}
);
```

As funções utilizadas podem ser encontradas em `/lib/auth/activedirectory`. Para obter informações sobre `activedirectory2`, foi consultada a documentação.

Gestão de sessões

A gestão de sessões era o processo de tratamento de sessões de utilizador em aplicações Web, em que os dados da sessão eram normalmente armazenados como um cookie. Uma biblioteca de gestão de sessões fornecia ferramentas para criar, manter e terminar sessões de utilizador (cookies de autenticação), simplificando a autenticação e a gestão do estado.

Como a configuração do AD já tratava da maior parte da autenticação, não era necessária uma biblioteca de autenticação completa. Na verdade, uma biblioteca leve de gerenciamento de sessão fez o trabalho. O pacote `iron-session` foi escolhido devido à sua natureza baseada em sessões e aos seus recursos leves, seguros e fáceis de implementar.

Outra biblioteca de gestão de sessões chamada `jose` também foi considerada. No entanto, foi rapidamente rejeitada, uma vez que a autenticação baseada em tokens não era necessária para o projeto. Além disso, `iron-session` era mais leve e fácil de usar. Mais informações sobre os tipos de autenticação podem ser encontradas na secção "Autenticação baseada em sessão vs. autenticação baseada em token".

Além disso, foram exploradas diferentes opções de armazenamento do navegador para manter os dados de autenticação do utilizador. No entanto, a utilização desta opção de armazenamento para dados de autenticação do utilizador era inadequada, uma vez que o armazenamento da sessão expirava quando o separador era fechado, ao contrário dos cookies, que eram normalmente utilizados para persistir os dados da sessão.

Implementação da gestão de sessões

Quando um utilizador é autenticado com sucesso, a sua informação é armazenada na base de dados e, subsequentemente, num cookie de id de sessão encriptado gerado pelo `iron-session`.

Configuração
As sessões `iron-session` foram personalizadas através da modificação de um objeto de configuração:

O nome e a palavra-passe podem ser qualquer coisa, mas para maior segurança a palavra-passe foi gerada utilizando openssl
A sessão expirou após 24 horas em vez dos 14 dias predefinidos.

```
18  // Iron session config object
19  export const sessionOptions: SessionOptions = {
20    cookieName: "my-etpzp-app-session", // anything you want
21    password: process.env.SESSION_SECRET!, // TypeScript non-null assertion operator
22
23    // Optional fields
24    ttl: 60 * 60 * 24, // cookie expiration from now in seconds (we want 24h)
25    cookieOptions: {
26      // prevent client side js from accessing the cookie
27      httpOnly: true,
28      // Secure only works in `https` environments. So if the environment is `https`, it'll return true.
29      secure: process.env.NODE_ENV === "production",
30    },
31  };
```

Ficheiro de configuração de autenticação localizado em `/lib/auth/config.ts`

Funções auxiliares
Uma função auxiliar simples chamada `getSession` foi criada para envolver a API Iron Session, que no lado do servidor recuperava a sessão ativa do cookie ou criava uma nova se não existisse nenhuma. O objeto de configuração `sessionOptions` previamente personalizado foi utilizado como um dos argumentos passados para a função `getIronSession`.

```
8   // helper function for getting the current session
9   export async function getSession(req?: NextRequest, res?: NextResponse) {
10    const session =
11      req && res
12        ? await getIronSession<SessionData>(req, res, sessionOptions)
13        : await getIronSession<SessionData>(await cookies(), sessionOptions);
14
15    // For security, you can double-check the user's existence in the database or AD server, but this slows down the app.
16    return session;
17  }
```

Função auxiliar getSession localizada em `/lib/auth/sessions.ts`

Foi criada outra função auxiliar com o objetivo de criar uma nova sessão. Esta utilizou a função `getSession` para obter primeiro a sessão atual e, em seguida, anexou à sessão informações úteis sobre o utilizador e se este estava ou não autenticado e se era ou não um administrador. Por fim, as modificações à sessão eram aplicadas (linha 29).

```
19  export async function createSession(user: SessionData) {
20    const session = await getSession();
21
22    // Store user data in the cookie by mapping over each of the object's property
23    Object.entries(user).forEach(([key, value]) => {
24      if (!(key in session)) {
25        (session as any)[key] = value;
26      }
27    });
28
29    await session.save();
30  }
```

Função auxiliar createSession localizada em `/lib/auth/sessions.ts`

Para obter informações sobre o pacote `iron-session`, foi consultada a documentação.

Fluxo de autenticação

Com o Active Diretory inicial e a configuração da sessão de ferro fora do caminho, a implementação final pode ser escrita.

Em resumo, o fluxo de autenticação envolveu as seguintes etapas:

Início de sessão do utilizador: Os utilizadores introduziram as suas credenciais (nome de utilizador e palavra-passe) no cliente e enviaram o formulário para o servidor.
Autenticação AD: A aplicação verificou estas credenciais com o servidor do Active Diretory.
Criação da sessão de ferro: Se for bem sucedida, a Sessão de Ferro cria um novo cookie de sessão e as informações do utilizador são guardadas na base de dados PostgreSQL.
Recuperação da sessão: Nos pedidos subsequentes, a aplicação verificou se a sessão do utilizador ainda era válida, desencriptando o cookie no servidor e verificando se a propriedade `isAuthenticated` estava definida como `verdadeira`.

A lógica geral foi tratada na **função de início de sessão**, que começou por obter os valores apresentados, validou-os utilizando zod (linhas 18 e 19), chamou a **função de autenticação** e, por último, devolveu a resposta adequada ao cliente, criando uma nova sessão se o utilizador tiver sido autenticado com êxito.

```
11  export async function login(
12    formData: FormData
13  ): Promise<ActionResponse<Login>> {
14    // 1. Type validation
15    const email = formData.get("email") as string;
16    const password = formData.get("password") as string;
17
18    const validatedData = LoginSchema.safeParse({ email, password });
19    if (!validatedData.success) {
20      return {
21        success: false,
22        message: ["common:fix_zod_errors"],
23        inputs: { email, password },
24        errors: validatedData.error.flatten().fieldErrors,
25      };
26    }
27
28    // 2. Authenticate user using AD and save to db
29    const user: SessionData = await authenticate({
30      email,
31      password,
32    });
33
34    if (!user.isAuthenticated) {
35      return {
36        success: false,
37        message: ["server-wrong_credentials"],
38        inputs: { email, password },
39      };
40    }
41
42    // 3. Create new session cookie
43    await createSession(user);
44    return {
45      success: true,
46      message: [
47        "server-auth_success_header",
48        "server-auth_success_header_caption",
49      ],
50    };
51  }
```

Função de início de sessão localizada em `/lib/auth/index.ts`

A **função de autenticação** continha a lógica importante para autenticar o utilizador com o servidor AD e guardar o resultado na base de dados. Primeiro, a existência da conta no servidor AD era verificada e, em caso negativo, a resposta correspondente era devolvida mais cedo. Se existisse, os privilégios da conta eram inspeccionados e, por último, os resultados destas consultas eram guardados na base de dados e devolvidos à função de início de sessão.

```
10  export default async function authenticate({
11    email,
12    password,
13  }: {
14    email: string;
15    password: string;
16  }): Promise<SessionData & { errors: string[] }> {
17    const ad = new ActiveDirectory(activeDirectoryConfig);
18
19    // 1. Check if user even exists in the active directory server
20    const exists = await userExists(ad, email, password);
21    if (!exists.success) {
22      return {
23        isAuthenticated: false,
24        isAdmin: false,
25        errors: [exists.error ? exists.error : ""],
26      };
27    }
28    // 2. Check if user is allowed to use the app
29    const userGroup = "Utilizadores-SMS";
30    const hasAppPermission = await userInGroup(ad, email, userGroup);
31
32    // 3. Check if user has admin privileges
33    const adminGroup = "Administradores-SMS";
34    const hasAdminPermission = await userInGroup(ad, email, adminGroup);
35
36    // 4. Sync all of this with the database
37    const userResult = await saveUser(ad, email, hasAdminPermission.success);
38
39    return {
40      user: userResult.success ? userResult.data : undefined,
41      isAuthenticated: hasAppPermission.success,
42      isAdmin: hasAdminPermission.success,
43      errors: [
44        exists.error !== null
45          ? exists.error
46          : "An error occurred while checking if user exists",
47        hasAppPermission.error !== null
48          ? hasAppPermission.error
49          : "An error occurred while checking if user is allowed to use the app",
50        hasAdminPermission.error !== null
51          ? hasAdminPermission.error
52          : "An error occurred while checking if user is an admin",
53      ],
54    };
55  }
```

A função Authenticate está localizada em `/lib/auth/activedirectory/authenticate.ts`

Para verificar se o utilizador existia (linha 20), foi utilizado o método `ad.authenticate()` e para verificar se a conta existia no grupo AD específico (linhas 30 e 34), o método `ad.isUserMemberOf()`. O código detalhado para isto estava localizado nos ficheiros em `lib/auth/activedirectory/`.

Solicitações subsequentes

Em cada solicitação, o Next.js reconheceu automaticamente o arquivo `/middleware.ts` e executou a exportação padrão desse arquivo antes que qualquer página fosse servida. Isto tornou-o o local perfeito para verificar a autenticação do utilizador. O código incluía o redireccionamento de utilizadores autenticados em `/login` para `/`, enquanto os utilizadores não autenticados eram redireccionados para `/login` se ainda não estivessem lá.

```tsx
export default async function middleware(request: NextRequest) {
  // Handle i18n routing
  const i18nResponse = i18nRouter(request, i18nConfig);
  const session = await getSession(request, i18nResponse);
  const { pathname } = request.nextUrl;
  const locale = request.cookies.get("NEXT_LOCALE")?.value || "en";

  // Pathname checks use `.includes()` instead of `.startsWith()`, because of possible locale between url segments.
  if (session.isAuthenticated && pathname.includes("/login")) {
    // Redirect logged in users to home
    return NextResponse.redirect(new URL(`/${locale}/`, request.url));
  }

  if (!session.isAuthenticated && !pathname.includes("/login")) {
    // Redirect unauthorized users to login
    return NextResponse.redirect(new URL(`/${locale}/login`, request.url));
  }

  // Return the i18n-router response for all other cases
  return i18nResponse;
}
```

Como o Next.js recomendava tratar as ações do servidor como rotas de API públicas, a autenticação do usuário também era **verificada em cada ação do servidor**.

Além disso, as verificações das permissões de administrador foram distribuídas pela aplicação para lhes mostrar certas coisas que os utilizadores normais não deveriam ver, sendo o ponto mais crítico um redireccionamento programático no layout do painel de administração.

```tsx
export default async function DashboardLayout({
  children,
  params,
}: LayoutProps) {
  const i18nNamespaces = ["dashboard-page", "errors", "common", "navigation"];
  const { locale } = await params;
  const { resources } = await initTranslations(locale, i18nNamespaces);

  // Prevent non-admins from viewing the admin-dashboard and display an authorization message.
  const session = await getSession();
  if (!session?.isAdmin) return <UnauthorizedPage />;

```

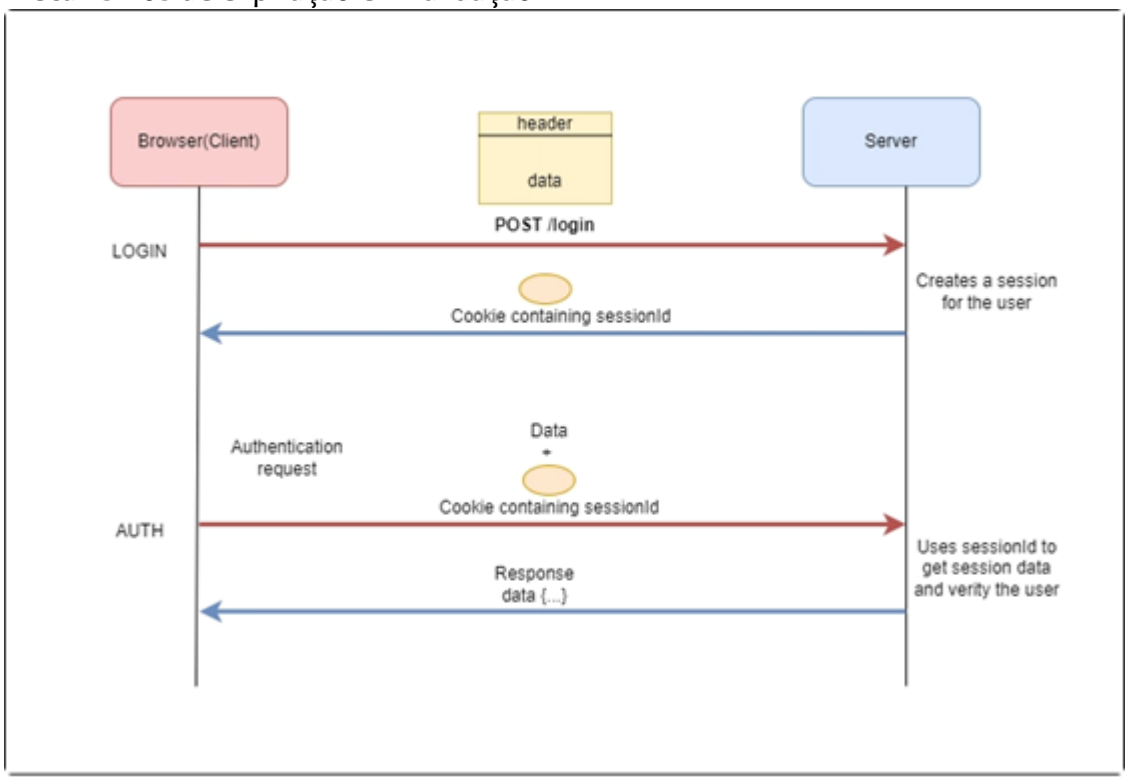
Layout do painel de controlo localizado em `/app/[locale]/dashboard/layout.tsx`

Autenticação baseada em sessão vs. autenticação baseada em token

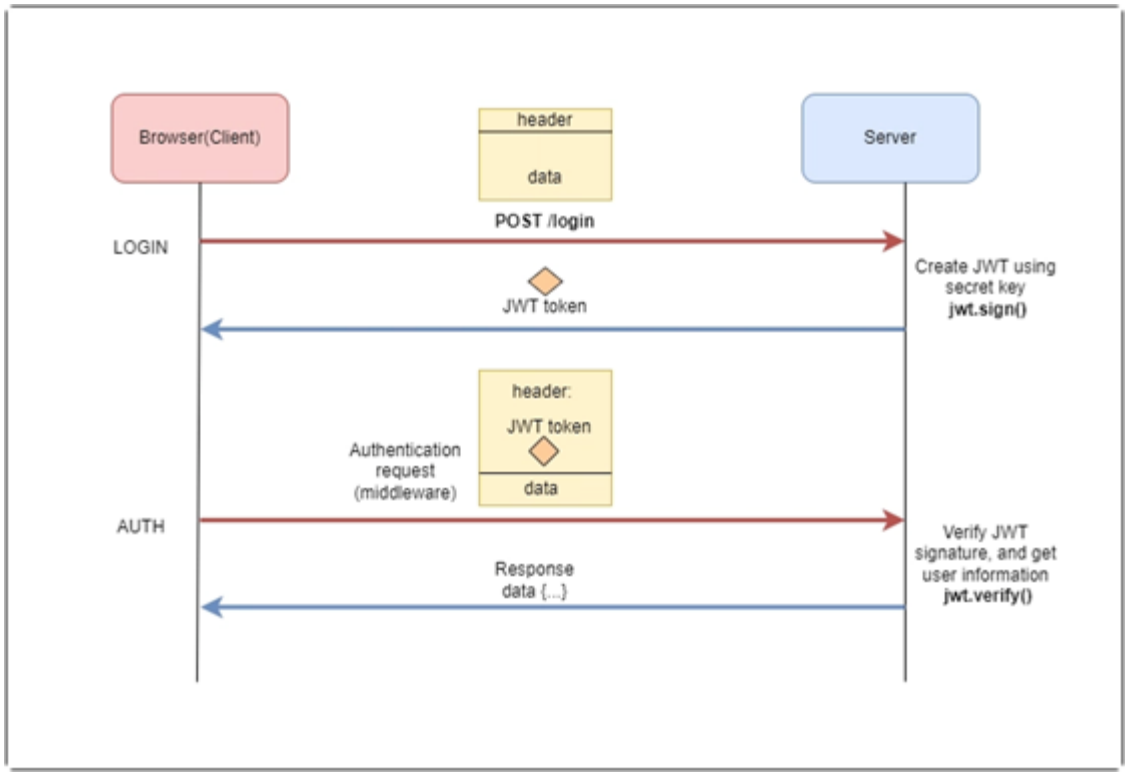
Foram examinados dois métodos para manter as sessões de utilizador em segurança:

Autenticação baseada na sessão: Foi gerado um ID de sessão único aquando do início de sessão, armazenado no servidor. O ID da sessão era enviado para o cliente **como um cookie** para verificar a identidade do utilizador em pedidos subsequentes, com

mecanismos de expiração e invalidação.



Autenticação baseada em token: Um JSON Web Token (JWT) foi gerado após o início de sessão, contendo informações do utilizador e um carimbo de data/hora de expiração. O cliente armazenava o token e enviava-o no **cabeçalho de autorização** com cada pedido. Este método permitiu a **autenticação sem estado**, uma vez que o servidor não manteve o estado da sessão, e suportou a autenticação entre domínios e a integração de aplicações móveis.



Dado o escopo limitado da aplicação e a hospedagem em um único servidor, a autenticação baseada em sessão foi considerada apropriada.

Fontes:

1. https://dev.to/fidalmathew/session-based-vs-token-based-authentication-which-is-better-2270
2. https://www.geeksforgeeks.org/session-vs-token-based-authentication/

INTERNACIONALIZAÇÃO (i18n)

A internacionalização (i18n) era o processo de conceção e desenvolvimento de software que podia ser facilmente adaptado a diferentes línguas, contextos culturais e regiões sem grandes alterações à base de código principal. Incluía a tradução da interface do utilizador, o tratamento do Unicode e a separação do conteúdo do código, garantindo que a aplicação era acessível e utilizável por um público global.

A I18next foi escolhida como a biblioteca de base para a i18n, juntamente com pacotes adicionais. Foi incluído um serviço externo chamado i18nexus, que fornece uma interface gráfica do utilizador (GUI) para gerir traduções e a capacidade de traduzir automaticamente cadeias de caracteres da língua de base para outras línguas.

Os termos especiais deste capítulo incluem:

espaço de nomes: Uma forma de organizar as chaves de tradução em grupos separados, permitindo uma melhor gestão e estruturação das traduções no i18next.
cadeia de tradução: Um par chave-valor em que a chave era um identificador único para uma cadeia de texto específica e o valor era o texto efetivamente traduzido na língua de destino.
interpolador: Uma funcionalidade do i18next que permitia a inserção dinâmica de variáveis nas cadeias de tradução, possibilitando a criação de traduções mais flexíveis e conscientes do contexto.

Implementação

No início, **o React-Intl** foi considerado como uma biblioteca i18n. No entanto, **o i18next** foi considerado a melhor escolha para internacionalização em aplicativos React devido ao seu conjunto abrangente de recursos, integração mais fácil, API mais intuitiva, comunidade maior e mais ativa e melhor desempenho, tornando-o a solução preferível para o projeto.

Para além do **i18next**, foram utilizados outros pacotes:

A i18next era a biblioteca de internacionalização principal que fornecia a funcionalidade básica para gerir traduções e localização.
react-i18next foi o pacote que integrou o i18next com o React, fornecendo ganchos e componentes que facilitaram o trabalho com traduções em componentes React.
i18next-resources-to-backend foi o plugin que permitiu o carregamento de recursos de tradução a partir de um servidor backend. Era particularmente útil para a renderização do lado do servidor (SSR), permitindo que a aplicação fosse buscar traduções dinamicamente com base na localidade do utilizador.
next-i18n-router foi projetado especificamente para projetos de roteadores de aplicativos Next.js. Ele implementou o roteamento internacionalizado e a deteção de localidade, permitindo que os desenvolvedores gerenciem facilmente as rotas com base no idioma selecionado sem ter que construir a lógica de roteamento do zero.

Configuração

Os pacotes foram instalados.

Foi criado um ficheiro de configuração (`/i18n.config.ts`)

```ts
export const i18nConfig = {
  locales: ["pt", "de", "en"],
  defaultLocale: "pt",

  // Set to `true` if you want the default locale to be included in the url
  prefixDefault: false,
};
```

Especificou uma propriedade `locales`, que era um conjunto de idiomas que a aplicação iria suportar.

A propriedade `defaultLocale` era o idioma para o qual os visitantes voltariam se a aplicação não suportasse o seu idioma.

Foi criado um segmento dinâmico dentro do diretório `/app` para conter todas as páginas e esquemas, denominado `[locale]`.

O middleware foi atualizado (`/middleware.ts`)

```ts
import { i18nRouter } from "next-i18n-router";
import { i18nConfig } from "./i18n.config";
import { NextRequest, NextResponse } from "next/server";
import { getSession } from "./lib/auth/sessions";

export default async function middleware(request: NextRequest) {
  // Handle i18n routing
  const i18nResponse = i18nRouter(request, i18nConfig);
  const session = await getSession(request, i18nResponse);
  const { pathname } = request.nextUrl;
  const locale = request.cookies.get("NEXT_LOCALE")?.value || "en";

  // Pathname checks use `.includes()` instead of `.startsWith()`,
  if (session.isAuthenticated && pathname.includes("/login")) {…
  }

  if (!session.isAuthenticated && !pathname.includes("/login")) {…
  }

  // Return the i18n-router response for all other cases
  return i18nResponse;
}

// applies this middleware only to files in the app directory
export const config = {
  matcher: "/((?!api|static|.*\\..*|_next).*)",
};
```

O pacote next-i18n-router facilitou o processo, uma vez que devolveu o valor da **função i18nRouter**, tratando de toda a lógica de encaminhamento de localidades.
Foi criada a função initTranslations (`/app/i18n.js`), que utilizou **o i18next-resources-to-backend** para carregar as traduções do lado do servidor, com código copiado do tutorial.
Foi adicionado o TranslationsProvider (`/contexts/translations-provider.jsx`), que envolveu os componentes onde foi utilizada a função `t` do react-i18next, com código copiado do tutorial.
A função generateStaticParams da API Next.js foi adicionada ao layout raiz para **gerar estaticamente** rotas no momento da construção, em vez de sob demanda no momento da solicitação.
A aplicação foi ligada à plataforma i18nexus, com mais pormenores disponíveis nas secções "i18nexus" e "integração i18nexus".

Para a configuração, foram consultados os tutoriais da **i18nexus**:

- Tutorial escrito
- Tutorial em vídeo (30 min)

Utilização
Num componente cliente, a função de tradução (chamada `t`) foi obtida através da sua desestruturação a partir do hook useTranslation do **react-i18next**.

```
import { useTranslation } from "react-i18next";
const { t } = useTranslation(["dashboard-page", "errors", "common"]);
```

Numa componente de servidor, a função de tradução (chamada `t`) foi obtida através da sua desestruturação a partir da função initTranslations criada na configuração, passando o locale atual e um conjunto de espaços de nomes.

```
const { t, resources } = await initTranslations(locale, i18nNamespaces);
```

Depois disso, a função `t` pode ser utilizada em qualquer parte do componente, passando a cadeia de tradução e, se aplicável, o interpolador.

```
t("header_long", {
  first_name: "Peter",
})
```

O i18next incluía uma série de outras funcionalidades, mas estas foram as únicas utilizadas neste projeto.

i18nexus

A i18nexus era uma plataforma que simplificava a internacionalização (i18n) e a localização (l10n) de aplicações de software. O método antigo envolvia a criação manual de vários ficheiros JSON para cada espaço de nome e idioma. No entanto, as traduções eram escritas e geridas na Interface Gráfica do Utilizador (GUI) fornecida pelo i18nexus. As traduções eram primeiro escritas numa língua base (Inglês) e depois traduzidas automaticamente para outras línguas.

Um aspeto notável foi o facto de a aplicação não depender de um serviço externo. Foi possível puxar todos os ficheiros JSON de tradução para o projeto utilizando um comando no terminal.

Inicialmente, foi utilizado apenas o plano gratuito, mas mais tarde foi adquirido o plano básico devido ao facto de as cadeias de tradução se terem esgotado. Depois de terminar a aplicação, este plano foi cancelado e a aplicação continuou a funcionar.

Um problema detectado foi o facto de a plataforma utilizar a API do Google Translate nos planos gratuito e básico, que apenas suportava o português do Brasil. Após contactar o suporte, foi possível implementar rapidamente uma correção em que a plataforma utilizava o tradutor DeepL para o português europeu, mesmo nos níveis inferiores.

Ao utilizar o Roteador de aplicativos com o i18next, era uma boa prática "namespace" as cadeias de caracteres por página. Esta abordagem permitiu evitar o carregamento de todas as cadeias de caracteres de toda a aplicação ao visualizar uma página, permitindo-lhe carregar apenas as cadeias de caracteres dessa página específica de cada vez.

Integração do i18nexus

Para ligar a aplicação ao i18nexus, foram seguidos estes passos:

O i18nexus-cli foi instalado globalmente (`bun i i18nexus-cli -g`) e como uma dependência de desenvolvimento (`bun i i18nexus-cli --save-dev`), que era a interface de linha de comandos utilizada para puxar os ficheiros de tradução para o projeto.
A chave da API do projeto foi adicionada ao ficheiro `.env` com a variável denominada `I18NEXUS_API_KEY`.
O comando `i18nexus pull` foi executado a partir do terminal no diretório raiz do projeto para extrair ou atualizar as localidades.
Por conveniência, este comando também foi adicionado aos scripts `package.json`, de modo a que as traduções mais actualizadas fossem automaticamente retiradas sempre que um servidor fosse ativado.

```
"scripts": {
  "dev": "i18nexus pull && next dev",
  "build": "i18nexus pull && next build",
  "start": "i18nexus pull && next start",
```

AUTO-HOSPEDAGEM E IMPLANTAÇÃO

A aplicação foi implementada num computador da escola num contentor Docker. Para simplificar o acesso, obteve um nome de domínio gratuito do No-IP, um fornecedor de DDNS. O tráfego fluía da seguinte forma:

Um cliente solicitou `etpzp-sms.ddns.net`.
O router recebeu o pedido e encaminhou-o para o Nginx.
O Nginx redireccionou para HTTPS, se necessário, e encaminhou o pedido para o contentor Docker.
O servidor Node.js no contentor processou o pedido.
Esta configuração permitiu um acesso fácil à aplicação através de um nome de domínio simples, garantindo simultaneamente o encaminhamento e a segurança adequados.

Docker

O Docker foi escolhido como uma plataforma de código aberto que permite aos programadores empacotar aplicações e as suas dependências em contentores leves e portáteis, simplificando a implementação e melhorando a portabilidade em diferentes ambientes.

Durante a produção, havia dois contentores Docker separados: um para a aplicação Web com o próprio servidor Node.js e outro para a base de dados PostgreSQL. Qualquer um deles executava o Alpine Linux, que era um sistema operativo Linux muito leve.

Outros ficheiros relacionados com o Docker que não foram explicados incluem `.env.docker` e `.dockerignore`, que foram utilizados para gerir variáveis de ambiente e especificar ficheiros e diretórios que devem ser excluídos do contexto de compilação do Docker, respetivamente.

`Dockerfile` explicado

Um Dockerfile era um ficheiro de texto que continha uma série de instruções para construir uma imagem Docker, especificando o ambiente da aplicação, as dependências e a configuração necessária para executar a aplicação. Havia 2 Dockerfiles na aplicação.

Dockerfile do Node.js

Este foi o Dockerfile mais importante localizado em `/Dockerfile`:

```
1  FROM oven/bun:alpine AS base
2
3  # Install Node.js, npm, and i18nexus for translations
4  RUN apk add --no-cache nodejs npm
5  RUN bun i -g i18nexus-cli
6
7  # Stage 1: Install dependencies
8  FROM base AS deps
9  # set a path to for the following commands to be run on
10 WORKDIR /app
11 COPY package.json bun.lock ./
12 RUN bun install
13
14 # Stage 2: Build the application
15 FROM base AS builder
16 WORKDIR /app
17 COPY --from=deps /app/node_modules ./node_modules
18 COPY . .
19 RUN bun run build
20
21 # Stage 3: Production server
22 FROM base AS runner
23 WORKDIR /app
24 ENV NODE_ENV=production
25 COPY --from=builder /app/public ./public
26
27 COPY --from=builder /app/.next/standalone ./
28 COPY --from=builder /app/.next/static ./.next/static
29 # If it at some point doesn't work anymore, copy the entire directory
30 # COPY --from=builder /app/.next ./.next
31
32 # This is for the `start` script, but when replacing the start command with server.js (also part of the build) I can't access the site on localhost
33 COPY --from=builder /app/package.json ./
34 COPY --from=builder /app/node_modules ./node_modules
35
36 EXPOSE 3000
37 CMD ["bun", "run", "start"]
```

Imagem de base: Foi definida uma imagem de base utilizando um ambiente leve Alpine Linux com Bun (linha 1).

Instalar o Node.js e o i18nexus: O Node.js e o npm foram instalados, juntamente com o i18nexus CLI para traduções (linhas 4-5).

Estágio de Dependências: Um novo estágio chamado `deps` foi criado para instalar as dependências do aplicativo. Ele definiu o diretório de trabalho, copiou os arquivos necessários e executou o comando de instalação (linhas 8-12).

Estágio de construção: A fase de construção foi iniciada, onde definiu o diretório de trabalho, copiou as dependências instaladas da fase anterior e construiu a aplicação (linhas 15-19).

Etapa do servidor de produção: A fase final, `runner`, definiu o diretório de trabalho e definiu a variável de ambiente para produção. Os ficheiros da aplicação compilados na fase anterior foram copiados (linhas 22-28).

Copiando arquivos adicionais: O código também copiou o `package.json` e `node_modules` do estágio anterior para garantir que todos os arquivos necessários estivessem disponíveis (linhas 33-34).

Expor porta: O `Dockerfile` expôs a porta 3000, permitindo o acesso externo ao aplicativo (linha 36).

Comando de início: Finalmente, foi especificado um comando para iniciar a aplicação (linha 37).

Dockerfile do banco de dados

Esta foi a configuração mais simples localizada em `/lib/db/Dockerfile` para configurar e propagar a base de dados:

Imagem de base: A imagem de base foi definida usando uma versão específica do PostgreSQL, que foi baseada numa variante leve do Alpine Linux (linha 1).
Copiar script de seed: Um arquivo SQL chamado `seed.sql` foi copiado para um diretório designado dentro do container do PostgreSQL usado para semear o banco de dados quando o container foi iniciado (linha 2).

Explicação do `docker-compose.yaml`

Um arquivo `docker-compose.yaml` era um arquivo de texto que definia um aplicativo Docker com vários contêineres. Ele especificava os serviços (contêineres) que compunham o aplicativo, suas configurações e como eles interagiam uns com os outros. Esse arquivo permitia a definição e o gerenciamento de toda a pilha de aplicativos, incluindo rede, volumes e variáveis de ambiente, em um único arquivo.

Teria sido possível alcançar os mesmos resultados sem o Docker Compose, criando e gerindo os contentores Docker individuais, redes, volumes e outros recursos necessários para a aplicação. No entanto, isso teria sido mais complexo e demorado.

O ficheiro Docker Compose, localizado em `/docker-compose.yaml`, definiu 2 serviços: `web` e `base de dados`.

```yaml
services:
  ▷Run Service
  web:
    build:
      context: .
      dockerfile: Dockerfile
    env_file: .env.docker
    ports:
      - "3000:3000"
    depends_on:
      database:
        condition: service_healthy
  ▷Run Service
  database:
    build:
      context: ./lib/db
      dockerfile: Dockerfile
    container_name: postgres
    env_file: .env.docker
    ports:
      - ${POSTGRES_PORT}:${POSTGRES_PORT}
    volumes:
      - database-v:/var/lib/postgresql/data
    healthcheck:
      test:
        [
          "CMD-SHELL",
          "pg_isready -p ${POSTGRES_PORT} -U ${POSTGRES_USER} -d ${POSTGRES_DB}",
        ]
      start_period: 0s
      interval: 5s
      timeout: 5s
      retries: 5
volumes:
  database-v:
    name: "database-v"
```

O serviço `Web`:
Ele construiu a imagem do Docker usando o Dockerfile no diretório atual.
Carregou variáveis de ambiente a partir do ficheiro `.env.docker`.

Expôs a porta `3000` no anfitrião e mapeou-a para a porta `3000` no contentor.
Dependia do serviço de `base de dados` e aguardava que este estivesse operacional antes de iniciar.
O serviço de `base de dados`:
Construiu a imagem do Docker usando o Dockerfile no diretório `./lib/db`.
Definiu o nome do contentor para `postgres`.
Carregou variáveis de ambiente a partir do ficheiro `.env.docker`.
Expôs a variável de ambiente `POSTGRES_PORT` no anfitrião e mapeou-a para a mesma porta no contentor.
Montou um volume chamado `database-v` no diretório `/var/lib/postgresql/data` no contentor.
Ele definiu um healthcheck que verificava se o PostgreSQL estava pronto para aceitar conexões a cada 5 segundos, com um timeout de 5 segundos e um máximo de 5 tentativas.
O volume `database-v` foi definido para manter os dados PostgreSQL.

Sem IP e reencaminhamento de portas

Para simplificar o acesso dos utilizadores, foi obtido um nome de domínio gratuito junto do No-IP, um fornecedor dinâmico de serviços de nomes de domínio (DNS). Isto permitiu aos utilizadores ligarem-se à aplicação utilizando um nome de domínio memorável em vez do endereço IP do router, que pode ter mudado frequentemente.

O No-IP actualizava automaticamente o nome de domínio para refletir o endereço IP atual do router, garantindo um acesso consistente. Este recurso era particularmente útil em ambientes onde o endereçamento IP dinâmico era comum. Por outras palavras, basicamente fazia com que o IP dinâmico se comportasse como um IP estático.

Para configurar o No-IP, foi consultado este guia que explica o que foi feito para configurar o No-IP:

Criar uma conta: Foi criada uma nova conta no sítio Web do No-IP e foram preenchidas as informações necessárias.
Confirmar a conta: Foi verificada a existência de uma ligação de confirmação no correio eletrónico e clicou-se nela.
Iniciar sessão: Acedeu à conta utilizando o e-mail e a palavra-passe.
Adicionar um nome de anfitrião: Foi criado um nome de anfitrião para o servidor
(Opcional) Criar uma chave DNS dinâmica: Foi criada uma chave DNS dinâmica para maior segurança e compatibilidade.
Tornando o host dinâmico: O Cliente de Atualização Dinâmica (DUC) do No-IP foi instalado e configurou o dispositivo para actualizações.
Configuração do router: O encaminhamento de portas foi configurado para os serviços necessários (por exemplo, web, FTP).
Execução dos serviços: A configuração foi verificada com uma ferramenta de verificação de portas e começou a utilizar os serviços.

O encaminhamento de portas foi configurado no router para direcionar o tráfego de entrada para a porta específica do contentor Docker que executa a aplicação. Esta configuração permitiu que os utilizadores acedessem à aplicação facilmente e a partir de redes que não apenas a rede da escola. Para a configuração, este guia foi referenciado.

Nginx

O Nginx forneceu vários recursos, servindo principalmente como um servidor web e funcionando como um proxy reverso para redirecionar o tráfego para outros servidores. Neste projeto, foi utilizado para configurar certificados SSL, redirecionar o tráfego http para https e redirecionar o tráfego para a aplicação executada no Docker. Para aprender os conceitos básicos do Nginx, foi consultado **este tutorial**.

O Nginx foi bastante fácil de configurar:

1. O Nginx foi instalado.
2. Ele foi iniciado usando o comando `nginx`.
3. Primeiro foi gerado um certificado SSL auto-assinado usando este comando: `openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout nginx-selfsigned.key -out nginx-selfsigned.crt`
4. Em seguida, foi feita uma tentativa de usar um comando do Certbot para gerar um certificado SSL assinado por autoridade gratuitamente. No entanto, este processo deparou-se com um problema porque o Certbot não era compatível com o computador da escola com Windows.
5. O resto do trabalho consistiu em editar o ficheiro `nginx.conf`, onde foi definido todo o comportamento do Nginx.

`nginx.conf`

Apesar de o ficheiro de configuração do Nginx (`/nginx.conf`) ter sido enviado para o repositório, não foi lido a partir deste ficheiro. Ele existia para garantir a disponibilidade quando necessário. A localização real do arquivo de configuração pode ser verificada executando `nginx -V`, permitindo que o caminho que contém o arquivo de configuração `nginx.conf` seja copiado. Aqui estava a configuração básica:

```
# Main context (this is the global configuration)
worker_processes  1;

events {
    worker_connections  1024;
}

http {
    include mime.types;

    # Optional server block for HTTP to HTTPS redirection
    server {
        listen 80;
        server_name localhost;

        # Redirect all HTTP requests to HTTPS
        return 301 https://\$host\$request_uri;
    }

    # Main server block
    server {
        listen 443 ssl; # Listen on port 443 for HTTPS
        server_name localhost;

        # Here are my self signed certs. In actual production you would let these be signed by a organization
        ssl_certificate /Users/<your_user>/nginx-certs/nginx-selfsigned.crt;
        ssl_certificate_key /Users/<your_user>/nginx-certs/nginx-selfsigned.key;

        # Proxying requests to the Docker container (assuming it is running on port 3000)
        location / {
            # Tell nginx to act as a reverse proxy to forward requests to the node servers
            proxy_pass http://localhost:3000;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
            proxy_set_header X-Forwarded-Host $host;
            proxy_set_header X-Forwarded-Port $server_port;
            proxy_set_header Cookie $http_cookie;  # Forward cookies
        }
    }
}
```

Ele configurou o Nginx com 1 processo de trabalho e 1024 conexões.
Redireccionou todo o tráfego HTTP (porta 80) para HTTPS (porta 443).
Utilizava certificados SSL auto-assinados para HTTPS.
Fazia proxy de pedidos para um serviço de backend executado na porta 3000.
Transmitia informações do cliente através de cabeçalhos para o backend.

Comandos úteis

1. `nginx -s reload` recarregou a configuração sem deixar cair as conexões.
2. `nginx -s stop` pára graciosamente o servidor.
3. `nginx -s quit` parou o servidor imediatamente após fechar as ligações actuais.
   Para obter mais informações sobre o nginx, a documentação foi referenciada.

Certificados SSL

Obter um certificado SSL **auto-assinado** foi feito facilmente usando o seguinte comando. Ele gerou uma chave auto-assinada, que foi colocada em `~/nginx-certs/` e então referenciada a partir do arquivo de configuração do Nginx usando o caminho absoluto. Ele mudou para esse diretório recém-criado:

openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout nginx-selfsigned.key -out nginx-selfsigned.crt

A obtenção de um certificado SSL **assinado por uma autoridade** foi feita usando o Certbot, completando um "desafio". Para isso, foi necessário o nome de domínio, leia mais aqui:

sudo certbot --nginx -d< your_domain_name> .com

Depois disso, ele ainda tinha que fazer mais tarefas, como gerar um link simbólico. Ele seguiu este tutorial para todas as etapas.

CONCLUSÃO

Em conclusão, foi determinado que a aplicação abordou eficazmente os elevados custos das mensagens de texto para a escola, oferecendo uma solução de comunicação SMS eficiente e económica, acessível a todos os utilizadores. O projeto era totalmente reativo e utilizável em dispositivos móveis, com todas as funcionalidades planeadas implementadas e outras adicionais incluídas. Com capacidades como o envio de mensagens para vários destinatários, envios programados e uma interface de fácil utilização, simplificou a comunicação e integrou-se perfeitamente no Active Diretory da escola para autenticação.

O sítio Web foi reconhecido pela sua rápida produção, graças a tecnologias robustas e a uma forte ênfase no desempenho. Esta ênfase resultou numa interface rápida com latência mínima, melhorando a experiência do utilizador e fazendo com que se parecesse mais com uma aplicação do que com um sítio Web. Construído com tecnologias de topo como Next.js e PostgreSQL, mostrou o potencial de alavancar APIs REST para a funcionalidade SMS. Por último, a sua implementação utilizando o Docker e o Nginx num computador local da escola foi excelente para aprender as noções básicas de alojamento e demonstrou a aplicação prática destas tecnologias.

Embora o projeto tenha apresentado desafios, foi particularmente difícil encerrá-lo no final, o que acabou por servir como uma experiência de aprendizagem significativa. Com os conhecimentos adquiridos, previa-se que as futuras iterações de aplicações semelhantes pudessem ser desenvolvidas de forma mais eficiente e eficaz.

Arrependimentos

Utilizar apenas as funcionalidades básicas do i18next sem o i18nexus, evitando o plural e a ramificação da tradução.

Repetição frequente de cálculos complexos da área de deslocação em vez de os gerir num único local

Configuração complexa das definições do front-end com algumas definições a serem tratadas por bibliotecas

Realização lenta da incompatibilidade entre os componentes `Sheet` e `ScrollArea` do ShadCN

A utilização de `.safeParse` em vez de `.parse` com `zod` faz com que o código de tratamento de erros fique no bloco try, o que não respeita a separação de preocupações

Sem snippets para código repetitivo

Não existe uma forma uniforme de tratar os erros nas acções do servidor

Não há convenções de nomenclatura claras/consistentes para funções, tipos TypeScript e esquemas Zod

Caraterísticas omitidas

A caraterística mais importante não implementada foi a **sondagem da API quanto ao estado de entrega das SMS**. Era crucial porque, embora os erros imediatos fossem geridos tanto do lado do utilizador como do lado da API de gateway, as mensagens podiam não chegar ao destinatário final devido a questões como um número de telefone inválido ou problemas com o telefone do destinatário. Este estado devia ser apresentado na aplicação.

Para obter informações sobre como pesquisar a API, foi consultada a documentação da GatewayAPI.

Embora a API recomendasse a utilização de webhooks em vez de sondagens por motivos de eficiência, necessitava de sondagens de mensagens devido à sua configuração de auto-hospedagem. Esta abordagem permitiu-lhe gerir situações em que o servidor podia ser desligado durante as férias, garantindo que ainda podia recuperar o estado da entrega de SMS quando o servidor estivesse novamente online.

Embora esta funcionalidade não tenha sido implementada, foram tomadas notas:

Do ponto de vista lógico, o `estado do` campo da base de dados das mensagens programadas não deve ser "PROGRAMADO" quando a data de entrega é atingida.
No início de sessão do utilizador, as mensagens podem ser verificadas quanto ao sinalizador `confirmed_delivery`.
Os erros de entrega para destinatários individuais podem ser apresentados no ecrã de mensagens.
Um campo como `was_scheduled` ou `scheduled_send` poderia ser adicionado para indicar como a mensagem foi enviada.
Para atualizar os indicadores de quantidade, pode ser adicionado na estrutura de raiz um temporizador de atualização de 5 minutos para sondar os estados de entrega de mensagens programadas.

Outras caraterísticas

Ligações para o item modificado/criado em mensagens de sucesso para um acesso fácil aos detalhes
Apenas administradores:
Ligações para a página de início de sessão da GatewayAPI no painel de administração
Definição de max-age do cookie de autenticação
Opção para efetuar cópias de segurança e restaurar a base de dados
Opção para especificar as opções de seleção disponíveis para o nome do remetente
Mais informações de contacto apresentadas em cada item de mensagem da lista
Fotografias de perfil de contacto
Os valores não definidos devem ser passados como `nulos` para a base de dados

ANEXO I - MANUAL DO UTILIZADOR

Este capítulo fornece explicações claras e passo a passo de procedimentos comuns e processos não intuitivos para ajudar os novos utilizadores a navegar no projeto e a aceder às ferramentas necessárias para a expansão. Dicas e guias adicionais para configurações específicas podem ser encontrados noutros capítulos.

Como começar

Este é um projeto de router de aplicações Next.js 15.

O utilizador instala o gestor de pacotes Bun seguindo as instruções do seu sítio Web.
O utilizador define as variáveis de ambiente necessárias encontradas na secção ANEXAS. Estas incluem `.env e .env.docker`, que vão ambas para o diretório raiz do projeto.
O utilizador navega para o diretório correto a partir de uma aplicação terminal à sua escolha.
O utilizador instala os pacotes executando o comando de terminal `bun install`.
O utilizador inicia o servidor de desenvolvimento executando o comando de terminal bun dev. Se ocorrer um erro, o utilizador pode utilizar o comando alternativo: `bun next dev`.

> Nota: O usuário pode utilizar qualquer gerenciador de pacotes que preferir, mas o autor recomenda o uso do Bun por ser o gerenciador de pacotes mais rápido e eficiente, além de fornecer uma API quase idêntica ao `npm`.

GitHub

Primeiros passos: O utilizador cria uma conta GitHub e configura o Git na sua máquina local. Ele configura seu nome de usuário e e-mail com `git config --global user.name "Seu nome"` e `git config --global user.email "your.email@example.com"`.

Clonando o repositório: O utilizador utiliza o comando `git clone <repositório-url>` para copiar um repositório remoto para a sua máquina local, permitindo-lhe trabalhar no projeto localmente.

Adicionando um novo ramo e configurando o upstream: O usuário cria um novo ramo com `git checkout -b <nome do ramo>` e, em seguida, faz o push para o repositório remoto pela primeira vez usando `git push -u origin <nome do ramo>`. O sinalizador `-u` define a referência de rastreamento upstream, vinculando o ramo local ao ramo remoto. Os ramos são criados apenas para novos recursos e, quando um recurso é concluído, testado e funcionando, ele pode ser mesclado no ramo principal.

Enviando para o repositório: Depois de fazer as alterações, o usuário as encena com `git add .`, faz o commit com `git commit -m "Sua mensagem"`, e faz o push para o repositório remoto usando `git push origin <branch-name>`.

Trabalhar num ambiente de desenvolvimento

Este processo pode ser complicado em diferentes plataformas, mas a configuração e algumas dicas para desenvolver este projeto são explicadas abaixo.

Servidor Web Node.js
Para iniciar um servidor de desenvolvimento, é utilizado o seguinte comando:

desenvolvimento de pão

Se não houver ligação à Internet ou ocorrer outro erro, é utilizado o comando abaixo:

bun next dev

Depurando o banco de dados PostgreSQL
No macOS, o Postgres.app deve estar em execução em segundo plano para funcionar corretamente. Para consultar a base de dados diretamente, o utilizador executa o comando `psql` num terminal para aceder à shell `psql`, onde todas as consultas podem ser executadas.

No Windows, o Postgres deve estar sempre em execução. O utilizador abre a aplicação `psql`, que contém a shell `psql` para execução de consultas.

> Sugestão: Os problemas de ligação devem-se provavelmente a credenciais inválidas.

Comandos PostgreSQL

Para semear a base de dados, o utilizador executa`your_project_file_path/lib/db/seed.sql` na shell `psql`. Este comando é o mesmo para macOS e Windows. No entanto, se surgirem problemas no Windows, o utilizador deve tentar usar barras invertidas (\) em vez de barras (/).
Comandos SQL:
Para eliminar todas as tabelas: `DROP TABLE IF EXISTS destinatário, contacto, mensagem, public.user;`
Para verificar quantas mensagens foram enviadas nos últimos 30 dias: `SELECT COUNT(*) FROM message WHERE send_time >= CURRENT_DATE - INTERVAL '1 months' AND in_trash = false AND status NOT IN ('FAILED', 'DRAFTED');`

Trabalhar num ambiente de produção (implantação)

O utilizador garante que o motor Docker está a funcionar abrindo a aplicação Docker.
O utilizador inicia os contentores Docker com o seguinte comando:

docker-compose up --build

Se o Nginx não estiver em execução, o utilizador executa este comando:

nginx

O utilizador reinicia o servidor Web Nginx utilizando o comando:

nginx -s recarregar

> **Observação**: No primeiro comando, o sinalizador `--build` é opcional. Ele solicita que o Docker reconstrua as imagens e deve ser usado quando há alterações que precisam ser aplicadas. Se omitido, o Docker Compose usa imagens existentes, acelerando o processo.

Depuração do Docker

Aceder a um contentor Docker: Para aceder a um contentor Docker em execução, o utilizador executa o seguinte comando:

docker exec -it< nome_do_contentor_ou_id> /bin/sh

O utilizador substitui `<nome_do_contentor_ou_id>` pelo nome real ou ID do contentor. Como o Alpine está em uso, o usuário acessa o shell `sh` em vez do `bash`.

Acessando um banco de dados PostgreSQL em um container Docker: Para acessar um banco de dados PostgreSQL através do shell `psql` em execução dentro de um contêiner Docker, o usuário executa:

docker exec -it< postgres_container_name_or_id> psql -U< username> -d< database_name>

O utilizador substitui `<postgres_container_name_or_id>`, `<username>` e `<database_name>` pelos valores apropriados.

Mais comandos:

Listagem de contentores em execução: `docker ps`
Parar um contentor: `docker stop <nome_do_contentor_ou_id>`
Iniciando um contêiner: `docker start <nome_do_contêiner_ou_id>`
Removendo um contêiner: `docker rm <nome_do_contêiner_ou_id>`
Ver os registos do contentor: `docker logs <nome_do_contentor_ou_id>`

Zona de perigo:

Remoção de contentores parados: `docker container prune`
Remoção de imagens não utilizadas: `docker image prune`
Remoção de volumes não utilizados: `docker volume prune`
Remoção de redes não utilizadas: `docker network prune`
Remoção de todos os recursos do Docker: `docker system prune -a --volumes`

Trabalhar com a i18nexus

> **Aviso**: Optar por não utilizar o i18nexus e editar manualmente os ficheiros JSON resulta na perda permanente de alterações quando se executa o comando pull, uma vez que o diretório `/locales` não é confirmado no git.

O utilizador inicia sessão na plataforma i18nexus com a conta fornecida.
O utilizador faz alterações na plataforma. O plano gratuito limita as cadeias de tradução, impedindo a adição de novas. Está disponível um espaço de nomes "arquivo (não utilizado em lado nenhum)" para mover e editar traduções não utilizadas.
O utilizador sincroniza as alterações executando:

i18nexus pull

ANEXO II - FICHEIROS DE CÓDIGO

/middleware.ts

```ts
import { i18nRouter } from "next-i18n-router";
import { i18nConfig } from "./i18n.config";
import { NextRequest, NextResponse } from "next/server";
import { getSession } from "./lib/auth/sessions";

export default async function middleware(request: NextRequest) {
  // Handle i18n routing
  const i18nResponse = i18nRouter(request, i18nConfig);
  const session = await getSession(request, i18nResponse);
  const { pathname } = request.nextUrl;
  const locale = request.cookies.get("NEXT_LOCALE")?.value || "en";

  // Pathname checks use `.includes()` instead of `.startsWith()`, because
of possible locale between url segments.
  if (session.isAuthenticated && pathname.includes("/login")) {
    // Redirect logged in users to home
    return NextResponse.redirect(new URL(`/${locale}/`, request.url));
  }

  if (!session.isAuthenticated && !pathname.includes("/login")) {
    // Redirect unauthorized users to login
    return NextResponse.redirect(new URL(`/${locale}/login`, request.url));
  }

  // Return the i18n-router response for all other cases
  return i18nResponse;
}

// applies this middleware only to files in the app directory
export const config = {
  matcher: "/((?!api|static|.*\\..*|_next).*)",
};
```

/docker-compose.yaml

```yaml
services:
  web:
    build:
      context: .
      dockerfile: Dockerfile
    env_file: .env.docker
    ports:
      - "3000:3000"
    depends_on:
      database:
```

```
        condition: service_healthy
  database:
    build:
      context: ./lib/db
      dockerfile: Dockerfile
    container_name: postgres
    env_file: .env.docker
    ports:
      - ${POSTGRES_PORT}:${POSTGRES_PORT}
    volumes:
      - database-v:/var/lib/postgresql/data
    healthcheck:
      test:
        [
          "CMD-SHELL",
          "pg_isready -p ${POSTGRES_PORT} -U ${POSTGRES_USER} -d ${POSTGRES
_DB}",
        ]
      start_period: 0s
      interval: 5s
      timeout: 5s
      retries: 5
volumes:
  database-v:
    name: "database-v"
```

/types/contact.ts

```ts
export type DBContact = {
  id: string;
  phone: string;

  // contact information
  user_id: string;
  name: string;
  description?: string; // Optional field
  created_at: Date;
  updated_at: Date;
};
```

/types/dashboard.ts

```ts
export type LightDBMessage = {
  user_id: string;
  send_time: Date;
  cost: string;
};
```

/types/action.ts

```ts
import { ContactSchema } from "@/lib/form.schemas";
import { DBContact } from "./contact";
import { z } from "zod";

// this is for useActionState() forms
export type ActionResponse<T> = {
  success: boolean;
  message: string[];
  errors?: {
    [K in keyof T]?: string[];
  };
  inputs?: {
    [K in keyof T]?: string;
  };
};

export type DraftActionResponse<T> = {
  success: boolean;
  message: string[];
  draftId?: T;
};

export type DataActionResponse<T> = {
  success: boolean;
  message: string[];
  data?: T;
};

export type UpdateSettingResponse = {
  success: boolean;
  name?: string;
  input: string;
  error?: string;
  data?: any;
};

export type CreateContactResponse = {
  success: boolean;
  message: string[];
  data?: DBContact;
  errors?: {
    [K in keyof z.infer<typeof ContactSchema>]?: string[];
  };
  inputs?: {
    [K in keyof z.infer<typeof ContactSchema>]?: string;
  };
};
```

/types/recipient.ts

```ts
type BaseRecipient = {
  phone: string;
  // if it is a contact
  contact?: {
    id: string;
    name?: string;
    phone: string;
    description?: string;
  };
};

// Recipients used in the new message form.
export type NewRecipient = {
  formattedPhone?: string;
  isValid: boolean;
  error?: {
    type?: "error" | "warning";
    message?: string;
  };
  proneForDeletion: boolean;
} & BaseRecipient;

export type WithContact = {
  id: string;
} & BaseRecipient;

// No joins - normal query directly from the DB
export type DBRecipient = {
  id: string;
  phone: string;
};

export type FetchedRecipient = DBRecipient & { last_used: Date };
export type RankedRecipient = DBRecipient & { usageCount: number };
```

/types/index.ts

```ts
import { z } from "zod";
import { DBRecipient, NewRecipient } from "./recipient";
import { MessageSchema } from "@/lib/form.schemas";

export type StatusEnums = "SENT" | "SCHEDULED" | "FAILED" | "DRAFTED";
export type CategoryEnums =
  | "SENT"
  | "SCHEDULED"
```

```
  | "FAILED"
  | "DRAFTS"
  | "TRASH";

// export type StringBoolMap = { [key: string]: boolean };
export type Modals = {
  schedule: boolean;
  scheduleAlert: boolean;
  contact: {
    create: boolean;
    edit: boolean;
    info: boolean;
    insert: boolean;
  };
};

export type Message = z.infer<typeof MessageSchema> & {
  recipients: NewRecipient[];
};

export type DBMessage = {
  id: string;
  user_id: string;
  sender?: string;
  subject?: string | null;
  body: string;
  created_at: Date;
  send_time: Date;
  status: StatusEnums;
  in_trash: boolean;
  api_error_code: number | null;
  api_error_details_json: string | null;
  recipients: DBRecipient[];
  sms_reference_id: string;
  cost: number | null;
  cost_currency: string | null;
};

export type AmountIndicators = {
  sent: number;
  scheduled: number;
  failed: number;
  drafts: number;
  trash: number;
  contacts: number;
};
```

/types/theme.ts

```
import { themes } from "@/lib/theme.colors";
```

```ts
export type ThemeProperties = {
  background: string;
  foreground: string;
  card: string;
  cardForeground: string;
  popover: string;
  popoverForeground: string;
  primary: string;
  primaryForeground: string;
  secondary: string;
  secondaryForeground: string;
  muted: string;
  mutedForeground: string;
  accent: string;
  accentForeground: string;
  destructive: string;
  destructiveForeground: string;
  border: string;
  input: string;
  ring: string;
  radius: string;
};

export type Theme = {
  light: ThemeProperties;
  dark: ThemeProperties;
};

export type Themes = {
  [key: string]: Theme;
};

export type ThemeColors = keyof typeof themes; // This will be 'Orange' | '
Blue' | 'Green' | 'Rose' | 'Zinc'

export type ThemeMode = "light" | "dark";
```

/types/user.ts

```ts
export const validSettingNames = [
  "lang",
  "display_name",
  "profile_color_id",

  "primary_color_id",
  "appearance_layout",
  "dark_mode",
];
```

```
export const appearanceLayoutValues = ["MODERN", "SIMPLE"] as const; // thi
s is needed for zod
export type LayoutType = (typeof appearanceLayoutValues)[number];
export type UserSettings = {
  lang: string;

  profile_color_id: number;
  display_name: string;

  dark_mode: boolean;
  primary_color_id: number;
  appearance_layout: LayoutType;
};

export type User = {
  id: string;
  name: string;
  email: string;
  first_name: string;
  last_name: string;
};

// All user fields
export type DBUser = User &
  UserSettings & {
    role: "USER" | "ADMIN";
    created_at?: Date;
    updated_at?: Date;
  };

export type SettingName =
  | "lang"
  | "profile_color_id"
  | "display_name"
  | "primary_color_id"
  | "appearance_layout"
  | "dark_mode";
```

/global.config.ts

```
import { MessageState } from "./contexts/use-new-message";

// These date formats are used for the date-fns library
export const PT_DATE_FORMAT = "dd/MM/yyyy HH:mm";
export const PT_DATE_FORMAT_NO_TIME = "dd/MM/yyyy";
export const ISO8601_DATE_FORMAT = "yyyy-MM-dd";
export const DEFAULT_START_DATE = "2025-01-01";

export const EMPTY_MESSAGE: MessageState = {
  sender: "ETPZP",
```

```
  subject: "",
  recipients: [],
  body: "",
  recipientInput: {
    recipientsExpanded: false,
    value: "",
    error: undefined,
    isHidden: false,
  },
  scheduledDate: new Date(),
  scheduledDateModified: false,
  scheduledDateConfirmed: false,
};

// This is used in the metadata
export const METADATA_APP_NAME = "ETPZP SMS | ";
```

/contexts/use-modal.tsx

```
"use client";

import { Modals } from "@/types";
import React, {
  createContext,
  Dispatch,
  SetStateAction,
  useContext,
  useEffect,
  useState,
} from "react";

const ModalContext = createContext<{
  modal: Modals;
  setModal: Dispatch<SetStateAction<Modals>>;
  scheduleDropdown: boolean;
  setScheduleDropdown: Dispatch<SetStateAction<boolean>>;
} | null>(null);

// These are popups used to work with contacts (create, edit, insert into n
ew message, view more info) used on /contacts and /new-message.
export function ModalProvider({
  children,
}: {
  children: Readonly<React.ReactNode>;
}) {
  const [modal, setModal] = useState<Modals>({
    schedule: false,
    scheduleAlert: false,
    contact: { create: false, edit: false, insert: false, info: false },
  });
```

```
  const [scheduleDropdown, setScheduleDropdown] = useState(false);

  return (
    <ModalContext.Provider
      value={{ modal, setModal, scheduleDropdown, setScheduleDropdown }}
    >
      {children}
    </ModalContext.Provider>
  );
}

export function useModal() {
  const context = useContext(ModalContext);
  if (!context) {
    throw new Error("ModalContext must be within ModalProvider");
  }
  return context;
}
```

/contexts/use-layout.tsx

```
"use client";

import { fetchAmountIndicators } from "@/lib/db/general";
import { AmountIndicators } from "@/types";
import {
  createContext,
  Dispatch,
  SetStateAction,
  useContext,
  useEffect,
  useState,
} from "react";

type LayoutContextType = {
  amountIndicators: AmountIndicators | undefined;
  fallbackLayout: number[];

  layout: number[];
  setLayout: Dispatch<SetStateAction<number[]>>;

  isCollapsed: boolean;
  setIsCollapsed: Dispatch<SetStateAction<boolean>>;

  mobileNavPanel: boolean;
  setMobileNavPanel: Dispatch<SetStateAction<boolean>>;

  isFullscreen: boolean;
  setIsFullscreen: Dispatch<SetStateAction<boolean>>;
```

```
  refetchAmountIndicators: () => void;
};

const LayoutContext = createContext<LayoutContextType | undefined>(undefine
d);

export function LayoutProvider({
  children,
  initialLayout,
  initialIsCollapsed,
  initialAmountIndicators,
}: {
  children: React.ReactNode;
  initialLayout: number[];
  initialIsCollapsed: boolean;
  initialAmountIndicators: AmountIndicators | undefined;
}) {
  // desktop layout 3 column react-resizable-panels data
  const [layout, setLayout] = useState(initialLayout);
  const [isCollapsed, setIsCollapsed] = useState(initialIsCollapsed);
  const fallbackLayout = [20, 32, 48];
  const [amountIndicators, setAmountIndicators] = useState(
    initialAmountIndicators
  );
  // Simple state to keep track of whether the mobile nav panel is open
  const [mobileNavPanel, setMobileNavPanel] = useState(false);
  const [isFullscreen, setIsFullscreen] = useState(false);

  const refetchAmountIndicators = async () => {
    const amountIndicators = await fetchAmountIndicators();

    if (amountIndicators) {
      setAmountIndicators(amountIndicators);
    }
  };

  useEffect(() => {
    setAmountIndicators(initialAmountIndicators);
  }, [initialAmountIndicators]);
  return (
    <LayoutContext.Provider
      value={{
        layout,
        setLayout,
        isCollapsed,
        setIsCollapsed,
        fallbackLayout,
        amountIndicators,
        mobileNavPanel,
        setMobileNavPanel,
        isFullscreen,
```

```
        setIsFullscreen,
        refetchAmountIndicators,
      }}
    >
      {children}
    </LayoutContext.Provider>
  );
}

export function useLayout() {
  const context = useContext(LayoutContext);
  if (context === undefined) {
    throw new Error("useLayout must be used within a LayoutProvider");
  }
  return context;
}
```

/contexts/use-new-message.tsx

```
"use client";

import type React from "react";
import {
  createContext,
  useState,
  useContext,
  useCallback,
  useMemo,
  useEffect,
} from "react";
import { toast } from "sonner";
import type { Message } from "@/types";
import type { DBContact } from "@/types/contact";
import type {
  DBRecipient,
  NewRecipient,
  RankedRecipient,
  WithContact,
} from "@/types/recipient";
import {
  convertToRecipient,
  getUniques,
  matchContactsToRecipients,
  validatePhoneNumber,
} from "@/lib/utils";
import { useContacts } from "./use-contacts";
import { useTranslation } from "react-i18next";
import { z } from "zod";
import { MessageSchema } from "@/lib/form.schemas";
import InsertContactModal from "@/components/modals/insert-contact";
```

```
import CreateContactModal from "@/components/modals/create-contact";
import RecipientInfoModal from "@/components/modals/recipient-info";
import { EMPTY_MESSAGE } from "@/global.config";
import ScheduleMessageModal, {
  ScheduleAlertModal,
} from "@/components/modals/schedule-modals";

// This is our biggest state where we store all data related to the active
message, that should be persisted during draft saving re-renders
// MessageState is only used here & for EMPTY_MESSAGE
export type MessageState = Message & {
  // This is only for the front end composing of the message and will not b
e used on the server
  recipientInput: {
    value: string;
    error?: string;
    isHidden: boolean;
    recipientsExpanded: boolean;
  };
  serverStateErrors?: { [K in keyof z.infer<typeof MessageSchema>]?: string
[] };
  invalidRecipients?: NewRecipient[];

  scheduledDate: Date;
  scheduledDateModified: boolean;
  scheduledDateConfirmed: boolean;
};
type DraftState = {
  id: string | null;
  pending: boolean;
  lastSaveSuccessful: boolean;
};

type MessageContextValues = {
  // Message state
  message: MessageState;
  setMessage: React.Dispatch<React.SetStateAction<MessageState>>;

  // Recipient management
  recipients: NewRecipient[];
  addRecipient: (phone: string) => void;
  removeRecipient: (
    recipient: NewRecipient,
    replaceWithRecipient?: NewRecipient
  ) => void;

  // Recipient search and suggestions
  searchRecipients: (searchTerm: string) => void;
  suggestedRecipients: WithContact[];

  // UI state
  showInfoAbout: React.Dispatch<React.SetStateAction<NewRecipient | null>>;
```

```
  selectedPhone: string | null;
  updateSelectedPhone: (direction: "ArrowDown" | "ArrowUp") => void;

  revalidateRecipients: () => void;
  focusedInput: string | null;
  setFocusedInput: React.Dispatch<React.SetStateAction<string | null>>;

  form: HTMLFormElement | null;
  setForm: React.Dispatch<React.SetStateAction<HTMLFormElement | null>>;
  draft: DraftState;
  setDraft: React.Dispatch<React.SetStateAction<DraftState>>;
};
type ContextProps = {
  children: React.ReactNode;
  rankedRecipients: RankedRecipient[];
  initialMessage?: MessageState;
  draftId: string | null;
};

const NewMessageContext = createContext<MessageContextValues | null>(null);

export function NewMessageProvider({
  children,
  rankedRecipients,
  initialMessage,
  draftId,
}: ContextProps) {
  // Message state
  const [message, setMessage] = useState<MessageState>(
    initialMessage || EMPTY_MESSAGE
  );
  // keep draft state separate because we don't want the draft saver to get
triggered when this data gets updated
  const [draft, setDraft] = useState<DraftState>({
    id: draftId,
    pending: false,
    lastSaveSuccessful: !!initialMessage ? true : false,
  });
  const { contacts } = useContacts();
  const { t } = useTranslation(["new-message-page"]);

  // Associate contacts with matching phone numbers  to recipients
  const initialRecipients: WithContact[] =
    matchContactsToRecipients(rankedRecipients, contacts) || [];

  // UI state
  const [moreInfoOn, showInfoAbout] = useState<NewRecipient | null>(null);
  const [selectedPhone, setSelectedPhone] = useState<string | null>(null);
  const [suggestedRecipients, setSuggestedRecipients] =
    useState(initialRecipients);
  const [focusedInput, setFocusedInput] = useState<string | null>(null);
  const [form, setForm] = useState<HTMLFormElement | null>(null);
```

```
  // Memoized values
  const recommendedRecipients: WithContact[] = useMemo(() => {
    // adjust this to your liking
    const AMOUNT = 10;
    const topRecipients = initialRecipients.slice(0, AMOUNT);

    if (topRecipients.length === AMOUNT) {
      // Check if there are enough topRecipients
      return topRecipients;
    } else {
      // If not look for unused contacts to fill the gap
      const extraContacts: WithContact[] = contacts
        // 1. Filter out the ones that already exist in the top recipients
        .filter(
          (contact) => !topRecipients.some((top) => top.phone === contact.p
hone)
        )
        // 2. Get only the extra ones we need to fill the gap
        .slice(0, AMOUNT - topRecipients.length)
        // 3. Adjust the contacts to match the other recipients in the arra
y
        .map(({ id, phone, name, description }) => ({
          id,
          phone,
          contact: {
            id,
            name,
            phone,
            description,
          },
        }));

      return [...topRecipients, ...extraContacts] as WithContact[];
    }
  }, [contacts]);

  // Helper functions
  const revalidateRecipients = () => {
    setMessage((prevMessage) => ({
      // For some reason this inner part gets run twice while the outer fun
ction only gets run once
      ...prevMessage,
      recipients: prevMessage.recipients.map((recipient, index) => {
        const foundContact = contacts.find(
          (contact) => contact.phone === recipient.phone
        );

        if (foundContact) {
          return { ...recipient, contact: foundContact };
        } else return recipient;
      }),
```

```
    }));
  };

  const DEFAULT_SELECTED_PHONE_INDEX = null;
  // Recipient management functions
  const addRecipient = (phone: string) => {
    if (message.recipients.some((item) => item.phone === phone)) {
      // I know this is not on the server, but I wanted to keep the same fo
rmat
      return toast.error(t("server-duplicate_recipients_error"), {
        description: t("server-duplicate_recipients_error_caption"),
      });
    }

    setMessage((prev) => {
      const validatedRecipient = validatePhoneNumber(phone);
      const foundContact = contacts.find((contact) => contact.phone === pho
ne);
      return {
        ...prev,
        recipients: [
          ...prev.recipients,
          // In case `recipientWithContact` has some old fields
          {
            ...validatedRecipient,
            contact: foundContact
              ? convertToRecipient(foundContact).contact
              : undefined,
          },
        ],
      };
    });

    // Update selectedPhone to the next available recipient
    setSelectedPhone((prevSelected) => {
      if (prevSelected === phone) {
        const nextRecipient = suggestedRecipients.find(
          (r) => r.phone !== phone
        );
        return nextRecipient ? nextRecipient.phone : null;
      }
      return prevSelected;
    });
    // Reset the input and search:
    setMessage((m) => ({
      ...m,
      recipientInput: { ...m.recipientInput, value: "" },
      recipients: m.recipients.map((r) => ({
        ...r,
        proneForDeletion: false,
      })),
    }));
  };
```

```
  const removeRecipient = useCallback(
    (recipient: NewRecipient, replaceWithRecipient?: NewRecipient) => {
      setMessage((prev) => ({
        ...prev,
        // recipientInput: { ...prev.recipientInput, value: "" },
        recipients: prev.recipients
          .map((r) => (r === recipient ? replaceWithRecipient : r))
          .filter((r) => r !== undefined), // Filter out undefined values
      }));
    },
    []
  );

  // Search and suggestion functions
  const searchRecipients = (rawSearchTerm: string) => {
    const searchTerm = rawSearchTerm.trim().toLowerCase();
    if (!suggestedRecipients.length && !recommendedRecipients.length) {
      // Searched suggested- and recommended recipients are empty -
      // All recipients from the suggested list have already been added!
      return setSelectedPhone(null);
    }

    // There are still suggested recipients that haven't been added yet, so
do additional checks
    if (searchTerm.length) {
      const filteredRecipients = getUniques(
        message.recipients,
        initialRecipients.filter(
          (recipient) =>
            (recipient.contact?.name?.toLowerCase().includes(searchTerm) ||
              recipient.phone.toLowerCase().includes(searchTerm)) &&
            !message.recipients.some((r) => r.phone === recipient.phone)
        )
      );
      setSuggestedRecipients(filteredRecipients);

      if (!filteredRecipients.length) {
        // No recipients found (the suggested panel will be hidden) - desel
ect the previous phone
        setSelectedPhone(null);
      } else {
        setSelectedPhone(
          DEFAULT_SELECTED_PHONE_INDEX
            ? filteredRecipients[DEFAULT_SELECTED_PHONE_INDEX]?.phone
            : DEFAULT_SELECTED_PHONE_INDEX
        );
      }
    } else {
      setSuggestedRecipients(
        getUniques(message.recipients, recommendedRecipients)
      );
```

```
      setSelectedPhone(
        DEFAULT_SELECTED_PHONE_INDEX
          ? recommendedRecipients[DEFAULT_SELECTED_PHONE_INDEX]?.phone
          : DEFAULT_SELECTED_PHONE_INDEX
      );
    }
  };

  // UI update functions
  const updateSelectedPhone = useCallback(
    (input: "ArrowDown" | "ArrowUp") => {
      setSelectedPhone((prevPhone) => {
        const currentIndex = suggestedRecipients.findIndex(
          (item) => item.phone === prevPhone
        );
        const length = suggestedRecipients.length;
        const newIndex =
          input === "ArrowUp"
            ? (currentIndex - 1 + length) % length
            : (currentIndex + 1) % length;
        return suggestedRecipients[newIndex]?.phone;
      });
    },
    [suggestedRecipients]
  );

  useEffect(() => {
    // Revalidate recipients when contacts get re-fetched
    revalidateRecipients();
  }, [contacts]);

  useEffect(() => {
    if (!!initialMessage === false) {
      // If initialMessage is undefined, reset all the controlled inputs to
an empty value
      setMessage(EMPTY_MESSAGE);
    }
  }, [initialMessage]);

  // When recipients change do this:
  useEffect(() => {
    // If we still freshly have the invalid recipients error
    if (message.invalidRecipients) {
      const validRecipientExists = !!message.recipients.find(
        (r) => r.isValid === true
      );
      if (validRecipientExists) {
        // If the new recipient is valid, we clear the error, allowing erro
r pulsing for more invalid recipients.
        setMessage((prev) => ({ ...prev, invalidRecipients: undefined }));
      }
    }
```

```
    // Note: Works only correctly here; won't update correctly with add/rem
ove operations.
    searchRecipients(message.recipientInput.value);
  }, [message.recipients]);
  return (
    <NewMessageContext.Provider
      value={{
        message,
        setMessage,
        recipients: message.recipients,
        addRecipient,
        removeRecipient,
        suggestedRecipients,
        searchRecipients,

        showInfoAbout,
        selectedPhone,
        updateSelectedPhone,
        revalidateRecipients,
        focusedInput,
        setFocusedInput,

        form,
        setForm,
        draft,
        setDraft,
      }}
    >
      {/* We move modals here, because unlike the form component, this does
n't re-render when a draft gets saved */}
      <InsertContactModal />
      <ScheduleMessageModal />
      <ScheduleAlertModal />
      {/* This should always be defined as we pass a defaultPhone and may c
reate a contact from scratch. */}
      <CreateContactModal
        defaultPhone={moreInfoOn?.phone}
        onCreateSuccess={(contact) => {
          // After creating the new contact, replace the old recipient
          const oldRecipient = message.recipients.find(
            (r) => r.phone == moreInfoOn?.phone
          );
          const newRecipient = convertToRecipient(contact);
          showInfoAbout(newRecipient);
          if (oldRecipient) {
            removeRecipient(oldRecipient, newRecipient);
          }
        }}
      />

      {moreInfoOn && (
```

```
        <RecipientInfoModal recipient={moreInfoOn} allowContactCreation />
      )}
      {children}
    </NewMessageContext.Provider>
  );
}

export function useNewMessage() {
  const context = useContext(NewMessageContext);
  if (!context) {
    throw new Error("useNewMessage must be used within a NewMessageProvider
");
  }
  return context;
}
```

/contexts/use-settings.tsx

```
"use client";

import { useThemeContext } from "@/contexts/theme-data-provider";
import { i18nConfig } from "@/i18n.config";
import { fetchUserSettings } from "@/lib/db/general";
import { usePathname, useRouter } from "next/navigation";
import { useTheme as useNextTheme } from "next-themes";
import {
  createContext,
  Dispatch,
  SetStateAction,
  useContext,
  useEffect,
  useState,
} from "react";
import { LayoutType } from "@/types/user";
import useIsMounted from "@/hooks/use-mounted";

type SettingsState = {
  displayName?: string;
  profileColorId?: number;
  layout: LayoutType | undefined;
};

type SettingsContext = {
  settings: SettingsState;
  setSettings: Dispatch<SetStateAction<SettingsState>>;
  updateLanguageCookie: (newLocale: string) => void;
  normalizePath: (path: string) => string;
  // hasLanguageCookie: () => boolean; not used outside as of now
  syncWithDB: () => Promise<void>;
  resetLocalSettings: () => void;
```

```
};

const SettingsContext = createContext<SettingsContext | null>(null);

export function SettingsProvider({
  children,
  currentLocale,
}: {
  children: Readonly<React.ReactNode>;
  currentLocale: string;
}) {
  const isMounted = useIsMounted();
  // Localstorage state without theme color (primary_color) and theme mode
because those are handled internally by our packages
  const [settings, setSettings] = useState<SettingsState>({
    displayName: localStorage.getItem("display_name") || undefined,
    profileColorId:
      Number(localStorage.getItem("profile_color_id")) || undefined,
    layout:
      (localStorage.getItem("appearance_layout") as LayoutType) || undefine
d,
  });

  const router = useRouter();
  const currentPathname = usePathname();
  const { setThemeColor } = useThemeContext();
  const { setTheme } = useNextTheme();

  // Helper function to normalize paths
  function normalizePath(path: string) {
    const defaultLocale = i18nConfig.defaultLocale as string;

    // Remove leading slash and split into segments
    const segments = path.replace(/^\//, "").split("/");

    // If the first segment is a locale and it's not the default, remove it
    if (segments[0] === currentLocale && currentLocale !== defaultLocale) {
      segments.shift();
    }

    return "/" + segments.join("/");
  }

  const updateLanguageCookie = (newLocale: string) => {
    // set cookie for next-i18n-router
    const days = 30;
    const date = new Date();
    date.setTime(date.getTime() + days * 24 * 60 * 60 * 1000);
    const expires = date.toUTCString();
    document.cookie = `NEXT_LOCALE=${newLocale};expires=${expires};path=/`;

    // redirect to the new locale path
```

```
    if (
      currentLocale === i18nConfig.defaultLocale &&
      !i18nConfig.prefixDefault
    ) {
      router.push("/" + newLocale + currentPathname);
    } else {
      router.push(
        currentPathname.replace(`/${currentLocale}`, `/${newLocale}`)
      );
    }

    router.refresh();
  };

  const hasLanguageCookie = () => {
    const cookies = document.cookie.split(";").map((cookie) => cookie.trim(
));
    return cookies.some((cookie) =>
      cookie.startsWith("NEXT_LOCALE=")
    ) as boolean;
  };

  const syncWithDB = async () => {
    const settings = await fetchUserSettings();

    if (settings) {
      const {
        profile_color_id,
        display_name,
        dark_mode,
        primary_color_id,
        lang,
        appearance_layout,
      } = settings;
      // Profile
      localStorage.setItem("profile_color_id", profile_color_id.toString())
;
      localStorage.setItem("display_name", display_name);

      // Appearance
      setTheme(dark_mode === true ? "dark" : "light"); // theme is stored a
s strings because we are using next-themes
      setThemeColor(primary_color_id);
      localStorage.setItem("appearance_layout", appearance_layout);

      // Language - this comes last because it will refresh the page, which
might cause issues
      updateLanguageCookie(lang);

      // Update components when localstorage settings change
      setSettings({
        displayName: display_name,
```

```
        profileColorId: profile_color_id,
        layout: appearance_layout,
      });
    }
  };

  const resetLocalSettings = () => {
    localStorage.clear();
    setTheme("light");
    setThemeColor(1);
    updateLanguageCookie(i18nConfig.defaultLocale);
  };

  // This will also get triggered on load
  useEffect(() => {
    const referenceHeaderHeight = parseInt(
      getComputedStyle(document.documentElement).getPropertyValue(
        "--simple-header-height"
      ),
      10 // base 10 integer
    );
    if (settings.layout === "MODERN") {
      document.documentElement.style.setProperty(
        "--header-height",
        `${referenceHeaderHeight * 2}px`
      );
    } else if (settings.layout === "SIMPLE") {
      document.documentElement.style.setProperty(
        "--header-height",
        `${referenceHeaderHeight}px`
      );
    }
  }, [settings.layout]);
  useEffect(() => {
    if (isMounted) {
      if (
        localStorage.getItem("profile_color_id") == null ||
        localStorage.getItem("display_name") == null ||
        localStorage.getItem("primary_color_id") == null ||
        localStorage.getItem("theme") == null ||
        hasLanguageCookie() === false
      ) {
        syncWithDB();
      }
    }
  }, [isMounted]);

  return (
    <SettingsContext.Provider
      value={{
        settings,
        setSettings,
        updateLanguageCookie,
```

```
        normalizePath,
        // hasLanguageCookie,
        syncWithDB,
        resetLocalSettings,
      }}
    >
      {children}
    </SettingsContext.Provider>
  );
}

export function useSettings() {
  const context = useContext(SettingsContext);
  if (!context) {
    throw new Error("SettingsContext must be within SettingsProvider");
  }
  return context;
}
```

/contexts/theme-data-provider.tsx

```
"use client";

import setGlobalColorTheme from "@/lib/theme.colors";
import { ThemeProviderProps, useTheme as useNextTheme } from "next-themes";
import React, { createContext, useContext, useEffect, useState } from "reac
t";

type ThemeColorStateParams = {
  themeColor: number;
  setThemeColor: React.Dispatch<React.SetStateAction<number>>;
};
const ThemeContext = createContext<ThemeColorStateParams>(
  {} as ThemeColorStateParams
);

export default function ThemeDataProvider({ children }: ThemeProviderProps)
{
  if (typeof localStorage === "undefined") {
    return null;
  }
  const getSavedThemeColor = (): number => {
    return Number(localStorage.getItem("primary_color_id")) || 1;
  };

  const { theme } = useNextTheme();
  const [themeColor, setThemeColor] = useState<number>(getSavedThemeColor()
);
  const [isMounted, setIsMounted] = useState<boolean>(false);
```

```
  useEffect(() => {
    localStorage.setItem("primary_color_id", themeColor.toString());
    setGlobalColorTheme(theme as "light" | "dark", themeColor);

    if (!isMounted) {
      setIsMounted(true);
    }
  }, [themeColor, theme, isMounted]);
  if (!isMounted) {
    return null;
  }

  return (
    <ThemeContext.Provider value={{ themeColor, setThemeColor }}>
      {children}
    </ThemeContext.Provider>
  );
}

export function useThemeContext() {
  return useContext(ThemeContext);
}
```

/contexts/use-contacts.tsx

```
"use client";

import { fetchContacts } from "@/lib/db/contact";
import { DBContact } from "@/types/contact";
import React, { createContext, useContext, useEffect, useState } from "reac
t";
import { useTranslation } from "react-i18next";

type ContactContextValues = {
  contacts: DBContact[];
  refetchContacts: () => void;
  contactFetchError: string | null;
};

const ContactsContext = createContext<ContactContextValues | null>(null);

export function ContactsProvider({
  children,
  initialContacts,
}: {
  children: Readonly<React.ReactNode>;
  initialContacts: DBContact[] | undefined;
}) {
  const { t } = useTranslation(["contacts-page"]);
  const [contacts, setContacts] = useState<DBContact[]>(initialContacts ||
```

```
[]);
  const unknownFetchError = t("fetch_error");
  const [error, setError] = useState<string | null>(
    initialContacts === undefined ? unknownFetchError : null
  );

  const refetchContacts = async () => {
    const newContacts = await fetchContacts();
    setContacts(newContacts || []);

    if (newContacts === undefined) {
      setError(unknownFetchError);
    }
  };

  return (
    <ContactsContext.Provider
      value={{ contacts, refetchContacts, contactFetchError: error }}
    >
      {children}
    </ContactsContext.Provider>
  );
}

export function useContacts() {
  const context = useContext(ContactsContext);
  if (!context) {
    throw new Error("ContactsContext must be within ContactsProvider");
  }
  return context;
}
```

/contexts/translations-provider.jsx

```
"use client";

import { I18nextProvider } from "react-i18next";
import initTranslations from "@/app/i18n";
import { createInstance } from "i18next";

// This provider is for client-component useTranslation() hook
export default function TranslationsProvider({
  children,
  locale,
  namespaces,
  resources,
}) {
  const i18n = createInstance();

  initTranslations(locale, namespaces, i18n, resources);
```

```
  return <I18nextProvider i18n={i18n}>{children}</I18nextProvider>;
}
```

/app/[locale]/(root)/(message-layout)/drafts/loading.tsx

```
import MessagesPageSkeleton from "@/components/messages-page-skeleton";

export default function Loading() {
  return <MessagesPageSkeleton category="DRAFTS" />;
}
```

/app/[locale]/(root)/(message-layout)/drafts/page.tsx

```
import initTranslations from "@/app/i18n";
import MessagesPage from "@/components/messages-page";
import { METADATA_APP_NAME } from "@/global.config";
import { fetchMessagesByStatus } from "@/lib/db/message";

export default async function Page() {
  const messages = await fetchMessagesByStatus("DRAFTED");

  return (
    <MessagesPage
      messages={messages || []}
      error={messages === undefined}
      category="DRAFTS"
    />
  );
}

export async function generateMetadata({
  params,
}: {
  params: Promise<{ locale: string }>;
}) {
  const { locale } = await params;
  const { t } = await initTranslations(locale, ["metadata"]);

  return {
    title: METADATA_APP_NAME + t("drafts-title"),
    description: t("drafts-description"),
  };
}
```

/app/[locale]/(root)/(message-layout)/contacts/loading.tsx

```tsx
import ContactsPageSkeleton from "@/components/contacts-page-skeleton";

export default function Loading() {
  return <ContactsPageSkeleton />;
}
```

/app/[locale]/(root)/(message-layout)/contacts/page.tsx

```tsx
import ContactsPage from "@/components/contacts-page";
import { ModalProvider } from "@/contexts/use-modal";
import initTranslations from "@/app/i18n";
import { METADATA_APP_NAME } from "@/global.config";

export default async function Page() {
  return (
    <ModalProvider>
      <ContactsPage />
    </ModalProvider>
  );
}

export async function generateMetadata({
  params,
}: {
  params: Promise<{ locale: string }>;
}) {
  const { locale } = await params;
  const { t } = await initTranslations(locale, ["metadata"]);

  return {
    title: METADATA_APP_NAME + t("contacts-title"),
    description: t("contacts-description"),
  };
}
```

/app/[locale]/(root)/(message-layout)/trash/loading.tsx

```tsx
import MessagesPageSkeleton from "@/components/messages-page-skeleton";

export default function Loading() {
  return <MessagesPageSkeleton category="TRASH" />;
}
```

/app/[locale]/(root)/(message-layout)/trash/page.tsx

```tsx
import initTranslations from "@/app/i18n";
import MessagesPage from "@/components/messages-page";
import { METADATA_APP_NAME } from "@/global.config";
import { fetchTrashedMessages } from "@/lib/db/message";

export default async function Page() {
  const messages = await fetchTrashedMessages();

  return (
    <MessagesPage
      messages={messages || []}
      error={messages === undefined}
      category="TRASH"
    />
  );
}

export async function generateMetadata({
  params,
}: {
  params: Promise<{ locale: string }>;
}) {
  const { locale } = await params;
  const { t } = await initTranslations(locale, ["metadata"]);

  return {
    title: METADATA_APP_NAME + t("trash-title"),
    description: t("trash-description"),
  };
}
```

/app/[locale]/(root)/(message-layout)/sent/loading.tsx

```tsx
import MessagesPageSkeleton from "@/components/messages-page-skeleton";

export default function Loading() {
  return <MessagesPageSkeleton category="SENT" />;
}
```

/app/[locale]/(root)/(message-layout)/sent/page.tsx

```tsx
import initTranslations from "@/app/i18n";
import MessagesPage from "@/components/messages-page";
import { METADATA_APP_NAME } from "@/global.config";
```

```
import { fetchSentIn } from "@/lib/db/message";

export default async function Page() {
  const messages = await fetchSentIn("PAST");

  return (
    <MessagesPage
      messages={messages || []}
      error={messages === undefined}
      category="SENT"
    />
  );
}

export async function generateMetadata({
  params,
}: {
  params: Promise<{ locale: string }>;
}) {
  const { locale } = await params;
  const { t } = await initTranslations(locale, ["metadata"]);

  return {
    title: METADATA_APP_NAME + t("sent-title"),
    description: t("sent-description"),
  };
}
```

/app/[locale]/(root)/(message-layout)/failed/loading.tsx

```
import MessagesPageSkeleton from "@/components/messages-page-skeleton";

export default function Loading() {
  return <MessagesPageSkeleton category="FAILED" />;
}
```

/app/[locale]/(root)/(message-layout)/failed/page.tsx

```
import initTranslations from "@/app/i18n";
import MessagesPage from "@/components/messages-page";
import { METADATA_APP_NAME } from "@/global.config";
import { fetchMessagesByStatus } from "@/lib/db/message";

export default async function Page() {
  const messages = await fetchMessagesByStatus("FAILED");

  return (
```

```
    <MessagesPage
      messages={messages || []}
      error={messages === undefined}
      category="FAILED"
    />
  );
}

export async function generateMetadata({
  params,
}: {
  params: Promise<{ locale: string }>;
}) {
  const { locale } = await params;
  const { t } = await initTranslations(locale, ["metadata"]);

  return {
    title: METADATA_APP_NAME + t("failed-title"),
    description: t("failed-description"),
  };
}
```

/app/[locale]/(root)/(message-layout)/layout.tsx

```
import initTranslations from "@/app/i18n";
import TranslationsProvider from "@/contexts/translations-provider";
import { ContactsProvider } from "@/contexts/use-contacts";
import { fetchContacts } from "@/lib/db/contact";

type LayoutProps = Readonly<{
  children: React.ReactNode;
  params: Promise<{ locale: string }>;
}>;

export default async function TranslationLayout({
  children,
  params,
}: LayoutProps) {
  // Internationalization (i18n) stuff
  const i18nNamespaces = [
    "messages-page",
    "contacts-page",
    "modals",
    "common",
    "errors",
  ];
  const { locale } = await params;
  const { resources } = await initTranslations(locale, i18nNamespaces);

  return (
```

```
    /* This is a client layout component containing the translation provide
r for the nav panel */
    <TranslationsProvider
      /* Only wrap what's necessary with the TranslationsProvider */
      resources={resources}
      locale={locale}
      namespaces={i18nNamespaces}
    >
      <ContactsProvider initialContacts={(await fetchContacts()) || []}>
        {children}
      </ContactsProvider>
    </TranslationsProvider>
  );
}
```

/app/[locale]/(root)/(message-layout)/error.tsx

```
"use client";

import ChildrenPanel from "@/components/shared/children-panel";
import ErrorComponent from "@/components/shared/error-component";
import { Button } from "@/components/ui/button";
import { useEffect } from "react";
import { useTranslation } from "react-i18next";

export default function Error({
  error,
  reset,
}: {
  error: Error & { digest?: string };
  reset: () => void;
}) {
  const { t } = useTranslation(["errors"]);

  useEffect(() => {
    // Log the error to an error reporting service
    console.error(error);
  }, [error]);
  return (
    <ChildrenPanel>
      <ErrorComponent
        title={t("error-header")}
        subtitle={t("error-header_caption")}
      >
        <Button
          onClick={
            // Attempt to recover by trying to re-render the segment
            () => reset()
          }
        >
```

```
          {t("try_again")}
        </Button>
      </ErrorComponent>
    </ChildrenPanel>
  );
}
```

/app/[locale]/(root)/(message-layout)/scheduled/loading.tsx

```tsx
import MessagesPageSkeleton from "@/components/messages-page-skeleton";

export default function Loading() {
  return <MessagesPageSkeleton category="SCHEDULED" />;
}
```

/app/[locale]/(root)/(message-layout)/scheduled/page.tsx

```tsx
import initTranslations from "@/app/i18n";
import MessagesPage from "@/components/messages-page";
import { METADATA_APP_NAME } from "@/global.config";
import { fetchSentIn } from "@/lib/db/message";

export default async function Page() {
  const messages = await fetchSentIn("FUTURE");

  return (
    <MessagesPage
      messages={messages || []}
      error={messages === undefined}
      category="SCHEDULED"
    />
  );
}

export async function generateMetadata({
  params,
}: {
  params: Promise<{ locale: string }>;
}) {
  const { locale } = await params;
  const { t } = await initTranslations(locale, ["metadata"]);

  return {
    title: METADATA_APP_NAME + t("scheduled-title"),
    description: t("scheduled-description"),
  };
}
```

/app/[locale]/(root)/layout.tsx

```tsx
import initTranslations from "@/app/i18n";
import AppLayout from "@/components/app-layout";
import { SettingsProvider } from "@/contexts/use-settings";
import { METADATA_APP_NAME } from "@/global.config";

type LayoutProps = Readonly<{
  children: React.ReactNode;
  params: Promise<{ locale: string }>;
}>;

export default async function NavPanelLayout({
  children,
  params,
}: LayoutProps) {
  // Internationalization (i18n) stuff - no need to include errors namespac
e as we only put in more specific locations
  const i18nNamespaces = ["navigation", "welcome-page", "modals", "common"]
;
  const { locale } = await params;
  const { resources } = await initTranslations(locale, i18nNamespaces);

  return (
    <SettingsProvider currentLocale={locale}>
      <AppLayout
        /* This is a client layout component containing the translation pro
vider for the nav panel */
        resources={resources}
        locale={locale}
        namespaces={i18nNamespaces}
      >
        {children}
      </AppLayout>
    </SettingsProvider>
  );
}

export async function generateMetadata({
  params,
}: {
  params: Promise<{ locale: string }>;
}) {
  const { locale } = await params;
  const { t } = await initTranslations(locale, ["metadata"]);

  return {
    title: METADATA_APP_NAME + t("welcome-title"),
    description: t("welcome-description"),
```

```
  };
}
```

/app/[locale]/(root)/page.tsx

```
"use client";
import ChildrenPanel from "@/components/shared/children-panel";
import { useLayout } from "@/contexts/use-layout";
import Link from "next/link";
import { cn } from "@/lib/utils";
import { buttonVariants } from "@/components/ui/button";
import { Trans, useTranslation } from "react-i18next";
import LinkCard from "@/components/cards";
import { useThemeContext } from "@/contexts/theme-data-provider";
import Envelope from "@/public/icons/envelope-solid.svg";
import Contact from "@/public/icons/user-solid.svg";
import { PageHeader } from "@/components/headers";
import { ScrollArea } from "@/components/ui/scroll-area";
import { useIsMobile } from "@/hooks/use-mobile";

export default function WelcomePage() {
  const { amountIndicators } = useLayout();
  const { themeColor } = useThemeContext();
  const onMobile = useIsMobile();

  const { t, i18n } = useTranslation(["welcome-page"]);
  const gradientStyle = {
    fontSize: "48px", // Adjust the font size as needed
    fontWeight: "bold", // Make the text bold
    background: `linear-gradient(135deg, ${themeColor}, orange`, // Diagona
l gradient using CSS variables
    WebkitBackgroundClip: "text", // Clip the background to the text
    WebkitTextFillColor: "transparent", // Make the text color transparent
    display: "inline-block", // Ensure the gradient applies correctly
  };
  return (
    <ChildrenPanel>
      <ScrollArea className="h-full">
        {onMobile && <PageHeader />}

        <div className="flex-1 flex flex-col p-4 min-h-[calc(100vh-var(--si
mple-header-height))]">
          <div className="flex-1 flex flex-col items-center justify-center
gap-10">
            {/* <PageHeader title="Welcome to the Etpzp SMS App!" /> */}

            <div className="text-center">
              <span className="text-xl text-muted-foreground block">
                {t("welcome_message")}{" "}
              </span>
```

```
            <span className="text-6xl leading-tighter gradient-text">
              ETPZP-SMS
            </span>
          </div>

          {/*  */}
          <div className="flex flex-col xs:flex-row gap-2 w-full justify-
center items-center">
            <LinkCard
              href="/contacts"
              heroValue={amountIndicators?.contacts || 0}
              Icon={Contact}
              title={t("card_1-title")}
            />
            <LinkCard
              href="/sent"
              heroValue={
                (amountIndicators?.sent || 0) +
                (amountIndicators?.scheduled || 0)
              }
              Icon={Envelope}
              title={t("card_2-title")}
            />
          </div>
        </div>

        <p className="text-sm text-center my-8" /**mb-12 */>
          {t("developer_credit")}{" "}
          <Link
            href="https://github.com/devdogfish"
            className={cn(
              buttonVariants({ variant: "link" }),
              "p-0 h-min"
              // "underline hover:no-underline"
            )}
            target="_blank"
          >
            Luigi Girke
          </Link>
        </p>
      </div>
    </ScrollArea>
  </ChildrenPanel>
 );
}
```

/app/[locale]/(root)/(other)/_seed/page.tsx

```
import ChildrenPanel from "@/components/shared/children-panel";
import db from "@/lib/db";
```

```
// Function to generate a random date up to 3 years ago
function getRandomDate() {
  const now = new Date();
  const threeYearsAgo = new Date(now.setFullYear(now.getFullYear() - 2));
  const randomDate = new Date(
    threeYearsAgo.getTime() +
      Math.random() * (Date.now() - threeYearsAgo.getTime())
  );
  return randomDate;
}

// Function to generate random message data
function getRandomMessageData() {
  const users = Array.from({ length: 10 }, (_, i) => i + 1); // User IDs fr
om 1 to 10
  const subjects = [
    "Hello",
    "Meeting Reminder",
    "Invoice",
    "Newsletter",
    "Promotion",
  ];
  const bodies = [
    "This is a test message.",
    "Don't forget about our meeting tomorrow.",
    "Your invoice is attached.",
    "Check out our latest newsletter.",
    "Exclusive offer just for you!",
  ];
  const statuses = ["SENT", "SCHEDULED", "FAILED", "DRAFTED"];

  return {
    user_id: users[Math.floor(Math.random() * users.length)],
    sender: `user${Math.floor(Math.random() * 10) + 1}@example.com`,
    subject: subjects[Math.floor(Math.random() * subjects.length)],
    body: bodies[Math.floor(Math.random() * bodies.length)],
    send_time: getRandomDate(),
    status: statuses[Math.floor(Math.random() * statuses.length)],
    in_trash: false, // Randomly true or false
    cost: parseFloat((Math.random() * 0.1).toFixed(4)), // Random cost betw
een 0.0 and 0.1
    cost_currency: "EUR",
  };
}

// Function to insert a message into the database
async function insertMessage() {
  const messageData = getRandomMessageData();

  const query = `
        INSERT INTO "message" (user_id, sender, subject, body, send_time, s
```

```
tatus, in_trash, cost, cost_currency)
        VALUES ($1, $2, $3, $4, $5, $6, $7, $8, $9)
    `;

  const values = [
    messageData.user_id,
    messageData.sender,
    messageData.subject,
    messageData.body,
    messageData.send_time,
    messageData.status,
    messageData.in_trash,
    messageData.cost,
    messageData.cost_currency,
  ];

  try {
    await db(query, values);
    console.log("Message inserted successfully:", messageData);
  } catch (err) {
    console.error("Error inserting message:", err);
  }
}

async function insertUsers() {
  try {
    const result = await db(
      `
        INSERT INTO "user" (name, email, role, created_at, updated_at, firs
t_name, last_name, lang, profile_color_id, display_name, dark_mode, primary
_color_id)
        VALUES
          ('Alice Johnson', 'alice@example.com', 'USER', NOW(), NOW(), 'Ali
ce', 'Johnson', 'en', 1, 'Alice J.', false, 1),
          ('Bob Smith', 'bob@example.com', 'USER', NOW(), NOW(), 'Bob', 'Sm
ith', 'en', 1, 'Bob S.', false, 1),
          ('Charlie Brown', 'charlie@example.com', 'ADMIN', NOW(), NOW(), '
Charlie', 'Brown', 'en', 1, 'Charlie B.', false, 1),
          ('David Wilson', 'david@example.com', 'USER', NOW(), NOW(), 'Davi
d', 'Wilson', 'pt', 1, 'David W.', false, 1),
          ('Eve Davis', 'eve@example.com', 'ADMIN', NOW(), NOW(), 'Eve', 'D
avis', 'pt', 1, 'Eve D.', true, 1),
          ('Frank Miller', 'frank@example.com', 'USER', NOW(), NOW(), 'Fran
k', 'Miller', 'en', 1, 'Frank M.', false, 1),
          ('Grace Lee', 'grace@example.com', 'USER', NOW(), NOW(), 'Grace',
'Lee', 'en', 1, 'Grace L.', false, 1),
          ('Hank Green', 'hank@example.com', 'USER', NOW(), NOW(), 'Hank',
'Green', 'pt', 1, 'Hank G.', true, 1),
          ('Irene Taylor', 'irene@example.com', 'ADMIN', NOW(), NOW(), 'Ire
ne', 'Taylor', 'en', 1, 'Irene T.', false, 1),
          ('Jack White', 'jack@example.com', 'USER', NOW(), NOW(), 'Jack',
'White', 'pt', 1, 'Jack W.', false, 1);
    `
```

```
    );
    console.log("Users inserted successfully:", result.rows);
  } catch (err) {
    console.error("Error inserting users:", err);
  }
}

// Call the function to insert a message
export default async function Page() {
  await insertUsers();
  for (let i = 1; i <= 300; i++) {
    await insertMessage();
  }

  return (
    <ChildrenPanel>
      <div className="centered">Seeded successfully</div>
    </ChildrenPanel>
  );
}
```

/app/[locale]/(root)/(other)/settings/layout.tsx

```
import initTranslations from "@/app/i18n";
import TranslationsProvider from "@/contexts/translations-provider";
import ChildrenPanel from "@/components/shared/children-panel";
import { METADATA_APP_NAME } from "@/global.config";

type LayoutProps = Readonly<{
  children: React.ReactNode;
  params: Promise<{ locale: string }>;
}>;

export default async function TranslationLayout({
  children,
  params,
}: LayoutProps) {
  // Internationalization (i18n) stuff
  const i18nNamespaces = ["settings-page", "common", "errors"];
  const { locale } = await params;
  const { resources } = await initTranslations(locale, i18nNamespaces);

  return (
    /* This is a client layout component containing the translation provide
r for the nav panel */
    <TranslationsProvider
      /* Only wrap what's necessary with the TranslationsProvider */
      resources={resources}
      locale={locale}
      namespaces={i18nNamespaces}
```

```
    >
      <ChildrenPanel>{children}</ChildrenPanel>
    </TranslationsProvider>
  );
}

export async function generateMetadata({
  params,
}: {
  params: Promise<{ locale: string }>;
}) {
  const { locale } = await params;
  const { t } = await initTranslations(locale, ["metadata"]);

  return {
    title: METADATA_APP_NAME + t("settings-title"),
    description: t("settings-description"),
  };
}
```

/app/[locale]/(root)/(other)/settings/error.tsx

```
"use client";

import ChildrenPanel from "@/components/shared/children-panel";
import ErrorComponent from "@/components/shared/error-component";
import { Button } from "@/components/ui/button";
import { useEffect } from "react";
import { useTranslation } from "react-i18next";

export default function Error({
  error,
  reset,
}: {
  error: Error & { digest?: string };
  reset: () => void;
}) {
  const { t } = useTranslation(["errors"]);

  useEffect(() => {
    // Log the error to an error reporting service
    console.error(error);
  }, [error]);
  return (
    <ErrorComponent
      title={t("error-header")}
      subtitle={t("error-header_caption")}
    >
      <Button
        onClick={
```

```
          // Attempt to recover by trying to re-render the segment
          () => reset()
        }
      >
        {t("try_again")}
      </Button>
    </ErrorComponent>
  );
}
```

/app/[locale]/(root)/(other)/settings/page.tsx

```
"use client";

import { PageHeader, SectionHeader } from "@/components/headers";
import {
  LanguageChanger,
  ThemeToggle,
  ThemeColorChanger,
  createSelectItems,
  ColorDropdown,
} from "@/components/settings";
import { Button, buttonVariants } from "@/components/ui/button";
import {
  Select,
  SelectContent,
  SelectItem,
  SelectTrigger,
  SelectValue,
} from "@/components/ui/select";
import { useTranslation } from "react-i18next";
import SettingsItem from "../../../../../components/settings-item";
import { cn } from "@/lib/utils";
import { useTheme as useNextTheme } from "next-themes";
import { useThemeContext } from "@/contexts/theme-data-provider";
import { ScrollArea } from "@/components/ui/scroll-area";
import { useIsMobile } from "@/hooks/use-mobile";

export default function Settings() {
  const { t } = useTranslation();
  const { theme } = useNextTheme();
  const { themeColor, setThemeColor } = useThemeContext();
  const onMobile = useIsMobile();

  const initialValues = {
    profile: {
      displayName:
        localStorage.getItem("display_name") || "Initial display name",
      colorId: localStorage.getItem("profile_color_id") || undefined,
    },
```

```
    appearance: {
      darkMode: theme,
      layout: localStorage.getItem("appearance_layout") || "MODERN",
      primaryColor: themeColor.toString(),
    },
  };

  return (
    <>
      <PageHeader title={t("header")} />

      <ScrollArea
        className={
          onMobile
            ? "h-[calc(100vh-var(--simple-header-height))]"
            : "h-[calc(100vh-var(--header-height))]"
        }
      >
        <div
          className="p-4" /* Inside looks better with rimless bottom on scr
oll on scroll */
        >
          <div className="space-y-12">
            <SectionHeader
              title={t("language-header")}
              subtitle={t("language-header_caption")}
              anchorName="language"
            >
              <SettingsItem
                name="lang"
                label={t("language-language_label")}
                caption={t("language-language_label_caption")}
                renderInput={({
                  value,
                  onChange,
                  onBlur,
                  id,
                  isPending,
                  setServerState,
                }) => {
                  return (
                    <LanguageChanger
                      // This component has custom behavior—only select pro
ps are used as it handles its own submission,
                      // and setServerState is passed so elements update wi
th errors.
                      id={id}
                      value={value}
                      onChange={onChange}
                      onBlur={onBlur}
                      isPending={isPending}
                      setServerState={setServerState}
                    />
```

```
                );
              }}
            />
          </SectionHeader>

          <SectionHeader
            title={t("profile-header")}
            subtitle={t("profile-header_caption")}
            anchorName="profile"
          >
            <SettingsItem
              name="profile_color_id" // this might need to be the exact
database field
              label={t("profile-color_label")}
              caption={t("profile-color_label_caption")}
              renderInput={({ value, onChange, onBlur, id, isPending }) =
> (
                <ColorDropdown
                  initialValue={initialValues.profile.colorId}
                  id={id}
                  value={value}
                  isPending={isPending}
                  // We need to do nothing here because the this type of
setting is handled internally (in settings-item)
                  onValueChange={(colorIndex: string) => {}}
                  onChange={onChange}
                  onBlur={onBlur}
                />
              )}
            />
            <SettingsItem
              name="display_name"
              label={t("profile-name_label")}
              caption={t("profile-name_label_caption")}
              initialValue={initialValues.profile.displayName}
            />
          </SectionHeader>

          <SectionHeader
            title={t("appearance-header")}
            subtitle={t("appearance-header_caption")}
            anchorName="appearance"
          >
            <SettingsItem
              name="primary_color_id" // this might need to be the exact
database field
              label={t("appearance-color_label")}
              caption={t("appearance-color_label_caption")}
              renderInput={({ value, onChange, onBlur, id, isPending }) =
> (
                <ColorDropdown
                  initialValue={initialValues.appearance.primaryColor}
                  // Initial value handled internally
```

```
                    id={id}
                    value={value}
                    isPending={isPending}
                    onValueChange={(colorIndex: string) =>
                      setThemeColor(Number(colorIndex))
                    }
                    // we call these in onValueChange
                    onChange={onChange}
                    onBlur={onBlur}
                  />
                )}
              />

              <SettingsItem
                name="appearance_layout" // this might need to be the exact
database field
                label={t("appearance-layout_label")}
                caption={t("appearance-layout_label_caption")}
                renderInput={({ value, onChange, onBlur, id, isPending }) =
> {
                  const layouts = [
                    {
                      value: "MODERN",
                      name: "Modern",
                    },
                    {
                      value: "SIMPLE",
                      name: "Simple",
                    },
                  ];
                  return (
                    <Select
                      defaultValue={initialValues.appearance.layout}
                      onValueChange={(value) => {
                        onChange(value);
                        setTimeout(() => {
                          onBlur(undefined, value);
                        }, 200);
                      }}
                      disabled={isPending}
                    >
                      <SelectTrigger
                        id={id}
                        className={cn(
                          buttonVariants({ variant: "outline" }),
                          "w-[200px] appearance-none font-normal justify-be
tween"

                        )}
                      >
                        <SelectValue />
                      </SelectTrigger>
                      <SelectContent>
                        {createSelectItems(layouts, theme)}
```

```
                    </SelectContent>
                  </Select>
                );
              }}
            />
            <SettingsItem
              name="dark_mode"
              label={t("appearance-theme_label")}
              caption={t("appearance-theme_label_caption")}
              renderInput={({ value, onChange, onBlur, id, isPending }) =
> (
                <ThemeToggle
                  id={id}
                  value={value}
                  onChange={onChange}
                  onBlur={onBlur}
                  className="order-2"
                  initialValue={initialValues.appearance.darkMode}
                  isPending={isPending}
                />
              )}
            />
          </SectionHeader>
        </div>
      </div>
    </ScrollArea>
  </>
);
}
```

/app/[locale]/(root)/(other)/new-message/layout.tsx

```
import initTranslations from "@/app/i18n";
import TranslationsProvider from "@/contexts/translations-provider";
import ChildrenPanel from "@/components/shared/children-panel";
import { ContactsProvider } from "@/contexts/use-contacts";
import { fetchContacts } from "@/lib/db/contact";
import { METADATA_APP_NAME } from "@/global.config";

type LayoutProps = Readonly<{
  children: React.ReactNode;
  params: Promise<{ locale: string }>;
}>;

export default async function TranslationLayout({
  children,
  params,
}: LayoutProps) {
  // Internationalization (i18n) stuff
  const i18nNamespaces = ["new-message-page", "modals", "common", "errors"]
```

```
;
  const { locale } = await params;
  const { resources } = await initTranslations(locale, i18nNamespaces);

  return (
    /* This is a client layout component containing the translation provide
r for the nav panel */
    <TranslationsProvider
      /* Only wrap what's necessary with the TranslationsProvider */
      resources={resources}
      locale={locale}
      namespaces={i18nNamespaces}
    >
      <ChildrenPanel>
        <ContactsProvider initialContacts={(await fetchContacts()) || []}>
          {children}
        </ContactsProvider>
      </ChildrenPanel>
    </TranslationsProvider>
  );
}

export async function generateMetadata({
  params,
}: {
  params: Promise<{ locale: string }>;
}) {
  const { locale } = await params;
  const { t } = await initTranslations(locale, ["metadata"]);

  return {
    title: METADATA_APP_NAME + t("new_message-title"),
    description: t("new_message-description"),
  };
}
```

/app/[locale]/(root)/(other)/new-message/error.tsx

```
"use client";

import ErrorComponent from "@/components/shared/error-component";
import { Button } from "@/components/ui/button";
import { useEffect } from "react";
import { useTranslation } from "react-i18next";

export default function Error({
  error,
  reset,
}: {
  error: Error & { digest?: string };
```

```
  reset: () => void;
}) {
  const { t } = useTranslation(["errors"]);

  useEffect(() => {
    // Log the error to an error reporting service
    console.error(error);
  }, [error]);
  return (
    <ErrorComponent
      title={t("error-header")}
      subtitle={t("error-header_caption")}
    >
      <Button
        onClick={
          // Attempt to recover by trying to re-render the segment
          () => reset()
        }
      >
        {t("try_again")}
      </Button>
    </ErrorComponent>
  );
}
```

/app/[locale]/(root)/(other)/new-message/loading.tsx

```
"use client";

import { Separator } from "@/components/ui/separator";
import {
  ChevronDown,
  Maximize2,
  Minimize2,
  Send,
  Trash2,
  X,
} from "lucide-react";
import { useTranslation } from "react-i18next";
import { PageHeader } from "@/components/headers";
import { Button, buttonVariants } from "@/components/ui/button";
import { usePathname, useRouter, useSearchParams } from "next/navigation";
import {
  Select,
  SelectContent,
  SelectItem,
  SelectTrigger,
  SelectValue,
} from "@/components/ui/select";
import { useLayout } from "@/contexts/use-layout";
```

```
import { useIsMobile } from "@/hooks/use-mobile";

import Skeleton from "react-loading-skeleton";
import { cn } from "@/lib/utils";

const PULSE_BODY_WIDTH = "70%";
const PULSE_SUBJECT_WIDTH = "25%";

export default function Loading() {
  const { t } = useTranslation(["new-message-page"]);
  const router = useRouter();
  const { isFullscreen, setIsFullscreen } = useLayout();

  const onMobile = useIsMobile();

  return (
    <div classNAme="">
      <PageHeader title={t("header")} skeleton>
        <p>{t("common:loading")}</p>
        {!onMobile && (
          <Button variant="ghost" size="icon" disabled>
            {isFullscreen ? (
              <Minimize2 className="h-4 w-4" />
            ) : (
              <Maximize2 className="h-4 w-4" />
            )}
          </Button>
        )}

        <Button
          variant="ghost"
          className={cn(buttonVariants({ variant: "ghost" }), "aspect-1 p-0
")}
          disabled
        >
          <X className="h-4 w-4" />
        </Button>
      </PageHeader>
      <div className="h-screen flex flex-col">
        <div className="flex flex-col h-[calc(100vh-var(--header-height))]"
>
          <div className="flex flex-col px-4 mt-2">
            <div className={cn("border-b focus-within:border-black")}>
              <Select name="sender" defaultValue="ETPZP" disabled>
                {/** It defaults to the first SelectItem */}
                <SelectTrigger className="w-full rounded-none border-none s
hadow-none focus:ring-0 px-5 py-1 h-11">
                  <SelectValue placeholder="ETPZP" />
                </SelectTrigger>
                <SelectContent>
                  <SelectItem value="ETPZP">ETPZP</SelectItem>
                  <SelectItem value="Test">Test</SelectItem>
```

```
              </SelectContent>
            </Select>
          </div>

          <InputSkeleton title={t("common:to")} />
          <InputSkeleton />
        </div>
        <div className="px-4 flex-grow mt-[1.25rem] mb-2 w-full">
          <span className="mb-1 flex items-center text-sm text-muted-fore
ground flex-1 min-w-8">
            <Skeleton
              height={16}
              containerClassName={`min-w-[${PULSE_BODY_WIDTH}]`}
            />
          </span>
        </div>

        <Separator />
        <div className="flex px-4 py-2 justify-end gap-2">
          <Button
            variant="secondary"
            type="button"
            className="w-max"
            disabled
          >
            <Trash2 className="h-4 w-4" />
            {t("discard")}
          </Button>

          <div className="flex">
            <Button
              className="rounded-tr-none rounded-br-none border-primary-f
oreground border-r"
              disabled
            >
              <Send className="w-4 h-4" />
              {t("submit_btn-normal")}
            </Button>
            <div
              className={cn("flex gap-3 items-center justify-start w-full
")}
            >
              <Button
                className="px-[1px] rounded-tl-none rounded-bl-none shado
w-none"
                type="button"
                disabled
              >
                <ChevronDown className={cn("h-4 w-4 transition-transform"
)} />
              </Button>
            </div>
          </div>
```

```
          </div>
        </div>
      </div>
    </div>
  );
}

function InputSkeleton({ title }: { title?: string }) {
  return (
    <div className="flex-1 py-1 relative ">
      <div className="max-h-24 overflow-auto">
        <div className="w-full flex flex-wrap items-center gap-x-1 py-1 h-f
ull border-b px-5 z-50 min-h-[45px]">
          {title ? (
            <span className="my-0.5 mr-0.5 px-0 flex items-center text-sm t
ext-muted-foreground">
              {title}
            </span>
          ) : (
            <span className="mb-1 flex items-center text-sm text-muted-fore
ground flex-1 min-w-8">
              <Skeleton
                height={16}
                width=""
                containerClassName={`min-w-[${PULSE_SUBJECT_WIDTH}]`}
              />
            </span>
          )}
        </div>
      </div>
    </div>
  );
}
```

/app/[locale]/(root)/(other)/new-message/page.tsx

```
import NewMessageForm from "@/components/new-message-form";
import { MessageState, NewMessageProvider } from "@/contexts/use-new-messag
e";
import { fetchRecipients } from "@/lib/db/recipients";
import { fetchDraft } from "@/lib/db/message";
import { rankRecipients, validatePhoneNumber } from "@/lib/utils";
import { ModalProvider } from "@/contexts/use-modal";
import { EMPTY_MESSAGE } from "@/global.config";

type NewMessagePageProps = {
  searchParams: Promise<{ message_id: string }>;
};

export default async function Page({ searchParams }: NewMessagePageProps) {
```

```
  const rawRecipients = await fetchRecipients();
  const draftInUrl = await searchParams;
  const fetchedDraft = await fetchDraft(draftInUrl.message_id);

  return (
    <ModalProvider>
      <NewMessageProvider
        rankedRecipients={rankRecipients(rawRecipients || []) || []}
        // initialMessage={fetchedDraft || EMPTY_MESSAGE}
        initialMessage={
          fetchedDraft
            ? {
                body: fetchedDraft?.body || EMPTY_MESSAGE.body,
                subject: fetchedDraft?.subject || EMPTY_MESSAGE.subject,
                sender: fetchedDraft?.sender || EMPTY_MESSAGE.sender,
                recipients:
                  fetchedDraft?.recipients.map((r) => {
                    return {
                      ...r,
                      ...validatePhoneNumber(r.phone),
                    };
                  }) || EMPTY_MESSAGE.recipients,
                recipientInput: EMPTY_MESSAGE.recipientInput,
                scheduledDate:
                  fetchedDraft.send_time || EMPTY_MESSAGE.scheduledDate,
                scheduledDateModified: EMPTY_MESSAGE.scheduledDateModified,
                scheduledDateConfirmed: EMPTY_MESSAGE.scheduledDateConfirme
d,
              }
            : undefined
        }
        draftId={fetchedDraft?.id || null}
      >
        <NewMessageForm message_id={fetchedDraft} />
      </NewMessageProvider>
    </ModalProvider>
  );
}
```

/app/[locale]/dashboard/layout.tsx

```
import TranslationsProvider from "@/contexts/translations-provider";
import initTranslations from "@/app/i18n";
import { SettingsProvider } from "@/contexts/use-settings";
import { getSession } from "@/lib/auth/sessions";
import UnauthorizedPage from "@/components/403";
import { METADATA_APP_NAME } from "@/global.config";

type LayoutProps = {
  children: React.ReactNode;
```

```
  params: Promise<{ locale: string }>;
};

export default async function DashboardLayout({
  children,
  params,
}: LayoutProps) {
  const i18nNamespaces = ["dashboard-page", "errors", "common", "navigation
"];
  const { locale } = await params;
  const { resources } = await initTranslations(locale, i18nNamespaces);

  // Prevent non-admins from viewing the admin-dashboard and display an aut
horization message.
  const session = await getSession();
  if (!session?.isAdmin) return <UnauthorizedPage />;

  return (
    <TranslationsProvider
      resources={resources}
      locale={locale}
      namespaces={i18nNamespaces}
    >
      <SettingsProvider currentLocale={locale}>{children}</SettingsProvider
>
    </TranslationsProvider>
  );
}

export async function generateMetadata({
  params,
}: {
  params: Promise<{ locale: string }>;
}) {
  const { locale } = await params;
  const { t } = await initTranslations(locale, ["metadata"]);

  return {
    title: METADATA_APP_NAME + t("dashboard-title"),
    description: t("dashboard-description"),
  };
}
```

/app/[locale]/dashboard/page.tsx

```
import {
  fetchCountryStats,
  fetchMessagesInDateRange,
  fetchUsers,
} from "@/lib/db/dashboard";
```

```
import { format } from "date-fns";
import AdminDashboard from "@/components/admin-dashboard";
import { DEFAULT_START_DATE, ISO8601_DATE_FORMAT } from "@/global.config";

export type CountryStat = { country: string; amount: number; cost: number }
;

export default async function Dashboard({
  searchParams,
}: {
  searchParams?: Promise<{
    // We expect both of these to be in ISO 8601 format (YYYY-MM-DD)
    start_date?: string;
    end_date?: string;
  }>;
}) {
  const s = await searchParams;
  const dateRange = {
    startDate: s?.start_date || format(DEFAULT_START_DATE, ISO8601_DATE_FOR
MAT),
    endDate: s?.end_date || format(new Date(), ISO8601_DATE_FORMAT),
  };
  const messages = await fetchMessagesInDateRange(dateRange);
  const users = await fetchUsers();
  const countryData = await fetchCountryStats(dateRange);

  return (
    <AdminDashboard
      messages={messages || []}
      users={users || []}
      countryStats={countryData}
    />
  );
}
```

/app/[locale]/login/layout.tsx

```
import TranslationsProvider from "@/contexts/translations-provider";
import initTranslations from "@/app/i18n";
import { SettingsProvider } from "@/contexts/use-settings";
import { METADATA_APP_NAME } from "@/global.config";

type LayoutProps = {
  children: React.ReactNode;
  params: Promise<{ locale: string }>;
};

export default async function LoginLayout({ children, params }: LayoutProps
) {
  const i18nNamespaces = ["login-page", "common"];
```

```
  const { locale } = await params;
  const { resources } = await initTranslations(locale, i18nNamespaces);

  return (
    <TranslationsProvider
      resources={resources}
      locale={locale}
      namespaces={i18nNamespaces}
    >
      <SettingsProvider currentLocale={locale}>{children}</SettingsProvider
>
    </TranslationsProvider>
  );
}

export async function generateMetadata({
  params,
}: {
  params: Promise<{ locale: string }>;
}) {
  const { locale } = await params;
  const { t } = await initTranslations(locale, ["metadata"]);

  return {
    title: METADATA_APP_NAME + t("login-title"),
    description: t("login-description"),
  };
}
```

/app/[locale]/login/page.tsx

```
import LoginForm from "@/components/login-form";

export default async function LoginPage() {
  return <LoginForm />;
}
```

/app/scattered-profiles.module.css

```
/* Base class for absolute centering (if needed) */
.profile-absolute {
  position: absolute;
  /* Reducing width/height and increasing scale makes the text inside the e
lement bigger */
  width: 67%;
  height: 67%;
  /* Uncomment this if you prefer
```

```
    opacity: 0.9; */
}

/* Big profile: centered in container with scale */
.profile-big {
  z-index: 1;
  /* Center using helper translate and scale */
  top: 40%;
  left: 52%;
  transform: translate(-50%, -50%) scale(1.15);
}

.profile-top-left {
  top: 0;
  left: -7px;
  transform: scale(0.7);
  transform-origin: top left;
}

.profile-bottom-left {
  left: 0;
  bottom: 0;
  transform-origin: bottom left;
  transform: scale(0.4);
}

.profile-top-right {
  top: -5px;
  right: 0;
  transform: scale(0.3);
  transform-origin: top right;
}

.profile-bottom-right {
  z-index: 2;

  /* This works, while top:100% and left: 100% places it way outside of the
parent. */
  right: 0;
  bottom: -3px;
  transform-origin: bottom right;
  transform: scale(0.82);
  font-weight: 700;
}
```

/app/layout.tsx

```
import localFont from "next/font/local";
import "./globals.css";
import { dir } from "i18next";
```

```
import { ThemeProvider as NextThemesProvider } from "next-themes";
import ThemeProvider from "@/contexts/theme-data-provider";
import { cookies } from "next/headers";
import { TooltipProvider } from "@/components/ui/tooltip";
import { LayoutProvider } from "@/contexts/use-layout";
import { Toaster } from "sonner";
import { fetchAmountIndicators } from "@/lib/db/general";
import { i18nConfig } from "@/i18n.config";

// We can't export this, because in layout or page files Next.js expects on
ly components and some other stuff to be exported
const disketMonoRegular = localFont({
  src: "./fonts/Disket-Mono-Bold.ttf",
  variable: "--font-disket-mono-regular",
  weight: "100 900",
});

// Let Next.js statically generate pages for each of our languages
export function generateStaticParams() {
  return i18nConfig.locales.map((locale) => ({ locale }));
}

export default async function RootLayout({
  children,
}: {
  children: React.ReactNode;
}) {
  // We can't get the locale from the url params, thus we parse it from the
locale cookie
  const cookieStore = await cookies();
  const currentLocale = cookieStore.get("NEXT_LOCALE")?.value;

  const layoutCookie = cookieStore.get("react-resizable-panels:layout:app")
;
  const collapsedCookie = cookieStore.get("react-resizable-panels:collapsed
");

  const initialLayout: number[] = layoutCookie
    ? JSON.parse(layoutCookie.value)
    : undefined;
  const initialIsCollapsed: boolean = collapsedCookie
    ? JSON.parse(collapsedCookie.value)
    : undefined;

  const amountIndicators = await fetchAmountIndicators();

  return (
    <html
      lang={currentLocale}
      dir={dir(currentLocale)}
      suppressHydrationWarning
    >
```

```
      <body
        className={`${disketMonoRegular.variable} antialiased flex flex-col
h-screen`}
      >
        <NextThemesProvider
          attribute="class"
          defaultTheme="light"
          enableSystem
          disableTransitionOnChange
        >
          <ThemeProvider>
            <TooltipProvider delayDuration={0}>
              <LayoutProvider
                initialLayout={initialLayout}
                initialIsCollapsed={initialIsCollapsed}
                initialAmountIndicators={amountIndicators}
              >
                <Toaster richColors position="top-center" />
                {children}
              </LayoutProvider>
            </TooltipProvider>
          </ThemeProvider>
        </NextThemesProvider>
      </body>
    </html>
  );
}
```

/app/i18n.js

```
import { createInstance } from "i18next";
import { initReactI18next } from "react-i18next/initReactI18next";
import resourcesToBackend from "i18next-resources-to-backend";
import { i18nConfig } from "@/i18n.config";

export default async function initTranslations(
  locale,
  namespaces,
  i18nInstance,
  resources
) {
  i18nInstance = i18nInstance || createInstance();

  i18nInstance.use(initReactI18next);

  if (!resources) {
    i18nInstance.use(
      resourcesToBackend((language, namespace) =>
        import(`@/locales/${language}/${namespace}.json`)
      )
```

```
    );
  }

  await i18nInstance.init({
    lng: locale,
    resources,
    fallbackLng: i18nConfig.defaultLocale,
    supportedLngs: i18nConfig.locales,
    defaultNS: namespaces[0],
    fallbackNS: namespaces[0],
    ns: namespaces,
    preload: resources ? [] : i18nConfig.locales,
  });

  return {
    i18n: i18nInstance,
    resources: i18nInstance.services.resourceStore.data,
    t: i18nInstance.t,
  };
}
```

/app/globals.css

```
@tailwind base;
@tailwind components;
@tailwind utilities;

@layer base {
  :root {
    /* Custom variables here */
    /* --simple-header-height is a constant for the SIMPLE layout, used as
a reference for other layouts. */
    --simple-header-height: 52px;
    /* header-height on the other hand is dynamic */
    --header-height: 52px;
  }

  :root {
    --background: 0 0% 100%;
    --foreground: 20 14.3% 4.1%;
    --card: 0 0% 100%;
    --cardForeground: 20 14.3% 4.1%;
    --popover: 0 0% 100%;
    --popoverForeground: 20 14.3% 4.1%;
    --primary: 47.9 95.8% 53.1%;
    --primaryForeground: 26 83.3% 14.1%;
    --secondary: 60 4.8% 95.9%;
    --secondaryForeground: 24 9.8% 10%;
    --muted: 60 4.8% 95.9%;
    --mutedForeground: 25 5.3% 44.7%;
```

```
    --accent: 60 4.8% 95.9%;
    --accentForeground: 24 9.8% 10%;
    --destructive: 0 84.2% 60.2%;
    --destructiveForeground: 60 9.1% 97.8%;
    --border: 240 5.9% 90%;
    --input: 20 5.9% 90%;
    --ring: 20 14.3% 4.1%;
    --radius: 0.5rem;
    --chart1: 207 90% 57%;
    --chart2: 100.15, 63.11%, 59.61%;
    --chart3: 51 100% 50%;
    --chart4: 36 100% 50%;
    --chart5: 262 52% 47%;
    --sidebar-background: 0 0% 98%;
    --sidebar-foreground: 240 5.3% 26.1%;
    --sidebar-primary: 240 5.9% 10%;
    --sidebar-primary-foreground: 0 0% 98%;
    --sidebar-accent: 240 4.8% 95.9%;
    --sidebar-accent-foreground: 240 5.9% 10%;
    --sidebar-border: 220 13% 91%;
    --sidebar-ring: 217.2 91.2% 59.8%;
  }

  .dark {
    --background: 20 14.3% 4.1%;
    --foreground: 60 9.1% 97.8%;
    --card: 20 14.3% 4.1%;
    --cardForeground: 60 9.1% 97.8%;
    --popover: 20 14.3% 4.1%;
    --popoverForeground: 60 9.1% 97.8%;
    --primary: 47.9 95.8% 53.1%;
    --primaryForeground: 26 83.3% 14.1%;
    --secondary: 12 6.5% 15.1%;
    --secondaryForeground: 60 9.1% 97.8%;
    --muted: 12 6.5% 15.1%;
    --mutedForeground: 24 5.4% 63.9%;
    --accent: 12 6.5% 15.1%;
    --accentForeground: 60 9.1% 97.8%;
    --destructive: 0 62.8% 30.6%;
    --destructiveForeground: 60 9.1% 97.8%;
    --border: 240 3.7% 15.9%;
    --input: 12 6.5% 15.1%;
    --ring: 35.5 91.7% 32.9%;
    --chart1: 207 90% 57%;
    --chart2: 100.15, 63.11%, 59.61%;
    --chart3: 51 100% 50%;
    --chart4: 36 100% 50%;
    --chart5: 262 52% 47%;
    --sidebar-background: 240 5.9% 10%;
    --sidebar-foreground: 240 4.8% 95.9%;
    --sidebar-primary: 224.3 76.3% 48%;
    --sidebar-primary-foreground: 0 0% 100%;
    --sidebar-accent: 240 3.7% 15.9%;
```

```
    --sidebar-accent-foreground: 240 4.8% 95.9%;
    --sidebar-border: 240 3.7% 15.9%;
    --sidebar-ring: 217.2 91.2% 59.8%;
  }
}

@layer base {
  * {
    @apply border-border;
  }
  html {
    @apply scroll-smooth;
  }
  body {
    @apply bg-background text-foreground overscroll-none;
    /* font-feature-settings: "rlig" 1, "calt" 1; */
    font-synthesis-weight: none;
    text-rendering: optimizeLegibility;
  }

  @supports (font: -apple-system-body) and (-webkit-appearance: none) {
    [data-wrapper] {
      @apply min-[1800px]:border-t;
    }
  }

  /* Custom scrollbar styling. Thanks @pranathiperii. */
  ::-webkit-scrollbar {
    width: 5px;
  }
  ::-webkit-scrollbar-track {
    background: transparent;
  }
  ::-webkit-scrollbar-thumb {
    background: hsl(var(--border));
    border-radius: 5px;
  }
  * {
    scrollbar-width: thin;
    scrollbar-color: hsl(var(--border)) transparent;
  }
}

h1 {
  @apply text-4xl font-bold;
}
h2 {
  @apply text-xl font-bold;
}
h3 {
  @apply text-lg font-medium;
}
```

```
p.subtitle {
  @apply text-sm text-muted-foreground;
}

h6 {
  font-size: 1.15em;
  line-height: 1.5;
}
* {
  box-sizing: border-box;
}

body {
  overflow: hidden;
}
.font-disket-mono-regular {
  font-family: var(--font-disket-mono-regular);
}

.gradient-text {
  font-weight: bold; /* Make the text bold */
  /* background: linear-gradient(135deg, var(--border), orange); /* Diagona
l gradient */
  background: linear-gradient(
    135deg,
    hsl(var(--primary)),
    hsl(var(--primaryForeground))
  );
  -webkit-background-clip: text; /* Clip the background to the text */
  -webkit-text-fill-color: transparent; /* Make the text color transparent
*/
  display: inline-block; /* Ensure the gradient applies correctly */
}
.focus-primary-ring {
  @apply focus-visible:ring-1 focus-visible:ring-primary focus-visible:outl
ine-none;
}

.user-select-none {
  user-select: none; /* Prevent text selection */
}
.new-message-input {
  @apply h-11 rounded-none pl-5 shadow-none border-0 border-b-[1px] border-
border focus-visible:border-b-ring disabled:opacity-100 placeholder:text-mu
ted-foreground;
}

.shadcn-input {
  @apply flex h-9 w-full rounded-md bg-transparent px-3 py-1 text-base shad
ow-sm transition-colors file:border-0 file:bg-transparent file:text-sm file
:font-medium file:text-accent-foreground placeholder:text-accent-foreground
focus-visible:outline-none focus-visible:ring-1 focus-visible:ring-ring dis
```

```
abled:cursor-not-allowed disabled:opacity-50 md:text-sm;
}

.centered {
  text-align: center;
  align-content: center;
}
.flex-centered {
  display: flex;
  align-items: center;
  justify-content: center;
}
.closeX:hover * {
  color: var(--background);
}

.error-border-pulse {
  animation: pulse 1000ms infinite;
}
@keyframes pulse {
  0% {
    @apply border-border;
  }
  50% {
    @apply border-destructive;
  }
  100% {
    @apply border-border;
  }
}

/* Helper class to center an element absolutely using transform */
.center-absolute {
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
}

.ellipsis {
  white-space: nowrap; /* Prevents text from wrapping */
  overflow: hidden; /* Hides overflowed text */
  text-overflow: ellipsis; /* Adds ellipsis (...) */
}

.container-overlay {
  position: absolute;
  width: 100%;
  height: 100%;
}

.frozen {
```

```
  pointer-events: none; /* Prevents all mouse events */
  user-select: none; /* Prevents text selection */
  opacity: 0.5; /* Optional: make it look "frozen" */
}
```

/app/not-found.tsx

```
import { buttonVariants } from "@/components/ui/button";
import Link from "next/link";
import { cookies } from "next/headers";
import initTranslations from "./i18n";
import { i18nConfig } from "@/i18n.config";
import ErrorComponent from "@/components/shared/error-component";

export default async function NotFound() {
  // we have to get it directly from the cookie here, because we are not in
the [locale] route segment
  const cookieStore = await cookies();
  const currentLocale = cookieStore.get("NEXT_LOCALE")?.value;

  const { t } = await initTranslations(
    currentLocale || i18nConfig.defaultLocale,
    ["errors", "common"]
  );

  return (
    <ErrorComponent
      title={t("404_error-header")}
      subtitle={t("404_error-header_caption")}
    >
      <Link href="/" className={buttonVariants({ variant: "default" })}>
        {t("common:go_back")}
      </Link>
    </ErrorComponent>
  );
}
```

/postcss.config.mjs

```
/** @type {import('postcss-load-config').Config} */
const config = {
  plugins: {
    tailwindcss: {},
  },
};

export default config;
```

/Dockerfile

```
FROM oven/bun:alpine AS base

# Install Node.js, npm, and i18nexus for translations
RUN apk add --no-cache nodejs npm
RUN bun i -g i18nexus-cli

# Stage 1: Install dependencies
FROM base AS deps
# set a path to for the following commands to be run on
WORKDIR /app
COPY package.json bun.lock ./
RUN bun install

# Stage 2: Build the application
FROM base AS builder
WORKDIR /app
COPY --from=deps /app/node_modules ./node_modules
COPY . .
RUN bun run build

# Stage 3: Production server
FROM base AS runner
WORKDIR /app
ENV NODE_ENV=production
COPY --from=builder /app/public ./public

COPY --from=builder /app/.next/standalone ./
COPY --from=builder /app/.next/static ./.next/static
# If it at some point doesn't work anymore, copy the entire directory
# COPY --from=builder /app/.next ./.next

# This is for the `start` script, but when replacing the start command with
server.js (also part of the build) I can't access the site on localhost
COPY --from=builder /app/package.json ./
COPY --from=builder /app/node_modules ./node_modules

EXPOSE 3000
CMD ["bun", "run", "start"]
```

/i18n.config.ts

```
export const i18nConfig = {
  locales: ["pt", "de", "en"],
  defaultLocale: "pt",
```

```
  // Set to `true` if you want the default locale to be included in the url
  prefixDefault: false,
};
```

/next-env.d.ts

```
/// <reference types="next" />
/// <reference types="next/image-types/global" />

// NOTE: This file should not be edited
// see https://nextjs.org/docs/app/api-reference/config/typescript for more information.
```

/.prettierignore

```
README.md
.env**
```

/README.md

This is a **Next.js 15** app router project

Getting Started

First, run the development server:

```
npm run dev
# or
yarn dev
# or
pnpm dev
# or
bun dev
```

Open **http://localhost:3000** with your browser to see the result.

/tailwind.config.ts

```ts
import type { Config } from "tailwindcss";
import tailwindcssAnimate from "tailwindcss-animate";

export default {
  darkMode: ["class"],
  content: [
    "./pages/**/*.{js,ts,jsx,tsx,mdx}",
    "./components/**/*.{js,ts,jsx,tsx,mdx}",
    "./app/**/*.{js,ts,jsx,tsx,mdx}",
  ],
  theme: {
    extend: {
      colors: {
        background: "hsl(var(--background))",
        foreground: "hsl(var(--foreground))",
        card: {
          DEFAULT: "hsl(var(--card))",
          foreground: "hsl(var(--cardForeground))",
        },
        popover: {
          DEFAULT: "hsl(var(--popover))",
          foreground: "hsl(var(--popoverForeground))",
        },
        primary: {
          DEFAULT: "hsl(var(--primary))",
          foreground: "hsl(var(--primaryForeground))",
        },
        secondary: {
          DEFAULT: "hsl(var(--secondary))",
          foreground: "hsl(var(--secondaryForeground))",
        },
        muted: {
          DEFAULT: "hsl(var(--muted))",
          foreground: "hsl(var(--mutedForeground))",
```

```
    },
    accent: {
      DEFAULT: "hsl(var(--accent))",
      foreground: "hsl(var(--accentForeground))",
    },
    destructive: {
      DEFAULT: "hsl(var(--destructive))",
      foreground: "hsl(var(--destructiveForeground))",
    },
    border: "hsl(var(--border))",
    input: "hsl(var(--input))",
    ring: "hsl(var(--ring))",
    chart: {
      "1": "hsl(var(--chart1))",
      "2": "hsl(var(--chart2))",
      "3": "hsl(var(--chart3))",
      "4": "hsl(var(--chart4))",
      "5": "hsl(var(--chart5))",
    },
    sidebar: {
      DEFAULT: "hsl(var(--sidebar-background))",
      foreground: "hsl(var(--sidebar-foreground))",
      primary: "hsl(var(--sidebar-primary))",
      "primary-foreground": "hsl(var(--sidebar-primary-foreground))",
      accent: "hsl(var(--sidebar-accent))",
      "accent-foreground": "hsl(var(--sidebar-accent-foreground))",
      border: "hsl(var(--sidebar-border))",
      ring: "hsl(var(--sidebar-ring))",
    },
  },
  borderRadius: {
    lg: "var(--radius)",
    md: "calc(var(--radius) - 2px)",
    sm: "calc(var(--radius) - 4px)",
  },
  keyframes: {
    "accordion-down": {
      from: {
        height: "0",
      },
      to: {
        height: "var(--radix-accordion-content-height)",
      },
    },
    "accordion-up": {
      from: {
        height: "var(--radix-accordion-content-height)",
      },
      to: {
        height: "0",
      },
    },
  },
```

```
      animation: {
        "accordion-down": "accordion-down 0.2s ease-out",
        "accordion-up": "accordion-up 0.2s ease-out",
      },
    },
    screens: {
      sm: "640px",
      md: "768px",
      lg: "1024px",
      xl: "1280px",
      "2xl": "1536px",
      // Custom breakpoints
      xs: "435px",
    },
    aspectRatio: {
      "1": "1 / 1",
    },
  },
  plugins: [tailwindcssAnimate],
} satisfies Config;
```

/components/settings.tsx

```
"use client";

import { Button, buttonVariants } from "@/components/ui/button";
import {
  Select,
  SelectContent,
  SelectItem,
  SelectTrigger,
  SelectValue,
} from "@/components/ui/select";
import { useThemeContext } from "@/contexts/theme-data-provider";
import { cn } from "@/lib/utils";
import { useTheme as useNextTheme } from "next-themes";
import { useTranslation } from "react-i18next";
import { RenderInputArgs } from "@/components/settings-item";
import { useEffect, useState } from "react";
import { updateSetting } from "@/lib/actions/user.actions";
import { useSettings } from "@/contexts/use-settings";

export function LanguageChanger({
  // value,
  onChange,
  id,
  setServerState,
}: RenderInputArgs) {
  const { t, i18n } = useTranslation();
  const currentLocale = i18n.language;
```

```
  const { updateLanguageCookie } = useSettings();
  const [isPending, setIsPending] = useState<boolean>(false);

  const handleChange = async (newLocale: string) => {
    // Update the database first
    setIsPending(true);
    const formData = new FormData();
    formData.append("name", "lang");
    formData.append("value", newLocale);

    const result = await updateSetting(formData);
    if (setServerState) setServerState(result);
    setIsPending(false);

    updateLanguageCookie(newLocale);
  };
  return (
    <Select
      defaultValue={currentLocale}
      // When turning into a controlled input by passing in a value, the ap
p breaks - I'm not sure why.
      // value={value}
      onValueChange={handleChange}
      disabled={isPending}
    >
      <SelectTrigger
        id={id}
        className={cn(
          buttonVariants({ variant: "outline" }),
          "w-[200px] appearance-none font-normal justify-between"
        )}
      >
        <SelectValue placeholder="Select Language" />
      </SelectTrigger>
      <SelectContent>
        <SelectItem value="en">English</SelectItem>
        <SelectItem value="pt">Português</SelectItem>
        <SelectItem value="de">Deutsch</SelectItem>
      </SelectContent>
    </Select>
  );
}

const COLORS = [
  {
    value: "1",
    name: "Zinc",
    light: "bg-zinc-900",
    dark: "bg-zinc-700",
  },
  {
    value: "2",
```

```
    name: "Rose",
    light: "bg-rose-600",
    dark: "bg-rose-700",
  },
  {
    value: "3",
    name: "Blue",
    light: "bg-blue-600",
    dark: "bg-blue-700",
  },
  {
    value: "4",
    name: "Green",
    light: "bg-green-600",
    dark: "bg-green-500",
  },
  {
    value: "5",
    name: "Orange",
    light: "bg-orange-500",
    dark: "bg-orange-700",
  },
  {
    value: "6",
    name: "Yellow",
    light: "bg-yellow-300",
    dark: "bg-yellow-500",
  },
];
export function ThemeColorChanger({
  onChange,
  onBlur,
  id,
  isPending,
}: RenderInputArgs) {
  const { themeColor, setThemeColor } = useThemeContext();
  const { theme } = useNextTheme();

  const handleChange = (colorIndex: string) => {
    setThemeColor(Number(colorIndex));
    onChange(colorIndex);

    // Remove this if you are sure that it works this way
    // setTimeout(() => {
    onBlur(undefined, colorIndex);
    // }, 200);
  };

  return (
    <Select
      defaultValue={themeColor.toString()}
      onValueChange={handleChange}
      disabled={isPending}
```

```
    >
      <SelectTrigger
        id={id}
        className={cn(
          buttonVariants({ variant: "outline" }),
          "w-[200px] appearance-none font-normal justify-between"
        )}
      >
        <SelectValue />
      </SelectTrigger>
      <SelectContent>{createSelectItems(COLORS, theme)}</SelectContent>
    </Select>
  );
}
export function ColorDropdown({
  onValueChange,
  id,
  isPending,
  onChange,
  onBlur,
  initialValue,
}: RenderInputArgs & { onValueChange: (value: string) => void }) {
  const { theme } = useNextTheme();

  return (
    <Select
      defaultValue={initialValue}
      onValueChange={(colorIndex) => {
        onValueChange(colorIndex);
        onChange(colorIndex);
        onBlur(undefined, colorIndex);
      }}
      disabled={isPending}
    >
      <SelectTrigger
        id={id}
        className={cn(
          buttonVariants({ variant: "outline" }),
          "w-[200px] appearance-none font-normal justify-between"
        )}
      >
        <SelectValue />
      </SelectTrigger>
      <SelectContent>{createSelectItems(COLORS, theme)}</SelectContent>
    </Select>
  );
}

export function ThemeToggle({
  onChange,
  onBlur,
  id,
  initialValue,
```

```
  className,
  isPending,
}: RenderInputArgs) {
  const { theme, setTheme } = useNextTheme();
  const { t } = useTranslation();
  const activeString = `(${t("common:active").toLowerCase()})`;

  const handleChange = (value: string) => {
    setTheme(value);
    onChange(value);
    setTimeout(() => {
      onBlur(undefined, value);
    }, 200);
  };
  return (
    <div
      className={cn(
        className,
        "flex flex-col gap-1 sm:flex-row sm:gap-8 max-w-md pt-2"
      )}
    >
      <div
        onClick={isPending ? () => {} : () => handleChange("light")}
        className={cn(isPending && "opacity-50 cursor-not-allowed")}
      >
        <div className="items-center rounded-md border-2 border-muted p-1 h
over:border-accent">
          <div className="space-y-2 rounded-sm bg-[#ecedef] p-2">
            <div className="space-y-2 rounded-md bg-white p-2 shadow-sm">
              <div className="h-2 w-[80px] rounded-lg bg-[#ecedef]" />
              <div className="h-2 w-[100px] rounded-lg bg-[#ecedef]" />
            </div>
            <div className="flex items-center space-x-2 rounded-md bg-white
p-2 shadow-sm">
              <div className="h-4 w-4 rounded-full bg-[#ecedef]" />
              <div className="h-2 w-[100px] rounded-lg bg-[#ecedef]" />
            </div>
            <div className="flex items-center space-x-2 rounded-md bg-white
p-2 shadow-sm">
              <div className="h-4 w-4 rounded-full bg-[#ecedef]" />
              <div className="h-2 w-[100px] rounded-lg bg-[#ecedef]" />
            </div>
          </div>
        </div>
        <label className="block w-full p-2 text-center font-normal text-sm"
>
          {t("appearance-theme_light")}{" "}
          {!isPending && theme === "light" && activeString}
        </label>
      </div>

      <div
        onClick={isPending ? () => {} : () => handleChange("dark")}
```

```
        className={cn(isPending && "opacity-50 cursor-not-allowed")}
      >
        <div
          className={cn(
            "items-center rounded-md border-2 border-muted bg-popover p-1",
            !isPending && "hover:bg-accent hover:text-accent-foreground"
          )}
        >
          <div className="space-y-2 rounded-sm bg-slate-950 p-2">
            <div className="space-y-2 rounded-md bg-slate-800 p-2 shadow-sm
">
              <div className="h-2 w-[80px] rounded-lg bg-slate-400" />
              <div className="h-2 w-[100px] rounded-lg bg-slate-400" />
            </div>
            <div className="flex items-center space-x-2 rounded-md bg-slate
-800 p-2 shadow-sm">
              <div className="h-4 w-4 rounded-full bg-slate-400" />
              <div className="h-2 w-[100px] rounded-lg bg-slate-400" />
            </div>
            <div className="flex items-center space-x-2 rounded-md bg-slate
-800 p-2 shadow-sm">
              <div className="h-4 w-4 rounded-full bg-slate-400" />
              <div className="h-2 w-[100px] rounded-lg bg-slate-400" />
            </div>
          </div>
          <label className="block w-full p-2 text-center font-normal text-sm"
>
            {t("appearance-theme_dark")}{" "}
            {!isPending && theme === "dark" && activeString}
          </label>
        </div>
      </div>
    );
}

export const createSelectItems = (data: any[], theme: string | undefined) =
> {
  return data.map(({ name, light, dark, value }) => (
    <SelectItem key={value} value={value || name}>
      <div className="flex gap-2">
        {light && dark && (
          <div
            className={cn(
              "w-[20px]",
              "h-[20px]",
              "rounded-full",
              theme === "light" ? light : dark
            )}
          />
        )}
        <div className="text-sm">{name}</div>
      </div>
```

```
    </SelectItem>
  ));
};
```

/components/ui/alert-dialog.tsx

```
"use client";

import * as React from "react";
import * as AlertDialogPrimitive from "@radix-ui/react-alert-dialog";

import { cn } from "@/lib/utils";
import { buttonVariants } from "@/components/ui/button";

const AlertDialog = AlertDialogPrimitive.Root;

const AlertDialogTrigger = AlertDialogPrimitive.Trigger;

const AlertDialogPortal = AlertDialogPrimitive.Portal;

const AlertDialogOverlay = React.forwardRef<
  React.ElementRef<typeof AlertDialogPrimitive.Overlay>,
  React.ComponentPropsWithoutRef<typeof AlertDialogPrimitive.Overlay>
>(({ className, ...props }, ref) => (
  <AlertDialogPrimitive.Overlay
    className={cn(
      "fixed inset-0 z-50 bg-black/80 data-[state=open]:animate-in data-[st
ate=closed]:animate-out data-[state=closed]:fade-out-0 data-[state=open]:fa
de-in-0",
      className
    )}
    {...props}
    ref={ref}
  />
));
AlertDialogOverlay.displayName = AlertDialogPrimitive.Overlay.displayName;

const AlertDialogContent = React.forwardRef<
  React.ElementRef<typeof AlertDialogPrimitive.Content>,
  React.ComponentPropsWithoutRef<typeof AlertDialogPrimitive.Content>
>(({ className, ...props }, ref) => (
  <AlertDialogPortal>
    <AlertDialogOverlay />
    <AlertDialogPrimitive.Content
      ref={ref}
      className={cn(
        "fixed left-[50%] top-[50%] z-50 grid w-full bg-background max-w-lg
translate-x-[-50%] translate-y-[-50%] gap-4 border bg-background p-6 shadow
-lg duration-200 data-[state=open]:animate-in data-[state=closed]:animate-o
ut data-[state=closed]:fade-out-0 data-[state=open]:fade-in-0 data-[state=c
```

```
losed]:zoom-out-95 data-[state=open]:zoom-in-95 data-[state=closed]:slide-o
ut-to-left-1/2 data-[state=closed]:slide-out-to-top-[48%] data-[state=open]
:slide-in-from-left-1/2 data-[state=open]:slide-in-from-top-[48%] sm:rounde
d-lg",
        className
      )}
      {...props}
    />
  </AlertDialogPortal>
));
AlertDialogContent.displayName = AlertDialogPrimitive.Content.displayName;

const AlertDialogHeader = ({
  className,
  ...props
}: React.HTMLAttributes<HTMLDivElement>) => (
  <div
    className={cn(
      "flex flex-col space-y-2 text-center sm:text-left",
      className
    )}
    {...props}
  />
);
AlertDialogHeader.displayName = "AlertDialogHeader";

const AlertDialogFooter = ({
  className,
  ...props
}: React.HTMLAttributes<HTMLDivElement>) => (
  <div
    className={cn(
      "flex flex-col-reverse sm:flex-row sm:justify-end sm:space-x-2",
      className
    )}
    {...props}
  />
);
AlertDialogFooter.displayName = "AlertDialogFooter";

const AlertDialogTitle = React.forwardRef<
  React.ElementRef<typeof AlertDialogPrimitive.Title>,
  React.ComponentPropsWithoutRef<typeof AlertDialogPrimitive.Title>
>(({ className, ...props }, ref) => (
  <AlertDialogPrimitive.Title
    ref={ref}
    className={cn("text-lg font-semibold", className)}
    {...props}
  />
));
AlertDialogTitle.displayName = AlertDialogPrimitive.Title.displayName;
```

```
const AlertDialogDescription = React.forwardRef<
  React.ElementRef<typeof AlertDialogPrimitive.Description>,
  React.ComponentPropsWithoutRef<typeof AlertDialogPrimitive.Description>
>(({ className, ...props }, ref) => (
  <AlertDialogPrimitive.Description
    ref={ref}
    className={cn("text-sm text-muted-foreground ", className)}
    {...props}
  />
));
AlertDialogDescription.displayName =
  AlertDialogPrimitive.Description.displayName;

const AlertDialogAction = React.forwardRef<
  React.ElementRef<typeof AlertDialogPrimitive.Action>,
  React.ComponentPropsWithoutRef<typeof AlertDialogPrimitive.Action>
>(({ className, ...props }, ref) => (
  <AlertDialogPrimitive.Action
    ref={ref}
    className={cn(buttonVariants(), className)}
    {...props}
  />
));
AlertDialogAction.displayName = AlertDialogPrimitive.Action.displayName;

const AlertDialogCancel = React.forwardRef<
  React.ElementRef<typeof AlertDialogPrimitive.Cancel>,
  React.ComponentPropsWithoutRef<typeof AlertDialogPrimitive.Cancel>
>(({ className, ...props }, ref) => (
  <AlertDialogPrimitive.Cancel
    ref={ref}
    className={cn(
      buttonVariants({ variant: "outline" }),
      "mt-2 sm:mt-0",
      className
    )}
    {...props}
  />
));
AlertDialogCancel.displayName = AlertDialogPrimitive.Cancel.displayName;

export {
  AlertDialog,
  AlertDialogPortal,
  AlertDialogOverlay,
  AlertDialogTrigger,
  AlertDialogContent,
  AlertDialogHeader,
  AlertDialogFooter,
  AlertDialogTitle,
  AlertDialogDescription,
  AlertDialogAction,
```

```
  AlertDialogCancel,
};
```

/components/ui/tabs.tsx

```
"use client";

import * as React from "react";
import * as TabsPrimitive from "@radix-ui/react-tabs";

import { cn } from "@/lib/utils";

const Tabs = TabsPrimitive.Root;

const TabsList = React.forwardRef<
  React.ElementRef<typeof TabsPrimitive.List>,
  React.ComponentPropsWithoutRef<typeof TabsPrimitive.List>
>(({ className, ...props }, ref) => (
  <TabsPrimitive.List
    ref={ref}
    className={cn(
      "inline-flex h-9 items-center justify-center rounded-lg bg-muted p-1
text-muted-foreground",
      className
    )}
    {...props}
  />
));
TabsList.displayName = TabsPrimitive.List.displayName;

const TabsTrigger = React.forwardRef<
  React.ElementRef<typeof TabsPrimitive.Trigger>,
  React.ComponentPropsWithoutRef<typeof TabsPrimitive.Trigger>
>(({ className, ...props }, ref) => (
  <TabsPrimitive.Trigger
    ref={ref}
    className={cn(
      "inline-flex items-center justify-center whitespace-nowrap rounded-md
px-3 py-1 text-sm font-medium ring-offset-ring transition-all focus-visible
:outline-none focus-visible:ring-2 focus-visible:ring-ring focus-visible:ri
ng-offset-2 disabled:pointer-events-none disabled:opacity-50 data-[state=ac
tive]:bg-background data-[state=active]:text-foreground data-[state=active]
:shadow",
      className
    )}
    {...props}
  />
));
TabsTrigger.displayName = TabsPrimitive.Trigger.displayName;
```

```
const TabsContent = React.forwardRef<
  React.ElementRef<typeof TabsPrimitive.Content>,
  React.ComponentPropsWithoutRef<typeof TabsPrimitive.Content>
>(({ className, ...props }, ref) => (
  <TabsPrimitive.Content
    ref={ref}
    className={cn(
      "mt-2 ring-offset-white focus-visible:outline-none focus-visible:ring
-2 focus-visible:ring-slate-950 focus-visible:ring-offset-2 dark:ring-offse
t-slate-950 dark:focus-visible:ring-primary",
      className
    )}
    {...props}
  />
));
TabsContent.displayName = TabsPrimitive.Content.displayName;

export { Tabs, TabsList, TabsTrigger, TabsContent };
```

/components/ui/card.tsx

```
import * as React from "react";

import { cn } from "@/lib/utils";

const Card = React.forwardRef<
  HTMLDivElement,
  React.HTMLAttributes<HTMLDivElement>
>(({ className, ...props }, ref) => (
  <div
    ref={ref}
    className={cn(
      "rounded-xl border bg-background text-foreground shadow",
      className
    )}
    {...props}
  />
));
Card.displayName = "Card";

const CardHeader = React.forwardRef<
  HTMLDivElement,
  React.HTMLAttributes<HTMLDivElement>
>(({ className, ...props }, ref) => (
  <div
    ref={ref}
    className={cn("flex flex-col space-y-1.5 p-6", className)}
    {...props}
  />
));
```

```
CardHeader.displayName = "CardHeader";

const CardTitle = React.forwardRef<
  HTMLDivElement,
  React.HTMLAttributes<HTMLDivElement>
>(({ className, ...props }, ref) => (
  <div
    ref={ref}
    className={cn("font-semibold leading-none tracking-tight", className)}
    {...props}
  />
));
CardTitle.displayName = "CardTitle";

const CardDescription = React.forwardRef<
  HTMLDivElement,
  React.HTMLAttributes<HTMLDivElement>
>(({ className, ...props }, ref) => (
  <div
    ref={ref}
    className={cn("text-sm text-muted-foreground ", className)}
    {...props}
  />
));
CardDescription.displayName = "CardDescription";

const CardContent = React.forwardRef<
  HTMLDivElement,
  React.HTMLAttributes<HTMLDivElement>
>(({ className, ...props }, ref) => (
  <div ref={ref} className={cn("p-6 pt-0", className)} {...props} />
));
CardContent.displayName = "CardContent";

const CardFooter = React.forwardRef<
  HTMLDivElement,
  React.HTMLAttributes<HTMLDivElement>
>(({ className, ...props }, ref) => (
  <div
    ref={ref}
    className={cn("flex items-center p-6 pt-0", className)}
    {...props}
  />
));
CardFooter.displayName = "CardFooter";

export {
  Card,
  CardHeader,
  CardFooter,
  CardTitle,
  CardDescription,
```

```
  CardContent,
};
```

/components/ui/popover.tsx

```
"use client";

import * as React from "react";
import * as PopoverPrimitive from "@radix-ui/react-popover";

import { cn } from "@/lib/utils";

const Popover = PopoverPrimitive.Root;

const PopoverTrigger = PopoverPrimitive.Trigger;

const PopoverAnchor = PopoverPrimitive.Anchor;

const PopoverContent = React.forwardRef<
  React.ElementRef<typeof PopoverPrimitive.Content>,
  React.ComponentPropsWithoutRef<typeof PopoverPrimitive.Content>
>(({ className, align = "center", sideOffset = 4, ...props }, ref) => (
  <PopoverPrimitive.Portal>
    <PopoverPrimitive.Content
      ref={ref}
      align={align}
      sideOffset={sideOffset}
      className={cn(
        "z-50 w-72 rounded-md border bg-background p-4 text-slate-950 shado
w-md outline-none data-[state=open]:animate-in data-[state=closed]:animate-
out data-[state=closed]:fade-out-0 data-[state=open]:fade-in-0 data-[state=
closed]:zoom-out-95 data-[state=open]:zoom-in-95 data-[side=bottom]:slide-i
n-from-top-2 data-[side=left]:slide-in-from-right-2 data-[side=right]:slide
-in-from-left-2 data-[side=top]:slide-in-from-bottom-2 dark:border-slate-80
0 bg-background dark:text-slate-50",
        className
      )}
      {...props}
    />
  </PopoverPrimitive.Portal>
));
PopoverContent.displayName = PopoverPrimitive.Content.displayName;

export { Popover, PopoverTrigger, PopoverContent, PopoverAnchor };
```

/components/ui/chart.tsx

```tsx
"use client";

import * as React from "react";
import * as RechartsPrimitive from "recharts";

import { cn } from "@/lib/utils";

// Format: { THEME_NAME: CSS_SELECTOR }
const THEMES = { light: "", dark: ".dark" } as const;

export type ChartConfig = {
  [k in string]: {
    label?: React.ReactNode;
    icon?: React.ComponentType;
  } & (
    | { color?: string; theme?: never }
    | { color?: never; theme: Record<keyof typeof THEMES, string> }
  );
};

type ChartContextProps = {
  config: ChartConfig;
};

const ChartContext = React.createContext<ChartContextProps | null>(null);

function useChart() {
  const context = React.useContext(ChartContext);

  if (!context) {
    throw new Error("useChart must be used within a <ChartContainer />");
  }

  return context;
}

const ChartContainer = React.forwardRef<
  HTMLDivElement,
  React.ComponentProps<"div"> & {
    config: ChartConfig;
    children: React.ComponentProps<
      typeof RechartsPrimitive.ResponsiveContainer
    >["children"];
  }
>(({ id, className, children, config, ...props }, ref) => {
  const uniqueId = React.useId();
  const chartId = `chart-${id || uniqueId.replace(/:/g, "")}`;

  return (
    <ChartContext.Provider value={{ config }}>
```

```
      <div
        data-chart={chartId}
        ref={ref}
        className={cn(
          "flex aspect-video justify-center text-xs [&_.recharts-cartesian-
axis-tick_text]:fill-muted-foreground [&_.recharts-cartesian-grid_line[stro
ke='#ccc']]:stroke-border/50 [&_.recharts-curve.recharts-tooltip-cursor]:st
roke-border [&_.recharts-dot[stroke='#fff']]:stroke-transparent [&_.rechart
s-layer]:outline-none [&_.recharts-polar-grid_[stroke='#ccc']]:stroke-borde
r [&_.recharts-radial-bar-background-sector]:fill-muted [&_.recharts-rectan
gle.recharts-tooltip-cursor]:fill-muted [&_.recharts-reference-line_[stroke
='#ccc']]:stroke-border [&_.recharts-sector[stroke='#fff']]:stroke-transpar
ent [&_.recharts-sector]:outline-none [&_.recharts-surface]:outline-none",
          className
        )}
        {...props}
      >
        <ChartStyle id={chartId} config={config} />
        <RechartsPrimitive.ResponsiveContainer>
          {children}
        </RechartsPrimitive.ResponsiveContainer>
      </div>
    </ChartContext.Provider>
  );
});
ChartContainer.displayName = "Chart";

const ChartStyle = ({ id, config }: { id: string; config: ChartConfig }) =>
{
  const colorConfig = Object.entries(config).filter(
    ([, config]) => config.theme || config.color
  );

  if (!colorConfig.length) {
    return null;
  }

  return (
    <style
      dangerouslySetInnerHTML={{
        __html: Object.entries(THEMES)
          .map(
            ([theme, prefix]) => `
${prefix} [data-chart=${id}] {
${colorConfig
  .map(([key, itemConfig]) => {
    const color =
      itemConfig.theme?.[theme as keyof typeof itemConfig.theme] ||
      itemConfig.color;
    return color ? `  --color-${key}: ${color};` : null;
  })
  .join("\n")}
}
```

```
`
          )
          .join("\n"),
      }}
    />
  );
};

const ChartTooltip = RechartsPrimitive.Tooltip;

const ChartTooltipContent = React.forwardRef<
  HTMLDivElement,
  React.ComponentProps<typeof RechartsPrimitive.Tooltip> &
    React.ComponentProps<"div"> & {
      hideLabel?: boolean;
      hideIndicator?: boolean;
      indicator?: "line" | "dot" | "dashed";
      nameKey?: string;
      labelKey?: string;
    }
>(
  (
    {
      active,
      payload,
      className,
      indicator = "dot",
      hideLabel = false,
      hideIndicator = false,
      label,
      labelFormatter,
      labelClassName,
      formatter,
      color,
      nameKey,
      labelKey,
    },
    ref
  ) => {
    const { config } = useChart();

    const tooltipLabel = React.useMemo(() => {
      if (hideLabel || !payload?.length) {
        return null;
      }

      const [item] = payload;
      const key = `${labelKey || item?.dataKey || item?.name || "value"}`;
      const itemConfig = getPayloadConfigFromPayload(config, item, key);
      const value =
        !labelKey && typeof label === "string"
          ? config[label as keyof typeof config]?.label || label
```

```
          : itemConfig?.label;

      if (labelFormatter) {
        return (
          <div className={cn("font-medium", labelClassName)}>
            {labelFormatter(value, payload)}
          </div>
        );
      }

      if (!value) {
        return null;
      }

      return <div className={cn("font-medium", labelClassName)}>{value}</di
v>;
    }, [
      label,
      labelFormatter,
      payload,
      hideLabel,
      labelClassName,
      config,
      labelKey,
    ]);

    if (!active || !payload?.length) {
      return null;
    }

    const nestLabel = payload.length === 1 && indicator !== "dot";

    return (
      <div
        ref={ref}
        className={cn(
          "grid min-w-[8rem] items-start gap-1.5 rounded-lg border border-s
late-200/50 bg-background px-2.5 py-1.5 text-xs shadow-xl dark:border-slate
-800 dark:border-slate-800/50 bg-background",
          className
        )}
      >
        {!nestLabel ? tooltipLabel : null}
        <div className="grid gap-1.5">
          {payload.map((item, index) => {
            const key = `${nameKey || item.name || item.dataKey || "value"}
`;
            const itemConfig = getPayloadConfigFromPayload(config, item, ke
y);
            const indicatorColor = color || item.payload.fill || item.color
;
```

```
            return (
              <div
                key={item.dataKey}
                className={cn(
                  "flex w-full flex-wrap items-stretch gap-2 [&>svg]:h-2.5
[&>svg]:w-2.5 [&>svg]:text-muted-foreground dark:[&>svg]:text-muted-foregro
und",
                  indicator === "dot" && "items-center"
                )}
              >
                {formatter && item?.value !== undefined && item.name ? (
                  formatter(item.value, item.name, item, index, item.payloa
d)
                ) : (
                  <>
                    {itemConfig?.icon ? (
                      <itemConfig.icon />
                    ) : (
                      !hideIndicator && (
                        <div
                          className={cn(
                            "shrink-0 rounded-[2px] border-[--color-border]
bg-[--color-bg]",

                            {
                              "h-2.5 w-2.5": indicator === "dot",
                              "w-1": indicator === "line",
                              "w-0 border-[1.5px] border-dashed bg-transpar
ent":

                                indicator === "dashed",
                              "my-0.5": nestLabel && indicator === "dashed"
,

                            }
                          )}
                          style={
                            {
                              "--color-bg": indicatorColor,
                              "--color-border": indicatorColor,
                            } as React.CSSProperties
                          }
                        />
                      )
                    )}
                    <div
                      className={cn(
                        "flex flex-1 justify-between leading-none",
                        nestLabel ? "items-end" : "items-center"
                      )}
                    >
                      <div className="grid gap-1.5">
                        {nestLabel ? tooltipLabel : null}
                        <span className="text-muted-foreground ">
                          {itemConfig?.label || item.name}
                        </span>
```

```
                    </div>
                    {item.value && (
                      <span className="font-mono font-medium tabular-nums
text-slate-950 dark:text-slate-50">
                        {item.value.toLocaleString()}
                      </span>
                    )}
                  </div>
                </>
              )}
            </div>
          );
        })}
      </div>
    </div>
  );
}
);
ChartTooltipContent.displayName = "ChartTooltip";

const ChartLegend = RechartsPrimitive.Legend;

const ChartLegendContent = React.forwardRef<
  HTMLDivElement,
  React.ComponentProps<"div"> &
    Pick<RechartsPrimitive.LegendProps, "payload" | "verticalAlign"> & {
      hideIcon?: boolean;
      nameKey?: string;
    }
>(
  (
    { className, hideIcon = false, payload, verticalAlign = "bottom", nameK
ey },
    ref
  ) => {
    const { config } = useChart();

    if (!payload?.length) {
      return null;
    }

    return (
      <div
        ref={ref}
        className={cn(
          "flex items-center justify-center gap-4",
          verticalAlign === "top" ? "pb-3" : "pt-3",
          className
        )}
      >
        {payload.map((item) => {
          const key = `${nameKey || item.dataKey || "value"}`;
```

```
          const itemConfig = getPayloadConfigFromPayload(config, item, key)
;

          return (
            <div
              key={item.value}
              className={cn(
                "flex items-center gap-1.5 [&>svg]:h-3 [&>svg]:w-3 [&>svg]:
text-muted-foreground dark:[&>svg]:text-muted-foreground"
              )}
            >
              {itemConfig?.icon && !hideIcon ? (
                <itemConfig.icon />
              ) : (
                <div
                  className="h-2 w-2 shrink-0 rounded-[2px]"
                  style={{
                    backgroundColor: item.color,
                  }}
                />
              )}
              {itemConfig?.label}
            </div>
          );
        })}
      </div>
    );
  }
);
ChartLegendContent.displayName = "ChartLegend";

// Helper to extract item config from a payload.
function getPayloadConfigFromPayload(
  config: ChartConfig,
  payload: unknown,
  key: string
) {
  if (typeof payload !== "object" || payload === null) {
    return undefined;
  }

  const payloadPayload =
    "payload" in payload &&
    typeof payload.payload === "object" &&
    payload.payload !== null
      ? payload.payload
      : undefined;

  let configLabelKey: string = key;

  if (
    key in payload &&
```

```
    typeof payload[key as keyof typeof payload] === "string"
  ) {
    configLabelKey = payload[key as keyof typeof payload] as string;
  } else if (
    payloadPayload &&
    key in payloadPayload &&
    typeof payloadPayload[key as keyof typeof payloadPayload] === "string"
  ) {
    configLabelKey = payloadPayload[
      key as keyof typeof payloadPayload
    ] as string;
  }

  return configLabelKey in config
    ? config[configLabelKey]
    : config[key as keyof typeof config];
}

export {
  ChartContainer,
  ChartTooltip,
  ChartTooltipContent,
  ChartLegend,
  ChartLegendContent,
  ChartStyle,
};
```

/components/ui/sheet.tsx

```
"use client";

import * as React from "react";
import * as SheetPrimitive from "@radix-ui/react-dialog";
import { cva, type VariantProps } from "class-variance-authority";
import { X } from "lucide-react";

import { cn } from "@/lib/utils";

const Sheet = SheetPrimitive.Root;

const SheetTrigger = SheetPrimitive.Trigger;

const SheetClose = SheetPrimitive.Close;

const SheetPortal = SheetPrimitive.Portal;

const SheetOverlay = React.forwardRef<
  React.ElementRef<typeof SheetPrimitive.Overlay>,
  React.ComponentPropsWithoutRef<typeof SheetPrimitive.Overlay>
```

```
>(({ className, ...props }, ref) => (
  <SheetPrimitive.Overlay
    className={cn(
      "fixed inset-0 z-50 bg-black/80 data-[state=open]:animate-in data-[st
ate=closed]:animate-out data-[state=closed]:fade-out-0 data-[state=open]:fa
de-in-0",
      className
    )}
    {...props}
    ref={ref}
  />
));
SheetOverlay.displayName = SheetPrimitive.Overlay.displayName;

const sheetVariants = cva(
  // change nav-panel duration here
  "fixed z-50 gap-4 bg-background p-6 shadow-lg transition ease-in-out data
-[state=closed]:duration-500 data-[state=open]:duration-500 data-[state=ope
n]:animate-in data-[state=closed]:animate-out bg-background",
  {
    variants: {
      side: {
        top: "inset-x-0 top-0 border-b data-[state=closed]:slide-out-to-top
data-[state=open]:slide-in-from-top",
        bottom:
          "inset-x-0 bottom-0 border-t data-[state=closed]:slide-out-to-bot
tom data-[state=open]:slide-in-from-bottom",
        left: "inset-y-0 left-0 h-full w-3/4 border-r data-[state=closed]:s
lide-out-to-left data-[state=open]:slide-in-from-left sm:max-w-sm",
        right:
          "inset-y-0 right-0 h-full w-3/4 border-l data-[state=closed]:slid
e-out-to-right data-[state=open]:slide-in-from-right sm:max-w-sm",
      },
    },
    defaultVariants: {
      side: "right",
    },
  }
);

interface SheetContentProps
  extends React.ComponentPropsWithoutRef<typeof SheetPrimitive.Content>,
    VariantProps<typeof sheetVariants> {}

const SheetContent = React.forwardRef<
  React.ElementRef<typeof SheetPrimitive.Content>,
  SheetContentProps
>(({ side = "right", className, children, ...props }, ref) => (
  <SheetPortal>
    <SheetOverlay />
    <SheetPrimitive.Content
      ref={ref}
      className={cn(sheetVariants({ side }), className)}
```

```
      {...props}
    >
      <SheetPrimitive.Close className="absolute right-4 top-4 rounded-sm op
acity-70 ring-offset-white transition-opacity hover:opacity-100 focus:outli
ne-none focus:ring-2 focus:ring-slate-950 focus:ring-offset-2 disabled:poin
ter-events-none data-[state=open]:bg-slate-100 dark:ring-offset-slate-950 d
ark:focus:ring-slate-300 dark:data-[state=open]:bg-slate-800">
        <X className="h-4 w-4" />
        <span className="sr-only">Close</span>
      </SheetPrimitive.Close>
      {children}
    </SheetPrimitive.Content>
  </SheetPortal>
));
SheetContent.displayName = SheetPrimitive.Content.displayName;

const SheetHeader = ({
  className,
  ...props
}: React.HTMLAttributes<HTMLDivElement>) => (
  <div
    className={cn(
      "flex flex-col space-y-2 text-center sm:text-left",
      className
    )}
    {...props}
  />
);
SheetHeader.displayName = "SheetHeader";

const SheetFooter = ({
  className,
  ...props
}: React.HTMLAttributes<HTMLDivElement>) => (
  <div
    className={cn(
      "flex flex-col-reverse sm:flex-row sm:justify-end sm:space-x-2",
      className
    )}
    {...props}
  />
);
SheetFooter.displayName = "SheetFooter";

const SheetTitle = React.forwardRef<
  React.ElementRef<typeof SheetPrimitive.Title>,
  React.ComponentPropsWithoutRef<typeof SheetPrimitive.Title>
>(({ className, ...props }, ref) => (
  <SheetPrimitive.Title
    ref={ref}
    className={cn(
      "text-lg font-semibold text-slate-950 dark:text-slate-50",
      className
```

```
    )}
    {...props}
  />
));
SheetTitle.displayName = SheetPrimitive.Title.displayName;

const SheetDescription = React.forwardRef<
  React.ElementRef<typeof SheetPrimitive.Description>,
  React.ComponentPropsWithoutRef<typeof SheetPrimitive.Description>
>(({ className, ...props }, ref) => (
  <SheetPrimitive.Description
    ref={ref}
    className={cn("text-sm text-muted-foreground ", className)}
    {...props}
  />
));
SheetDescription.displayName = SheetPrimitive.Description.displayName;

export {
  Sheet,
  SheetPortal,
  SheetOverlay,
  SheetTrigger,
  SheetClose,
  SheetContent,
  SheetHeader,
  SheetFooter,
  SheetTitle,
  SheetDescription,
};
```

/components/ui/scroll-area.tsx

```
"use client";

import * as React from "react";
import * as ScrollAreaPrimitive from "@radix-ui/react-scroll-area";

import { cn } from "@/lib/utils";

const ScrollArea = React.forwardRef<
  React.ElementRef<typeof ScrollAreaPrimitive.Root>,
  React.ComponentPropsWithoutRef<typeof ScrollAreaPrimitive.Root>
>(({ className, children, ...props }, ref) => (
  <ScrollAreaPrimitive.Root
    ref={ref}
    className={cn("relative overflow-hidden", className)}
    {...props}
  >
    <ScrollAreaPrimitive.Viewport className="h-full w-full rounded-[inherit
```

```
]" >
      {children}
    </ScrollAreaPrimitive.Viewport>
    <ScrollBar />
    <ScrollAreaPrimitive.Corner />
  </ScrollAreaPrimitive.Root>
));
ScrollArea.displayName = ScrollAreaPrimitive.Root.displayName;

const ScrollBar = React.forwardRef<
  React.ElementRef<typeof ScrollAreaPrimitive.ScrollAreaScrollbar>,
  React.ComponentPropsWithoutRef<typeof ScrollAreaPrimitive.ScrollAreaScrol
lbar>
>(({ className, orientation = "vertical", ...props }, ref) => (
  <ScrollAreaPrimitive.ScrollAreaScrollbar
    ref={ref}
    orientation={orientation}
    className={cn(
      "flex touch-none select-none transition-colors",
      orientation === "vertical" &&
        "h-full w-2.5 border-l border-l-transparent p-[1px]",
      orientation === "horizontal" &&
        "h-2.5 flex-col border-t border-t-transparent p-[1px]",
      className
    )}
    {...props}
  >
    <ScrollAreaPrimitive.ScrollAreaThumb className="relative flex-1 rounded
-full bg-border" />
  </ScrollAreaPrimitive.ScrollAreaScrollbar>
));
ScrollBar.displayName = ScrollAreaPrimitive.ScrollAreaScrollbar.displayName
;

export { ScrollArea, ScrollBar };
```

/components/ui/resizable.tsx

```
"use client";

import { GripVertical } from "lucide-react";
import * as ResizablePrimitive from "react-resizable-panels";

import { cn } from "@/lib/utils";

const ResizablePanelGroup = ({
  className,
  ...props
}: React.ComponentProps<typeof ResizablePrimitive.PanelGroup>) => (
  <ResizablePrimitive.PanelGroup
```

```
    className={cn(
      "flex h-full w-full data-[panel-group-direction=vertical]:flex-col",
      className
    )}
    {...props}
  />
);

const ResizablePanel = ResizablePrimitive.Panel;

const ResizableHandle = ({
  withHandle,
  className,
  ...props
}: React.ComponentProps<typeof ResizablePrimitive.PanelResizeHandle> & {
  withHandle?: boolean;
}) => (
  <ResizablePrimitive.PanelResizeHandle
    className={cn(
      "relative flex w-px items-center justify-center bg-border after:absol
ute after:inset-y-0 after:left-1/2 after:w-1 after:-translate-x-1/2 focus-v
isible:outline-none focus-visible:ring-1 focus-visible:ring-ring focus-visi
ble:ring-offset-1 data-[panel-group-direction=vertical]:h-px data-[panel-gr
oup-direction=vertical]:w-full data-[panel-group-direction=vertical]:after:
left-0 data-[panel-group-direction=vertical]:after:h-1 data-[panel-group-di
rection=vertical]:after:w-full data-[panel-group-direction=vertical]:after:
-translate-y-1/2 data-[panel-group-direction=vertical]:after:translate-x-0
[&[data-panel-group-direction=vertical]>div]:rotate-90",
      className
    )}
    {...props}
  >
    {withHandle && (
      <div className="z-10 flex h-4 w-3 items-center justify-center rounded
-sm border bg-border">
        {" "}
        {/* bg-border or bg-background - both looks good */}
        <GripVertical className="h-2.5 w-2.5 text-accent-foreground" />
      </div>
    )}
  </ResizablePrimitive.PanelResizeHandle>
);

export { ResizablePanelGroup, ResizablePanel, ResizableHandle };
```

/components/ui/label.tsx

```
"use client"

import * as React from "react"
```

```
import * as LabelPrimitive from "@radix-ui/react-label"
import { cva, type VariantProps } from "class-variance-authority"

import { cn } from "@/lib/utils"

const labelVariants = cva(
  "text-sm font-medium leading-none peer-disabled:cursor-not-allowed peer-d
isabled:opacity-70"
)

const Label = React.forwardRef<
  React.ElementRef<typeof LabelPrimitive.Root>,
  React.ComponentPropsWithoutRef<typeof LabelPrimitive.Root> &
    VariantProps<typeof labelVariants>
>(({ className, ...props }, ref) => (
  <LabelPrimitive.Root
    ref={ref}
    className={cn(labelVariants(), className)}
    {...props}
  />
))
Label.displayName = LabelPrimitive.Root.displayName

export { Label }
```

/components/ui/tooltip.tsx

```
"use client"

import * as React from "react"
import * as TooltipPrimitive from "@radix-ui/react-tooltip"

import { cn } from "@/lib/utils"

const TooltipProvider = TooltipPrimitive.Provider

const Tooltip = TooltipPrimitive.Root

const TooltipTrigger = TooltipPrimitive.Trigger

const TooltipContent = React.forwardRef<
  React.ElementRef<typeof TooltipPrimitive.Content>,
  React.ComponentPropsWithoutRef<typeof TooltipPrimitive.Content>
>(({ className, sideOffset = 4, ...props }, ref) => (
  <TooltipPrimitive.Portal>
    <TooltipPrimitive.Content
      ref={ref}
      sideOffset={sideOffset}
      className={cn(
```

```
        "z-50 overflow-hidden rounded-md bg-slate-900 px-3 py-1.5 text-xs t
ext-slate-50 animate-in fade-in-0 zoom-in-95 data-[state=closed]:animate-ou
t data-[state=closed]:fade-out-0 data-[state=closed]:zoom-out-95 data-[side
=bottom]:slide-in-from-top-2 data-[side=left]:slide-in-from-right-2 data-[s
ide=right]:slide-in-from-left-2 data-[side=top]:slide-in-from-bottom-2 dark
:bg-slate-50 dark:text-slate-900",
        className
      )}
      {...props}
    />
  </TooltipPrimitive.Portal>
))
TooltipContent.displayName = TooltipPrimitive.Content.displayName

export { Tooltip, TooltipTrigger, TooltipContent, TooltipProvider }
```

/components/ui/alert.tsx

```
import * as React from "react";
import { cva, type VariantProps } from "class-variance-authority";

import { cn } from "@/lib/utils";

const alertVariants = cva(
  "relative w-full rounded-lg border px-4 py-3 text-sm [&>svg+div]:translat
e-y-[-3px] [&>svg]:absolute [&>svg]:left-4 [&>svg]:top-4 [&>svg]:text-slate
-950 [&>svg~*]:pl-7 dark:border-slate-800 dark:[&>svg]:text-slate-50",
  {
    variants: {
      variant: {
        default:
          "bg-background text-slate-950 bg-background dark:text-slate-50",
        destructive:
          "border-red-500/50 text-red-500 dark:border-red-500 [&>svg]:text-
red-500 dark:border-red-900/50 dark:text-red-900 dark:dark:border-red-900 d
ark:[&>svg]:text-red-900",
      },
    },
    defaultVariants: {
      variant: "default",
    },
  }
);

const Alert = React.forwardRef<
  HTMLDivElement,
  React.HTMLAttributes<HTMLDivElement> & VariantProps<typeof alertVariants>
>(({ className, variant, ...props }, ref) => (
  <div
    ref={ref}
```

```
    role="alert"
    className={cn(alertVariants({ variant }), className)}
    {...props}
  />
));
Alert.displayName = "Alert";

const AlertTitle = React.forwardRef<
  HTMLParagraphElement,
  React.HTMLAttributes<HTMLHeadingElement>
>(({ className, ...props }, ref) => (
  <h5
    ref={ref}
    className={cn("mb-1 font-medium leading-none tracking-tight", className
)}
    {...props}
  />
));
AlertTitle.displayName = "AlertTitle";

const AlertDescription = React.forwardRef<
  HTMLParagraphElement,
  React.HTMLAttributes<HTMLParagraphElement>
>(({ className, ...props }, ref) => (
  <div
    ref={ref}
    className={cn("text-sm [&_p]:leading-relaxed", className)}
    {...props}
  />
));
AlertDescription.displayName = "AlertDescription";

export { Alert, AlertTitle, AlertDescription };
```

/components/ui/switch.tsx

```
"use client";

import * as React from "react";
import * as SwitchPrimitives from "@radix-ui/react-switch";

import { cn } from "@/lib/utils";

const Switch = React.forwardRef<
  React.ElementRef<typeof SwitchPrimitives.Root>,
  React.ComponentPropsWithoutRef<typeof SwitchPrimitives.Root>
>(({ className, ...props }, ref) => (
  <SwitchPrimitives.Root
    className={cn(
      "peer inline-flex h-5 w-9 shrink-0 cursor-pointer items-center rounde
```

```
d-full border-2 border-transparent shadow-sm transition-colors focus-visibl
e:outline-none focus-visible:ring-2 focus-visible:ring-slate-950 focus-visi
ble:ring-offset-2 focus-visible:ring-offset-white disabled:cursor-not-allow
ed disabled:opacity-50 data-[state=checked]:bg-slate-900 data-[state=unchec
ked]:bg-slate-200 dark:focus-visible:ring-primary dark:focus-visible:ring-o
ffset-slate-950 dark:data-[state=checked]:bg-slate-50 dark:data-[state=unch
ecked]:bg-slate-800",
      className
    )}
    {...props}
    ref={ref}
  >
    <SwitchPrimitives.Thumb
      className={cn(
        "pointer-events-none block h-4 w-4 rounded-full bg-background shado
w-lg ring-0 transition-transform data-[state=checked]:translate-x-4 data-[s
tate=unchecked]:translate-x-0 bg-background"
      )}
    />
  </SwitchPrimitives.Root>
));
Switch.displayName = SwitchPrimitives.Root.displayName;

export { Switch };
```

/components/ui/calendar.tsx

```
"use client";

import * as React from "react";
import { ChevronLeft, ChevronRight } from "lucide-react";
import { DayPicker } from "react-day-picker";

import { cn } from "@/lib/utils";
import { buttonVariants } from "@/components/ui/button";

export type CalendarProps = React.ComponentProps<typeof DayPicker>;

function Calendar({
  className,
  classNames,
  showOutsideDays = true,
  ...props
}: CalendarProps) {
  return (
    <DayPicker
      showOutsideDays={showOutsideDays}
      className={cn("p-3", className)}
      classNames={{
        months: "flex flex-col sm:flex-row space-y-4 sm:space-x-4 sm:space-
```

```
y-0",
        month: "space-y-4",
        caption: "flex justify-center pt-1 relative items-center",
        caption_label: "text-sm font-medium",
        nav: "space-x-1 flex items-center",
        nav_button: cn(
          buttonVariants({ variant: "outline" }),
          "h-7 w-7 bg-transparent p-0 opacity-50 hover:opacity-100"
        ),
        nav_button_previous: "absolute left-1",
        nav_button_next: "absolute right-1",
        table: "w-full border-collapse space-y-1",
        head_row: "flex",
        head_cell:
          "text-slate-500 rounded-md w-8 font-normal text-[0.8rem] dark:tex
t-slate-400",
        row: "flex w-full mt-2",
        cell: cn(
          "relative p-0 text-center text-sm focus-within:relative focus-wit
hin:z-20 [&:has([aria-selected])]:bg-slate-100 [&:has([aria-selected].day-o
utside)]:bg-slate-100/50 [&:has([aria-selected].day-range-end)]:rounded-r-m
d dark:[&:has([aria-selected])]:bg-muted dark:[&:has([aria-selected].day-ou
tside)]:bg-muted/50",
          props.mode === "range"
            ? "[&:has(>.day-range-end)]:rounded-r-md [&:has(>.day-range-sta
rt)]:rounded-l-md first:[&:has([aria-selected])]:rounded-l-md last:[&:has([
aria-selected])]:rounded-r-md"
            : "[&:has([aria-selected])]:rounded-md"
        ),
        day: cn(
          buttonVariants({ variant: "ghost" }),
          "h-8 w-8 p-0 font-normal aria-selected:opacity-100"
        ),
        day_range_start: "day-range-start",
        day_range_end: "day-range-end",
        day_selected:
          "bg-primary text-primary-foreground hover:bg-primary hover:text-p
rimary-foreground",
        day_today: "border border-muted",
        day_outside:
          "invisible day-outside text-slate-500 aria-selected:bg-slate-100/
50 aria-selected:text-slate-500 dark:text-slate-400 dark:aria-selected:bg-m
uted/50 dark:aria-selected:text-slate-400",
        day_disabled: "text-muted-foreground opacity-50",
        day_range_middle:
          "aria-selected:bg-slate-100 aria-selected:text-slate-900 dark:ari
a-selected:bg-muted dark:aria-selected:text-slate-50",
        day_hidden: "invisible",
        ...classNames,
      }}
      components={{
        IconLeft: ({ ...props }) => <ChevronLeft className="h-4 w-4" />,
        IconRight: ({ ...props }) => <ChevronRight className="h-4 w-4" />,
```

```
      }}
      disabled={{ before: new Date() }}
      {...props}
    />
  );
}
Calendar.displayName = "Calendar";

export { Calendar };
```

/components/ui/radio-group.tsx

```
"use client";

import * as React from "react";
import * as RadioGroupPrimitive from "@radix-ui/react-radio-group";
import { Circle } from "lucide-react";

import { cn } from "@/lib/utils";

const RadioGroup = React.forwardRef<
  React.ElementRef<typeof RadioGroupPrimitive.Root>,
  React.ComponentPropsWithoutRef<typeof RadioGroupPrimitive.Root>
>(({ className, ...props }, ref) => {
  return (
    <RadioGroupPrimitive.Root
      className={cn("grid gap-2", className)}
      {...props}
      ref={ref}
    />
  );
});
RadioGroup.displayName = RadioGroupPrimitive.Root.displayName;

const RadioGroupItem = React.forwardRef<
  React.ElementRef<typeof RadioGroupPrimitive.Item>,
  React.ComponentPropsWithoutRef<typeof RadioGroupPrimitive.Item>
>(({ className, ...props }, ref) => {
  return (
    <RadioGroupPrimitive.Item
      ref={ref}
      className={cn(
        "aspect-square h-4 w-4 rounded-full border border-slate-900 text-sl
ate-900 shadow focus:outline-none focus-visible:ring-1 focus-visible:ring-s
late-950 disabled:cursor-not-allowed disabled:opacity-50 dark:border-slate-
800 dark:border-slate-50 dark:text-slate-50 dark:focus-visible:ring-primary
",
        className
      )}
      {...props}
```

```
    >
      <RadioGroupPrimitive.Indicator className="flex items-center justify-c
enter">
        <Circle className="h-3.5 w-3.5 fill-primary" />
      </RadioGroupPrimitive.Indicator>
    </RadioGroupPrimitive.Item>
  );
});
RadioGroupItem.displayName = RadioGroupPrimitive.Item.displayName;

export { RadioGroup, RadioGroupItem };
```

/components/ui/command.tsx

```
"use client";

import * as React from "react";
import { type DialogProps } from "@radix-ui/react-dialog";
import { Command as CommandPrimitive } from "cmdk";
import { Search } from "lucide-react";

import { cn } from "@/lib/utils";
import { Dialog, DialogContent } from "@/components/ui/dialog";

const Command = React.forwardRef<
  React.ElementRef<typeof CommandPrimitive>,
  React.ComponentPropsWithoutRef<typeof CommandPrimitive>
>(({ className, ...props }, ref) => (
  <CommandPrimitive
    ref={ref}
    className={cn(
      "flex h-full w-full flex-col overflow-hidden rounded-md bg-background
text-slate-950 bg-background dark:text-slate-50",
      className
    )}
    {...props}
  />
));
Command.displayName = CommandPrimitive.displayName;

const CommandDialog = ({ children, ...props }: DialogProps) => {
  return (
    <Dialog {...props}>
      <DialogContent className="overflow-hidden p-0">
        <Command className="[&_[cmdk-group-heading]]:px-2 [&_[cmdk-group-he
ading]]:font-medium [&_[cmdk-group-heading]]:text-muted-foreground [&_[cmdk
-group]:not([hidden])_~[cmdk-group]]:pt-0 [&_[cmdk-group]]:px-2 [&_[cmdk-in
put-wrapper]_svg]:h-5 [&_[cmdk-input-wrapper]_svg]:w-5 [&_[cmdk-input]]:h-1
2 [&_[cmdk-item]]:px-2 [&_[cmdk-item]]:py-3 [&_[cmdk-item]_svg]:h-5 [&_[cmd
k-item]_svg]:w-5 dark:[&_[cmdk-group-heading]]:text-muted-foreground">
```

```
          {children}
        </Command>
      </DialogContent>
    </Dialog>
  );
};

const CommandInput = React.forwardRef<
  React.ElementRef<typeof CommandPrimitive.Input>,
  React.ComponentPropsWithoutRef<typeof CommandPrimitive.Input>
>(({ className, ...props }, ref) => (
  <div className="flex items-center border-b px-3" cmdk-input-wrapper="">
    <Search className="mr-2 h-4 w-4 shrink-0 opacity-50" />
    <CommandPrimitive.Input
      ref={ref}
      className={cn(
        "flex h-10 w-full rounded-md bg-transparent py-3 text-sm outline-no
ne placeholder:text-muted-foreground disabled:cursor-not-allowed disabled:o
pacity-50 dark:placeholder:text-muted-foreground",
        className
      )}
      {...props}
    />
  </div>
));

CommandInput.displayName = CommandPrimitive.Input.displayName;

const CommandList = React.forwardRef<
  React.ElementRef<typeof CommandPrimitive.List>,
  React.ComponentPropsWithoutRef<typeof CommandPrimitive.List>
>(({ className, ...props }, ref) => (
  <CommandPrimitive.List
    ref={ref}
    className={cn("max-h-[300px] overflow-y-auto overflow-x-hidden", classN
ame)}
    {...props}
  />
));

CommandList.displayName = CommandPrimitive.List.displayName;

const CommandEmpty = React.forwardRef<
  React.ElementRef<typeof CommandPrimitive.Empty>,
  React.ComponentPropsWithoutRef<typeof CommandPrimitive.Empty>
>((props, ref) => (
  <CommandPrimitive.Empty
    ref={ref}
    className="py-6 text-center text-sm"
    {...props}
  />
));
```

```
CommandEmpty.displayName = CommandPrimitive.Empty.displayName;

const CommandGroup = React.forwardRef<
  React.ElementRef<typeof CommandPrimitive.Group>,
  React.ComponentPropsWithoutRef<typeof CommandPrimitive.Group>
>(({ className, ...props }, ref) => (
  <CommandPrimitive.Group
    ref={ref}
    className={cn(
      "overflow-hidden p-1 text-slate-950 [&_[cmdk-group-heading]]:px-2 [&_
[cmdk-group-heading]]:py-1.5 [&_[cmdk-group-heading]]:text-xs [&_[cmdk-grou
p-heading]]:font-medium [&_[cmdk-group-heading]]:text-muted-foreground dark
:text-slate-50 dark:[&_[cmdk-group-heading]]:text-muted-foreground",
      className
    )}
    {...props}
  />
));

CommandGroup.displayName = CommandPrimitive.Group.displayName;

const CommandSeparator = React.forwardRef<
  React.ElementRef<typeof CommandPrimitive.Separator>,
  React.ComponentPropsWithoutRef<typeof CommandPrimitive.Separator>
>(({ className, ...props }, ref) => (
  <CommandPrimitive.Separator
    ref={ref}
    className={cn("-mx-1 h-px bg-slate-200 dark:bg-slate-800", className)}
    {...props}
  />
));
CommandSeparator.displayName = CommandPrimitive.Separator.displayName;

const CommandItem = React.forwardRef<
  React.ElementRef<typeof CommandPrimitive.Item>,
  React.ComponentPropsWithoutRef<typeof CommandPrimitive.Item>
>(({ className, ...props }, ref) => (
  <CommandPrimitive.Item
    ref={ref}
    className={cn(
      "relative flex cursor-default gap-2 select-none items-center rounded-
sm px-2 py-1.5 text-sm outline-none data-[disabled=true]:pointer-events-non
e data-[selected=true]:bg-slate-100 data-[selected=true]:text-slate-900 dat
a-[disabled=true]:opacity-50 [&_svg]:pointer-events-none [&_svg]:size-4 [&_
svg]:shrink-0 dark:data-[selected=true]:bg-slate-800 dark:data-[selected=tr
ue]:text-slate-50",
      className
    )}
    {...props}
  />
));
```

```
CommandItem.displayName = CommandPrimitive.Item.displayName;

const CommandShortcut = ({
  className,
  ...props
}: React.HTMLAttributes<HTMLSpanElement>) => {
  return (
    <span
      className={cn(
        "ml-auto text-xs tracking-widest text-muted-foreground ",
        className
      )}
      {...props}
    />
  );
};
CommandShortcut.displayName = "CommandShortcut";

export {
  Command,
  CommandDialog,
  CommandInput,
  CommandList,
  CommandEmpty,
  CommandGroup,
  CommandItem,
  CommandShortcut,
  CommandSeparator,
};
```

/components/ui/avatar.tsx

```
"use client"

import * as React from "react"
import * as AvatarPrimitive from "@radix-ui/react-avatar"

import { cn } from "@/lib/utils"

const Avatar = React.forwardRef<
  React.ElementRef<typeof AvatarPrimitive.Root>,
  React.ComponentPropsWithoutRef<typeof AvatarPrimitive.Root>
>(({ className, ...props }, ref) => (
  <AvatarPrimitive.Root
    ref={ref}
    className={cn(
      "relative flex h-10 w-10 shrink-0 overflow-hidden rounded-full",
      className
    )}
```

```
    {...props}
  />
))
Avatar.displayName = AvatarPrimitive.Root.displayName

const AvatarImage = React.forwardRef<
  React.ElementRef<typeof AvatarPrimitive.Image>,
  React.ComponentPropsWithoutRef<typeof AvatarPrimitive.Image>
>(({ className, ...props }, ref) => (
  <AvatarPrimitive.Image
    ref={ref}
    className={cn("aspect-square h-full w-full", className)}
    {...props}
  />
))
AvatarImage.displayName = AvatarPrimitive.Image.displayName

const AvatarFallback = React.forwardRef<
  React.ElementRef<typeof AvatarPrimitive.Fallback>,
  React.ComponentPropsWithoutRef<typeof AvatarPrimitive.Fallback>
>(({ className, ...props }, ref) => (
  <AvatarPrimitive.Fallback
    ref={ref}
    className={cn(
      "flex h-full w-full items-center justify-center rounded-full bg-slate
-100 dark:bg-slate-800",
      className
    )}
    {...props}
  />
))
AvatarFallback.displayName = AvatarPrimitive.Fallback.displayName

export { Avatar, AvatarImage, AvatarFallback }
```

/components/ui/dialog.tsx

```
"use client";

import * as React from "react";
import * as DialogPrimitive from "@radix-ui/react-dialog";
import { X } from "lucide-react";

import { cn } from "@/lib/utils";

const Dialog = DialogPrimitive.Root;

const DialogTrigger = DialogPrimitive.Trigger;
```

```
const DialogPortal = DialogPrimitive.Portal;

const DialogClose = DialogPrimitive.Close;

const DialogOverlay = React.forwardRef<
  React.ElementRef<typeof DialogPrimitive.Overlay>,
  React.ComponentPropsWithoutRef<typeof DialogPrimitive.Overlay>
>(({ className, ...props }, ref) => (
  <DialogPrimitive.Overlay
    ref={ref}
    className={cn(
      "fixed inset-0 z-50 bg-black/80 data-[state=open]:animate-in data-[st
ate=closed]:animate-out data-[state=closed]:fade-out-0 data-[state=open]:fa
de-in-0",
      className
    )}
    {...props}
  />
));
DialogOverlay.displayName = DialogPrimitive.Overlay.displayName;

const DialogContent = React.forwardRef<
  React.ElementRef<typeof DialogPrimitive.Content>,
  React.ComponentPropsWithoutRef<typeof DialogPrimitive.Content>
>(({ className, children, ...props }, ref) => (
  <DialogPortal>
    <DialogOverlay />
    <DialogPrimitive.Content
      ref={ref}
      className={cn(
        "fixed left-[50%] top-[50%] z-50 grid w-full max-w-lg translate-x-[
-50%] translate-y-[-50%] gap-4 border bg-background p-6 shadow-lg duration-
200 data-[state=open]:animate-in data-[state=closed]:animate-out data-[stat
e=closed]:fade-out-0 data-[state=open]:fade-in-0 data-[state=closed]:zoom-o
ut-95 data-[state=open]:zoom-in-95 data-[state=closed]:slide-out-to-left-1/
2 data-[state=closed]:slide-out-to-top-[48%] data-[state=open]:slide-in-fro
m-left-1/2 data-[state=open]:slide-in-from-top-[48%] sm:rounded-lg",
        className
      )}
      {...props}
    >
      {children}
      <DialogPrimitive.Close className="absolute right-4 top-4 rounded-sm o
pacity-70 ring-offset-white transition-opacity hover:opacity-100 focus:outl
ine-none focus:ring-2 focus:ring-slate-950 focus:ring-offset-2 disabled:poi
nter-events-none data-[state=open]:bg-slate-100 data-[state=open]:text-mute
d-foreground dark:ring-offset-slate-950 dark:focus:ring-slate-300 dark:data
-[state=open]:bg-slate-800 dark:data-[state=open]:text-muted-foreground">
        <X className="h-4 w-4" />
        <span className="sr-only">Close</span>
      </DialogPrimitive.Close>
    </DialogPrimitive.Content>
  </DialogPortal>
```

```
));
DialogContent.displayName = DialogPrimitive.Content.displayName;

const DialogHeader = ({
  className,
  ...props
}: React.HTMLAttributes<HTMLDivElement>) => (
  <div
    className={cn(
      "flex flex-col space-y-1.5 text-center sm:text-left",
      className
    )}
    {...props}
  />
);
DialogHeader.displayName = "DialogHeader";

const DialogFooter = ({
  className,
  ...props
}: React.HTMLAttributes<HTMLDivElement>) => (
  <div
    className={cn(
      "mt-5 flex gap-2 flex-col-reverse sm:flex-row sm:justify-between  sm:
space-x-2",
      className
    )}
    {...props}
  />
);
DialogFooter.displayName = "DialogFooter";

const DialogTitle = React.forwardRef<
  React.ElementRef<typeof DialogPrimitive.Title>,
  React.ComponentPropsWithoutRef<typeof DialogPrimitive.Title>
>(({ className, ...props }, ref) => (
  <DialogPrimitive.Title
    ref={ref}
    className={cn(
      "text-lg font-semibold leading-none tracking-tight",
      className
    )}
    {...props}
  />
));
DialogTitle.displayName = DialogPrimitive.Title.displayName;

const DialogDescription = React.forwardRef<
  React.ElementRef<typeof DialogPrimitive.Description>,
  React.ComponentPropsWithoutRef<typeof DialogPrimitive.Description>
>(({ className, ...props }, ref) => (
  <DialogPrimitive.Description
```

```
    ref={ref}
    className={cn("text-sm text-muted-foreground", className)}
    {...props}
  />
));
DialogDescription.displayName = DialogPrimitive.Description.displayName;

export {
  Dialog,
  DialogPortal,
  DialogOverlay,
  DialogTrigger,
  DialogClose,
  DialogContent,
  DialogHeader,
  DialogFooter,
  DialogTitle,
  DialogDescription,
};
```

/components/ui/badge.tsx

```
import * as React from "react";
import { cva, type VariantProps } from "class-variance-authority";

import { cn } from "@/lib/utils";

const badgeVariants = cva(
  "inline-flex items-center rounded-md border px-2.5 py-0.5 text-xs font-se
mibold transition-colors focus:outline-none focus:ring-2 focus:ring-slate-9
50 focus:ring-offset-2 dark:border-slate-800 dark:focus:ring-slate-300",
  {
    variants: {
      variant: {
        default:
          "border-transparent bg-slate-900 text-slate-50 shadow hover:bg-sl
ate-900/80 dark:bg-slate-50 dark:text-slate-900 dark:hover:bg-slate-50/80",
        secondary:
          "border-transparent bg-slate-100 text-slate-900 hover:bg-muted/80
dark:bg-slate-800 dark:text-slate-50 dark:hover:bg-slate-800/80",
        destructive:
          "border-transparent bg-red-500 text-slate-50 shadow hover:bg-red-
500/80 dark:bg-red-900 dark:text-slate-50 dark:hover:bg-red-900/80",
        outline: "text-slate-950 dark:text-slate-50",
      },
    },
    defaultVariants: {
      variant: "default",
    },
  }
```

```
);

export interface BadgeProps
  extends React.HTMLAttributes<HTMLDivElement>,
    VariantProps<typeof badgeVariants> {}

function Badge({ className, variant, ...props }: BadgeProps) {
  return (
    <div className={cn(badgeVariants({ variant }), className)} {...props} /
>
  );
}

export { Badge, badgeVariants };
```

/components/ui/table.tsx

```
import * as React from "react";

import { cn } from "@/lib/utils";

const Table = React.forwardRef<
  HTMLTableElement,
  React.HTMLAttributes<HTMLTableElement>
>(({ className, ...props }, ref) => (
  <div className="relative w-full overflow-auto">
    <table
      ref={ref}
      className={cn("w-full caption-bottom text-sm", className)}
      {...props}
    />
  </div>
));
Table.displayName = "Table";

const TableHeader = React.forwardRef<
  HTMLTableSectionElement,
  React.HTMLAttributes<HTMLTableSectionElement>
>(({ className, ...props }, ref) => (
  <thead ref={ref} className={cn("[&_tr]:border-b", className)} {...props}
/>
));
TableHeader.displayName = "TableHeader";

const TableBody = React.forwardRef<
  HTMLTableSectionElement,
  React.HTMLAttributes<HTMLTableSectionElement>
>(({ className, ...props }, ref) => (
  <tbody
    ref={ref}
```

```
    className={cn("[&_tr:last-child]:border-0", className)}
    {...props}
  />
));
TableBody.displayName = "TableBody";

const TableFooter = React.forwardRef<
  HTMLTableSectionElement,
  React.HTMLAttributes<HTMLTableSectionElement>
>(({ className, ...props }, ref) => (
  <tfoot
    ref={ref}
    className={cn(
      "border-t bg-slate-100/50 font-medium [&>tr]:last:border-b-0 dark:bg-
slate-800/50",
      className
    )}
    {...props}
  />
));
TableFooter.displayName = "TableFooter";

const TableRow = React.forwardRef<
  HTMLTableRowElement,
  React.HTMLAttributes<HTMLTableRowElement>
>(({ className, ...props }, ref) => (
  <tr
    ref={ref}
    className={cn(
      "border-b transition-colors hover:bg-muted/50 data-[state=selected]:b
g-slate-100 dark:hover:bg-slate-800/50 dark:data-[state=selected]:bg-slate-
800",
      className
    )}
    {...props}
  />
));
TableRow.displayName = "TableRow";

const TableHead = React.forwardRef<
  HTMLTableCellElement,
  React.ThHTMLAttributes<HTMLTableCellElement>
>(({ className, ...props }, ref) => (
  <th
    ref={ref}
    className={cn(
      "h-10 px-2 text-left align-middle font-medium text-muted-foreground [
&:has([role=checkbox])]:pr-0 [&>[role=checkbox]]:translate-y-[2px] ",
      className
    )}
    {...props}
  />
));
```

```
TableHead.displayName = "TableHead";

const TableCell = React.forwardRef<
  HTMLTableCellElement,
  React.TdHTMLAttributes<HTMLTableCellElement>
>(({ className, ...props }, ref) => (
  <td
    ref={ref}
    className={cn(
      "p-2 align-middle [&:has([role=checkbox])]:pr-0 [&>[role=checkbox]]:t
ranslate-y-[2px]",
      className
    )}
    {...props}
  />
));
TableCell.displayName = "TableCell";

const TableCaption = React.forwardRef<
  HTMLTableCaptionElement,
  React.HTMLAttributes<HTMLTableCaptionElement>
>(({ className, ...props }, ref) => (
  <caption
    ref={ref}
    className={cn("mt-4 text-sm text-muted-foreground ", className)}
    {...props}
  />
));
TableCaption.displayName = "TableCaption";

export {
  Table,
  TableHeader,
  TableBody,
  TableFooter,
  TableHead,
  TableRow,
  TableCell,
  TableCaption,
};
```

/components/ui/separator.tsx

```
"use client";

import * as React from "react";
import * as SeparatorPrimitive from "@radix-ui/react-separator";

import { cn } from "@/lib/utils";
```

```
const Separator = React.forwardRef<
  React.ElementRef<typeof SeparatorPrimitive.Root>,
  React.ComponentPropsWithoutRef<typeof SeparatorPrimitive.Root>
>(
  (
    { className, orientation = "horizontal", decorative = true, ...props },
    ref
  ) => (
    <SeparatorPrimitive.Root
      ref={ref}
      decorative={decorative}
      orientation={orientation}
      className={cn(
        "shrink-0 bg-border",
        orientation === "horizontal" ? "h-[1px] w-full" : "h-full w-[1px]",
        className
      )}
      {...props}
    />
  )
);
Separator.displayName = SeparatorPrimitive.Root.displayName;

export { Separator };
```

/components/ui/button.tsx

```
import * as React from "react";
import { Slot } from "@radix-ui/react-slot";
import { cva, type VariantProps } from "class-variance-authority";

import { cn } from "@/lib/utils";

const buttonVariants = cva(
  "inline-flex items-center justify-center gap-2 whitespace-nowrap rounded-
md text-sm font-medium transition-colors focus-visible:outline-none focus-v
isible:ring-1 focus-visible:ring-primary disabled:pointer-events-none disab
led:opacity-50 [&_svg]:pointer-events-none [&_svg]:size-4 [&_svg]:shrink-0"
,
  {
    variants: {
      variant: {
        default:
          "bg-primary text-primary-foreground shadow hover:bg-primary/90 fo
cus-visible:ring-white",
        destructive:
          "bg-red-500 text-slate-50 shadow-sm hover:bg-red-500/90 dark:bg-r
ed-900 dark:text-slate-50 dark:hover:bg-red-900/90",
        outline: "border shadow-sm hover:bg-accent",
        secondary: "bg-accent/70 text-foreground shadow-sm hover:bg-accent"
```

```
,
        ghost:
          "hover:bg-accent hover:text-accent-foreground ring-1 ring-transpa
rent focus-visible:ring-1",
        link: "text-slate-900 underline-offset-4 hover:underline dark:text-
slate-50",
        none: "",
      },
      size: {
        default: "h-9 px-4 py-2",
        sm: "h-8 rounded-md px-3 text-xs",
        lg: "h-10 rounded-md px-8",
        icon: "h-9 w-9",
      },
    },
    defaultVariants: {
      variant: "default",
      size: "default",
    },
  }
);

export interface ButtonProps
  extends React.ButtonHTMLAttributes<HTMLButtonElement>,
    VariantProps<typeof buttonVariants> {
  asChild?: boolean;
}

const Button = React.forwardRef<HTMLButtonElement, ButtonProps>(
  ({ className, variant, size, asChild = false, ...props }, ref) => {
    const Comp = asChild ? Slot : "button";
    return (
      <Comp
        className={cn(buttonVariants({ variant, size, className }))}
        ref={ref}
        {...props}
      />
    );
  }
);
Button.displayName = "Button";

export { Button, buttonVariants };
```

/components/ui/checkbox.tsx

```
"use client";

import * as React from "react";
import * as CheckboxPrimitive from "@radix-ui/react-checkbox";
```

```
import { Check } from "lucide-react";

import { cn } from "@/lib/utils";

const Checkbox = React.forwardRef<
  React.ElementRef<typeof CheckboxPrimitive.Root>,
  React.ComponentPropsWithoutRef<typeof CheckboxPrimitive.Root>
>(({ className, ...props }, ref) => (
  <CheckboxPrimitive.Root
    ref={ref}
    className={cn(
      "peer h-4 w-4 shrink-0 rounded-sm border border-slate-900 shadow focu
s-visible:outline-none focus-visible:ring-1 focus-visible:ring-slate-950 di
sabled:cursor-not-allowed disabled:opacity-50 data-[state=checked]:bg-slate
-900 data-[state=checked]:text-slate-50 dark:border-slate-800 dark:border-s
late-50 dark:focus-visible:ring-primary dark:data-[state=checked]:bg-slate-
50 dark:data-[state=checked]:text-slate-900",
      className
    )}
    {...props}
  >
    <CheckboxPrimitive.Indicator
      className={cn("flex items-center justify-center text-current")}
    >
      <Check className="h-4 w-4" />
    </CheckboxPrimitive.Indicator>
  </CheckboxPrimitive.Root>
));
Checkbox.displayName = CheckboxPrimitive.Root.displayName;

export { Checkbox };
```

/components/ui/collapsible.tsx

```
"use client"

import * as CollapsiblePrimitive from "@radix-ui/react-collapsible"

const Collapsible = CollapsiblePrimitive.Root

const CollapsibleTrigger = CollapsiblePrimitive.CollapsibleTrigger

const CollapsibleContent = CollapsiblePrimitive.CollapsibleContent

export { Collapsible, CollapsibleTrigger, CollapsibleContent }
```

/components/ui/dropdown-menu.tsx

```
"use client";

import * as React from "react";
import * as DropdownMenuPrimitive from "@radix-ui/react-dropdown-menu";
import { Check, ChevronRight, Circle } from "lucide-react";

import { cn } from "@/lib/utils";

const DropdownMenu = DropdownMenuPrimitive.Root;

const DropdownMenuTrigger = DropdownMenuPrimitive.Trigger;

const DropdownMenuGroup = DropdownMenuPrimitive.Group;

const DropdownMenuPortal = DropdownMenuPrimitive.Portal;

const DropdownMenuSub = DropdownMenuPrimitive.Sub;

const DropdownMenuRadioGroup = DropdownMenuPrimitive.RadioGroup;

const DropdownMenuSubTrigger = React.forwardRef<
  React.ElementRef<typeof DropdownMenuPrimitive.SubTrigger>,
  React.ComponentPropsWithoutRef<typeof DropdownMenuPrimitive.SubTrigger> &
{
    inset?: boolean;
  }
>(({ className, inset, children, ...props }, ref) => (
  <DropdownMenuPrimitive.SubTrigger
    ref={ref}
    className={cn(
      "flex cursor-default gap-2 select-none items-center rounded-sm px-2 p
y-1.5 text-sm outline-none focus:bg-slate-100 data-[state=open]:bg-slate-10
0 [&_svg]:pointer-events-none [&_svg]:size-4 [&_svg]:shrink-0 dark:focus:bg
-slate-800 dark:data-[state=open]:bg-slate-800",
      inset && "pl-8",
      className
    )}
    {...props}
  >
    {children}
    <ChevronRight className="ml-auto" />
  </DropdownMenuPrimitive.SubTrigger>
));
DropdownMenuSubTrigger.displayName =
  DropdownMenuPrimitive.SubTrigger.displayName;

const DropdownMenuSubContent = React.forwardRef<
  React.ElementRef<typeof DropdownMenuPrimitive.SubContent>,
  React.ComponentPropsWithoutRef<typeof DropdownMenuPrimitive.SubContent>
>(({ className, ...props }, ref) => (
  <DropdownMenuPrimitive.SubContent
```

```
    ref={ref}
    className={cn(
      "z-50 min-w-[8rem] bg-background overflow-hidden rounded-md border p-
1 shadow-lg data-[state=open]:animate-in data-[state=closed]:animate-out da
ta-[state=closed]:fade-out-0 data-[state=open]:fade-in-0 data-[state=closed
]:zoom-out-95 data-[state=open]:zoom-in-95 data-[side=bottom]:slide-in-from
-top-2 data-[side=left]:slide-in-from-right-2 data-[side=right]:slide-in-fr
om-left-2 data-[side=top]:slide-in-from-bottom-2",
      className
    )}
    {...props}
  />
));
DropdownMenuSubContent.displayName =
  DropdownMenuPrimitive.SubContent.displayName;

const DropdownMenuContent = React.forwardRef<
  React.ElementRef<typeof DropdownMenuPrimitive.Content>,
  React.ComponentPropsWithoutRef<typeof DropdownMenuPrimitive.Content>
>(({ className, sideOffset = 4, ...props }, ref) => (
  <DropdownMenuPrimitive.Portal>
    <DropdownMenuPrimitive.Content
      ref={ref}
      sideOffset={sideOffset}
      className={cn(
        "z-50 min-w-[8rem] bg-background overflow-hidden rounded-md border
p-1 shadow-md",
        "data-[state=open]:animate-in data-[state=closed]:animate-out data-
[state=closed]:fade-out-0 data-[state=open]:fade-in-0 data-[state=closed]:z
oom-out-95 data-[state=open]:zoom-in-95 data-[side=bottom]:slide-in-from-to
p-2 data-[side=left]:slide-in-from-right-2 data-[side=right]:slide-in-from-
left-2 data-[side=top]:slide-in-from-bottom-2",
        className
      )}
      {...props}
    />
  </DropdownMenuPrimitive.Portal>
));
DropdownMenuContent.displayName = DropdownMenuPrimitive.Content.displayName
;

const DropdownMenuItem = React.forwardRef<
  React.ElementRef<typeof DropdownMenuPrimitive.Item>,
  React.ComponentPropsWithoutRef<typeof DropdownMenuPrimitive.Item> & {
    inset?: boolean;
  }
>(({ className, inset, ...props }, ref) => (
  <DropdownMenuPrimitive.Item
    ref={ref}
    className={cn(
      "relative flex cursor-default select-none items-center gap-2 rounded-
sm px-2 py-1.5 text-sm outline-none transition-colors hover:bg-accent data-
[disabled]:pointer-events-none data-[disabled]:opacity-50 [&>svg]:size-4 [&
```

```
>svg]:shrink-0",
      inset && "pl-8",
      className
    )}
    {...props}
  />
));
DropdownMenuItem.displayName = DropdownMenuPrimitive.Item.displayName;

const DropdownMenuCheckboxItem = React.forwardRef<
  React.ElementRef<typeof DropdownMenuPrimitive.CheckboxItem>,
  React.ComponentPropsWithoutRef<typeof DropdownMenuPrimitive.CheckboxItem>
>(({ className, children, checked, ...props }, ref) => (
  <DropdownMenuPrimitive.CheckboxItem
    ref={ref}
    className={cn(
      "relative flex cursor-default select-none items-center rounded-sm py-
1.5 pl-8 pr-2 text-sm outline-none transition-colors hover:bg-accent data-[
disabled]:pointer-events-none data-[disabled]:opacity-50",
      className
    )}
    checked={checked}
    {...props}
  >
    <span className="absolute left-2 flex h-3.5 w-3.5 items-center justify-
center">
      <DropdownMenuPrimitive.ItemIndicator>
        <Check className="h-4 w-4" />
      </DropdownMenuPrimitive.ItemIndicator>
    </span>
    {children}
  </DropdownMenuPrimitive.CheckboxItem>
));
DropdownMenuCheckboxItem.displayName =
  DropdownMenuPrimitive.CheckboxItem.displayName;

const DropdownMenuRadioItem = React.forwardRef<
  React.ElementRef<typeof DropdownMenuPrimitive.RadioItem>,
  React.ComponentPropsWithoutRef<typeof DropdownMenuPrimitive.RadioItem>
>(({ className, children, ...props }, ref) => (
  <DropdownMenuPrimitive.RadioItem
    ref={ref}
    className={cn(
      "relative flex cursor-default select-none items-center rounded-sm py-
1.5 pl-8 pr-2 text-sm outline-none transition-colors focus:bg-slate-100 foc
us:text-slate-900 data-[disabled]:pointer-events-none data-[disabled]:opaci
ty-50 dark:focus:bg-slate-800 dark:focus:text-slate-50",
      className
    )}
    {...props}
  >
    <span className="absolute left-2 flex h-3.5 w-3.5 items-center justify-
center">
```

```
      <DropdownMenuPrimitive.ItemIndicator>
        <Circle className="h-2 w-2 fill-current" />
      </DropdownMenuPrimitive.ItemIndicator>
    </span>
    {children}
  </DropdownMenuPrimitive.RadioItem>
));
DropdownMenuRadioItem.displayName = DropdownMenuPrimitive.RadioItem.display
Name;

const DropdownMenuLabel = React.forwardRef<
  React.ElementRef<typeof DropdownMenuPrimitive.Label>,
  React.ComponentPropsWithoutRef<typeof DropdownMenuPrimitive.Label> & {
    inset?: boolean;
  }
>(({ className, inset, ...props }, ref) => (
  <DropdownMenuPrimitive.Label
    ref={ref}
    className={cn(
      "px-2 py-1.5 text-sm font-semibold",
      inset && "pl-8",
      className
    )}
    {...props}
  />
));
DropdownMenuLabel.displayName = DropdownMenuPrimitive.Label.displayName;

const DropdownMenuSeparator = React.forwardRef<
  React.ElementRef<typeof DropdownMenuPrimitive.Separator>,
  React.ComponentPropsWithoutRef<typeof DropdownMenuPrimitive.Separator>
>(({ className, ...props }, ref) => (
  <DropdownMenuPrimitive.Separator
    ref={ref}
    className={cn("-mx-1 my-1 h-px bg-border", className)}
    {...props}
  />
));
DropdownMenuSeparator.displayName = DropdownMenuPrimitive.Separator.display
Name;

const DropdownMenuShortcut = ({
  className,
  ...props
}: React.HTMLAttributes<HTMLSpanElement>) => {
  return (
    <span
      className={cn("ml-auto text-xs tracking-widest opacity-60", className
)}
      {...props}
    />
  );
};
```

```
DropdownMenuShortcut.displayName = "DropdownMenuShortcut";

export {
  DropdownMenu,
  DropdownMenuTrigger,
  DropdownMenuContent,
  DropdownMenuItem,
  DropdownMenuCheckboxItem,
  DropdownMenuRadioItem,
  DropdownMenuLabel,
  DropdownMenuSeparator,
  DropdownMenuShortcut,
  DropdownMenuGroup,
  DropdownMenuPortal,
  DropdownMenuSub,
  DropdownMenuSubContent,
  DropdownMenuSubTrigger,
  DropdownMenuRadioGroup,
};
```

/components/ui/select.tsx

```
"use client";

import * as React from "react";
import * as SelectPrimitive from "@radix-ui/react-select";
import { Check, ChevronDown, ChevronUp } from "lucide-react";

import { cn } from "@/lib/utils";

const Select = SelectPrimitive.Root;

const SelectGroup = SelectPrimitive.Group;

const SelectValue = SelectPrimitive.Value;

const SelectTrigger = React.forwardRef<
  React.ElementRef<typeof SelectPrimitive.Trigger>,
  React.ComponentPropsWithoutRef<typeof SelectPrimitive.Trigger>
>(({ className, children, ...props }, ref) => (
  <SelectPrimitive.Trigger
    ref={ref}
    className={cn(
      "flex h-9 items-center justify-between whitespace-nowrap rounded-md b
order bg-transparent px-3 py-2 text-sm shadow-sm ring-offset-white placehol
der:text-muted-foreground focus:outline-none focus:ring-1 focus:ring-slate-
950 disabled:cursor-not-allowed disabled:opacity-50 [&>span]:line-clamp-1",
      className
    )}
    {...props}
```

```
  >
    {children}
    <SelectPrimitive.Icon asChild>
      <ChevronDown className="h-4 w-4 opacity-50" />
    </SelectPrimitive.Icon>
  </SelectPrimitive.Trigger>
));
SelectTrigger.displayName = SelectPrimitive.Trigger.displayName;

const SelectScrollUpButton = React.forwardRef<
  React.ElementRef<typeof SelectPrimitive.ScrollUpButton>,
  React.ComponentPropsWithoutRef<typeof SelectPrimitive.ScrollUpButton>
>(({ className, ...props }, ref) => (
  <SelectPrimitive.ScrollUpButton
    ref={ref}
    className={cn(
      "flex cursor-default items-center justify-center py-1",
      className
    )}
    {...props}
  >
    <ChevronUp className="h-4 w-4" />
  </SelectPrimitive.ScrollUpButton>
));
SelectScrollUpButton.displayName = SelectPrimitive.ScrollUpButton.displayNa
me;

const SelectScrollDownButton = React.forwardRef<
  React.ElementRef<typeof SelectPrimitive.ScrollDownButton>,
  React.ComponentPropsWithoutRef<typeof SelectPrimitive.ScrollDownButton>
>(({ className, ...props }, ref) => (
  <SelectPrimitive.ScrollDownButton
    ref={ref}
    className={cn(
      "flex cursor-default items-center justify-center py-1",
      className
    )}
    {...props}
  >
    <ChevronDown className="h-4 w-4" />
  </SelectPrimitive.ScrollDownButton>
));
SelectScrollDownButton.displayName =
  SelectPrimitive.ScrollDownButton.displayName;

const SelectContent = React.forwardRef<
  React.ElementRef<typeof SelectPrimitive.Content>,
  React.ComponentPropsWithoutRef<typeof SelectPrimitive.Content>
>(({ className, children, position = "popper", ...props }, ref) => (
  <SelectPrimitive.Portal>
    <SelectPrimitive.Content
      ref={ref}
      className={cn(
```

```
        "relative z-50 max-h-96 min-w-[8rem] overflow-hidden rounded-md bor
der bg-background text-foreground shadow-md data-[state=open]:animate-in da
ta-[state=closed]:animate-out data-[state=closed]:fade-out-0 data-[state=op
en]:fade-in-0 data-[state=closed]:zoom-out-95 data-[state=open]:zoom-in-95
data-[side=bottom]:slide-in-from-top-2 data-[side=left]:slide-in-from-right
-2 data-[side=right]:slide-in-from-left-2 data-[side=top]:slide-in-from-bot
tom-2",
        position === "popper" &&
          "data-[side=bottom]:translate-y-1 data-[side=left]:-translate-x-1
data-[side=right]:translate-x-1 data-[side=top]:-translate-y-1",
        className
      )}
      position={position}
      {...props}
    >
      <SelectScrollUpButton />
      <SelectPrimitive.Viewport
        className={cn(
          "p-1",
          position === "popper" &&
            "h-[var(--radix-select-trigger-height)] w-full min-w-[var(--rad
ix-select-trigger-width)]"
        )}
      >
        {children}
      </SelectPrimitive.Viewport>
      <SelectScrollDownButton />
    </SelectPrimitive.Content>
  </SelectPrimitive.Portal>
));
SelectContent.displayName = SelectPrimitive.Content.displayName;

const SelectLabel = React.forwardRef<
  React.ElementRef<typeof SelectPrimitive.Label>,
  React.ComponentPropsWithoutRef<typeof SelectPrimitive.Label>
>(({ className, ...props }, ref) => (
  <SelectPrimitive.Label
    ref={ref}
    className={cn("px-2 py-1.5 text-sm font-semibold", className)}
    {...props}
  />
));
SelectLabel.displayName = SelectPrimitive.Label.displayName;

const SelectItem = React.forwardRef<
  React.ElementRef<typeof SelectPrimitive.Item>,
  React.ComponentPropsWithoutRef<typeof SelectPrimitive.Item>
>(({ className, children, ...props }, ref) => (
  <SelectPrimitive.Item
    ref={ref}
    className={cn(
      "relative flex w-full hover:bg-accent cursor-default select-none item
s-center rounded-sm py-1.5 pl-2 pr-8 text-sm outline-none data-[disabled]:p
```

```
ointer-events-none data-[disabled]:opacity-50 ",
      className
    )}
    {...props}
  >
    <span className="absolute right-2 flex h-3.5 w-3.5 items-center justify
-center">
      <SelectPrimitive.ItemIndicator>
        <Check className="h-4 w-4" />
      </SelectPrimitive.ItemIndicator>
    </span>
    <SelectPrimitive.ItemText>{children}</SelectPrimitive.ItemText>
  </SelectPrimitive.Item>
));
SelectItem.displayName = SelectPrimitive.Item.displayName;

const SelectSeparator = React.forwardRef<
  React.ElementRef<typeof SelectPrimitive.Separator>,
  React.ComponentPropsWithoutRef<typeof SelectPrimitive.Separator>
>(({ className, ...props }, ref) => (
  <SelectPrimitive.Separator
    ref={ref}
    className={cn("-mx-1 my-1 h-px bg-background", className)}
    {...props}
  />
));
SelectSeparator.displayName = SelectPrimitive.Separator.displayName;

export {
  Select,
  SelectGroup,
  SelectValue,
  SelectTrigger,
  SelectContent,
  SelectLabel,
  SelectItem,
  SelectSeparator,
  SelectScrollUpButton,
  SelectScrollDownButton,
};
```

/components/ui/textarea.tsx

```
import * as React from "react";

import { cn } from "@/lib/utils";

const Textarea = React.forwardRef<
  HTMLTextAreaElement,
  React.ComponentProps<"textarea">
```

```
>(({ className, ...props }, ref) => {
  return (
    <textarea
      className={cn(
        "flex min-h-[60px] w-full rounded-md border bg-background px-3 py-2
text-base shadow-sm placeholder:text-muted-foreground focus-visible:outline
-none focus-visible:ring-1 focus-visible:ring-primary disabled:cursor-not-a
llowed disabled:opacity-50 md:text-sm",
        className
      )}
      ref={ref}
      {...props}
    />
  );
});
Textarea.displayName = "Textarea";

export { Textarea };
```

/components/ui/input.tsx

```
import * as React from "react";

import { cn } from "@/lib/utils";

const Input = React.forwardRef<HTMLInputElement, React.ComponentProps<"inpu
t">>(
  ({ className, type, ...props }, ref) => {
    return (
      <input
        type={type}
        className={cn(
          "flex h-9 w-full rounded-md border bg-background px-3 py-1 text-b
ase shadow-sm transition-colors file:border-0 file:bg-transparent file:text
-sm file:font-medium file:text-slate-950 placeholder:text-muted-foreground
focus-visible:outline-none focus-visible:ring-primary focus-visible:ring-1
disabled:cursor-not-allowed disabled:opacity-50 md:text-sm dark:file:text-s
late-50 dark:placeholder:text-muted-foreground",
          className
        )}
        ref={ref}
        {...props}
      />
    );
  }
);
Input.displayName = "Input";

export { Input };
```

/components/ui/form.tsx

```tsx
"use client";

import * as React from "react";
import * as LabelPrimitive from "@radix-ui/react-label";
import { Slot } from "@radix-ui/react-slot";
import {
  Controller,
  ControllerProps,
  FieldPath,
  FieldValues,
  FormProvider,
  useFormContext,
} from "react-hook-form";

import { cn } from "@/lib/utils";
import { Label } from "@/components/ui/label";

const Form = FormProvider;

type FormFieldContextValue<
  TFieldValues extends FieldValues = FieldValues,
  TName extends FieldPath<TFieldValues> = FieldPath<TFieldValues>
> = {
  name: TName;
};

const FormFieldContext = React.createContext<FormFieldContextValue>(
  {} as FormFieldContextValue
);

const FormField = <
  TFieldValues extends FieldValues = FieldValues,
  TName extends FieldPath<TFieldValues> = FieldPath<TFieldValues>
>({
  ...props
}: ControllerProps<TFieldValues, TName>) => {
  return (
    <FormFieldContext.Provider value={{ name: props.name }}>
      <Controller {...props} />
    </FormFieldContext.Provider>
  );
};

const useFormField = () => {
  const fieldContext = React.useContext(FormFieldContext);
  const itemContext = React.useContext(FormItemContext);
  const { getFieldState, formState } = useFormContext();
```

```
  const fieldState = getFieldState(fieldContext.name, formState);

  if (!fieldContext) {
    throw new Error("useFormField should be used within <FormField>");
  }

  const { id } = itemContext;

  return {
    id,
    name: fieldContext.name,
    formItemId: `${id}-form-item`,
    formDescriptionId: `${id}-form-item-description`,
    formMessageId: `${id}-form-item-message`,
    ...fieldState,
  };
};

type FormItemContextValue = {
  id: string;
};

const FormItemContext = React.createContext<FormItemContextValue>(
  {} as FormItemContextValue
);

const FormItem = React.forwardRef<
  HTMLDivElement,
  React.HTMLAttributes<HTMLDivElement>
>(({ className, ...props }, ref) => {
  const id = React.useId();

  return (
    <FormItemContext.Provider value={{ id }}>
      <div ref={ref} className={className} {...props} />
    </FormItemContext.Provider>
  );
});
FormItem.displayName = "FormItem";

const FormLabel = React.forwardRef<
  React.ElementRef<typeof LabelPrimitive.Root>,
  React.ComponentPropsWithoutRef<typeof LabelPrimitive.Root>
>(({ className, ...props }, ref) => {
  const { error, formItemId } = useFormField();

  return (
    <Label
      ref={ref}
      className={cn(error && "text-red-500 dark:text-red-900", className)}
      htmlFor={formItemId}
```

```
      {...props}
    />
  );
});
FormLabel.displayName = "FormLabel";

const FormControl = React.forwardRef<
  React.ElementRef<typeof Slot>,
  React.ComponentPropsWithoutRef<typeof Slot>
>(({ ...props }, ref) => {
  const { error, formItemId, formDescriptionId, formMessageId } =
    useFormField();

  return (
    <Slot
      ref={ref}
      id={formItemId}
      aria-describedby={
        !error
          ? `${formDescriptionId}`
          : `${formDescriptionId} ${formMessageId}`
      }
      aria-invalid={!!error}
      {...props}
    />
  );
});
FormControl.displayName = "FormControl";

const FormDescription = React.forwardRef<
  HTMLParagraphElement,
  React.HTMLAttributes<HTMLParagraphElement>
>(({ className, ...props }, ref) => {
  const { formDescriptionId } = useFormField();

  return (
    <p
      ref={ref}
      id={formDescriptionId}
      className={cn("text-[0.8rem] text-muted-foreground ", className)}
      {...props}
    />
  );
});
FormDescription.displayName = "FormDescription";

const FormMessage = React.forwardRef<
  HTMLParagraphElement,
  React.HTMLAttributes<HTMLParagraphElement>
>(({ className, children, ...props }, ref) => {
  const { error, formMessageId } = useFormField();
  const body = error ? String(error?.message) : children;
```

```
  if (!body) {
    return null;
  }

  return (
    <p
      ref={ref}
      id={formMessageId}
      className={cn(
        "text-[0.8rem] font-medium text-red-500 dark:text-red-900",
        className
      )}
      {...props}
    >
      {body}
    </p>
  );
});
FormMessage.displayName = "FormMessage";

export {
  useFormField,
  Form,
  FormItem,
  FormLabel,
  FormControl,
  FormDescription,
  FormMessage,
  FormField,
};
```

/components/contacts-page-skeleton.tsx

```
"use client";

import React from "react";
import ChildrenPanel from "./shared/children-panel";
import { ResizableHandle, ResizablePanel } from "./ui/resizable";
import { useLayout } from "@/contexts/use-layout";
import { useTranslation } from "react-i18next";

import { cn } from "@/lib/utils";
import { useIsMobile } from "@/hooks/use-mobile";
import { PageHeader } from "./headers";
import Skeleton from "react-loading-skeleton";
import "react-loading-skeleton/dist/skeleton.css";
import { Button } from "./ui/button";
import { ArrowLeft, Edit, Share, Trash2, X } from "lucide-react";
```

```
export default function ContactsPageSkeleton() {
  const { layout, fallbackLayout, amountIndicators } = useLayout();
  const { t } = useTranslation(["contacts-page", "common"]);
  const onMobile = useIsMobile();
  const selected = null;
  const skeletonsAmount: number =
    typeof amountIndicators?.contacts === "number"
      ? amountIndicators?.contacts
      : 4;
  return (
    <>
      <ResizablePanel
        className={cn(onMobile && selected !== null && "hidden")} // If we
are on mobile and a message is selected we only want to show the column con
taining the selected message.
        // Check if the layout is a 3-column middle-bar panel. Use the prev
ious 3-column layout if available; otherwise, render the fallback for diffe
rent or undefined layouts.
        defaultSize={
          Array.isArray(layout) && layout.length === 3
            ? layout[1]
            : fallbackLayout[1]
        }
        minSize={22}
        maxSize={50}
      >
        <PageHeader title={t(`header`)} />

        <div className="rounded-md p-4 h-[68px]">
          <Skeleton className="h-9" style={{ borderRadius: "0.375rem" }} />
        </div>

        <div className="flex flex-col gap-2 p-4 pt-0 mt-2 overflow-hidden">
          {skeletonsAmount > 0 ? (
            // Math.min() makes it so that the maximum will be x, even if t
he variable has a larger number
            Array.from({ length: Math.min(skeletonsAmount, 10) }).map(
              (_, i) => {
                return <ContactSkeleton key={i} />;
              }
            )
          ) : (
            <div className="p-8 text-center text-muted-foreground">
              <Skeleton className="w-full" />
            </div>
          )}
        </div>
      </ResizablePanel>
      <ResizableHandle withHandle className={cn(onMobile && "hidden")} />
      <ChildrenPanel
        hasMiddleBar
        className={cn(onMobile && selected === null && "hidden")} // like a
bove we are using reverse logic here. If we are on mobile, and nothing is s
```

```
elected, this component should not be displayed.
      >
        <ContactDisplaySkeleton />
      </ChildrenPanel>
    </>
  );
}

function ContactSkeleton() {
  return (
    <div
      className={cn(
        "flex contacts-start items-center gap-2 rounded-lg border p-3 text-
left text-sm transition-all"
      )}
    >
      <Skeleton circle width={48} height={48} />
      <div className="w-1/2 space-y-1">
        <div className="w-1/4 font-semibold">
          <Skeleton />
        </div>
        <div className="w-2/3 text-xs font-medium">
          <Skeleton />
        </div>
      </div>
    </div>
  );
}

function ContactDisplaySkeleton() {
  const onMobile = useIsMobile();
  const { t } = useTranslation(["contacts-page"]);

  return (
    <div className={cn("flex h-full flex-col")}>
      <div className="flex items-center p-2 h-[var(--header-height)] border
-b">
        <div className="flex items-center gap-2">
          {onMobile && (
            <Button variant="ghost" size="icon">
              <ArrowLeft className="h-4 w-4" />
              <span className="sr-only">{t("common:go_back")}</span>
            </Button>
          )}

          <Button variant="ghost" size="icon" disabled>
            <Trash2 className="h-4 w-4" />
            <span className="sr-only">{t("common:delete_permanently")}</spa
n>
          </Button>

          <Button variant="ghost" size="icon" disabled>
```

```
            <Edit className="h-4 w-4" />
            <span className="sr-only">{t("common:edit")}</span>
          </Button>
        </div>
        <div className="ml-auto flex items-center gap-2">
          <Button variant="ghost" size="icon" disabled>
            <X className="h-4 w-4" />
            <span className="sr-only">{t("common:close")}</span>
          </Button>
        </div>
      </div>

      <div className="p-8 text-center text-muted-foreground">
        {t("none_selected")}
      </div>
    </div>
  );
}
```

/components/recipients-input.tsx

```
"use client";

import { Input } from "./shared/input";
import React, {
  useState,
  useRef,
  useEffect,
  type KeyboardEvent,
  type ChangeEvent,
} from "react";
import { UserPlus, X } from "lucide-react";

import { Button } from "./ui/button";
import { cn } from "@/lib/utils";
import { useNewMessage } from "@/contexts/use-new-message";
import { useModal } from "@/contexts/use-modal";
import { ScrollArea } from "./ui/scroll-area";
import { NewRecipient } from "@/types/recipient";
import ProfilePic from "./profile-pic";
import { useTranslation } from "react-i18next";
import {
  Tooltip,
  TooltipContent,
  TooltipProvider,
  TooltipTrigger,
} from "./ui/tooltip";

const OFF_FOCUSED_RECIPIENT_AMOUNT = 5;
React.memo(RecipientsInput);
```

```
export default function RecipientsInput({
  error,
  onFocus,
  onBlur,
}: {
  error: boolean;
  onFocus: () => void;
  onBlur: () => void;
}) {
  const container = useRef<HTMLDivElement | null>(null);
  const [isDropdownOpen, setIsDropdownOpen] = useState(false);
  const inputElement = useRef<HTMLInputElement | null>(null);

  const {
    message,
    setMessage,
    recipients,
    addRecipient,
    removeRecipient,
    suggestedRecipients,
    searchRecipients,
    showInfoAbout,
    focusedInput,

    // Which one in the suggested recipients/contacts is currently selected
. You can change the selection with up and down arrow keys.
    selectedPhone,
    updateSelectedPhone,
  } = useNewMessage();
  const { setModal } = useModal();
  const { t } = useTranslation(["new-message-page"]);

  // reset the input's value
  function clearInputValue() {
    setMessage((m) => ({
      ...m,
      recipientInput: {
        ...m.recipientInput,
        value: "",
      },
    }));
  }

  const handleKeyDown = (e: KeyboardEvent<HTMLInputElement>) => {
    setTimeout(() => {
      if (container.current) {
        // automatically scroll to the bottom of the recipients container w
hen user starts typing
        container.current.scrollTop += container.current.scrollHeight;
      }
    }, 0);
```

```
    const trimmedInput = message.recipientInput.value.trim();
    if (e.key === "Enter" || e.key === "Tab") {
      e.preventDefault();
      e.stopPropagation();
      if (selectedPhone) {
        addRecipient(selectedPhone);
        clearInputValue();
      } else if (trimmedInput !== "") {
        addRecipient(trimmedInput);

        clearInputValue();
      }
    } else if (e.key === "ArrowDown" || e.key === "ArrowUp") {
      updateSelectedPhone(e.key);
    }

    if (e.key === "Backspace" && trimmedInput === "" && recipients.length)
{
      const lastRecipientIndex = recipients.length - 1;
      const lastRecipient = recipients[lastRecipientIndex];
      if (lastRecipient && lastRecipient.proneForDeletion) {
        // Remove the last recipient if it is already prone for deletion
        removeRecipient(lastRecipient);
      } else {
        setMessage((prev) => {
          const lastRecipientIndex = prev.recipients.length - 1;

          // Create a new array of recipients with the last recipient marke
d as prone for deletion
          const newRecipients = prev.recipients.map((recipient, index) => {
            if (index === lastRecipientIndex) {
              return { ...recipient, proneForDeletion: true }; // Mark as p
rone for deletion
            }
            return recipient; // Return the other recipients unchanged
          });

          // Return the new state with the last recipient marked as prone f
or deletion
          return { ...prev, recipients: newRecipients };
        });
      }
    }
  };

  const onInputChange = (e: ChangeEvent<HTMLInputElement>) => {
    const value = e.target.value;
    setMessage((m) => ({
      ...m,
      recipientInput: {
        ...m.recipientInput,
        value,
```

```
      },
    }));
    setIsDropdownOpen(true);

    searchRecipients(value);
  };

  const showRecipientInfo = (recipient: NewRecipient) => {
    showInfoAbout(recipient);
    setModal((m) => ({ ...m, contact: { ...m.contact, info: true } }));
  };

  useEffect(() => {
    // automatically collapse the expanded recipients when another input ge
ts selected
    if (focusedInput !== "new-recipient" && typeof focusedInput == "string"
) {
      setMessage((prev) => ({
        ...prev,
        recipientInput: { ...prev.recipientInput, recipientsExpanded: false
},
      }));
    }
  }, [focusedInput]);
  return (
    <div className="flex-1 py-1 relative z-[1000]">
      <div className="max-h-24 overflow-auto" ref={container}>
        <div
          className={cn(
            "w-full min-h-[2.75rem] flex flex-wrap items-center gap-x-1 py-
1 h-full border-b px-5 z-50",
            focusedInput === "new-recipient" && "border-primary",
            error && "border-red-500"
          )}
        >
          <span className="my-0.5 mr-0.5 px-0 flex items-center text-sm tex
t-muted-foreground">
            {t("common:to")}
          </span>
          {/* Recipient chips */}
          {recipients.map((recipient, index) => {
            // Since we have so many recipients, only some should be shown
until the user clicks to see the rest
            if (
              index >= OFF_FOCUSED_RECIPIENT_AMOUNT &&
              message.recipientInput.recipientsExpanded === false
            ) {
              return;
            }
            // else, we show all of them
            return (
              <div
                key={recipient.phone}
```

```
                // Height of the row/container
                className="flex items-center h-7"
              >
                <div
                  // Height of the contact chip itself
                  className={cn("h-6")}
                >
                  <TooltipProvider delayDuration={1000}>
                    <Tooltip>
                      <TooltipTrigger asChild>
                        <div
                          className={cn(
                            "bg-background flex items-center text-xs border
rounded-xl hover:bg-muted dark:hover:bg-muted cursor-pointer whitespace-now
rap h-full hover:shadow-none",
                            error && "error-border-pulse",
                            recipient.proneForDeletion && "border-destructi
ve",
                            !recipient.isValid &&
                              "bg-red-100/70 dark:bg-red-900/50",
                            recipient.error?.type === "warning" &&
                              "bg-yellow-50 dark:bg-yellow-400/40"
                          )}
                        >
                          <div
                            onClick={() => showRecipientInfo(recipient)}
                            className="h-full flex items-center rounded-l-x
l pl-1.5"
                          >
                            {recipient?.contact?.name || recipient.phone}
                          </div>

                          <Button
                            variant="none"
                            className="h-full py-0 px-1.5 cursor-pointer cl
oseX rounded-l-none rounded-r-xl"
                            onClick={() => removeRecipient(recipient)}
                            type="button"
                          >
                            <X className="h-4 w-4 text-muted-foreground" />
                          </Button>
                        </div>
                      </TooltipTrigger>
                      <TooltipContent>
                        {t(
                          recipient.error?.message
                            ? recipient.error?.message
                            : ""
                        ) || t("tooltip-more_info")}
                      </TooltipContent>
                    </Tooltip>
                  </TooltipProvider>
                </div>
```

```
            </div>
          );
        })}

        {/* Button to show all recipients, when it's clicked we also focu
s the input */}
        {message.recipients.length > OFF_FOCUSED_RECIPIENT_AMOUNT &&
        message.recipientInput.recipientsExpanded === false ? (
          <Button
            variant="none"
            className="p-0 ml-2"
            type="button"
            onClick={() => {
              setMessage((prev) => ({
                ...prev,
                recipientInput: {
                  ...prev.recipientInput,
                  recipientsExpanded: true,
                },
              }));
              setTimeout(() => {
                if (inputElement.current) {
                  inputElement.current.focus();
                }
              }, 0);
            }}
          >
            {t("x_more", {
              x: message.recipients.length - OFF_FOCUSED_RECIPIENT_AMOUNT
,
            })}
          </Button>
        ) : (
          <></>
        )}

        <div
          className={cn(
            "h-7 min-w-[200px] flex-1 py-1 ml-3", // my-0
            message.recipients.length > OFF_FOCUSED_RECIPIENT_AMOUNT &&
              message.recipientInput.recipientsExpanded === false &&
              "hidden"
          )} /* we are taking advantage of the default positioning of abs
olute elements this common parent div */
        >
          <Input
            ref={inputElement}
            // this name only used for the focus state, not for submittin
g any value
            name="new-recipient"
            className={cn(
              "h-min text-sm w-full p-0 ring-0 focus:ring-0 shadow-none r
ounded-none placeholder:text-muted-foreground" //my-0
```

```
              )}
              placeholder={
                message.recipients.length <= OFF_FOCUSED_RECIPIENT_AMOUNT |
|
                message.recipientInput.recipientsExpanded
                  ? t("common:phone_number")
                  : ""
              }
              value={message.recipientInput.value}
              onChange={onInputChange}
              onKeyDown={handleKeyDown}
              onFocus={() => {
                setIsDropdownOpen(true);

                searchRecipients(message.recipientInput.value);
                onFocus();
              }}
              onBlur={() => {
                setMessage((m) => ({
                  ...m,
                  recipients: m.recipients.map((r) => ({
                    ...r,
                    proneForDeletion: false,
                  })),
                }));
                setIsDropdownOpen(false);

                // Create recipient from input value on blur if not empty
                if (message.recipientInput.value.trim() !== "") {
                  addRecipient(message.recipientInput.value);
                }

                onBlur();
              }}
            />

            {/* Begin suggested recipients dropdown */}
            {isDropdownOpen && suggestedRecipients.length !== 0 && (
              <div className="absolute top-[85%] bg-background rounded-lg b
order shadow-md dark:shadow-lg-light">
                <ScrollArea className="w-[230px] xs:w-[300px] h-[330px]">
                  <div
                    className="p-2" /* this is necessary to have a separate
container so that the items scroll all the way up to the end of the contain
er */
                  >
                    <h3 className="mb-2 px-2 text-sm font-medium">
                      {!message.recipientInput.value.length
                        ? t("suggestions")
                        : t("x_found", { x: suggestedRecipients.length })}
                    </h3>
                    <div className="flex flex-col gap-1">
```

```
                    {suggestedRecipients.map((recipient) => (
                      <button
                        key={recipient.phone}
                        className={cn(
                          "flex items-center w-full gap-2 rounded-lg bord
er p-3 text-left text-sm transition-all hover:bg-accent",
                          selectedPhone === recipient.phone &&
                            "border-primary"
                        )}
                        type="button"
                        onMouseDown={(e) => {
                          e.preventDefault();

                          addRecipient(recipient.phone);
                        }}
                      >
                        <ProfilePic
                          name={recipient.contact?.name || undefined}
                          size={10}
                          className="border"
                        />
                        <div className="space-y-1">
                          <div className="font-semibold">
                            {recipient.contact?.name || recipient.phone}
                          </div>
                          <div className="text-xs font-medium">
                            {recipient.contact?.name ? recipient.phone :
""}
                          </div>
                        </div>
                      </button>
                    ))}
                  </div>
                </div>
              </ScrollArea>
            </div>
          )}
        </div>
      </div>
      <Button
        className="absolute right-2 bottom-[6px] p-2 aspect-1 top-1/2 -tr
anslate-y-1/2 z-10"
        variant="ghost"
        type="button"
        onClick={() =>
          setModal((m) => ({ ...m, contact: { ...m.contact, insert: true
} }))
        }
      >
        <UserPlus className="h-1 w-1" />
      </Button>
    </div>
  </div>
```

```
  );
}
```

/components/contacts-list.tsx

```tsx
"use client";

import { cn } from "@/lib/utils";
import { ScrollArea } from "@/components/ui/scroll-area";
import ProfilePic from "./profile-pic";
import type { DBContact } from "@/types/contact";
import { useIsMobile } from "@/hooks/use-mobile";
import { Button } from "./ui/button";

type ContactListProps = {
  contacts: DBContact[];
  selectedContactId: string | null;
  setSelected: (contact: DBContact) => void;
};

export default function ContactsList({
  contacts,
  selectedContactId,
  setSelected,
}: ContactListProps) {
  const onMobile = useIsMobile();
  return (
    <ScrollArea
      className={
        onMobile
          ? `h-[calc(100vh-var(--simple-header-height)-68px)]`
          : `h-[calc(100vh-var(--header-height)-68px)]`
      }
    >
      <div className="flex flex-col gap-2 p-4 pt-0">
        {contacts.map((contact) => (
          <Button
            key={contact.id}
            variant="ghost"
            className={cn(
              "h-full flex items-center justify-start gap-2 rounded-lg bord
er p-3 text-left mt-[1px]",
              selectedContactId === contact.id && "bg-accent"
            )}
            onClick={() => setSelected(contact)}
          >
            <ProfilePic name={contact.name} size={10} className="border" />
            <div className="space-y-1">
              <div className="font-semibold">{contact.name}</div>
              <div className="text-xs font-medium">{contact.phone}</div>
```

```
            </div>
          </Button>
        ))}
      </div>
    </ScrollArea>
  );
}
```

/components/403.tsx

```
"use client";

import React from "react";
import ErrorComponent from "./shared/error-component";
import { useTranslation } from "react-i18next";
import Link from "next/link";
import { buttonVariants } from "./ui/button";

export default function UnauthorizedPage() {
  const { t } = useTranslation(["errors", "common"]);
  return (
    <ErrorComponent
      title={t("403_error-header")}
      subtitle={t("403_error-header_caption")}
    >
      <Link href="/sent" className={buttonVariants({ variant: "default" })}
>
        {t("common:go_back")}
      </Link>
    </ErrorComponent>
  );
}
```

/components/new-message-form.tsx

```
"use client";

import { Separator } from "./ui/separator";
import { Textarea } from "./ui/textarea";
import {
  Check,
  FileCheck,
  Loader2,
  Maximize2,
  Minimize2,
  Save,
  Trash2,
  X,
```

```
} from "lucide-react";
import SendButton from "./send-button";
import { capitalize, cn, toastActionResult } from "@/lib/utils";
import { useTranslation } from "react-i18next";
import { PageHeader } from "./headers";
import { sendMessage } from "@/lib/actions/message.create";
import {
  Tooltip,
  TooltipContent,
  TooltipTrigger,
} from "@/components/ui/tooltip";

// Form
import { Button, buttonVariants } from "@/components/ui/button";
import { Input } from "@/components/ui/input";
import React, {
  ChangeEvent,
  useCallback,
  useEffect,
  useRef,
  useState,
} from "react";

import RecipientsInput from "./recipients-input";
import { useNewMessage } from "@/contexts/use-new-message";
import { usePathname, useRouter, useSearchParams } from "next/navigation";
import {
  Select,
  SelectContent,
  SelectItem,
  SelectTrigger,
  SelectValue,
} from "@/components/ui/select";
import { toast } from "sonner";
import { NewRecipient } from "@/types/recipient";

import { useLayout } from "@/contexts/use-layout";
import type { DBMessage, Message } from "@/types";
import { ActionResponse } from "@/types/action";
import { deleteMessage, saveDraft } from "@/lib/actions/message.actions";
import useDebounce from "@/hooks/use-debounce";
import useIsMounted from "@/hooks/use-mounted";
import { format } from "date-fns";
import { useIsMobile } from "@/hooks/use-mobile";
import { EMPTY_MESSAGE, PT_DATE_FORMAT } from "@/global.config";
import { useModal } from "@/contexts/use-modal";

// apparently, when something gets revalidated or the url gets updated, thi
s component gets re-rendered, while the new-message-context keeps it's stat
e
const NewMessageForm = React.memo(function ({
  message_id,
```

```
}: {
  message_id?: DBMessage;
}) {
  const formRef = useRef<HTMLFormElement>(null);
  const { t } = useTranslation(["new-message-page"]);
  const router = useRouter();
  const {
    recipients,
    setMessage,
    message,
    focusedInput,
    setFocusedInput,
    form,
    setForm,
    draft,
    setDraft,
  } = useNewMessage();
  const { setModal } = useModal();
  const [loading, setLoading] = useState(false);
  const { isFullscreen, setIsFullscreen } = useLayout();
  const pathname = usePathname();
  const onMobile = useIsMobile();

  const isMounted = useIsMounted();
  const debouncedSaveDraft = useDebounce(message, 2000);
  const previousDraftRef = useRef(message);
  const searchParams = useSearchParams();

  // When the controlled inputs value changes, we update the state
  const handleInputChange = (
    e: ChangeEvent<HTMLInputElement | HTMLTextAreaElement>
  ) => {
    const { name, value } = e.target;
    setMessage((prev) => ({ ...prev, [name]: value }));
  };

  const handleSubmit = async (e: React.FormEvent<HTMLFormElement>) => {
    e.preventDefault();

    // Smaller than (<) means it is in the past, while larger than (>) mean
s in the future
    if (
      message.scheduledDateModified &&
      message.scheduledDateConfirmed === false &&
      message.scheduledDate.getTime() < Date.now()
    ) {
      // Prevent the rest of the code of getting executed if the invalid da
te has not been confirmed yet.
      return setModal((m) => ({ ...m, scheduleAlert: true }));
    }

    setLoading(true);
```

```
    setMessage((m) => ({ ...m, scheduledDateConfirmed: false }));

    const formData = new FormData(e.currentTarget);
    const result = await sendMessage(draft.id, {
      sender: /*formData.get("sender") as string */ "ETPZP",
      recipients: recipients as NewRecipient[],
      subject: formData.get("subject") as string,
      body: formData.get("body") as string,
      secondsUntilSend:
        message.scheduledDate.getTime() > new Date().getTime()
          ? (Math.floor(
              (message.scheduledDate.getTime() - Date.now()) / 1000
            ) as number)
          : undefined,
    });

    setLoading(false);

    // Update the message context with the result errors, so that they can
be persisted between draft re-renders
    setMessage((m) => ({
      ...m,
      serverStateErrors: result.errors,
      invalidRecipients: result.invalidRecipients,
    }));

    if (result.success) {
      // Message got sent successfully
      if (result.sendDate) {
        toast.success(
          `${t(result.message[0])} ${format(result.sendDate, PT_DATE_FORMAT
)}`
        );
      } else {
        toastActionResult(result, t);
      }
    } else {
      // Unable to send message due to an error:
      // 1. Display input specific error messages
      const zodErrors = result.errors || {};
      let waitTime = 0;
      const inBetweenTime = 300;
      Object.entries(zodErrors).forEach(
        ([input, errorArray], index) =>
          setTimeout(() => {
            toast.error(capitalize(input), {
              description: errorArray.map((error) => t(error)).join(", "),
            });
            waitTime += index * inBetweenTime;
          }, index * inBetweenTime) // Increase delay by 50ms for each erro
r
      );
```

```
    // 2. Display general error message
    setTimeout(() => {
      if (result.invalidRecipients) {
        toast.error(
          `${t(result.message)} ${result.invalidRecipients
            .map((r) => r.phone)
            .join(", ")}`
        );
      } else {
        toastActionResult(result, t);
      }
    }, Object.entries(zodErrors).length * inBetweenTime);
  }

  if (result.clearForm === true) {
    // 3. Reset the form
    setMessage(EMPTY_MESSAGE); // technically this isn't even needed
    router.push("/new-message");
  }
};

// When the user pressed discard at the bottom
const discardDraft = async () => {
  if (draft.id) {
    // Drafts should also be discarded (deleted) immediately
    const result: ActionResponse<null> = await deleteMessage(draft.id);
    toastActionResult(result, t);
  }

  // The navigation already re-fetches the amount indicators
  router.push("/sent");
};

function messageIsEmpty() {
  return (
    !message.body &&
    !message.subject &&
    !message.recipients.length &&
    message.sender === "ETPZP"
  );
}

// Draft saving logic
const handleSaveDraft = () => {
  const save = async () => {
    if (
      JSON.stringify(debouncedSaveDraft) !==
      JSON.stringify(previousDraftRef.current)
    ) {
      setDraft((prev) => ({ ...prev, pending: true }));
      const { draftId } = await saveDraft(draft.id || undefined, message)
```

```
;
        setDraft((prev) => ({ ...prev, pending: false }));

        if (draftId) {
          setDraft((prev) => ({
            ...prev,
            id: draftId || null,
            lastSaveSuccessful: true,
          }));
          // Updating the URL revalidates the server (including fetching am
ount indicators) and re-renders the component.
          const params = new URLSearchParams(searchParams.toString());
          params.set("message_id", draftId);
          router.replace(pathname + "?" + params.toString());
        } else {
          setDraft((prev) => ({ ...prev, lastSaveSuccessful: false }));
        }
      }
    };

    // Empty drafts should be deleted from db
    const discard = async () => {
      if (draft.id) {
        await deleteMessage(draft.id);

        // Updating the URL revalidates the server (including fetching amou
nt indicators) and re-renders the component.
        const params = new URLSearchParams(searchParams.toString());
        params.delete("message_id");
        router.replace(pathname + "?" + params.toString());
      }
    };

    if (messageIsEmpty()) {
      // Delete the old draft
      discard();
    } else {
      save();
    }
  };
  useEffect(() => {
    if (!isMounted) return;
    handleSaveDraft();
  }, [debouncedSaveDraft]);
  useEffect(() => {
    // Reapply input focus state - sender focusing logic not needed as it i
s a <Select>.
    if (focusedInput) {
      const inputElement = document.querySelector(
        `[name="${focusedInput}"]`
      ) as HTMLElement;
```

```
      // Move cursor to end of textarea to prevent default behavior of plac
ing it at the beginning.
      if (inputElement) {
        if (
          focusedInput === "body" &&
          inputElement instanceof HTMLTextAreaElement
        ) {
          // For textarea, set cursor at the end
          inputElement.focus();
          inputElement.setSelectionRange(
            inputElement.value.length,
            inputElement.value.length
          );
        } else {
          inputElement.focus();
        }
      }
    }
  }, [focusedInput]);

  useEffect(() => {
    if (formRef.current) {
      setForm(formRef.current);
    }
  }, [formRef]);
  useEffect(() => {
    if (isMounted) {
      setMessage((m) => ({ ...m, draft: { id: message_id?.id || null } }));
    }
  }, [isMounted]);
  return (
    <>
      <PageHeader
        title={
          message.subject
            ? message.subject.length > (onMobile ? 22 : 60)
              ? message.subject.substring(0, (onMobile ? 22 : 60) - 3) + ".
.."
              : message.subject
            : t("header")
        }
      >
        <Tooltip>
          <TooltipTrigger asChild>
            <Button
              variant="ghost"
              size="icon"
              type="button"
              onClick={() => {
                // We only disable it if the message is empty so we need mo
re checks here to prevent the user from clicking the button over and over
                if (draft.pending || messageIsEmpty()) return; // not empty
and not pending means it is saved
```

```
              handleSaveDraft();
            }}
            // disabled={messageIsEmpty()}
          >
            {draft.pending ? (
              <Loader2 className="animate-spin" />
            ) : messageIsEmpty() || !draft.lastSaveSuccessful ? (
              // draft is pending either it is saved or not
              // this comes first because we don't want to show the resul
t if message is empty
              <Save className="w-4 h-4" />
            ) : (
              <FileCheck className="h-4 w-4" />
            )}
          </Button>
        </TooltipTrigger>
        <TooltipContent>
          {draft.pending
            ? t("draft_btn-saving")
            : messageIsEmpty() || !draft.lastSaveSuccessful
            ? // draft is pending either it is saved or not
              // this comes first because we don't want to show the resul
t if message is empty
              t("draft_btn-save")
            : t("draft_btn-saved")}
        </TooltipContent>
      </Tooltip>
      {!onMobile && (
        <>
          <Tooltip>
            <TooltipTrigger asChild>
              <Button
                variant="ghost"
                size="icon"
                onClick={() =>
                  setIsFullscreen((prevFullscreen) => !prevFullscreen)
                }
              >
                {isFullscreen ? (
                  <Minimize2 className="h-4 w-4" />
                ) : (
                  <Maximize2 className="h-4 w-4" />
                )}
              </Button>
            </TooltipTrigger>
            <TooltipContent>{t("toggle_fullscreen")}</TooltipContent>
          </Tooltip>

          <Tooltip>
            <TooltipTrigger asChild>
              <Button
                variant="ghost"
                className={cn(
```

```
                  buttonVariants({ variant: "ghost" }),
                  "aspect-1 p-0"
                )}
                onClick={() => {
                  setIsFullscreen(false);
                  router.push("/sent");
                }}
              >
                <X className="h-4 w-4" />
              </Button>
            </TooltipTrigger>
            <TooltipContent>{t("common:close")}</TooltipContent>
          </Tooltip>
        </>
      )}
    </PageHeader>
    <form
      ref={formRef}
      onSubmit={handleSubmit}
      className="h-screen flex flex-col"
    >
      <div
        className={cn(
          "flex flex-col",
          isFullscreen || onMobile
            ? "h-[calc(100vh-var(--simple-header-height))]"
            : "h-[calc(100vh-var(--header-height))]"
        )}
      >
        <div className="flex flex-col px-4 mt-2">
          <div
            className={cn(
              "border-b focus-within:border-primary",
              message.serverStateErrors?.sender && "border-red-500"
            )}
          >
            <Select
              name="sender"
              defaultValue={/**message_id?.sender || */ "ETPZP"}
              onValueChange={(value) => {
                setMessage((prev) => ({ ...prev, sender: value }));
              }}
              disabled
            >
              {/** It defaults to the first SelectItem */}
              <SelectTrigger className="w-full rounded-none border-none s
hadow-none focus:ring-0 px-5 py-1 h-11">
                <SelectValue placeholder="ETPZP" />
              </SelectTrigger>
              <SelectContent>
                <SelectItem value="ETPZP">ETPZP</SelectItem>
                <SelectItem value="Test">Test</SelectItem>
              </SelectContent>
```

```
              </Select>
            </div>

            <RecipientsInput
              // Instead of a Zod error, we receive an invalid recipients a
rray for recipient errors.
              error={!!message.invalidRecipients?.length}
              onFocus={() => setFocusedInput("new-recipient")}
              onBlur={() => setFocusedInput(null)}
            />

            <Input
              name="subject"
              placeholder={t("subject_placeholder")}
              className={cn(
                "new-message-input focus-visible:ring-0 placeholder:text-mu
ted-foreground border-b  focus:border-primary"
              )}
              onChange={handleInputChange}
              value={message?.subject || EMPTY_MESSAGE.subject}
              onFocus={() => setFocusedInput("subject")}
              onBlur={() => setFocusedInput(null)}
            />
          </div>
          <div className="px-4 flex-grow mt-[1.25rem] mb-2">
            <Textarea
              name="body"
              className={cn(
                "border-none rounded-none h-full p-0 focus-visible:ring-0 s
hadow-none resize-none placeholder:text-muted-foreground",
                message.serverStateErrors?.body &&
                  "ring-red-500 placeholder:text-red-400 dark:placeholder:t
ext-red-400"
              )}
              placeholder={
                message.serverStateErrors?.body
                  ? t(message.serverStateErrors?.body[0])
                  : t("body_placeholder")
              }
              onChange={handleInputChange}
              value={message?.body || EMPTY_MESSAGE.body}
              onFocus={() => setFocusedInput("body")}
              onBlur={() => setFocusedInput(null)}
            />
          </div>

          <Separator />
          <div className="flex px-4 py-2 justify-end gap-2">
            <Button
              variant="secondary"
              type="button"
              className="w-max"
              onClick={discardDraft}
```

```
            >
              <Trash2 className="h-4 w-4" />
              <span className="hidden xs:inline">{t("discard")}</span>
            </Button>

            <SendButton loading={loading} />
          </div>
        </div>
      </form>
      {/* <UnloadListener /> */}
    </>
  );
});

export default NewMessageForm;
```

/components/headers.tsx

```
"use client";

import { Separator } from "@/components/ui/separator";
import { useIsMobile } from "@/hooks/use-mobile";
import { ArrowLeft, Menu } from "lucide-react";
import { Button, buttonVariants } from "./ui/button";
import { useLayout } from "@/contexts/use-layout";
import Link from "next/link";
import Skeleton from "react-loading-skeleton";
import { cn } from "@/lib/utils";
import Account from "./shared/account";
import { usePathname } from "next/navigation";
import { useTranslation } from "react-i18next";

type PageHeaderProps = {
  title?: string;
  skeleton?: boolean;
  marginRight?: boolean;
  className?: string;
  children?: React.ReactNode;
};

export function PageHeader({
  title,
  skeleton,
  marginRight = true,
  className,
  children,
}: PageHeaderProps) {
  const onMobile = useIsMobile();
  const pathname = usePathname();
```

```
  return (
    <>
      <div
        className={cn(
          "flex items-center gap-2 px-4 h-[var(--simple-header-height)]",
          title && "border-b",
          className
        )}
      >
        <div className="shrink flex items-center min-w-min whitespace-nowra
p">
          {onMobile &&
            (pathname.includes("/dashboard") ? (
              <Link
                href="/"
                className={buttonVariants({ variant: "ghost", size: "icon"
})}
              >
                <ArrowLeft className="w-4 h-4" />
              </Link>
            ) : (
              <MobileHamburgerButton className="mr-2" />
            ))}
          {skeleton ? (
            <Skeleton
              // Consider to set this to 158 later which is the width of `N
ew Message` title
              width=""
              height={28}
              containerClassName="mr-auto w-[30%]"
            />
          ) : (
            <h2 className={marginRight ? "mr-auto" : ""}>{title}</h2>
          )}
        </div>
        <div className="grow flex items-center gap-2 justify-end">
          {children}
          {onMobile && <Account profilePicPosition="RIGHT" hideNameRole />}
        </div>
      </div>
    </>
  );
}

type SectionHeaderProps = {
  title: string;
  subtitle: string;
  children?: React.ReactNode;
  anchorName: string;
};
export function SectionHeader({
  title,
  subtitle,
```

```
  children,
  anchorName,
}: SectionHeaderProps) {
  return (
    <div>
      <Link href={`#${anchorName}`} className="mr-auto">
        <h3 id={anchorName}>{title}</h3>
      </Link>
      <p className="subtitle">{subtitle}</p>
      <Separator className="mt-5 mb-3 lg:max-w-2xl" />
      <div className="space-y-5 px-5">{children}</div>
    </div>
  );
}

export function MobileHamburgerButton({ className }: { className: string })
{
  const { setMobileNavPanel } = useLayout();
  return (
    <Button
      variant="ghost"
      size="icon"
      className={cn("md:hidden", className)}
      type="button"
      onClick={() => setMobileNavPanel(true)}
    >
      <Menu className="h-5 w-5" />
      <span className="sr-only">Toggle mobile menu</span>
    </Button>
  );
}
```

/components/messages-page.tsx

```
"use client";

import { DBMessage, CategoryEnums } from "@/types";
import React, { useEffect, useState } from "react";
import ChildrenPanel from "./shared/children-panel";
import { ResizableHandle, ResizablePanel } from "./ui/resizable";
import { useLayout } from "@/contexts/use-layout";
import { PageHeader } from "./headers";
import { useTranslation } from "react-i18next";
import { MessageList } from "./messages-list";

import { cn, searchMessages } from "@/lib/utils";
import MessageDisplay from "./message-display";
import { useIsMobile } from "@/hooks/use-mobile";
import Search from "./shared/search";
import { useSearchParams } from "next/navigation";
```

```
import useIsMounted from "@/hooks/use-mounted";
import { ModalProvider } from "@/contexts/use-modal";

export default function MessagesPage({
  messages,
  error,
  category,
}: Readonly<{
  messages: DBMessage[];
  error: boolean;
  category: CategoryEnums;
}>) {
  const { layout, fallbackLayout } = useLayout();
  const { t } = useTranslation(["messages-page", "common"]); // and more
  const [filteredMessages, setFilteredMessages] = useState(messages);
  const [selected, setSelected] = useState<DBMessage | null>(
    filteredMessages[0] || null
  );
  const isMounted = useIsMounted();
  const [isLarge, setIsLarge] = useState({
    bool: window.matchMedia("(min-width: 1024px)").matches,
    breakpoint: window.matchMedia("(min-width: 1024px)").matches ? 29 : 44,
  });
  const onMobile = useIsMobile();
  const searchParams = useSearchParams();
  const query = searchParams.get("query") || "";
  const currentPage = Number(searchParams.get("page")) || 1;

  // Update ui based on search term
  const onSearch = (searchTerm: string) => {
    setFilteredMessages(searchMessages(messages, searchTerm, currentPage));
  };

  useEffect(() => {
    // Filter the messages with URLsearchParams on page load
    setFilteredMessages(searchMessages(messages, query, currentPage));
    if (selected && messages.some((msg) => msg.id === selected.id)) {
      // Keep the current selection
      setSelected(selected);
    } else {
      // If the selected message is not in the new messages, set it to null
or handle accordingly
      setSelected(messages[0] || null);
    }
  }, [messages]);

  useEffect(() => {
    if (isMounted && onMobile) {
      // On mobile, it should show the list by default without having the f
irst one selected like on desktop.
      setSelected(null);
    }
```

```
  }, [isMounted]);

  return (
    <>
      <ResizablePanel
        className={cn(onMobile && selected !== null && "hidden")} // If we
are on mobile and a message is selected we only want to show the column con
taining the selected message.
        // Check if the layout is a 3-column middle-bar panel. Use the prev
ious 3-column layout if available; otherwise, render the fallback for diffe
rent or undefined layouts.
        defaultSize={
          Array.isArray(layout) && layout.length === 3
            ? layout[1]
            : fallbackLayout[1]
        }
        minSize={22}
        maxSize={50}
      >
        <PageHeader title={t(`header_${category.toLowerCase()}`)} />
        <Search
          onSearch={onSearch}
          placeholder={t(`search_${category.toLowerCase()}`)}
          className="pl-8 placeholder:text-muted-foreground border"
        />

        {filteredMessages.length > 0 ? (
          <MessageList
            messages={filteredMessages}
            selectedMessageId={selected?.id || null}
            setSelected={setSelected}
          />
        ) : (
          <div className="p-8 text-center text-muted-foreground">
            {error || t("none_found")}
          </div>
        )}
      </ResizablePanel>
      <ResizableHandle withHandle className={cn(onMobile && "hidden")} />

      <ChildrenPanel
        hasMiddleBar
        // reverse logic like above: on mobile and with nothing selected, t
his component should be hidden.
        className={cn(onMobile && selected === null && "hidden")}
      >
        {/* If you need other modals somewhere else, move the provider up t
he component tree. And don't forget to update the skeleton too! */}
        <ModalProvider>
          <MessageDisplay
            message={selected}
            reset={() => setSelected(null)}
            category={category}
```

```
          />
        </ModalProvider>
      </ChildrenPanel>
    </>
  );
}
```

/components/nav-links.tsx

```
"use client";

import Link from "next/link";
import { LucideIcon } from "lucide-react";
import { cn } from "@/lib/utils";
import { Button, buttonVariants } from "@/components/ui/button";
import {
  Tooltip,
  TooltipContent,
  TooltipTrigger,
} from "@/components/ui/tooltip";
import { usePathname } from "next/navigation";
import { useSettings } from "@/contexts/use-settings";
import { useTranslation } from "react-i18next";

type NavLink = {
  title: string;
  label?: string;
  icon: LucideIcon;
  href?: string;
  action?: () => void;
  variant: "default" | "ghost";
  size?: "sm" | "md" | "xl";
  hidden?: boolean;
  isNewButton?: boolean;
};
type NavProps = {
  isCollapsed: boolean;
  links: NavLink[];
  onMobile?: boolean;
};

export default function NavLinks({ links, isCollapsed, onMobile }: NavProps
) {
  const pathname = usePathname();
  const { i18n } = useTranslation();
  const { normalizePath } = useSettings();

  const activeStyles =
    "bg-accent text-primary-accent hover:bg-accent hover:text-accent-foregr
ound";
```

```
  // isActive takes Link, compares it to the current url, and returns wheth
er it is the same link we are on or not.
  const isActive = (href: string, isNewButton: boolean | undefined) => {
    return !isNewButton && normalizePath(href) === normalizePath(pathname);
  };
  return (
    <div
      data-collapsed={isCollapsed}
      className={cn(
        `group flex flex-col gap-4 py-2 data-[collapsed=true]:py-2`,
        onMobile && "w-[250px]"
      )}
    >
      <nav className="grid gap-1 px-2 group-[data-collapsed=true](data-coll
apsed=true):justify-center group-[data-collapsed=true](data-collapsed=true)
:px-2">
        {links.map((link, index) => {
          const desktopItemClassName = cn(
            buttonVariants({
              variant: link.variant,
              size: link.isNewButton ? "lg" : "sm",
            }),
            "w-full justify-start",
            link.href && isActive(link.href, link.isNewButton) && activeSty
les,
            link.isNewButton && "justify-center",
            link.hidden === true && "hidden"
          );

          return isCollapsed ? ( // NavPanel is collapsed = render with too
ltips
            <Tooltip key={index} delayDuration={0}>
              <TooltipTrigger
                className={cn(
                  buttonVariants({ variant: link.variant, size: "icon" }),
                  link.isNewButton && "mb-3",
                  "h-9 w-9",
                  link.href &&
                    isActive(link.href, link.isNewButton) &&
                    activeStyles,
                  link.hidden === true && "hidden"
                )}
                asChild
              >
                {link.href ? (
                  <Link href={link.href}>
                    <link.icon className="h-4 w-4" />
                    <span className="sr-only">{link.title}</span>
                  </Link>
                ) : (
                  <Button onClick={link.action} variant="none">
                    <link.icon className="h-4 w-4" />
```

```
                  <span className="sr-only">{link.title}</span>
                </Button>
              )}
            </TooltipTrigger>
            <TooltipContent side="right" className="flex items-center gap
-4">
              {link.title}
              {link.label && (
                <span className="ml-auto text-muted-foreground">
                  {link.label}
                </span>
              )}
            </TooltipContent>
          </Tooltip>
        ) : (
          // NavPanel is not collapsed = render links normally without to
oltips
          <div key={index}>
            {link.href ? (
              <Link href={link.href} className={desktopItemClassName}>
                {!link.isNewButton && <link.icon className="mr-2 h-4 w-4"
/>}
                {link.title}
                {link.label && (
                  <span
                    className={cn(
                      "ml-auto",
                      link.variant === "default" &&
                        "text-background dark:text-white"
                    )}
                  >
                    {link.label}
                  </span>
                )}
              </Link>
            ) : (
              <Button
                onClick={link.action}
                variant="none"
                className={desktopItemClassName}
              >
                {!link.isNewButton && <link.icon className="mr-2 h-4 w-4"
/>}
                {link.title}
                {link.label && (
                  <span
                    className={cn(
                      "ml-auto",
                      link.variant === "default" &&
                        "text-background dark:text-white"
                    )}
                  >
                    {link.label}
```

```
                  </span>
                )}
              </Button>
            )}
          </div>
        );
      })}
    </nav>
  </div>
  );
}
```

/components/example-client.tsx

```
"use client";
import { useTranslation } from "react-i18next"; // the client side function
for translations from `react`

export default function Greeting() {
  const { t } = useTranslation(["Common"]);

  // in this case I used username variable interpolation, so pass that as w
ell
  const name = "Peter Fox";
  return <div>{t("welcome", { name })}
  <h2>test: {t("admin_dashboard")}
    </h2></div>;
}
```

/components/message-display.tsx

```
"use client";

import styles from "@/app/scattered-profiles.module.css";
import { format } from "date-fns/format";
import {
  AlertTriangle,
  ArchiveRestore,
  ArrowLeft,
  ChevronDown,
  Edit,
  MessageCircleX,
  Send,
  Trash2,
  X,
} from "lucide-react";
import { Button } from "@/components/ui/button";
import { Separator } from "@/components/ui/separator";
```

```
import {
  Tooltip,
  TooltipContent,
  TooltipTrigger,
} from "@/components/ui/tooltip";
import { CategoryEnums, DBMessage } from "@/types";
import { useIsMobile } from "@/hooks/use-mobile";
import { cn, shuffleArray, toastActionResult } from "@/lib/utils";
import {
  cancelCurrentlyScheduled,
  deleteMessage,
  saveDraft,
  toggleTrash,
} from "@/lib/actions/message.actions";
import { toast } from "sonner";
import { ActionResponse } from "@/types/action";
import { usePathname, useRouter } from "next/navigation";
import ProfilePic from "./profile-pic";
import { DBRecipient, NewRecipient } from "@/types/recipient";
import { useTranslation } from "react-i18next";
import { PT_DATE_FORMAT } from "@/global.config";
import { useContacts } from "@/contexts/use-contacts";
import { PROFILE_COLOR_CSS_NAMES } from "@/lib/theme.colors";
import React, { useEffect, useMemo, useState } from "react";
import { useModal } from "@/contexts/use-modal";
import RecipientInfoModal from "./modals/recipient-info";
import { ScrollArea } from "./ui/scroll-area";

function MessageDisplay({
  message,
  category,
  reset,
}: {
  message: DBMessage | null;
  category?: CategoryEnums;
  reset: () => void;
}) {
  const today = new Date();
  const onMobile = useIsMobile();
  const router = useRouter();
  const { t } = useTranslation(["messages-page"]);
  const pathname = usePathname();
  const [moreInfoRecipient, setMoreInfoRecipient] =
    useState<NewRecipient | null>(null);
  const { setModal } = useModal();

  const [recipientsExpanded, setRecipientsExpanded] = useState(false);
  const { contacts, contactFetchError } = useContacts();
  // State to store random colors for each item
  const [profileColors, setProfileColors] = useState<string[]>([]);
  const showInfoAbout = (recipient: NewRecipient) => {
    setMoreInfoRecipient(recipient);
    setModal((m) => ({ ...m, contact: { ...m.contact, info: true } }));
```

```
  };

  const handleTrashButtonClick = async () => {
    if (message) {
      let result: ActionResponse<null>;

      // Drafts should also be discarded (deleted) immediately
      if (message.in_trash || message.status === "DRAFTED") {
        result = await deleteMessage(message.id, pathname);
      } else {
        result = await toggleTrash(message.id, true);
      }

      toastActionResult(result, t);
    }
  };

  const resend = async () => {
    if (message) {
      const newDraft = await saveDraft(undefined, {
        sender: message.sender,
        subject: message.subject || undefined,
        body: message.body,
        // convert DBRecipient to NewRecipient
        recipients: message.recipients.map((r) => ({
          phone: r.phone,
          // This is a temporary solution. Maybe change the type later to n
ot be NewRecipient[]
          isValid: true,
          proneForDeletion: false,
        })),
      });

      if (newDraft.draftId) {
        router.push(`/new-message?message_id=${newDraft.draftId}`);
      }
    }
  };
  const retry = () => {
    if (message) {
      router.push(`/new-message?message_id=${message.id}`);
    }
  };

  const putBack = async () => {
    if (message) {
      const result = await toggleTrash(message.id, false);

      toastActionResult(result, t);
    }
  };
```

```
  const cancelSend = async () => {
    if (message) {
      const smsReferenceId = parseInt(message.sms_reference_id);

      if (smsReferenceId && !isNaN(smsReferenceId)) {
        const result = await cancelCurrentlyScheduled(smsReferenceId);

        toastActionResult(result, t);
      } else {
        toast.error(t("messages-page:server-cancel_scheduled_invalid_id"));
      }
    }
  };

  const initialColors = PROFILE_COLOR_CSS_NAMES;
  let colors = [...initialColors]; // Create a copy of the array by spreadi
ng it.
  useEffect(() => {
    if (message) {
      shuffleArray(colors);

      setProfileColors(
        message.recipients.map((recipient, index) => {
          // Create a stable color for each item by using the index or item
(in case the order doesn't change)
          if (colors.length === 0) {
            // All items have been used
            // Reset the array using the initial array and reshuffle
            colors = [...initialColors]; // Reset array to original values
            shuffleArray(colors); // Shuffle the reset array
          }

          // Pick and remove the first item from the shuffled colors
          return colors.pop() as string;
        })
      );
    }
  }, [message]);

  return (
    <div className={cn("flex h-full flex-col")}>
      {moreInfoRecipient && (
        <RecipientInfoModal
          recipient={moreInfoRecipient}
          allowContactCreation={false}
        />
      )}
      {/* Begin top bar with action buttons */}
      <div className="flex items-center p-2 h-[var(--simple-header-height)]
border-b">
        <div className="flex items-center gap-2">
          {onMobile && (
```

```
            <Tooltip>
              <TooltipTrigger asChild>
                <Button variant="ghost" size="icon" onClick={() => reset()}
>
                  <ArrowLeft className="h-4 w-4" />
                  <span className="sr-only">{t("common:go_back")}</span>
                </Button>
              </TooltipTrigger>
              <TooltipContent>{t("common:go_back")}</TooltipContent>
            </Tooltip>
          )}

          {/* Move message to trash or delete it */}
          <Tooltip>
            <TooltipTrigger asChild>
              <Button
                variant="ghost"
                size="icon"
                disabled={!message}
                onClick={handleTrashButtonClick}
              >
                <Trash2 className="h-4 w-4" />
                <span className="sr-only">
                  {message?.in_trash || message?.status === "DRAFTED"
                    ? t("common:delete_permanently")
                    : t("common:move_to_trash")}
                </span>
              </Button>
            </TooltipTrigger>
            <TooltipContent>
              {message?.in_trash || message?.status === "DRAFTED"
                ? t("common:delete_permanently")
                : t("common:move_to_trash")}
            </TooltipContent>
          </Tooltip>

          {/* Cancel the sending of a scheduled message */}
          {category === "SCHEDULED" && (
            <Tooltip>
              <TooltipTrigger asChild>
                <Button
                  variant="ghost"
                  size="icon"
                  disabled={!message}
                  onClick={cancelSend}
                >
                  <MessageCircleX className="w-4 h-4" />
                  <span className="sr-only">{t("btn-cancel_scheduled")}</sp
an>
                </Button>
              </TooltipTrigger>
              <TooltipContent>{t("btn-cancel_scheduled")}</TooltipContent>
            </Tooltip>
```

```
)}

{/* Put back / restore trashed message */}
{category === "TRASH" && (
  <Tooltip>
    <TooltipTrigger asChild>
      <Button
        variant="ghost"
        size="icon"
        disabled={!message}
        onClick={putBack}
      >
        <ArchiveRestore className="w-4 h-4" />
        <span className="sr-only">{t("btn-restore")}</span>
      </Button>
    </TooltipTrigger>
    <TooltipContent>{t("btn-restore")}</TooltipContent>
  </Tooltip>
)}

{/* Reply to all recipients in the message */}
{category !== "DRAFTS" && category !== "FAILED" && (
  <Tooltip>
    <TooltipTrigger asChild>
      <Button
        variant="ghost"
        size="icon"
        onClick={resend}
        disabled={!message}
      >
        <Send className="h-4 w-4" />
        <span className="sr-only">{t("btn-resend")}</span>
      </Button>
    </TooltipTrigger>
    <TooltipContent>{t("btn-resend")}</TooltipContent>
  </Tooltip>
)}
{/* On Failed page we want a retry button */}
{category === "FAILED" && (
  <Tooltip>
    <TooltipTrigger asChild>
      <Button
        variant="ghost"
        size="icon"
        onClick={retry}
        disabled={!message}
      >
        <Send className="h-4 w-4" />
        <span className="sr-only">{t("btn-retry")}</span>
      </Button>
    </TooltipTrigger>
    <TooltipContent>{t("btn-retry")}</TooltipContent>
  </Tooltip>
```

```
          )}

          {/* Reply to all recipients in the message */}
          {category === "DRAFTS" && (
            <Tooltip>
              <TooltipTrigger asChild>
                <Button
                  variant="ghost"
                  size="icon"
                  onClick={() =>
                    message
                      ? router.push(`/new-message?message_id=${message.id}`
)
                      : ""
                  }
                  disabled={!message}
                >
                  <Edit className="h-4 w-4" />
                  <span className="sr-only">{t("btn-continue_draft")}</span
>
                </Button>
              </TooltipTrigger>
              <TooltipContent>{t("btn-continue_draft")}</TooltipContent>
            </Tooltip>
          )}
        </div>
        <div className="ml-auto flex items-center gap-2">
          {/* Close (deselect) the selected message */}
          <Tooltip>
            <TooltipTrigger asChild>
              <Button
                variant="ghost"
                size="icon"
                onClick={() => reset()}
                disabled={!message}
              >
                <X className="h-4 w-4" />
                <span className="sr-only">{t("common:close")}</span>
              </Button>
            </TooltipTrigger>
            <TooltipContent>{t("common:close")}</TooltipContent>
          </Tooltip>
        </div>
      </div>
      {/* End top bar */}
      {/* <Separator /> */}
      {/* Begin message content */}
      <ScrollArea>
        <div
          className={
            onMobile
              ? `h-[calc(100vh-var(--simple-header-height))]`
              : `h-[calc(100vh-var(--header-height))]`
```

```
          }
        >
          <div className="flex flex-col h-full">
            {message ? (
              <div className="grow flex flex-col">
                <div className="flex justify-between p-4">
                  <div className="flex gap-4 text-sm w-full">
                    <div className="flex relative min-w-[50px] min-h-[50px]
h-[50px]">
                      {message.recipients.map(
                        (recipient: DBRecipient, index) => {
                          if (index >= 5) return; // Max recipients reached
; remaining will be shown as a single picture with count

                          let foundContactName: string | undefined = undefi
ned;

                          foundContactName = contacts.find(
                            (contact) => contact.phone === recipient.phone
                          )?.name;

                          if (index == 4) {
                            // the fifth recipient should be the number of
missing recipients
                            const missingRecipients =
                              message.recipients.length - index;
                            if (missingRecipients > 1) {
                              // if there are many missing recipients,
                              foundContactName = `+ ${missingRecipients}`;
                            }
                          }

                          return (
                            <ProfilePic
                              key={index}
                              // size={10}
                              name={foundContactName}
                              className={cn(
                                styles["profile-absolute"],
                                index === 0 &&
                                  cn("center-absolute", styles["profile-big
"]),
                                index === 1 && styles["profile-top-left"],
                                index === 2 && styles["profile-bottom-left"
],
                                index === 3 && styles["profile-top-right"],
                                index === 4 && styles["profile-bottom-right
"]
                              )}
                              // The dynamically generated class `bg-${chos
enColor}` won't work because Tailwind purges unused classes in production,
and it doesn't recognize dynamically created class names.
```

```
                                style={{
                                  // Only show color for saved contacts
                                  backgroundColor: foundContactName
                                    ? profileColors[index]
                                    : "",
                                }}
                              />
                            );
                          }
                        )}
                      </div>
                      <div className="flex flex-col gap-1 grow overflow-hidde
n">
                        <div className="flex justify-between items-center rel
ative">
                          <span className="font-semibold ellipsis">
                            {message.subject || t("no_subject")}
                          </span>
                          {message.send_time && (
                            <span
                              className="text-xs text-muted-foreground relati
ve whitespace-nowrap"
                              style={{ top: "1px" }}
                            >
                              {format(
                                new Date(message.send_time),
                                PT_DATE_FORMAT
                              )}
                            </span>
                          )}
                          <Button
                            onClick={() =>
                              setRecipientsExpanded(
                                (prevExpanded) => !prevExpanded
                              )
                            }
                            variant="none"
                            className="p-0 pl-1 h-min absolute right-0 bottom
-[-20px] bg-background z-10 rounded-none"
                          >
                            <ChevronDown
                              className={cn(
                                "duration-200",
                                !recipientsExpanded && "rotate-90"
                              )}
                            />
                          </Button>
                        </div>
                        <div className={cn("flex text-xs gap-1 relative")}>
                          {!recipientsExpanded && (
                            <div
                              // Have a div cover the recipients so that the
user has to expand the recipients first to be able to view more info
```

```
                            className="container-overlay"
                            onClick={() => setRecipientsExpanded(true)}
                          />
                        )}
                        <div
                          className={cn(
                            "flex gap-1",
                            recipientsExpanded ? "flex-wrap mr-5" : ""
                          )}
                        >
                          <div className="font-medium">{t("common:to")}:</d
iv>

                          {message.recipients.map(
                            (recipientWithoutContact, index) => {
                              const recipient: NewRecipient = {
                                ...recipientWithoutContact,
                                contact: contacts.find(
                                  (contact) =>
                                    contact.phone ===
                                    recipientWithoutContact.phone
                                ),
                                // This is a temporary solution. Maybe chan
ge the type later to not be NewRecipient[]
                                isValid: true,
                                proneForDeletion: false,
                              };
                              return (
                                <div key={recipient.phone} className="flex"
>
                                  <Button
                                    variant="none"
                                    onClick={() => showInfoAbout(recipient)
}
                                    className="whitespace-nowrap p-0 text-x
s h-min hover:bg-muted px-[2px]"
                                  >
                                    {recipient.contact?.name || recipient.p
hone}
                                  </Button>
                                  {index < message.recipients.length - 1 &&
                                    ", "}
                                </div>
                              );
                            }
                          )}
                        </div>
                      </div>
                    </div>
                  </div>
                </div>

                <Separator />
```

```
              <div className="flex-1 whitespace-pre-wrap p-4 text-sm">
                {message.body}
              </div>
            </div>
          ) : (
            <div className="p-8 text-center text-muted-foreground">
              {t("none_selected")}
            </div>
          )}

          {message && message.status === "FAILED" && (
            <>
              <Separator className="" />

              <div className="flex w-full p-4 gap-2">
                <AlertTriangle className="relative top-2 text-destructive
min-w-6 min-h-6" />
                <div className="flex flex-col gap-2">
                  <p className="text-destructive text-sm font-semibold ">
                    {t(`api_error_${message.api_error_code}`)}
                  </p>
                  <pre className="max-w-max whitespace-pre-wrap break-wor
ds bg-muted border p-2 rounded-lg text-xs">
                    {message.api_error_details_json
                      ? JSON.stringify(
                          JSON.parse(message.api_error_details_json),
                          null,
                          2
                        )
                      : t("no_json_available")}
                  </pre>
                </div>
              </div>

              {/* <p className="text-muted-foreground text-sm mb-4">
                  {t("api_error_caption")}
                </p> */}
            </>
          )}

          {/* You can remove the message check if you want to, I like it
better that this bottom bar only shows up on selection */}
          {message && message.status === "DRAFTED" ? (
            <>
              <Separator className="mt-auto" />
              <div className="flex px-4 py-2 justify-end gap-2">
                <Button
                  variant="default"
                  type="button"
                  className="w-max"
                  disabled={!message}
                  onClick={() =>
```

```
                    message
                      ? router.push(`/new-message?message_id=${message.id
}`)
                      : ""
                  }
                >
                  <Edit className="h-4 w-4" />
                  {t("btn-continue_draft")}
                </Button>
              </div>
            </>
          ) : (
            ""
          )}
        </div>
      </div>
    </ScrollArea>
  </div>
 );
}
export default React.memo(MessageDisplay);
```

/components/settings-item.tsx

```
"use client";

import { Input } from "@/components/ui/input";
import { updateSetting } from "@/lib/actions/user.actions";
import { cn } from "@/lib/utils";
import React, { SetStateAction } from "react";
import {
  useState,
  useTransition,
  type FormEvent,
  type InputHTMLAttributes,
  useEffect,
} from "react";
import type { UpdateSettingResponse } from "@/types/action";
import { useSettings } from "@/contexts/use-settings";

export type RenderInputArgs = {
  value: string;
  onChange: (newValue: string) => void;
  onBlur: (e?: FormEvent<Element>, submittedValue?: string) => void;
  id: string;
  initialValue?: string;
  className?: string;
  isPending: boolean;
  setServerState?: React.Dispatch<SetStateAction<UpdateSettingResponse>>;
};
```

```
type SettingsItemProps = InputHTMLAttributes<HTMLInputElement> & {
  name: string;
  initialValue?: string;
  label?: string;
  caption?: string;
  inputType?: string;
  renderInput?: (props: RenderInputArgs) => React.ReactNode;
  onUpdate?: (newValue: string) => void;
};

const initialState: UpdateSettingResponse = {
  success: false,
  input: "",
};

export function SettingsItem({
  name,
  initialValue = "",
  label,
  caption,
  inputType = "text",
  renderInput,
  onUpdate,
  ...inputProps
}: SettingsItemProps) {
  const [value, setValue] = useState<string>(initialValue);
  const [isPending, setIsPending] = useState<boolean>(false);
  const [serverState, setServerState] = useState(initialState);
  const { setSettings } = useSettings();

  async function handleSubmit(e?: FormEvent, submittedValue?: string) {
    if (e) e.preventDefault();
    setIsPending(true);

    const formData = new FormData();
    formData.append("name", name);
    formData.append("value", submittedValue || value);

    const result = await updateSetting(formData);
    setServerState(result);
    if (onUpdate) onUpdate(value);

    // these are currently the settings that we store in localstorage as we
ll as state
    const stateSettingNames = [
      "display_name",
      "profile_color_id",
      "appearance_layout",
    ];
    if (stateSettingNames.includes(name)) {
      // 1. Update localstorage itself
```

```
      localStorage.setItem(name, result.data || initialValue);

      // 2. Update state since localStorage changes don't trigger re-render
s.
      setSettings((prev) => ({
        displayName: name === "display_name" ? result.data : prev.displayNa
me,
        profileColorId:
          name === "profile_color_id" ? result.data : prev.profileColorId,
        layout: name === "appearance_layout" ? result.data : prev.layout,
      }));
    }
    setIsPending(false);
  }

  const handleChange = (newValue: string) => {
    setValue(newValue);
  };

  const defaultInput = (
    <Input
      id={name}
      type={inputType}
      value={value}
      onChange={(e) => handleChange(e.target.value)}
      onBlur={(e?: any, v?: any) => handleSubmit(e, v)}
      className="w-max"
      disabled={isPending}
      {...inputProps}
    />
  );

  const inputElement = renderInput
    ? renderInput({
        value,
        onChange: handleChange,
        onBlur: (e, submittedValue) => handleSubmit(e, submittedValue),
        id: name,
        initialValue,
        isPending,
        setServerState,
      })
    : defaultInput;

  return (
    <form
      onSubmit={handleSubmit}
      style={{ marginBottom: "1rem" }}
      className="space-y-2 flex flex-col"
    >
      <label
        className="text-sm font-medium leading-none peer-disabled:cursor-no
```

```
t-allowed peer-disabled:opacity-70"
        htmlFor={name}
      >
        {(isPending && "Saving...") || label || name}
      </label>
      {inputElement}
      <p
        className={cn(
          "text-[0.8rem] order-1",
          serverState.error ? "text-destructive" : "text-muted-foreground"
        )}
      >
        {serverState.error || caption}
      </p>
    </form>
  );
}

export default SettingsItem;
```

/components/nav-panel.tsx

```
"use client";

import { useCallback, useEffect, useState } from "react";
import {
  Sheet,
  SheetContent,
  SheetTitle,
  SheetTrigger,
} from "@/components/ui/sheet";
import { Button, buttonVariants } from "@/components/ui/button";
import {
  AlertTriangle,
  Calendar,
  LogOut,
  Menu,
  UserRoundPen,
} from "lucide-react";
import {
  MonitorCog,
  Settings,
  Trash2,
  Contact2,
  Pencil,
  MailCheck,
  FileText,
} from "lucide-react";
import { ResizableHandle, ResizablePanel } from "./ui/resizable";
import { cn } from "@/lib/utils";
```

```
import { Separator } from "./ui/separator";
import NavLinks from "./nav-links";
import { useTranslation } from "react-i18next";
import { useLayout } from "@/contexts/use-layout";
import { ScrollArea } from "./ui/scroll-area";
import { useIsMobile } from "@/hooks/use-mobile";
import { usePathname, useRouter } from "next/navigation";
import { useSession } from "@/hooks/use-session";
import { logout } from "@/lib/auth";
import {
  AlertDialog,
  AlertDialogAction,
  AlertDialogCancel,
  AlertDialogContent,
  AlertDialogDescription,
  AlertDialogFooter,
  AlertDialogHeader,
  AlertDialogTitle,
  AlertDialogTrigger,
} from "@/components/ui/alert-dialog";
import { useSettings } from "@/contexts/use-settings";
import Account from "./shared/account";
import AppLogo from "./logo";

export default function NavPanel() {
  const { layout, isCollapsed, setIsCollapsed, fallbackLayout, isFullscreen
} =
    useLayout();
  // In case we need to check for large screens
  let isExtraLargeScreen = window.innerWidth >= 1200;
  // the nav panel is a bit bigger than that, but the elements inside keep
it at its minimum size
  const COLLAPSED_SIZE = 2;

  const hidePanelClassName =
    ((isFullscreen || useIsMobile()) && "hidden") || undefined;
  return (
    <>
      <ResizablePanel
        className={cn(
          isCollapsed && "min-w-[50px] transition-all duration-300 ease-in-
out",
          hidePanelClassName
        )}
        defaultSize={layout ? layout[0] : fallbackLayout[0]}
        collapsedSize={COLLAPSED_SIZE}
        collapsible={true}
        minSize={13}
        maxSize={35}
        onCollapse={() => {
          setIsCollapsed(true);
          const cookieValue = JSON.stringify(true);
          const cookiePath = "/";
```

```
          document.cookie = `react-resizable-panels:collapsed=${cookieValue
}; path=${cookiePath};`;
        }}
        onResize={() => {
          setIsCollapsed(false);
          const cookieValue = JSON.stringify(false);
          const cookiePath = "/";
          document.cookie = `react-resizable-panels:collapsed=${cookieValue
}; path=${cookiePath};`;
        }}
      >
        <NavPanelContent isCollapsed={isCollapsed} />
      </ResizablePanel>
      <ResizableHandle withHandle className={hidePanelClassName} />
    </>
  );
}

export function MobileNavPanel() {
  const { mobileNavPanel, setMobileNavPanel } = useLayout();
  const router = useRouter();
  const { t } = useTranslation(["navigation"]);

  useEffect(() => {
    setMobileNavPanel(false);
  }, [router]);

  // add a click event listener to the nav element
  const handleNavClick = useCallback((event: React.MouseEvent<HTMLElement>)
=> {
    const target = event.target as HTMLElement;
    // when user clicks inside of this NavPanel, we check if the element cl
icked is a <Link> and close the NavPanel. This is so that we can have the n
ice closing animation
    if (target.tagName === "A" || target.closest("a")) {
      setMobileNavPanel(false);
    }
  }, []);

  return (
    <Sheet
      open={mobileNavPanel}
      onOpenChange={setMobileNavPanel}
      /* You can change the animation duration inside the shadCn component
(easiest way) */
    >
      <SheetContent side="left" className="w-[300px] p-0">
        <SheetTitle className="sr-only">{t("sr_only-nav_menu")}</SheetTitle
>
        <nav onClick={handleNavClick}>
          <NavPanelContent
            isCollapsed={false} // on mobile it will never be collapsed
```

```
          />
        </nav>
      </SheetContent>
    </Sheet>
  );
}

// We have to data sources for the user's profile:
// 1. Sensitive information is extracted from the encrypted session
// 2. Stuff that can be changed in the settings is encrypted from localstor
age
function NavPanelContent({ isCollapsed }: { isCollapsed: boolean }) {
  const { t, i18n } = useTranslation(["navigation", "modals", "common"]);
  const { amountIndicators } = useLayout();
  const router = useRouter();
  const { session, loading } = useSession();
  const { settings, resetLocalSettings } = useSettings();
  const onMobile = useIsMobile();

  const [confirmLogoutOpen, setConfirmLogoutOpen] = useState(false);
  const showAlertDialog = () => {
    // show the alert dialog
    setConfirmLogoutOpen(true);
  };

  const handleLogout = async () => {
    const { success } = await logout();
    if (success) {
      resetLocalSettings();
      router.push("/login");
    }
  };

  return (
    <>
      {(onMobile || settings.layout === "SIMPLE") && (
        <div
          className={cn(
            "h-[var(--simple-header-height)] border-b flex items-center gap
-2",
            !isCollapsed && "px-2",
            isCollapsed && "justify-center"
          )}
        >
          <AppLogo isCollapsed={isCollapsed} />
        </div>
      )}

      {/* <Separator /> */}
      <NavLinks
        isCollapsed={isCollapsed}
        links={[
```

```
          {
            title: t("new_message"),
            icon: Pencil,
            variant: "default",
            size: "xl",
            isNewButton: true,
            action: () => {
              // We need to manually refresh so all the inputs actually get
refreshed
              router.push("/new-message");
              router.refresh();
            },
          },
        ]}
      />

      {/* Maybe we need a fixed height here, but if everything works, all g
ood. Use div instead of ScrollArea, because otherwise it the Sheet componen
t glitches out */}
      <div className="overflow-auto">
        <div
          className="flex flex-col"
          // In tailwind, this doesn't work, and I don't know why
          style={{
            // 56 is the new-message button
            height: `calc(100vh - var(--simple-header-height) - 56px${
              isCollapsed ? " - 8px" : ""
            })`,
            width: "100%",
          }}
        >
          <div className="grow">
            <NavLinks
              isCollapsed={isCollapsed}
              links={[
                {
                  title: t("sent"),
                  label:
                    amountIndicators?.sent == 0
                      ? ""
                      : amountIndicators?.sent.toString(),
                  icon: MailCheck,
                  variant: "ghost",
                  href: "/sent",
                },
                {
                  title: t("scheduled"),
                  label:
                    amountIndicators?.scheduled == 0
                      ? ""
                      : amountIndicators?.scheduled.toString(),
                  icon: Calendar,
                  variant: "ghost",
```

```
        href: "/scheduled",
      },
      {
        title: t("failed"),
        label:
          amountIndicators?.failed == 0
            ? ""
            : amountIndicators?.failed.toString(),
        icon: AlertTriangle,
        variant: "ghost",
        href: "/failed",
      },
      {
        title: t("drafts"),
        label:
          amountIndicators?.drafts == 0
            ? ""
            : amountIndicators?.drafts.toString(),
        icon: FileText,
        variant: "ghost",
        href: "/drafts",
      },
      {
        title: t("trash"),
        label:
          amountIndicators?.trash == 0
            ? ""
            : amountIndicators?.trash.toString(),
        icon: Trash2,
        variant: "ghost",
        href: "/trash",
      },
    ]}
  />

  <Separator />
  <NavLinks
    isCollapsed={isCollapsed}
    links={[
      {
        title: t("settings"),
        label: "",
        icon: Settings,
        variant: "ghost",
        href: "/settings",
      },
      {
        title: t("contacts"),
        label:
          amountIndicators?.contacts == 0
            ? ""
            : amountIndicators?.contacts.toString(),
        icon: Contact2,
```

```
                variant: "ghost",
                href: "/contacts",
              },
              {
                title: t("dashboard"),
                label: "",
                icon: MonitorCog,
                variant: "ghost",
                href: "/dashboard",
                hidden: !session?.isAdmin,
              },
            ]}
          />
        </div>

        <Separator />
        <div className="shrink h-[var(--simple-header-height)] flex flex-
col justify-center">
          {/* Also show logout button in the mobile sheet, regardless of
the current layout */}
          {!onMobile && settings.layout === "SIMPLE" ? (
            <Account hideNameRole={isCollapsed} className="px-2" />
          ) : (
            <>
              <NavLinks
                isCollapsed={isCollapsed}
                links={[
                  {
                    title: t("log_out"),
                    label: "",
                    icon: LogOut,
                    variant: "ghost",
                    action: showAlertDialog,
                  },
                ]}
              />

              {/* "Confirm Logout" dialog */}
              <AlertDialog
                open={confirmLogoutOpen}
                onOpenChange={setConfirmLogoutOpen}
              >
                <AlertDialogContent>
                  <AlertDialogHeader>
                    <AlertDialogTitle>
                      {t("modals:logout-header")}
                    </AlertDialogTitle>
                    <AlertDialogDescription>
                      {t("modals:logout-header_caption")}
                    </AlertDialogDescription>
                  </AlertDialogHeader>
                  <AlertDialogFooter>
                    <AlertDialogCancel>
```

```
                    {t("common:cancel")}
                  </AlertDialogCancel>
                  <AlertDialogAction onClick={handleLogout}>
                    {t("common:continue")}
                  </AlertDialogAction>
                </AlertDialogFooter>
              </AlertDialogContent>
            </AlertDialog>
          </>
        )}
      </div>
    </div>
  </div>
 </>
 );
}
```

/components/admin-dashboard/index.tsx

```
"use client";

import MessagePieChart from "@/components/admin-dashboard/message-pie-chart
";
import MessageAreaChart from "@/components/admin-dashboard/message-area-cha
rt";
import UserRankingTable from "@/components/admin-dashboard/user-table";
import { PageHeader } from "@/components/headers";
import Account from "@/components/shared/account";
import { Button, buttonVariants } from "@/components/ui/button";
import { Card, CardContent, CardHeader, CardTitle } from "@/components/ui/c
ard";
import { useSettings } from "@/contexts/use-settings";
import { cn, extractFirstWord, getPercentageChange } from "@/lib/utils";
import { DBUser } from "@/types/user";
import Link from "next/link";
import { useTranslation } from "react-i18next";
import { CountryStat } from "../../app/[locale]/dashboard/page";
import { LightDBMessage } from "@/types/dashboard";
import { ScrollArea } from "../ui/scroll-area";
import { useIsMobile } from "@/hooks/use-mobile";
import { ArrowLeft } from "lucide-react";

export type TimeRange = {
  from: Date;
  to: Date;
};

export default function AdminDashboard({
  messages,
  users,
```

```
  countryStats,
}: {
  messages: LightDBMessage[];
  users: DBUser[];
  countryStats: CountryStat[] | undefined;
}) {
  const { t } = useTranslation(["dashboard-page", "errors", "common"]);
  const messageCounts = countMessages(messages);
  const { settings } = useSettings();
  const onMobile = useIsMobile();
  const onBigScreen = false;

  return (
    <div className="flex flex-col">
      <PageHeader
        title={
          onBigScreen
            ? t("header_long", {
                first_name: settings.displayName
                  ? extractFirstWord(settings.displayName)
                  : "User",
              })
            : t("header")
        }
        marginRight={onMobile}
      >
        {!onMobile && (
          <>
            <Link
              href="/"
              className={cn(buttonVariants({ variant: "link" }), "mx-2")}
            >
              <ArrowLeft className="h-4 w-4" />
              {t("back_to_app")}
            </Link>

            <Account className="ml-auto" profilePicPosition="RIGHT" />
          </>
        )}
      </PageHeader>

      <ScrollArea
        /** We always want simple header height here due to only having 1 s
imple nav-panel, regardless of any layout*/
        className="h-[calc(100vh-var(--simple-header-height))]"
      >
        <div
          className="p-4" /* Inside looks better with rimless bottom on scr
oll */
        >
          <div className="flex flex-col md:grid grid-cols-3 gap-4">
            <TextCard
              label={t("text_card_1-title")}
```

```
        value={messageCounts.today}
        caption={
          getPercentageChange(
            messageCounts.today,
            messageCounts.todayBefore
          ) < 0
            ? // Negative change (lower than before)
              t("text_card_1-caption_lower", {
                percentage: `${
                  getPercentageChange(
                    messageCounts.today,
                    messageCounts.todayBefore
                  ) * -1
                }%`,
              })
            : // Positive change (higher than before)
              t("text_card_1-caption_higher", {
                percentage: `${getPercentageChange(
                  messageCounts.today,
                  messageCounts.todayBefore
                )}%`,
              })
        }
      />
      <TextCard
        label={t("text_card_2-title")}
        value={messageCounts.last7Days}
        caption={
          getPercentageChange(
            messageCounts.last7Days,
            messageCounts.last7DaysBefore
          ) < 0
            ? // Negative change (lower than before)
              t("text_card_2-caption_lower", {
                percentage: `${
                  getPercentageChange(
                    messageCounts.last7Days,
                    messageCounts.last7DaysBefore
                  ) * -1
                }%`,
              })
            : // Positive change (higher than before)
              t("text_card_2-caption_higher", {
                percentage: `${getPercentageChange(
                  messageCounts.last7Days,
                  messageCounts.last7DaysBefore
                )}%`,
              })
        }
      />
      <TextCard
        label={t("text_card_3-title")}
        value={messageCounts.lastMonth}
```

```
              caption={
                getPercentageChange(
                  messageCounts.lastMonth,
                  messageCounts.lastMonthBefore
                ) < 0
                  ? // Negative change (lower than before)
                    t("text_card_3-caption_lower", {
                      percentage: `${
                        getPercentageChange(
                          messageCounts.lastMonth,
                          messageCounts.lastMonthBefore
                        ) * -1
                      }%`,
                    })
                  : // Positive change (higher than before)
                    t("text_card_3-caption_higher", {
                      percentage: `${getPercentageChange(
                        messageCounts.lastMonth,
                        messageCounts.lastMonthBefore
                      )}%`,
                    })
              }
            />
            {/* <Card>
            <CardHeader>
              <CardTitle>Sent This week</CardTitle>
            </CardHeader>
            <CardContent>{messageCounts.last7Days}</CardContent>
          </Card>
          <Card>
            <CardHeader>
              <CardTitle>Sent This Month</CardTitle>
            </CardHeader>
            <CardContent>{messageCounts.last3Months}</CardContent>
          </Card> */}
            <div className="col-span-3">
              <div className="h-min">
                <MessageAreaChart messages={messages || []} />
              </div>
            </div>
            <div className={cn("col-span-2", onMobile && "order-6")}>
              <UserRankingTable users={users || []} messages={messages || [
]} />
            </div>
            <MessagePieChart data={countryStats} />
          </div>
        </div>
      </ScrollArea>
    </div>
  );
}

function TextCard({
```

```
  label,
  value,
  caption,
}: {
  label: string;
  value: string | number;
  caption: string;
}) {
  return (
    <Card className="min-h-min">
      <CardContent className="p-6 flex flex-col gap">
        <p className="text-sm font-semibold text-foreground">{label}</p>
        <h1 className="leading-tight">{value}</h1>
        <p className="text-sm text-muted-foreground">{caption}</p>
      </CardContent>
    </Card>
  );
}

function countMessages(messages: LightDBMessage[]) {
  const now = new Date();
  const todayStart = new Date(now.getFullYear(), now.getMonth(), now.getDat
e());
  const yesterdayStart = new Date(todayStart);
  yesterdayStart.setDate(todayStart.getDate() - 1);
  const yesterdayEnd = new Date(todayStart);
  yesterdayEnd.setDate(todayStart.getDate() - 1);
  yesterdayEnd.setHours(23, 59, 59, 999); // End of yesterday

  const sevenDaysAgo = new Date(now);
  sevenDaysAgo.setDate(now.getDate() - 7);
  const weekBeforeStart = new Date(sevenDaysAgo);
  weekBeforeStart.setDate(sevenDaysAgo.getDate() - 7); // Start of the week
before last 7 days

  const oneMonthAgo = new Date(now);
  oneMonthAgo.setMonth(now.getMonth() - 1);
  const oneMonthBeforeStart = new Date(oneMonthAgo);
  oneMonthBeforeStart.setMonth(oneMonthAgo.getMonth() - 1); // Start of the
3 months before last 3 months

  const counts = {
    today: 0,
    todayBefore: 0,
    last7Days: 0,
    last7DaysBefore: 0, // New property for the week before last 7 days
    lastMonth: 0,
    lastMonthBefore: 0, // New property for the 3 months before last 3 mont
hs
  };

  messages.forEach((message) => {
```

```
    const sentAt = new Date(message.send_time);

    // Count messages sent today
    if (sentAt >= todayStart) {
      counts.today++;
    }
    // Count messages sent yesterday
    if (sentAt >= yesterdayStart && sentAt <= yesterdayEnd) {
      counts.todayBefore++;
    }

    // Count messages sent in the last 7 days
    if (sentAt >= sevenDaysAgo) {
      counts.last7Days++;
    }
    // Count messages sent in the week before the last 7 days
    if (sentAt >= weekBeforeStart && sentAt < sevenDaysAgo) {
      counts.last7DaysBefore++;
    }

    // Count messages sent in the last 3 months
    if (sentAt >= oneMonthAgo) {
      counts.lastMonth++;
    }
    // Count messages sent in the 3 months before the last 3 months
    if (sentAt >= oneMonthBeforeStart && sentAt < oneMonthAgo) {
      counts.lastMonthBefore++;
    }
  });

  return counts;
}
```

/components/admin-dashboard/message-pie-chart.tsx

```
"use client";

import { useState, useEffect, useMemo } from "react";
import { TrendingUp } from "lucide-react";
import { Cell, Pie, PieChart, ResponsiveContainer, Tooltip } from "recharts
";

import {
  Card,
  CardContent,
  CardDescription,
  CardFooter,
  CardHeader,
  CardTitle,
} from "@/components/ui/card";
```

```
import { useTranslation } from "react-i18next";
import { capitalize, cn, shuffleArray } from "@/lib/utils";
import { CountryStat } from "@/app/[locale]/dashboard/page";
import { themesArray } from "@/lib/theme.colors";
import { useTheme as useNextTheme } from "next-themes";
import { ThemeMode } from "@/types/theme";
import { useThemeContext } from "@/contexts/theme-data-provider";

export default function MessagePieChart({
  data,
}: {
  data: CountryStat[] | undefined;
}) {
  const { theme } = useNextTheme();
  const { themeColor } = useThemeContext();
  const userLikesZinc = themeColor === 1;
  const slicedArray = [
    ...themesArray.slice(
      userLikesZinc ? 0 : 1, // remove Zinc color if the user doesn't like
it, because it looks bad with the other colors. If he does, leave it in
      themesArray.length
    ),
  ];

  const [pieChartColors, setPieChartColors] = useState<string[]>([]);

  const totalMessages = useMemo(() => {
    return data?.reduce((acc, curr) => acc + curr.amount, 0);
  }, [data]);
  const { t } = useTranslation();

  const totalCost = useMemo(() => {
    return data?.reduce((acc, curr) => acc + curr.cost, 0).toFixed(2);
  }, [data]);

  // Find the country with the most messages
  const topCountry = useMemo(() => {
    if (data?.length === 0) return null;
    return data?.reduce((max, curr) => (max.amount > curr.amount ? max : cu
rr));
  }, [data]);

  // Custom center label renderer
  const renderCustomLabel = ({ cx, cy }: any) => {
    return (
      <g>
        <text
          x={cx}
          y={cy}
          fill="hsl(var(--foreground))"
          textAnchor="middle"
          dominantBaseline="central"
```

```
          className="text-3xl font-bold"
        >
          {totalMessages}
        </text>
        <text
          x={cx}
          y={cy + 24}
          fill="hsl(var(--foreground))"
          textAnchor="middle"
          dominantBaseline="central"
          className="text-sm text-muted-foreground opacity-75"
        >
          {t("messages_amount")}
        </text>
      </g>
    );
  };

  useEffect(() => {
    // Randomize colors on component mount only, so that when user changes
the input the colors don't change
    shuffleArray(slicedArray);
    setPieChartColors(
      slicedArray.map(
        (themeColor) =>
          `hsl(${themeColor.value[(theme as ThemeMode) || "light"].primary}
)`
      )
    );
  }, []);

  return (
    <Card className="flex flex-col min-h-[400px]">
      <CardHeader className="items-center md:items-start pb-0">
        <CardTitle>{t("pie_chart-title")}</CardTitle>
        <CardDescription>{t("pie_chart-title_caption")}</CardDescription>
      </CardHeader>

      {/* Error case */}
      {!data || !data.length ? (
        <CardContent className="h-full">
          {data === undefined ? (
            <p className="h-full centered text-destructive">
              {t("pie_chart-error")}
            </p>
          ) : (
            data?.length === 0 && (
              <p className="h-full centered">
                {/* No data case */}
                {t("pie_chart-no_data")}
              </p>
            )
          )}
```

```
        </CardContent>
      ) : (
        <>
          {/* Content gets rendered in all other conditions */}
          <CardContent className="flex-1 pb-0 h-[300px]">
            <div className="mx-auto aspect-square h-full max-w-[300px] flex
justify-center">
              <ResponsiveContainer
                /* Docs say percentages, but numerical values work better *
/
                width={250}
                aspect={1}
              >
                <PieChart className="">
                  <Tooltip content={<CustomTooltip />} />

                  <Pie
                    data={data}
                    cx="50%"
                    cy="50%"
                    labelLine={false}
                    label={renderCustomLabel}
                    innerRadius={60}
                    outerRadius={105}
                    // strokeWidth={3}
                    dataKey="amount"
                    nameKey="country"
                  >
                    {data.map((entry, index) => (
                      <Cell
                        key={`cell-${index}`}
                        fill={pieChartColors[index % pieChartColors.length]
}
                        // Adjust these values
                        strokeWidth={0.5}
                        stroke="hsl(var(--background))"
                        // strokeOpacity={0.3}
                      />
                    ))}
                  </Pie>
                </PieChart>
              </ResponsiveContainer>
              {/* </ChartContainer> */}
            </div>
          </CardContent>
          <CardFooter className="flex-col gap-2 text-sm pt-4">
            <div className="flex items-center gap-2 font-medium leading-non
e">
              {topCountry && (
                <>
                  {t("pie_chart-leading_country", {
                    country: topCountry.country,
                    amount: topCountry.amount,
```

```
                  })}
                  <TrendingUp className="h-4 w-4" />
                </>
              )}
            </div>
            <div className="leading-none text-muted-foreground">
              {t("pie_chart-total_cost")} ${totalCost}
            </div>
          </CardFooter>
        </>
      )}
    </Card>
  );
}

function CustomTooltip({ active, payload }: any) {
  const { t } = useTranslation();
  if (active && payload && payload.length) {
    return (
      <div
        className={cn(
          "grid min-w-[8rem] items-start gap-1.5 rounded-lg border border-s
late-200/50 bg-background px-2.5 py-1.5 text-xs shadow-xl dark:border-slate
-800 dark:border-slate-800/50"
        )}
      >
        <div className="grid gap-1.5">
          {payload.map((item: any, index: number) => {
            const key = item.name || item.dataKey || "value";
            // const itemConfig = getPayloadConfigFromPayload(
            //   config,
            //   item,
            //   key
            // );
            const indicatorColor = item.payload.fill || item.color;

            return (
              <div
                key={item.dataKey}
                className={cn("flex w-full flex-col items-stretch gap-2 ")}
//[&>svg]:h-2.5 [&>svg]:w-2.5 [&>svg]:text-muted-foreground dark:[&>svg]:te
xt-muted-foreground
              >
                <div className="flex items-center gap-1">
                  <div
                    style={{
                      width: 10,
                      height: 10,
                      borderRadius: 2,
                      backgroundColor: indicatorColor,
                    }}
                  />
                  <div className="font-medium">{item.name}</div>
```

```
                </div>

                {/* Key value pairs */}
                <KeyValueInTooltip name={t("cost")} value={item.payload.cos
t} />
                <KeyValueInTooltip
                  name={t("messages_amount")}
                  value={item.payload.amount}
                />
              </div>
            );
          })}
        </div>
      </div>
    );
  }

  return null;
}

function KeyValueInTooltip({
  name,
  value,
}: {
  name: string;
  value?: string | number;
}) {
  return (
    <div className={cn("flex flex-1 justify-between leading-none")}>
      <div className="grid gap-1.5">
        <span className="text-muted-foreground ">{name}</span>
      </div>
      {value && (
        <span className="font-mono font-medium tabular-nums text-slate-950
dark:text-slate-50">
          {value}
        </span>
      )}
    </div>
  );
}
```

/components/admin-dashboard/user-table.tsx

```
"use client";

import {
  Card,
  CardContent,
  CardDescription,
```

```
  CardHeader,
  CardTitle,
} from "@/components/ui/card";
import { DBUser } from "@/types/user";
import ProfilePic from "../profile-pic";
import { useTranslation } from "react-i18next";
import { useMemo } from "react";
import { LightDBMessage } from "@/types/dashboard";

export default function UserRankingTable({
  users,
  messages,
}: {
  users: DBUser[];
  messages: LightDBMessage[];
}) {
  const { t } = useTranslation();
  const usersWithMessageCounts = useMemo(() => {
    return users
      .map((user) => ({
        ...user,
        messageCount: messages.filter((m) => m.user_id === user.id).length,
      }))
      .sort((a, b) => b.messageCount - a.messageCount);
  }, [users, messages]);

  return (
    <Card className="h-full">
      <CardHeader>
        <CardTitle>{t("users_table-title")}</CardTitle>
        <CardDescription>{t("users_table-title_caption")}</CardDescription>
      </CardHeader>
      <CardContent className="">
        <div className="max-h-[300px] overflow-auto">
          <table className="w-full">
            <tbody>
              {usersWithMessageCounts.map((user, index) => (
                <tr key={user.id || index} className="text-left">
                  <td className="w-1/12 p-2">
                    {/* First column for index */}
                    <p>{index + 1}.</p>
                  </td>
                  <td className="w-1/12 p-2">
                    {/* Second column for profile picture */}
                    <ProfilePic name={user.name} className="border" />
                  </td>
                  <td className="p-2">
                    {/* Last column for user details */}
                    <div className="flex flex-col">
                      <p className="text-sm font-medium leading-none">
                        {user.name}
                      </p>
                      <p className="text-sm text-muted-foreground">
```

```
                    {user.email}
                  </p>
                </div>
              </td>
              <td className="w-1/12 p-2 text-sm font-semibold">
                {messages.filter((m) => m.user_id == user.id).length}
              </td>
            </tr>
          ))}
        </tbody>
      </table>
    </div>
  </CardContent>
</Card>
);
}

/**
```

```
<Popover>
          <PopoverTrigger asChild>
            <Button variant="outline" size="sm" className="ml-auto">
              Owner <ChevronDown className="text-muted-foreground" />
            </Button>
          </PopoverTrigger>
          <PopoverContent className="p-0" align="end">
            <Command>
              <CommandInput placeholder="Select new role..." />
              <CommandList>
                <CommandEmpty>No roles found.</CommandEmpty>
                <CommandGroup>
                  <CommandItem className="teamaspace-y-1 flex flex-col it
ems-start px-4 py-2">
                    <p>Viewer</p>
                    <p className="text-sm text-muted-foreground">
                      Can view and comment.
                    </p>
                  </CommandItem>
                  <CommandItem className="teamaspace-y-1 flex flex-col it
ems-start px-4 py-2">
                    <p>Developer</p>
                    <p className="text-sm text-muted-foreground">
                      Can view, comment and edit.
                    </p>
                  </CommandItem>
                  <CommandItem className="teamaspace-y-1 flex flex-col it
ems-start px-4 py-2">
                    <p>Billing</p>
                    <p className="text-sm text-muted-foreground">
```

```
                    Can view, comment and manage billing.
                  </p>
                </CommandItem>
                <CommandItem className="teamaspace-y-1 flex flex-col it
ems-start px-4 py-2">
                  <p>Owner</p>
                  <p className="text-sm text-muted-foreground">
                    Admin-level access to all resources.
                  </p>
                </CommandItem>
              </CommandGroup>
            </CommandList>
          </Command>
        </PopoverContent>
      </Popover>
 */
```

/components/admin-dashboard/message-area-chart.tsx

```
"use client";

import { useEffect, useMemo, useState } from "react";
import { Area, AreaChart, CartesianGrid, XAxis } from "recharts";

import {
  Card,
  CardContent,
  CardDescription,
  CardHeader,
  CardTitle,
} from "@/components/ui/card";
import {
  ChartConfig,
  ChartContainer,
  ChartLegend,
  ChartLegendContent,
  ChartTooltip,
  ChartTooltipContent,
} from "@/components/ui/chart";
import {
  Select,
  SelectContent,
  SelectItem,
  SelectTrigger,
  SelectValue,
} from "@/components/ui/select";
import { format, parseISO, subDays } from "date-fns";
import { capitalize, cn, getDateFnsLocale } from "@/lib/utils";
import { useTranslation } from "react-i18next";
import { usePathname, useRouter, useSearchParams } from "next/navigation";
```

```
import {
  DEFAULT_START_DATE,
  ISO8601_DATE_FORMAT,
  PT_DATE_FORMAT_NO_TIME,
} from "@/global.config";
import { LightDBMessage } from "@/types/dashboard";
import { zodISODate } from "@/lib/form.schemas";
import { buttonVariants } from "../ui/button";
import { useIsMobile } from "@/hooks/use-mobile";
import { getThemeByIndex } from "@/lib/theme.colors";
import { useSettings } from "@/contexts/use-settings";
import { useTheme as useNextTheme } from "next-themes";
import { ThemeMode } from "@/types/theme";

export default function MessageAreaChart({
  messages,
}: {
  messages: LightDBMessage[];
}) {
  const now = new Date();
  const { i18n, t } = useTranslation(["dashboard-page"]);
  const data = toChartData(messages);
  const router = useRouter();
  const pathname = usePathname();
  const onMobile = useIsMobile();
  const searchParams = useSearchParams();
  const { settings } = useSettings();
  const { theme } = useNextTheme();
  const areaChartColors = [
    `hsl(${
      getThemeByIndex(settings.profileColorId || 1, theme as ThemeMode)?.pr
imary
    })`, // Current profile theme color-props
    "hsl(var(--primary))", // Current appearance theme color-props
  ];

  // This should get updated by re-renders, if not, turn it into a useState
that gets set by a useEffect
  const selectedStartDate = {
    ISO_date: searchParams.get("start_date"),
    isValid: zodISODate.safeParse(searchParams.get("start_date")).success,
  };

  function toISO(date: Date) {
    return format(date, ISO8601_DATE_FORMAT);
  }
  const selectItems = [
    {
      label: t("area_chart-week"),
      date: subDays(now, 7), // Subtract 7 days
    },
    {
      label: t("area_chart-month"),
```

```
      date: subDays(now, 30), // Subtract 30 days (assuming a 30-day month)
    },
    {
      label: t("area_chart-3_months"),
      date: subDays(now, 90), // Subtract 90 days (assuming a 30-day months
)
    },
    {
      label: t("area_chart-all_time"),
      date: new Date(DEFAULT_START_DATE),
    },
  ];

  const chartConfig = {
    amount: {
      label: t("messages_amount"),
    },
    price: {
      label: t("cost"),
    },
  } satisfies ChartConfig;

  return (
    <Card>
      <CardHeader className="flex items-center gap-2 space-y-0 border-b py-
5 sm:flex-row">
        <div className="grid flex-1 gap-1 text-center sm:text-left">
          <CardTitle>
            {t("area_chart-title")} ({data.length})
          </CardTitle>
          <CardDescription>{t("area_chart-title_caption")}</CardDescription
>
        </div>
        <Select
          defaultValue={searchParams.get("start_date") || DEFAULT_START_DAT
E}
          onValueChange={(value) => {
            const params = new URLSearchParams(searchParams);

            if (value) {
              params.set("start_date", value);
            } else {
              params.delete("start_date");
            }
            if (params.has("end_date")) params.delete("end_date");
            router.replace(`${pathname}?${params.toString()}`, {
              scroll: false, // persist current scroll for better ux
            });
          }}
        >
          <SelectTrigger
            className={cn(
              buttonVariants({ variant: "outline" }),
```

```
            "w-min appearance-none font-normal justify-between"
          )}
          // className="w-[160px] rounded-lg sm:ml-auto"
          aria-label={t("common:aria_label-select")}
        >
          <SelectValue placeholder={t("area_chart-3_months")} />
        </SelectTrigger>
        <SelectContent align={onMobile ? "center" : "end"}>
          {selectItems.map((item) => (
            <SelectItem key={item.date.getTime()} value={toISO(item.date)
}>
              {item.label}
            </SelectItem>
          ))}
          {selectedStartDate.ISO_date &&
            !selectItems.some(
              (item) => toISO(item.date) === selectedStartDate.ISO_date
            ) && (
              <SelectItem value={selectedStartDate.ISO_date} disabled>
                {selectedStartDate.isValid
                  ? format(
                      new Date(selectedStartDate.ISO_date),
                      PT_DATE_FORMAT_NO_TIME
                    )
                  : selectedStartDate.ISO_date}
              </SelectItem>
            )}
        </SelectContent>
      </Select>
    </CardHeader>

    <CardContent className="px-2 pt-4 sm:px-6 sm:pt-6">
      <ChartContainer
        config={chartConfig}
        className="aspect-auto h-[250px] w-full"
      >
        <AreaChart data={data}>
          <defs>
            {/* Gradient of the chart waves */}
            <linearGradient id="fillPrice" x1="0" y1="0" x2="0" y2="1">
              <stop
                offset="5%"
                stopColor={areaChartColors[0]}
                stopOpacity={0.8}
              />
              <stop
                offset="95%"
                stopColor={areaChartColors[0]}
                stopOpacity={0.1}
              />
            </linearGradient>
            <linearGradient id="fillAmount" x1="0" y1="0" x2="0" y2="1">
              <stop
```

```
                  offset="5%"
                  stopColor={areaChartColors[1]}
                  stopOpacity={0.8}
                />
                <stop
                  offset="95%"
                  stopColor={areaChartColors[1]}
                  stopOpacity={0.1}
                />
              </linearGradient>
            </defs>
            <CartesianGrid vertical={false} />
            <XAxis
              dataKey="date"
              tickLine={false}
              axisLine={false}
              tickMargin={8}
              minTickGap={32}
              tickFormatter={(value) => {
                return format(new Date(value), "MMM d, yyyy", {
                  locale: getDateFnsLocale(i18n.language),
                });
              }}
            />
            <ChartTooltip
              cursor={false}
              wrapperClassName="z-80"
              content={
                <ChartTooltipContent
                  className="z-80"
                  labelFormatter={(dateString: string) => {
                    // The error we were having is that between state updat
es and re-renders, sometimes the label date was not a valid date, so we nee
d to handle the date formatting gracefully to prevent a thrown error from f
ormat
                    const parsedDate = parseISO(dateString);
                    return isNaN(parsedDate.getTime()) // check if the date
is valid before trying to format it
                      ? t("invalid_date")
                      : format(parsedDate, "MMM d, yyyy", {
                          locale: getDateFnsLocale(i18n.language),
                        });
                  }}
                  indicator="dot"
                />
              }
            />
            {/* Line at the top of the chart waves */}
            <Area
              dataKey="price"
              type="natural"
              fill="url(#fillPrice)"
              stroke={areaChartColors[0]}
```

```
              stackId="a"
            />
            <Area
              dataKey="amount"
              type="natural"
              fill="url(#fillAmount)"
              stroke={areaChartColors[1]}
              stackId="a"
            />
            <ChartLegend content={<ChartLegendContent />} />
          </AreaChart>
        </ChartContainer>
      </CardContent>
    </Card>
  );
}

const toChartData = (
  messages: LightDBMessage[]
): { date: string; price: number; amount: number }[] => {
  const chartDataMap: {
    [key: string]: { totalCost: number; messageCount: number };
  } = {};

  messages.forEach((message) => {
    // Format the date to YYYY-MM-DD
    const date = message.send_time.toISOString().split("T")[0];

    // Initialize the entry for the date if it doesn't exist
    if (!chartDataMap[date]) {
      chartDataMap[date] = { totalCost: 0, messageCount: 0 };
    }

    // Increment the message count
    chartDataMap[date].messageCount += 1;

    // Add to the total cost if the cost is not null
    if (message.cost) {
      // Ensure cost is treated as a number
      const costValue =
        typeof message.cost === "string"
          ? parseFloat(message.cost)
          : message.cost;
      chartDataMap[date].totalCost += costValue;
    }
  });

  // Convert the map to an array
  return Object.entries(chartDataMap).map(([date, counts]) => ({
    date,
    price: counts.totalCost,
    amount: counts.messageCount,
```

```
  }));
};
```

/components/modals/schedule-modals.tsx

```tsx
"use client";

import React, { ChangeEvent, useEffect, useState } from "react";
import {
  Dialog,
  DialogContent,
  DialogTrigger,
  DialogTitle,
  DialogFooter,
  DialogDescription,
  DialogHeader,
} from "../ui/dialog";
import {
  AlertDialog,
  AlertDialogAction,
  AlertDialogCancel,
  AlertDialogContent,
  AlertDialogDescription,
  AlertDialogFooter,
  AlertDialogHeader,
  AlertDialogTitle,
  AlertDialogTrigger,
} from "@/components/ui/alert-dialog";
import { Calendar } from "../ui/calendar";
import { Input } from "../ui/input";
import { Label } from "../ui/label";
import { Button, buttonVariants } from "../ui/button";
import { useTranslation } from "react-i18next";
import { useModal } from "@/contexts/use-modal";
import { useNewMessage } from "@/contexts/use-new-message";

export default function ScheduleMessageModal() {
  const now = new Date();
  const { t } = useTranslation();
  const { modal, setModal } = useModal();
  const { message, setMessage } = useNewMessage();
  const [selectedDate, setSelectedDate] = useState(message.scheduledDate);

  const handleCancelButtonClick = () => {
    if (selectedDate > new Date()) {
      // date is in the future - so reset it to now
      setSelectedDate(now);
    } else {
      setModal((m) => ({ ...m, schedule: false }));
    }
```

```
  };

  const applySelectedDate = () => {
    setMessage((m) => ({
      ...m,
      scheduledDate: selectedDate,
      scheduledDateModified: true,
    }));

    setModal((m) => ({ ...m, schedule: false }));
  };

  const handleKeyPress = (event: KeyboardEvent) => {
    if (modal.schedule === true && event.key === "Enter") {
      applySelectedDate();
    }
  };

  useEffect(() => {
    // Add event listener for keydown
    document.addEventListener("keydown", handleKeyPress);

    // Cleanup the event listener on component unmount
    return () => {
      document.removeEventListener("keydown", handleKeyPress);
    };
  }, [modal.schedule]);

  return (
    <Dialog
      open={modal.schedule}
      onOpenChange={() => setModal((m) => ({ ...m, schedule: false }))}
    >
      <DialogContent className="p-6 overflow-y-auto">
        <DialogHeader className="mb-5">
          <DialogTitle>{t("modals:schedule_message-header")}</DialogTitle>
          <DialogDescription>
            {t("modals:schedule_message-header_caption")}
          </DialogDescription>
        </DialogHeader>
        <div
          className="flex flex-col gap-4 items-center xs:items-start xs:fle
x-row h-[325px] max-w-[250px] xs:max-w-full mx-auto xs:mx-0 p-0" /** This i
s the exact maximum height of the calendar */
        >
          <Calendar
            mode="single"
            selected={selectedDate}
            onSelect={(date: Date | undefined) => {
              setSelectedDate((prev) => (date ? date : prev));
            }}
            className="rounded-md border"
```

```
          />
          <div className="flex flex-col justify-between h-full w-full pb-6
xs:pb-0">
            <div /** className="flex flex-col h-full justify-center" */>
              <div className="flex flex-col gap-2 mb-3">
                <Label htmlFor="hour">
                  {t("modals:schedule_message-hour_label")}
                </Label>
                <TimeInput
                  id="hour"
                  min={0}
                  max={23}
                  value={selectedDate.getHours()}
                  onChange={(value) => {
                    setSelectedDate((prev) => new Date(prev.setHours(value)
));
                  }}
                />
              </div>
              <div className="flex flex-col gap-2 mb-3">
                <Label htmlFor="minute">
                  {t("modals:schedule_message-minute_label")}
                </Label>
                <TimeInput
                  id="minute"
                  min={0}
                  max={59}
                  value={selectedDate.getMinutes()}
                  onChange={(value) =>
                    setSelectedDate((prev) => new Date(prev.setMinutes(valu
e)))
                  }
                />
              </div>
            </div>
            <div className="flex flex-wrap gap-2 justify-end">
              <Button
                variant="outline"
                onClick={handleCancelButtonClick}
                className="flex-1"
              >
                {selectedDate > now
                  ? t("modals:schedule_message-reset")
                  : t("common:cancel")}
              </Button>
              <Button onClick={applySelectedDate} className="flex-1">
                {selectedDate > now
                  ? t("modals:schedule_message-submit")
                  : t("modals:schedule_message-submit_now")}
              </Button>
            </div>
          </div>
        </div>
```

```
      </DialogContent>
    </Dialog>
  );
}

export function ScheduleAlertModal() {
  const [shouldSubmit, setShouldSubmit] = useState(false);
  const { modal, setModal } = useModal();
  const { form } = useNewMessage();
  const { t } = useTranslation();
  const { setMessage } = useNewMessage();

  useEffect(() => {
    if (shouldSubmit) {
      // Check if the form ref is set and then call requestSubmit
      form?.requestSubmit();
      setShouldSubmit(false); // Reset the flag after submission
    }
  }, [shouldSubmit]); // This effect runs when shouldSubmit changes
  return (
    <>
      {/* "Confirm Invalid Date" dialog */}
      <AlertDialog
        open={modal.scheduleAlert}
        onOpenChange={(value) =>
          setModal((m) => ({ ...m, scheduleAlert: value }))
        }
      >
        <AlertDialogContent>
          <AlertDialogHeader>
            <AlertDialogTitle>
              {t("modals:schedule_alert-header")}
            </AlertDialogTitle>
            <AlertDialogDescription>
              {t("modals:schedule_alert-header_caption")}
            </AlertDialogDescription>
          </AlertDialogHeader>
          <AlertDialogFooter>
            <AlertDialogCancel>{t("common:cancel")}</AlertDialogCancel>
            <AlertDialogAction
              onClick={() => {
                setMessage((m) => ({ ...m, scheduledDateConfirmed: true }))
;
                // Can't submit directly from here, because we need to wait
for the scheduleDateConfirmed flag to be set
                setShouldSubmit(true);
              }}
            >
              {t("common:continue")}
            </AlertDialogAction>
          </AlertDialogFooter>
        </AlertDialogContent>
      </AlertDialog>
```

```
    </>
  );
}

// Define the props type for the TimeInput component
type TimeInputProps = {
  id: string;
  value: number;
  onChange: (value: number) => void;
  min: number;
  max: number;
};

// TimeInput component
function TimeInput({ id, value, onChange, min, max }: TimeInputProps) {
  const [displayValue, setDisplayValue] = useState<string>(
    value < 10 ? `0${value}` : value.toString()
  );
  const [isFocused, setIsFocused] = useState(false);

  const handleChange = (e: React.ChangeEvent<HTMLInputElement>) => {
    const inputValue = e.target.value;
    setDisplayValue(inputValue);

    const numericValue = Number(inputValue);
    if (numericValue >= min && numericValue <= max) {
      onChange(numericValue);
    }
  };

  const handleBlur = () => {
    setIsFocused(false);
    const numericValue = Number(displayValue);
    if (numericValue >= min && numericValue <= max) {
      setDisplayValue(
        numericValue < 10 ? `0${numericValue}` : numericValue.toString()
      );
    }
  };

  useEffect(() => {
    // reflect the current date object in the inputs whenever they change
    if (isFocused === false) {
      setDisplayValue(value < 10 ? `0${value}` : value.toString());
    }
  }, [value]);

  return (
    <Input
      id={id}
      type="number"
      min={min}
```

```
      max={max}
      value={displayValue}
      onChange={handleChange}
      onBlur={handleBlur} // Add onBlur event handler
      onFocus={() => setIsFocused(true)}
    />
  );
}
```

/components/modals/recipient-info.tsx

```
"use client";

import {
  Dialog,
  DialogContent,
  DialogDescription,
  DialogHeader,
  DialogTitle,
  DialogFooter,
} from "../ui/dialog";
import { DialogClose } from "@/components/ui/dialog";
import { useModal } from "@/contexts/use-modal";
import { Separator } from "../ui/separator";
import { CopyButton } from "../shared/copy-button";
import { Button } from "../ui/button";
import { NewRecipient } from "@/types/recipient";
import { useTranslation } from "react-i18next";
import ProfilePic from "../profile-pic";
import { useEffect, useState } from "react";

export default function RecipientInfoModal({
  recipient,
  allowContactCreation = true,
}: {
  recipient: NewRecipient;
  allowContactCreation: boolean;
}) {
  const { modal, setModal } = useModal();
  const { t } = useTranslation(["modals"]);
  const [watchCreateModalClose, setWatchCreateModalClose] = useState(false)
;

  const showCreateModal = () => {
    setModal((m) => ({ ...m, contact: { ...m.contact, info: false } }));
    setModal((m) => ({ ...m, contact: { ...m.contact, create: true } }));
    setWatchCreateModalClose(true);
  };
  useEffect(() => {
    if (watchCreateModalClose && modal.contact.create === false) {
```

```
      setWatchCreateModalClose(false);
      setModal((m) => ({ ...m, contact: { ...m.contact, info: true } }));
    }
  }, [modal.contact]);
  return (
    <Dialog
      /* We do need these shits unfortunately */
      open={modal.contact.info}
      onOpenChange={(value: boolean) =>
        setModal((m) => ({ ...m, contact: { ...m.contact, info: value } }))
      }
    >
      <DialogContent>
        <DialogHeader>
          <DialogTitle>
            {/* make it so we can interpolate a one of these translations u
sing {{name}} into the actual one */}
            {recipient.contact
              ? t("info-header_contact")
              : t("info-header_recipient")}
          </DialogTitle>
          <DialogDescription>
            {recipient.contact
              ? t("info-header_caption_contact")
              : t("info-header_caption_recipient")}
          </DialogDescription>
        </DialogHeader>
        <div className="flex flex-1 flex-col">
          <div className="flex items-start p-4">
            <div className="flex items-center gap-4 text-sm">
              <ProfilePic name={recipient.contact?.name} className="border"
/>
              <h2>{recipient.contact?.name || t("info-name_fallback")}</h2>
            </div>
          </div>
          <Separator />
          <div className="flex gap-4 justify-between items-center p-4 text-
sm">
            <div>{t("common:phone_number")}</div>
            <CopyButton text={recipient.phone} variant="none" className="pr
-0">
              {recipient.phone}
            </CopyButton>
          </div>
          {recipient.contact && ( // Contact description information
            <>
              <Separator />
              <div className="flex gap-4 justify-between p-4 text-sm">
                <p>{t("common:description")}</p>

                {recipient.contact?.description?.trim() ? (
                  <p className="text-right">{recipient.contact?.description
}</p>
```

```
                ) : (
                  <p className="italic text-right">
                    {t("common:no_description")}
                  </p>
                )}
              </div>
            </>
          )}
        </div>
        {allowContactCreation && (
          <DialogFooter>
            <DialogClose asChild>
              <Button variant="outline">{t("common:close")}</Button>
            </DialogClose>
            {!recipient.contact?.id && (
              <Button onClick={showCreateModal}>
                {t("info-button_create_contact")}
              </Button>
            )}
          </DialogFooter>
        )}
      </DialogContent>
    </Dialog>
  );
}
```

/components/modals/edit-contact.tsx

```
"use client";

import React, { useEffect, useState } from "react";
import { useActionState } from "react";
import {
  Dialog,
  DialogContent,
  DialogDescription,
  DialogHeader,
  DialogTitle,
  DialogTrigger,
  DialogFooter,
} from "../ui/dialog";
import { Button, buttonVariants } from "../ui/button";
import { Input } from "@/components/ui/input";
import { Label } from "../ui/label";
import { updateContact } from "@/lib/actions/contact.actions";
import { DBContact } from "@/types/contact";
import { ContactSchema } from "@/lib/form.schemas";
import { CircleAlert, Loader2 } from "lucide-react";
import { DialogClose } from "@/components/ui/dialog";
import { cn, toastActionResult } from "@/lib/utils";
```

```
import { Textarea } from "../ui/textarea";
import { Alert, AlertDescription } from "../ui/alert";
import { useModal } from "@/contexts/use-modal";
import { ActionResponse } from "@/types/action";
import { useTranslation } from "react-i18next";
import { useContacts } from "@/contexts/use-contacts";

const initialState: ActionResponse<undefined> = {
  success: false,
  message: [],
};

export default function EditContactModal({ contact }: { contact: DBContact
}) {
  const { modal, setModal } = useModal();
  const [serverState, action, pending] = useActionState(
    updateContact.bind(null, contact.id),
    initialState
  );
  const { refetchContacts } = useContacts();
  const { t } = useTranslation(["modals"]);

  useEffect(() => {
    if (serverState.success) {
      toastActionResult(serverState, t);
      handleOpenChange(false);
      // Refetch contacts context after mutation.
      refetchContacts();
    }
  }, [serverState]);

  const handleOpenChange = (value: boolean) => {
    setModal((m) => ({ ...m, contact: { ...m.contact, edit: value } }));
    clearInputs();
  };
  const clearInputs = () => {
    // This is unfortunately the easiest way to reset this shit
    serverState.errors = undefined;
    serverState.message = [];
    serverState.inputs = {};
  };
  return (
    <Dialog
      /* We do need these shits unfortunately */
      open={modal.contact.edit}
      onOpenChange={handleOpenChange}
    >
      <DialogContent>
        <DialogHeader>
          <DialogTitle>{t("edit_contact-header")}</DialogTitle>
          <DialogDescription>
            {t("edit_contact-header_caption")}
```

```
          </DialogDescription>
        </DialogHeader>
        <form action={action} className="space-y-6">
          <div className="space-y-2">
            <Label htmlFor="name">{t("common:name")}</Label>
            <Input
              name="name"
              id="name"
              placeholder={t("name_placeholder")}
              defaultValue={serverState.inputs?.name || contact.name}
              // required
              // minLength={5}
              // maxLength={100}
              aria-describedby="name-error"
              className={serverState.errors?.name ? "border-red-500" : ""}
            />
            {serverState.errors?.name && (
              <p id="name-error" className="text-sm text-red-500">
                {t(serverState.errors.name[0])}
              </p>
            )}
          </div>

          <div className="space-y-2">
            <Label htmlFor="phone">{t("common:phone_number")}</Label>
            <Input
              name="phone"
              id="phone"
              placeholder={t("phone_placeholder")}
              defaultValue={serverState.inputs?.phone || contact.phone}
              // required
              // minLength={5}
              // maxLength={100}
              aria-describedby="phone-error"
              className={serverState.errors?.phone ? "border-red-500" : ""}
            />
            {serverState.errors?.phone && (
              <p id="phone-error" className="text-sm text-red-500">
                {t(serverState.errors.phone[0])}
              </p>
            )}
          </div>

          <div className="space-y-2">
            <Label htmlFor="description">{t("common:description")}</Label>
            <Textarea
              name="description"
              id="description"
              placeholder={t("description_placeholder")}
              defaultValue={
                serverState.inputs?.description || contact.description
              }
              // required
```

```
              // minLength={5}
              // maxLength={100}
              aria-describedby="description-error"
              className={
                serverState.errors?.description ? "border-red-500" : ""
              }
            />
            {serverState.errors?.description && (
              <p id="description-error" className="text-sm text-red-500">
                {t(serverState.errors.description[0])}
              </p>
            )}
          </div>

          {serverState.message.length > 0 && (
            <Alert variant={serverState.success ? "default" : "destructive"
}>
              {!serverState.success && <CircleAlert className="w-4 h-4" />}
              <AlertDescription className="relative top-1">
                {t(serverState.message)}
              </AlertDescription>
            </Alert>
          )}

          <DialogFooter>
            <DialogClose
              type="button"
              className={cn(buttonVariants({ variant: "outline" }))}
            >
              {t("common:cancel")}
            </DialogClose>
            <Button type="submit" disabled={pending}>
              {pending && <Loader2 className="h-4 w-4 animate-spin" />}{" "
}
              {t("common:update")}
            </Button>
          </DialogFooter>
        </form>
      </DialogContent>
    </Dialog>
  );
}
```

/components/modals/insert-contact.tsx

```
"use client";

import { useEffect, useState } from "react";
import {
  Dialog,
```

```
  DialogContent,
  DialogDescription,
  DialogHeader,
  DialogTitle,
  DialogTrigger,
  DialogFooter,
  DialogPortal,
  DialogClose,
} from "../ui/dialog";
import { Button, buttonVariants } from "../ui/button";
import { cn } from "@/lib/utils";
import {
  Table,
  TableBody,
  TableCaption,
  TableCell,
  TableHead,
  TableHeader,
  TableRow,
} from "@/components/ui/table";
import { Checkbox } from "../ui/checkbox";
import { useNewMessage } from "@/contexts/use-new-message";
import { useModal } from "@/contexts/use-modal";
import { DBContact } from "@/types/contact";
import { useTranslation } from "react-i18next";
import { useContacts } from "@/contexts/use-contacts";

export default function InsertContactModal() {
  const { contacts } = useContacts();
  const { modal, setModal } = useModal();
  const { addRecipient, showInfoAbout, message, removeRecipient } =
    useNewMessage();
  const initialSelected: DBContact[] = [];
  message.recipients.forEach((r) => {
    const contactInMessage = contacts.find((c) => c.phone === r.phone);
    if (contactInMessage) initialSelected.push(contactInMessage);
  });

  const [selected, setSelected] = useState<DBContact[]>(initialSelected);
  const { t } = useTranslation(["modals", "common"]);
  const [watchCreateModalClose, setWatchCreateModalClose] = useState(false)
;

  // Only those contacts that are selected here should be inside the messag
e object
  const onInsert = () => {
    // 1. Remove the ones from the message that were deselected here
    const deselectedContacts = contacts.filter(
      (contact) =>
        !selected.some((selectedContact) => selectedContact === contact)
    );
    message.recipients.map((recipient) => {
      if (deselectedContacts.find((c) => c.phone === recipient.phone)) {
```

```
      removeRecipient(recipient);
    }
  });

  // 2. Add the ones that don't exist yet.
  const contactsNotInMessage = contacts.filter(
    (contact) =>
      !message.recipients.some(
        (messageContact) => messageContact.phone === contact.phone
      )
  );

  selected.forEach((selectedContact: DBContact) => {
    // pass add each selected selectedContact to the recipients context
    if (
      contactsNotInMessage.find(
        (notContact) => notContact.phone === selectedContact.phone
      )
    ) {
      addRecipient(selectedContact.phone);
    }
  });

  // close the modal
  setInsertModal(false);
};

const onSelectOne = (contact: DBContact) => {
  const isSelected = !!selected.find((item) => item.id === contact.id);
  isSelected
    ? // it is already checked, so uncheck it:
      setSelected((prevSelected) =>
        prevSelected.filter((s) => s.id !== contact.id)
      )
    : // it is not checked yet, so add it to the selectedArr
      setSelected((prevSelected) => [...prevSelected, contact]);
};
const onSelectAll = () => {
  selected.length === contacts.length
    ? setSelected([])
    : setSelected(contacts);
};
const showCreateModal = () => {
  showInfoAbout(null);
  setInsertModal(false);
  setModal((m) => ({ ...m, contact: { ...m.contact, create: true } }));
  setWatchCreateModalClose(true);
};
const setInsertModal = (value: boolean) => {
  setModal((m) => ({ ...m, contact: { ...m.contact, insert: value } }));
};
```

```
  useEffect(() => {
    if (watchCreateModalClose && modal.contact.create === false) {
      setWatchCreateModalClose(false);
      setInsertModal(true);
    }
    if (modal.contact.insert) setSelected(initialSelected);
  }, [modal.contact]);
  return (
    <>
      <Dialog open={modal.contact.insert} onOpenChange={setInsertModal}>
        <DialogContent>
          <DialogHeader>
            <DialogTitle>{t("insert_contact-header")}</DialogTitle>
            <DialogDescription>
              {t("insert_contact-header_caption")}
            </DialogDescription>
          </DialogHeader>
          {contacts.length ? (
            <div className="max-h-[400px] overflow-auto">
              <Table>
                {/* <TableCaption>A list of your contacts.</TableCaption> *
/}
                <TableHeader>
                  <TableRow className="cursor-pointer" onClick={onSelectAll
}>
                    <TableHead className="flex items-center">
                      <Checkbox
                        className="w-6 h-6 rounded-md"
                        checked={selected.length === contacts.length}
                        onClick={onSelectAll}
                      />
                    </TableHead>
                    <TableHead>{t("common:name")}</TableHead>
                    <TableHead>{t("common:phone_number")}</TableHead>
                  </TableRow>
                </TableHeader>
                <TableBody>
                  {contacts.map((contact) => {
                    const isSelected = !!selected.find(
                      (item) => item.id === contact.id
                    );
                    return (
                      <TableRow
                        key={contact.id}
                        className="cursor-pointer"
                        onClick={() => onSelectOne(contact)}
                      >
                        <TableCell
                          // This fixes the layout shifting
                          className="flex items-center h-[36.5px] font-medi
um"
                        >
                          <Checkbox
```

```
                                    className="h-6 w-6 rounded-md mb-1"
                                    style={{
                                      height: "24px !important",
                                      width: "24px !important",
                                    }}
                                    checked={isSelected}
                                    onClick={() => onSelectOne(contact)}
                                  />
                                </TableCell>
                                <TableCell>{contact.name}</TableCell>
                                <TableCell>{contact.phone}</TableCell>
                              </TableRow>
                            );
                          })}
                        </TableBody>
                      </Table>
                    </div>
                  ) : (
                    <div className="flex flex-col items-center gap-4 py-4">
                      <DialogDescription className="self-start sm:self-center text-
center text-red-400">
                        {t("insert_contact-no_contacts")}
                      </DialogDescription>
                      <Button
                        className="w-min"
                        onClick={() => {
                          setInsertModal(false);
                          showCreateModal();
                        }}
                      >
                        {t("insert_contact-button_create_new")}
                      </Button>
                    </div>
                  )}
                  <DialogFooter>
                    <DialogClose
                      className={cn(
                        "w-full sm:w-min",
                        buttonVariants({ variant: "outline" })
                      )}
                    >
                      {t("common:cancel")}
                    </DialogClose>
                    {contacts.length !== 0 && (
                      <Button
                        onClick={onInsert}
                        /* uncomment this if you prefer
                          disabled={!selected.length} */
                      >
                        {selected.length === 1
                          ? t("insert_contact-button_insert_one")
                          : t("insert_contact-button_insert_x", {
                              amount: selected.length,
```

```
                    })}
                </Button>
              )}
            </DialogFooter>
          </DialogContent>
        </Dialog>
      </>
    );
}
```

/components/modals/create-contact.tsx

```
"use client";

import React, { useEffect, useState } from "react";
import { useActionState } from "react";
import {
  Dialog,
  DialogContent,
  DialogDescription,
  DialogHeader,
  DialogTitle,
  DialogTrigger,
  DialogFooter,
} from "../ui/dialog";
import { Button, buttonVariants } from "../ui/button";
import { Input } from "@/components/ui/input";
import { Label } from "../ui/label";
import { createContact } from "@/lib/actions/contact.actions";
import { CircleAlert, Loader2 } from "lucide-react";
import { DialogClose } from "@/components/ui/dialog";
import { cn, toastActionResult } from "@/lib/utils";
import { Textarea } from "../ui/textarea";
import { Alert, AlertDescription } from "../ui/alert";
import { useModal } from "@/contexts/use-modal";
import { CreateContactResponse } from "@/types/action";
import { useTranslation } from "react-i18next";
import { DBContact } from "@/types/contact";
import { useContacts } from "@/contexts/use-contacts";

const initialState: CreateContactResponse = {
  success: false,
  message: [],
};

export default function CreateContactModal({
  defaultPhone,
  onCreateSuccess,
}: {
  defaultPhone?: string;
```

```
  onCreateSuccess?: (contact: DBContact) => void;
}) {
  const { modal, setModal } = useModal();
  const [serverState, action, pending] = useActionState(
    createContact,
    initialState
  );
  const { refetchContacts } = useContacts();
  const { t } = useTranslation(["modals"]);

  useEffect(() => {
    if (serverState.success) {
      toastActionResult(serverState, t);
      // Refetch contacts context after creation.
      refetchContacts();
      handleOpenChange(false);
      if (onCreateSuccess && serverState.data)
        onCreateSuccess(serverState.data);
    }
  }, [serverState]);

  const handleOpenChange = (value: boolean) => {
    setModal((m) => ({ ...m, contact: { ...m.contact, create: value } }));
    clearInputs();
  };
  const clearInputs = () => {
    // This is unfortunately the easiest way to reset this shit
    serverState.errors = undefined;
    serverState.message = [];
    serverState.inputs = {};
  };
  return (
    <Dialog
      /* We do need these shits unfortunately */
      open={modal.contact.create}
      onOpenChange={handleOpenChange}
    >
      <DialogContent>
        <DialogHeader>
          <DialogTitle>{t("create_contact-header")}</DialogTitle>
          <DialogDescription>
            {t("create_contact-header_caption")}
          </DialogDescription>
        </DialogHeader>
        <form action={action} className="space-y-6">
          <div className="space-y-2">
            <Label htmlFor="name">{t("common:name")}</Label>
            <Input
              name="name"
              id="name"
              placeholder={t("name_placeholder")}
              defaultValue={serverState.inputs?.name}
              // required
```

```
        // minLength={5}
        // maxLength={100}
        aria-describedby="name-error"
        className={serverState.errors?.name ? "border-red-500" : ""}
      />
      {serverState.errors?.name && (
        <p id="name-error" className="text-sm text-red-500">
          {t(serverState.errors.name[0])}
        </p>
      )}
    </div>

    <div className="space-y-2">
      <Label htmlFor="phone">{t("common:phone_number")}</Label>
      <Input
        name="phone"
        id="phone"
        placeholder={t("phone_placeholder")}
        defaultValue={serverState.inputs?.phone || defaultPhone}
        // required
        // minLength={5}
        // maxLength={100}
        aria-describedby="phone-error"
        className={serverState.errors?.phone ? "border-red-500" : ""}
      />
      {serverState.errors?.phone && (
        <p id="phone-error" className="text-sm text-red-500">
          {t(serverState.errors.phone[0])}
        </p>
      )}
    </div>

    <div className="space-y-2">
      <Label htmlFor="description">{t("common:description")}</Label>
      <Textarea
        name="description"
        id="description"
        placeholder={t("description_placeholder")}
        defaultValue={serverState.inputs?.description}
        // required
        // minLength={5}
        // maxLength={100}
        aria-describedby="description-error"
        className={
          serverState.errors?.description ? "border-red-500" : ""
        }
      />
      {serverState.errors?.description && (
        <p id="description-error" className="text-sm text-red-500">
          {t(serverState.errors.description[0])}
        </p>
      )}
    </div>
```

```
          {serverState.message.length > 0 && (
            <Alert variant={serverState.success ? "default" : "destructive"
}>
              {!serverState.success && <CircleAlert className="w-4 h-4" />}
              <AlertDescription className="relative top-1">
                {t(serverState.message.join(", "))}
              </AlertDescription>
            </Alert>
          )}

          <DialogFooter>
            <DialogClose
              type="button"
              className={cn(buttonVariants({ variant: "outline" }))}
            >
              {t("common:cancel")}
            </DialogClose>
            <Button type="submit" disabled={pending}>
              {pending && <Loader2 className="h-4 w-4 animate-spin" />}{" "
}
              {t("common:create")}
            </Button>
          </DialogFooter>
        </form>
      </DialogContent>
    </Dialog>
  );
}
```

/components/messages-page-skeleton.tsx

```
"use client";

import ChildrenPanel from "./shared/children-panel";
import { ResizableHandle, ResizablePanel } from "./ui/resizable";
import { useLayout } from "@/contexts/use-layout";
import { useTranslation } from "react-i18next";

import { cn } from "@/lib/utils";
import MessageDisplay from "./message-display";
import { useIsMobile } from "@/hooks/use-mobile";
import { PageHeader } from "./headers";
import Skeleton from "react-loading-skeleton";
import "react-loading-skeleton/dist/skeleton.css";
import { AmountIndicators, CategoryEnums } from "@/types";
import { ModalProvider } from "@/contexts/use-modal";

export default function MessagesPageSkeleton({
  category,
```

```
}: {
  category: CategoryEnums;
}) {
  const { layout, fallbackLayout, amountIndicators } = useLayout();
  const { t } = useTranslation(["messages-page", "common"]);
  const onMobile = useIsMobile();
  const selected = null;
  const skeletonsAmount: number = amountIndicators
    ? amountIndicators[category.toLowerCase() as keyof AmountIndicators]
    : 4;
  return (
    <>
      <ResizablePanel
        className={cn(onMobile && selected !== null && "hidden")} // If we
are on mobile and a message is selected we only want to show the column con
taining the selected message.
        // Check if the layout is a 3-column middle-bar panel. Use the prev
ious 3-column layout if available; otherwise, render the fallback for diffe
rent or undefined layouts.
        defaultSize={
          Array.isArray(layout) && layout.length === 3
            ? layout[1]
            : fallbackLayout[1]
        }
        minSize={22}
        maxSize={50}
      >
        <PageHeader title={t(`header_${category.toLowerCase()}`)} />

        <div className="rounded-md p-4 h-[68px]">
          <Skeleton className="h-9" style={{ borderRadius: "0.375rem" }} />
        </div>

        <div className="flex flex-col gap-2 p-4 pt-0 mt-2 overflow-hidden">
          {skeletonsAmount > 0 ? (
            // Math.min() makes it so that the maximum will be x, even if t
he variable has a larger number
            Array.from({ length: Math.min(skeletonsAmount, 10) }).map(
              (_, i) => {
                return <MessageSkeleton key={i} />;
              }
            )
          ) : (
            <div className="p-8 text-center text-muted-foreground">
              <Skeleton className="w-full" />
            </div>
          )}
        </div>
      </ResizablePanel>
      <ResizableHandle withHandle className={cn(onMobile && "hidden")} />
      <ChildrenPanel
        hasMiddleBar
        className={cn(onMobile && selected === null && "hidden")} // like a
```

```
bove we are using reverse logic here. If we are on mobile, and nothing is s
elected, this component should not be displayed.
      >
        {/* If you need other modals somewhere else, move the provider up t
he component tree. And don't forget to update the skeleton too! */}
        <ModalProvider>
          <MessageDisplay message={null} reset={() => {}} category={categor
y} />
        </ModalProvider>
      </ChildrenPanel>
    </>
  );
}

function MessageSkeleton() {
  return (
    <div className="flex flex-col items-start gap-2 rounded-lg border p-3 t
ext-left text-sm">
      <div className="flex w-full flex-col gap-1">
        <div className="flex items-center h-[20px] w-full">
          <h2 className="flex-1 mr-20">
            <Skeleton />
          </h2>
          <p className="w-[15%] self-center">
            <Skeleton height={16} />
          </p>
        </div>
        <div className="text-xs font-medium w-[40%] ">
          <Skeleton />
        </div>
      </div>
      <div className="line-clamp-2 text-xs text-muted-foreground w-full">
        <Skeleton count={2} />
      </div>
    </div>
  );
}
```

/components/shared/copy-button.tsx

```
"use client";

import React, { useState, useEffect, useRef, MouseEvent } from "react";
import { Copy, Check } from "lucide-react";
import { Button } from "@/components/ui/button";
import { cn } from "@/lib/utils";
import { toast } from "sonner";
import { useTranslation } from "react-i18next";

interface CopyButtonProps {
```

```
  children?: React.ReactNode;
  text: string;
  className?: string;
  variant?: "outline" | "none" | "ghost" | "link";
  size?: "sm" | "lg";
}

export function CopyButton({
  children,
  text,
  className = "",
  variant,
  size,
}: CopyButtonProps) {
  const [copied, setCopied] = useState(false);
  const timerRef = useRef<NodeJS.Timeout | null>(null);
  const { t } = useTranslation(["common"]);
  const successMessage = t("copy_btn-success");

  // We need this complex logic or it won't work in some browsers
  const handleCopy = async (e: MouseEvent<HTMLButtonElement>) => {
    if (!copied) {
      try {
        // Check if the Clipboard API is supported
        if (navigator.clipboard) {
          await navigator.clipboard.writeText(text);
          setCopied(true);
          toast.success(successMessage); // Notify success
        } else {
          // Fallback for browsers that do not support the Clipboard API
          const textarea = document.createElement("textarea");
          textarea.value = text;
          textarea.style.position = "fixed"; // Prevent scrolling to bottom
of page in MS Edge.
          textarea.style.opacity = "0"; // Make it invisible
          textarea.setAttribute("readonly", ""); // Make it read-only
          document.body.appendChild(textarea);
          textarea.select();
          const successful = document.execCommand("copy");
          document.body.removeChild(textarea);

          if (successful) {
            setCopied(true);
            toast.success(successMessage); // Notify success
          } else {
            throw new Error("Copy command was unsuccessful.");
          }
        }

        if (timerRef.current) clearTimeout(timerRef.current);
        timerRef.current = setTimeout(() => setCopied(false), 2000);
      } catch (error) {
```

```
        // Handle any errors that occur during the copy process
        toast.error("copy_btn-error");
      }
    }
  };

  return (
    <Button
      variant={variant}
      size={size}
      className={cn(className, "flex items-center")}
      onClick={handleCopy}
    >
      {copied ? (
        <Check style={{ width: ".8rem", height: ".8rem" }} />
      ) : (
        <Copy style={{ width: ".8rem", height: ".8rem" }} />
      )}{" "}
      <span>{children}</span>
    </Button>
  );
}
```

/components/shared/account.tsx

```
"use client";
import React, { useState } from "react";
import ProfilePic from "../profile-pic";
import { useSession } from "@/hooks/use-session";
import { cn } from "@/lib/utils";
import { useTranslation } from "react-i18next";
import {
  DropdownMenu,
  DropdownMenuContent,
  DropdownMenuGroup,
  DropdownMenuItem,
  DropdownMenuLabel,
  DropdownMenuSeparator,
  DropdownMenuTrigger,
} from "@/components/ui/dropdown-menu";
import Link from "next/link";
import { LogOut, MonitorCog, Settings, UserRoundPen } from "lucide-react";
import { useSettings } from "@/contexts/use-settings";
import { logout } from "@/lib/auth";
import { usePathname, useRouter } from "next/navigation";
import { getThemeByIndex, themes } from "@/lib/theme.colors";
import { useTheme as useNextTheme } from "next-themes";
import { ThemeMode } from "@/types/theme";

export default function Account({
```

```
  hideNameRole = false,
  hideNameRoleOnXS,
  profilePicPosition = "LEFT",
  className,
}: {
  hideNameRole?: boolean;
  hideNameRoleOnXS?: boolean;
  profilePicPosition?: "LEFT" | "RIGHT";
  className?: string;
}) {
  const { t } = useTranslation(["common"]);
  const { session, loading } = useSession();
  const { theme } = useNextTheme();

  const pathname = usePathname();
  const router = useRouter();
  const { settings, resetLocalSettings } = useSettings();

  const handleLogout = async () => {
    const { success } = await logout();
    if (success) {
      resetLocalSettings();
      router.push("/login");
    }
  };

  return (
    <div
      // className={className}
      className={cn(
        "flex h-[var(--header-header-height)] items-center justify-center",
// border-b
        className
      )}
    >
      <DropdownMenu>
        <DropdownMenuTrigger
          className={cn(
            "flex gap-3 items-center justify-start w-full focus-primary-rin
g",
            hideNameRole && "w-9 h-9"
          )}
        >
          <ProfilePic
            size={9}
            name={settings.displayName}
            colorObj={getThemeByIndex(
              settings.profileColorId || 1,
              theme as ThemeMode
            )}
            loading={loading}
            className={cn(profilePicPosition === "RIGHT" && "order-2")}
          />
```

```
          <div
            className={cn(
              "flex flex-col",
              profilePicPosition === "RIGHT" && "items-end", // align the t
ext to the right depending on layout
              (hideNameRole || loading) && "hidden",
              hideNameRoleOnXS && "hidden xs:flex"
            )}
          >
            <p className="font-semibold mb-[-3px]">
              {settings.displayName || t("common:account-no_name")}
            </p>
            <p className="text-xs text-muted-foreground text-start">
              {session?.isAdmin ? t("common:admin") : t("common:user")}
            </p>
          </div>
        </DropdownMenuTrigger>
        <DropdownMenuContent
          align={profilePicPosition === "LEFT" ? "start" : "end"}
          className="z-10"
        >
          <DropdownMenuGroup>
            <Link href="/settings#profile">
              <DropdownMenuItem>
                <UserRoundPen />
                {t("common:account-edit_profile")}
              </DropdownMenuItem>
            </Link>
            <Link href="/settings">
              <DropdownMenuItem>
                <Settings />
                {t("common:account-settings")}
              </DropdownMenuItem>
            </Link>
            {session?.isAdmin && (
              <Link href={pathname.includes("/dashboard") ? "/" : "/dashboa
rd"}>
                <DropdownMenuItem>
                  <MonitorCog />
                  {pathname.includes("/dashboard")
                    ? t("common:account-dashboard_leave")
                    : t("common:account-dashboard_enter")}
                </DropdownMenuItem>
              </Link>
            )}
          </DropdownMenuGroup>
          <DropdownMenuSeparator />
          <DropdownMenuItem onClick={handleLogout}>
            <LogOut />
            {t("common:account-log_out")}
          </DropdownMenuItem>
        </DropdownMenuContent>
      </DropdownMenu>
```

```
    </div>
  );
}
```

/components/shared/search.tsx

```
"use client";

import { Input } from "@/components/ui/input";
import { Search as SearchIcon } from "lucide-react";

type SearchProps = React.InputHTMLAttributes<HTMLInputElement> & {
  onSearch: (term: string) => void;
};

export default function Search({ onSearch, ...props }: SearchProps) {
  const url = new URL(window.location.href);
  const params = new URLSearchParams(url.search);
  const handleSearch = (term: string) => {
    // update data by calling parent function
    onSearch(term);

    // Use vanilla javascript to update the url.
    // According to the Next.js docs we should use useSearchParams, usePath
name, and useRouter, but that causes the component to re-render and re-fetc
h data.
    // For optimization purposes, we just fetch once for each page, and the
n filter that data using client-side javascript.

    // Update the search parameter or delete it if search bar is empty
    if (term) {
      params.set("query", term);
    } else {
      params.delete("query");
    }

    // Update the URL quietly without reloading the page
    url.search = params.toString();
    window.history.pushState({}, "", url);
  };
  return (
    <div className="p-4">
      <div className="relative">
        <SearchIcon className="absolute left-2 top-2.5 h-4 w-4 text-muted-f
oreground" />
        <Input /** this input is not part of a form, we are just using the
input element as it has handy event listeners */
          onChange={(e) => {
            handleSearch(e.target.value);
          }}
```

```
            className="focus-visible:ring-1 focus-visible:ring-primary"
            defaultValue={params.get("query")?.toString()}
            {...props}
          />
        </div>
      </div>
    );
}
```

/components/shared/children-panel.tsx

```
"use client";

import { ResizablePanel } from "../ui/resizable";
import { useLayout } from "@/contexts/use-layout";

export default function ChildrenPanel({
  children,
  hasMiddleBar,
  className,
}: {
  children: Readonly<React.ReactNode>;
  hasMiddleBar?: boolean;
  className?: string;
}) {
  const { layout, fallbackLayout } = useLayout();
  const middleBarWidth =
    Array.isArray(layout) && layout.length === 3 ? layout[2] : undefined;

  const fallbackWidth = Array.isArray(layout)
    ? 100 - layout[0]
    : fallbackLayout[0];

  return (
    <ResizablePanel
      // width at null means don't specify any width, if it has a value use
that, else use fallback
      defaultSize={hasMiddleBar ? middleBarWidth : fallbackWidth}
      className={className}
    >
      {children}
    </ResizablePanel>
  );
}
```

/components/shared/error-component.tsx

```
"use client";
import { Frown } from "lucide-react";

type ErrorComponentProps = {
  children?: React.ReactNode;
  title: string;
  subtitle: string;
};
export default function ErrorComponent({
  children,
  title,
  subtitle,
}: ErrorComponentProps) {
  return (
    <div className="h-full flex flex-col items-center justify-center gap-3"
>
      <div className="flex flex-col items-center gap-1">
        <Frown className="text-muted-foreground h-10 w-10 stroke-[1.2px]" /
>
        <div className="flex flex-col items-center">
          <h2>{title}</h2>
          <p className="text-sm">{subtitle}</p>
        </div>
      </div>
      {children}
    </div>
  );
}
```

/components/shared/submit-button.tsx

```
import React from "react";
import { Button } from "../ui/button";
import { Loader2 } from "lucide-react";
import { useFormStatus } from "react-dom";

export default function SubmitButton({
  children,
  ...props
}: React.ButtonHTMLAttributes<HTMLButtonElement>) {
  const { pending } = useFormStatus();
  return (
    <Button disabled={pending} {...props}>
      {pending && <Loader2 className="w-4 h-4 animate-spin" />}
      {children}
    </Button>
  );
}
```

/components/shared/unload-listener.tsx

```
"use client";
import React, { useEffect } from "react";

export default function UnloadListener() {
  useEffect(() => {
    const handleBeforeUnload = (event: Event) => {
      const message =
        "You have unsaved changes. Are you sure you want to leave this page
?";
      event.preventDefault();
      event.returnValue = !!message; // For most browsers
      return message; // For some older browsers
    };

    window.addEventListener("beforeunload", handleBeforeUnload);

    // Cleanup function to remove the event listener
    return () => {
      window.removeEventListener("beforeunload", handleBeforeUnload);
    };
  }, []);
  return <></>;
}
```

/components/shared/input.tsx

```
import * as React from "react";

import { cn } from "@/lib/utils";

const Input = React.forwardRef<HTMLInputElement, React.ComponentProps<"inpu
t">>(
  ({ className, type, ...props }, ref) => {
    return (
      <input
        type={type}
        className={cn(
          "focus-visible:ring-b-1 focus-visible:ring-ring flex h-9 w-full r
ounded-md bg-transparent px-3 py-1 text-base shadow-sm transition-colors fi
le:border-0 file:bg-transparent file:text-sm file:font-medium file:text-acc
ent-foreground placeholder:text-muted-foreground focus-visible:outline-none
disabled:cursor-not-allowed disabled:opacity-50 md:text-sm",
          className
        )}
        ref={ref}
```

```
        {...props}
      />
    );
  }
);
Input.displayName = "Input";

export { Input };
```

/components/clock-icon.tsx

```
import React, { JSX } from "react";
import {
  Clock1,
  Clock2,
  Clock3,
  Clock4,
  Clock5,
  Clock6,
  Clock7,
  Clock8,
  Clock9,
  Clock10,
  Clock11,
  Clock12,
} from "lucide-react";

function ClockIcon({ hour }: { hour: number }) {
  // Ensure the hour is between 1 and 12
  const validHour = Math.max(1, Math.min(12, hour));

  // Map the hour to the corresponding icon
  const icons: { [key: number]: JSX.Element } = {
    1: <Clock1 />,
    2: <Clock2 />,
    3: <Clock3 />,
    4: <Clock4 />,
    5: <Clock5 />,
    6: <Clock6 />,
    7: <Clock7 />,
    8: <Clock8 />,
    9: <Clock9 />,
    10: <Clock10 />,
    11: <Clock11 />,
    12: <Clock12 />,
  };

  return <div className="flex-centered h-4 w-4">{icons[validHour]}</div>;
}
```

```
export default ClockIcon;
```

/components/resizable-panel-wrapper.tsx

```
"use client";

import { ResizablePanelGroup } from "@/components/ui/resizable";
import { useLayout } from "@/contexts/use-layout";

export default function ResizablePanelWrapper({
  children,
}: Readonly<{ children: React.ReactNode }>) {
  const { setLayout } = useLayout();

  return (
    <ResizablePanelGroup
      direction="horizontal"
      onLayout={(sizes: number[]) => {
        setLayout(sizes);
        const cookieValue = JSON.stringify(sizes);
        const cookiePath = "/"; // Specify a url path. The layout should be
the same, no matter where it got saved.
        document.cookie = `react-resizable-panels:layout:app=${cookieValue}
; path=${cookiePath};`;
      }}
      className="h-full items-stretch"
    >
      {children}
    </ResizablePanelGroup>
  );
}
```

/components/messages-list.tsx

```
"use client";

import { cn, getDateFnsLocale } from "@/lib/utils";
import { ScrollArea } from "@/components/ui/scroll-area";
import { ComponentProps } from "react";
import { formatDistanceToNow } from "date-fns/formatDistanceToNow";
import { Badge } from "@/components/ui/badge";
import type { DBMessage } from "@/types";
import { useTranslation } from "react-i18next";
import ClockIcon from "./clock-icon";
import { useIsMobile } from "@/hooks/use-mobile";
import { Button } from "./ui/button";
```

```
type MessageListProps = {
  messages: DBMessage[];
  selectedMessageId: string | null;
  setSelected: (message: DBMessage) => void;
};

export function MessageList({
  messages,
  selectedMessageId,
  setSelected,
}: MessageListProps) {
  const { t, i18n } = useTranslation(["messages-page"]);
  const onMobile = useIsMobile();

  return (
    <ScrollArea
      className={
        onMobile
          ? `h-[calc(100vh-var(--simple-header-height)-68px)]`
          : `h-[calc(100vh-var(--header-height)-68px)]`
      }
    >
      <div className="flex flex-col gap-2 p-4 pt-0">
        {messages.map((message) => {
          const sendInFuture = message.send_time.getTime() > Date.now();
          const statusTranslationString = (
            message.status !== "SCHEDULED"
              ? message.status
              : sendInFuture
              ? "SCHEDULED"
              : "SENT"
          ).toLowerCase();
          return (
            <Button
              key={message.id}
              variant="ghost"
              className={cn(
                "h-full flex flex-col items-start gap-2 rounded-lg border p
-3 text-left mt-[1px]",
                selectedMessageId === message.id && "bg-accent"
              )}
              onClick={() => setSelected(message)}
            >
              <div className="flex w-full flex-col">
                <div className="flex items-center gap-1">
                  <div className="flex items-center gap-2">
                    <div className="font-semibold">
                      {message.subject
                        ? message.subject
                        : t("common:no_subject")}
                    </div>
                    {sendInFuture && message.status === "SCHEDULED" && (
```

```
                    <ClockIcon
                      hour={
                        Math.round(message.send_time.getHours() % 12) ||
12
                      }
                    />
                  )}
                  {message.status === "FAILED" && (
                    <div className="flex items-center gap-1 text-destruct
ive text-xs">
                      <div className="flex h-2 w-2 rounded-full bg-destru
ctive" />
                      {message.api_error_code}
                    </div>
                  )}
                </div>
                <div
                  className={cn(
                    "ml-auto text-xs",
                    selectedMessageId === message.id
                      ? "text-foreground"
                      : "text-muted-foreground"
                  )}
                >
                  {formatDistanceToNow(new Date(message.send_time), {
                    addSuffix: true,
                    locale: getDateFnsLocale(i18n.language),
                  })}
                </div>
              </div>
            </div>
            <div className="line-clamp-2 text-xs text-muted-foreground">
              {message.body.substring(0, 300)}
            </div>

            {/* If we are on the trash page, render a badge to show what
the message was before it got moved to the trash */}
            {message.in_trash == true && (
              <Badge
                variant="outline"
                className="tracking-widest text-xs text-muted-foreground"
                style={{ letterSpacing: "1px" }}
              >
                {/* Play around with the styles */}
                {t(`status_${statusTranslationString}`).toUpperCase()}
              </Badge>
            )}
          </Button>
        );
      })}
    </div>
  </ScrollArea>
);
```

```
}

function getBadgeVariantFromLabel(
  label: string
): ComponentProps<typeof Badge>["variant"] {
  // if (["success"].includes(label.toLowerCase())) {
  //   return "positive";
  // }

  if (["FAILED"].includes(label.toLowerCase())) {
    return "destructive";
  }

  if (["SCHEDULED"].includes(label.toLowerCase())) {
    return "outline";
  }

  return "secondary";
}
```

/components/cards.tsx

```
import React from "react";
import { Card, CardHeader, CardTitle } from "./ui/card";
import Link from "next/link";

type LinkCardProps = {
  href: string;
  title: string;
  heroValue: string | number;
  Icon: any;
};
export default function LinkCard({
  href,
  title,
  heroValue,
  Icon,
}: LinkCardProps) {
  return (
    <Link
      href={href}
      className="flex-1 max-w-[350px]  focus-primary-ring rounded-xl"
    >
      <Card className="shadow-none hover:bg-muted dark:hover:bg-muted relat
ive overflow-hidden ">
        <CardHeader>
          <div className="flex justify-between items-center gap-8">
            <div>
              <CardTitle>{title}</CardTitle>
              <h1 className="font-medium leading-tight">{heroValue}</h1>
```

```
            </div>
            <div className="">
              <Icon
                fill="hsl(var(--primary))"
                height={65}
                width={65}
                className="absolute rotate-[-15deg] bottom-[-3px] right-[25
px] opacity-25"
              />
              {/* you may change the order of these to see what works best
*/}
              <Icon
                fill="hsl(var(--primary))"
                height={70}
                width={70}
                className="absolute rotate-[-7deg] bottom-[-2px] right-[-5p
x] opacity-80"
              />
            </div>
          </div>
        </CardHeader>
      </Card>
    </Link>
  );
}
```

/components/contacts-page.tsx

```
"use client";

import React, { useEffect, useState } from "react";
import ChildrenPanel from "./shared/children-panel";
import { ResizableHandle, ResizablePanel } from "./ui/resizable";
import { useLayout } from "@/contexts/use-layout";
import { PageHeader } from "./headers";
import { useTranslation } from "react-i18next";
import ContactsList from "./contacts-list";

import { cn, searchContacts } from "@/lib/utils";
import ContactDisplay from "./contact-display";
import { useIsMobile } from "@/hooks/use-mobile";
import Search from "./shared/search";
import { useRouter, useSearchParams } from "next/navigation";
import { CirclePlus, Plus } from "lucide-react";
import { Button } from "./ui/button";
import { useModal } from "@/contexts/use-modal";
import { DBContact } from "@/types/contact";
import CreateContactModal from "./modals/create-contact";
import useIsMounted from "@/hooks/use-mounted";
import { useContacts } from "@/contexts/use-contacts";
```

```
export default function ContactsPage() {
  const { layout, fallbackLayout } = useLayout();
  const { t } = useTranslation(["contacts-page"]);
  const { contacts, contactFetchError } = useContacts();
  const [filteredContacts, setFilteredContacts] = useState(contacts);
  const onMobile = useIsMobile();
  const isMounted = useIsMounted();
  const { modal, setModal } = useModal();

  const [selected, setSelected] = useState<DBContact | null>(
    filteredContacts[0] || null
  );

  const searchParams = useSearchParams();

  const onSearch = (searchTerm: string) => {
    setFilteredContacts(searchContacts(contacts, searchTerm));
  };
  const showCreateModal = () => {
    setModal((m) => ({
      ...m,
      contact: { ...m.contact, create: true },
    }));
  };

  useEffect(() => {
    const oldSelected = contacts.find((c) => c.id === selected?.id);
    setFilteredContacts(searchContacts(contacts, searchParams.get("query"))
);
    if (oldSelected) {
      // Keep the current selection
      setSelected(oldSelected);
    }
  }, [contacts]);

  useEffect(() => {
    if (isMounted && onMobile) {
      // On mobile, it should show the list by default without having the f
irst one selected like on desktop.
      setSelected(null);
    }
  }, [isMounted]);

  return (
    <>
      <CreateContactModal
        onCreateSuccess={(contact: DBContact) => setSelected(contact)}
      />
      <ResizablePanel
        className={cn("relative", onMobile && selected !== null && "hidden"
)} // If we are on mobile and a contact is selected we only want to show th
```

```
e column containing the selected contact.
        // Check if the layout is a 3-column middle-bar panel. Use the prev
ious 3-column layout if available; otherwise, render the fallback for diffe
rent or undefined layouts.
        defaultSize={
          Array.isArray(layout) && layout.length === 3
            ? layout[1]
            : fallbackLayout[1]
        }
        minSize={22}
        maxSize={50}
      >
        <PageHeader title={t("header")}>
          {!onMobile && (
            <Button size="sm" onClick={showCreateModal}>
              <CirclePlus />
              {t("new")}
            </Button>
          )}
        </PageHeader>
        <Search
          onSearch={onSearch}
          placeholder={t("search_contacts")}
          className="pl-8 placeholder:text-muted-foreground border"
        />
        {filteredContacts.length > 0 ? (
          <ContactsList
            contacts={filteredContacts}
            selectedContactId={selected?.id || null}
            setSelected={setSelected}
          />
        ) : (
          <div className="p-8 text-center text-muted-foreground">
            {contactFetchError || t("none_found")}
          </div>
        )}
        {onMobile && (
          <Button
            className="absolute w-11 h-11 bg-primary bottom-0 right-0 m-8 r
ounded-full"
            onClick={() => {
              setModal((m) => ({
                ...m,
                contact: { ...m.contact, create: true },
              }));
            }}
          >
            <Plus />
          </Button>
        )}
      </ResizablePanel>
      <ResizableHandle withHandle className={cn(onMobile && "hidden")} />
```

```
      <ChildrenPanel
        hasMiddleBar
        // reverse logic like above: on mobile and with nothing selected, t
his component should be hidden.
        className={cn(onMobile && selected === null && "hidden")} // like a
bove we are using reverse logic here. If we are on mobile, and nothing is s
elected, this component should not be displayed.
      >
        <ContactDisplay contact={selected} reset={() => setSelected(null)}
/>
      </ChildrenPanel>
    </>
  );
}
```

/components/logo.tsx

```
import Image from "next/image";
import Link from "next/link";
import React from "react";

export default function AppLogo({ isCollapsed }: { isCollapsed: boolean })
{
  return (
    <>
      <Link
        href="/"
        className="flex items-center gap-2 user-select-none focus-primary-r
ing"
      >
        <Image
          src="/etpzp_sms-logo.png"
          alt="Application logo"
          width={48}
          height={48}
          className="user-select-none relative bottom-[2px]"
        />
        {!isCollapsed && (
          <span
            className="font-bold user-select-none tracking-tight text-xl fo
nt-disket-mono-regular" // or text-2xl
          >
            ETPZP-SMS
          </span>
        )}
      </Link>
    </>
  );
}
```

/components/form-input.tsx

```
"use client";
import React from "react";
import {
  FormControl,
  FormField,
  FormItem,
  FormLabel,
  FormMessage,
} from "./ui/form";
import { Input as ShadcnInput } from "./shared/input";
import { Control, FieldPath, FieldValues } from "react-hook-form";
import { cn } from "@/lib/utils";

interface InputProps<T extends FieldValues>
  extends React.InputHTMLAttributes<HTMLInputElement> {
  name: FieldPath<T>;
  control: Control<T>;
  label?: string;
  error?: boolean;
}

export function Input<T extends FieldValues>({
  name,
  control,
  label,
  error,
  ...props
}: InputProps<T>) {
  return (
    <FormField
      control={control}
      name={name}
      render={({ field }) => (
        <FormItem>
          {label && (
            <FormLabel
              className={cn("text-foreground", error && "text-destructive")
}
            >
              {label}
            </FormLabel>
          )}
          <FormControl>
            <ShadcnInput {...field} {...props} />
          </FormControl>
          <FormMessage />
        </FormItem>
      )}
```

```
    />
  );
}
```

/components/login-form.tsx

```
"use client";

import {
  Card,
  CardContent,
  CardDescription,
  CardHeader,
  CardTitle,
} from "@/components/ui/card";
import Image from "next/image";
import { Button } from "@/components/ui/button";
import { Input } from "@/components/ui/input";
import { login } from "@/lib/auth";
import { FormEvent, useState } from "react";
import { useRouter } from "next/navigation";
import { Label } from "./ui/label";
import { ActionResponse } from "@/types/action";
import { Login } from "@/lib/auth/config";
import SubmitButton from "./shared/submit-button";
import { Eye, Router } from "lucide-react";
import { useSettings } from "@/contexts/use-settings";
import { useTranslation } from "react-i18next";
import { toastActionResult } from "@/lib/utils";

const initialState: ActionResponse<Login> = {
  success: false,
  message: [],
};
export default function LoginForm() {
  const [passInputType, setPassInputType] = useState("password");
  const [serverState, setServerState] = useState(initialState);
  const [pending, setPending] = useState(false);
  const { syncWithDB } = useSettings();
  const router = useRouter();
  const { t } = useTranslation(["login-page", "common"]);

  async function handleSubmit(event: FormEvent<HTMLFormElement>) {
    event.preventDefault();
    setPending(true);
    // Create a FormData from the HTML form element
    const formData = new FormData(event.currentTarget);

    const result = await login(formData);
    setServerState(result);
```

```
    toastActionResult(result, t);
    if (result.success) {
      await syncWithDB(); // Fetch users settings from database on login
      router.replace("/");
    }
    setPending(false);
  }
  return (
    <main className="flex items-center justify-center w-screen h-screen p-3
">
      <form onSubmit={handleSubmit}>
        <Card className="mx-auto max-w-sm">
          <CardHeader>
            <div className="relative w-[60%] overflow-hidden mb-2">
              {/* Set a height for the parent */}
              <Image
                src="/etpzp_sms-logo.png"
                width={80}
                height={80}
                alt="Microsoft logo"
                // layout="fill" // This makes the image fill the parent co
ntainer
                // objectFit="cover" // This will crop the image to fill th
e container
                quality={100}
              />
            </div>
            <CardTitle className="text-2xl">{t("header")}</CardTitle>
            <CardDescription>{t("header_caption")}</CardDescription>
          </CardHeader>
          <CardContent className="flex flex-col gap-2">
            <div>
              <Label htmlFor="email">{t("email_label")}</Label>
              <Input
                name="email"
                id="email"
                type="email"
                defaultValue={serverState.inputs?.email}
                placeholder={t("email_placeholder")}
                aria-describedby="email"
                disabled={pending}
              />
              {serverState.errors?.email && (
                <p id="email-error" className="text-sm text-red-500">
                  {t(serverState.errors.email[0])}
                </p>
              )}
            </div>
            <div>
              <Label htmlFor="password">{t("password_label")}</Label>
              <div className="flex items-center gap-1 relative">
                <Input
                  name="password"
```

```
                  id="password"
                  type={passInputType}
                  defaultValue={serverState.inputs?.password}
                  aria-describedby="password"
                  disabled={pending}
                />
                <Button
                  className="absolute right-0 z-10"
                  type="button"
                  variant="none"
                  onClick={() =>
                    setPassInputType((prev) =>
                      prev === "text" ? "password" : "text"
                    )
                  }
                >
                  <Eye className="w-4 h-4" />
                </Button>
              </div>
              {serverState.errors?.password && (
                <p id="password-error" className="text-sm text-red-500">
                  {t(serverState.errors.password[0])}
                </p>
              )}
            </div>
            {!serverState.success && (
              <p className="text-sm text-destructive text-center">
                {t(serverState.message[0])}
              </p>
            )}
            <SubmitButton className="w-full">{t("button_submit")}</SubmitBu
tton>
          </CardContent>
        </Card>
      </form>
    </main>
  );
}
```

/components/profile-pic.tsx

```
"use client";

import React from "react";
import { cn, getNameInitials } from "@/lib/utils";
import { UserRound } from "lucide-react";
import Skeleton from "react-loading-skeleton";
import { ThemeProperties } from "@/types/theme";

type ProfilePicProps = {
```

```
  size?: number;
  name?: string;
  colorObj?: ThemeProperties | undefined;
  loading?: boolean;
  className?: string;
} & React.HTMLAttributes<HTMLDivElement>;

export default function ProfilePic({
  size = 9,
  name,
  // Will be filled use the colorObj's properties if it is provided
  colorObj,
  loading,
  className,
  ...props
}: ProfilePicProps) {
  if (loading)
    return (
      <Skeleton
        width={36}
        height={36}
        circle
        containerClassName={cn("flex", className)}
      />
    );

  return (
    <div
      className={cn(
        `flex justify-center items-center rounded-full`, // border border-m
uted-foreground - Don't like this
        className // add additional passed in classNames
      )}
      // For some reason we need to use inline styles for this, as it seems
to get overridden
      style={{
        width: `${size * 4}px`,
        height: `${size * 4}px`,
        backgroundColor: `hsl(${colorObj?.primary})`,
        color: `hsl(${colorObj?.primaryForeground})`,
      }}
      {...props}
    >
      {name ? (
        <p className={cn("text-sm")}>{getNameInitials(name)}</p>
      ) : (
        <UserRound
          className="height-full text-accent-foreground"
          strokeWidth={1.14}
        />
      )}
    </div>
```

```
  );
}
```

/components/app-layout.tsx

```tsx
"use client";

import ResizablePanelWrapper from "@/components/resizable-panel-wrapper";
import NavPanel, { MobileNavPanel } from "@/components/nav-panel";
import { useTheme as useNextTheme } from "next-themes";
import { SkeletonTheme } from "react-loading-skeleton";
import { useLayout } from "@/contexts/use-layout";
import { useSettings } from "@/contexts/use-settings";
import TranslationsProvider from "@/contexts/translations-provider";
import AppLogo from "./logo";
import { useIsMobile } from "@/hooks/use-mobile";
import Account from "./shared/account";
import { useEffect } from "react";

type LayoutProps = Readonly<{
  children: React.ReactNode;
  resources: any;
  locale: string;
  namespaces: string[];
}>;

export default function AppLayout({
  children,
  resources,
  locale,
  namespaces,
}: LayoutProps) {
  const { theme } = useNextTheme();
  const { settings } = useSettings();
  const onMobile = useIsMobile();
  const { isFullscreen } = useLayout();

  useEffect(() => {
    // Check if there's a hash in the URL
    if (window.location.hash) {
      // Scroll to the anchor
      const anchor = document.querySelector(window.location.hash);
      if (anchor) {
        anchor.scrollIntoView({ behavior: "smooth" });
      }
    }
  }, []); // Empty dependency array to run only on mount

  return (
    <SkeletonTheme
```

```
      // we are adjusting loading skeleton colors for dark mode - defaults
for light mode already look good
      baseColor={theme === "dark" ? "#2a2a2a" : undefined}
      highlightColor={theme === "dark" ? "#3a3a3a" : undefined}
    >
      {/* Modern layout bar here */}
      {settings.layout === "MODERN" && !isFullscreen && !onMobile && (
        <TranslationsProvider
          resources={resources}
          locale={locale}
          /* Currently account only uses `common` namespace */
          namespaces={["common"]}
        >
          <div className="w-full min-h-[var(--simple-header-height)] flex j
ustify-between items-center border-b px-2">
            <div className="flex items-center gap-2">
              <AppLogo isCollapsed={onMobile} />
            </div>
            <div className="">
              <Account profilePicPosition="RIGHT" />
            </div>
          </div>
        </TranslationsProvider>
      )}
      <ResizablePanelWrapper>
        <TranslationsProvider
          /* Only wrap what's necessary with the TranslationsProvider */
          resources={resources}
          locale={locale}
          /* should be ["navigation", "modals", "common"] */
          namespaces={namespaces}
        >
          {/* error.tsx catchall file would get its translations from here,
if one existed in /app/[locale]/(root)/error.tsx */}
          <NavPanel /* resizableHandle is inside here */ />
          <MobileNavPanel /* open state is managed in useLayout context */
/>
        </TranslationsProvider>

        {children}
      </ResizablePanelWrapper>
    </SkeletonTheme>
  );
}
```

/components/contact-display.tsx

```
"use client";

import { format } from "date-fns/format";
```

```
import { ArrowLeft, Edit, Trash2, X } from "lucide-react";
import { Button } from "@/components/ui/button";
import { Separator } from "@/components/ui/separator";
import {
  Tooltip,
  TooltipContent,
  TooltipTrigger,
} from "@/components/ui/tooltip";
import { useIsMobile } from "@/hooks/use-mobile";
import { cn, getNameInitials, toastActionResult } from "@/lib/utils";
import { CopyButton } from "./shared/copy-button";
import { deleteContact } from "@/lib/actions/contact.actions";
import { useModal } from "@/contexts/use-modal";
import EditContactModal from "./modals/edit-contact";
import { useRouter } from "next/navigation";
import { DBContact } from "@/types/contact";
import { saveDraft } from "@/lib/actions/message.actions";
import { useTranslation } from "react-i18next";
import ProfilePic from "./profile-pic";
import { PT_DATE_FORMAT } from "@/global.config";
import { ScrollArea } from "./ui/scroll-area";
import { useContacts } from "@/contexts/use-contacts";

export default function ContactDisplay({
  contact,
  reset,
}: {
  contact: DBContact | null;
  reset: () => void;
}) {
  const onMobile = useIsMobile();
  const router = useRouter();
  const { t } = useTranslation(["contacts-page", "common"]);
  const { setModal } = useModal();
  const { refetchContacts } = useContacts();

  const handleDelete = async () => {
    if (contact) {
      const result = await deleteContact(contact.id);
      toastActionResult(result, t);
      if (result.success) refetchContacts();
    }
  };
  const messageContact = async () => {
    if (contact) {
      const newDraft = await saveDraft(undefined, {
        body: "",
        recipients: [
          {
            phone: contact.phone,
            // This is a temporary solution. Maybe change the type later to
not be NewRecipient[]
            isValid: true,
```

```
            proneForDeletion: false,
          },
        ],
      });

      if (newDraft.success && newDraft.draftId) {
        router.push(`/new-message?message_id=${newDraft.draftId}`);
      } else {
        toastActionResult(newDraft, t);
      }
    }
  };
  return (
    <div className={cn("flex h-full flex-col")}>
      {contact && <EditContactModal contact={contact} />}
      <div className="flex items-center p-2 h-[var(--simple-header-height)]
border-b">
        <div className="flex items-center gap-2">
          {onMobile && (
            <Tooltip>
              <TooltipTrigger asChild>
                <Button variant="ghost" size="icon" onClick={() => reset()}
>
                  <ArrowLeft className="h-4 w-4" />
                  <span className="sr-only">{t("common:go_back")}</span>
                </Button>
              </TooltipTrigger>
              <TooltipContent>{t("common:go_back")}</TooltipContent>
            </Tooltip>
          )}

          <Tooltip>
            <TooltipTrigger asChild>
              <Button
                variant="ghost"
                size="icon"
                disabled={!contact}
                onClick={handleDelete}
              >
                <Trash2 className="h-4 w-4" />
                <span className="sr-only">
                  {t("common:delete_permanently")}
                </span>
              </Button>
            </TooltipTrigger>
            <TooltipContent>{t("common:delete_permanently")}</TooltipConten
t>
          </Tooltip>

          <Tooltip>
            <TooltipTrigger asChild>
              <Button
                variant="ghost"
```

```
                size="icon"
                onClick={() =>
                  setModal((m) => ({
                    ...m,
                    contact: { ...m.contact, edit: true },
                  }))
                }
                disabled={!contact}
              >
                <Edit className="h-4 w-4" />
                <span className="sr-only">{t("common:edit")}</span>
              </Button>
            </TooltipTrigger>
            <TooltipContent>{t("common:edit")}</TooltipContent>
          </Tooltip>
        </div>
        <div className="ml-auto flex items-center gap-2">
          {contact && (
            <Tooltip>
              <TooltipTrigger asChild>
                <Button variant="ghost" size="icon" onClick={() => reset()}
>
                  <X className="h-4 w-4" />
                  <span className="sr-only">{t("common:close")}</span>
                </Button>
              </TooltipTrigger>
              <TooltipContent>{t("common:close")}</TooltipContent>
            </Tooltip>
          )}
        </div>
      </div>
      {/* End top bar */}
      {/* <Separator /> */}

      <ScrollArea>
        <div
          className={
            onMobile
              ? `h-[calc(100vh-var(--simple-header-height))]`
              : `h-[calc(100vh-var(--header-height))]`
          }
        >
          {contact ? (
            <div className="flex flex-1 flex-col">
              <div className="flex items-start p-4">
                <div className="flex items-center gap-4 text-sm">
                  <ProfilePic
                    name={contact.name}
                    size={10}
                    className="border"
                  />
                  <h2>{contact.name}</h2>
                </div>
```

```
                {contact.created_at && (
                  <div className="ml-auto text-xs text-muted-foreground">
                    {`${t("common:created_on")} ${format(
                      new Date(contact.created_at),
                      PT_DATE_FORMAT
                    )}`}
                  </div>
                )}
              </div>
              <Separator />
              <div className="flex gap-4 justify-between items-center p-4 t
ext-sm">
                <p>{t("common:phone_number")}</p>
                <div className="flex">
                  <CopyButton
                    text={contact.phone}
                    variant="none"
                    className="pr-1"
                  />
                  <Button
                    variant="link"
                    className="p-0"
                    onClick={messageContact}
                  >
                    {contact.phone}
                  </Button>
                </div>
              </div>
              <Separator />
              <div className="flex gap-4 justify-between p-4 text-sm">
                <p>{t("common:description")}</p>

                {contact.description?.trim() ? (
                  <p className="text-right">{contact.description}</p>
                ) : (
                  <p className="italic text-right">
                    {t("common:no_description")}
                  </p>
                )}
              </div>
            </div>
          ) : (
            <div className="p-8 text-center text-muted-foreground">
              {t("none_selected")}
            </div>
          )}
        </div>
      </ScrollArea>
    </div>
  );
}
```

/components/send-button.tsx

```
"use client";

import { SetStateAction, useEffect, useState } from "react";
import { Button, buttonVariants } from "./ui/button";
import { ChevronDown, Clock, Loader2, Send } from "lucide-react";
import {
  DropdownMenu,
  DropdownMenuContent,
  DropdownMenuItem,
  DropdownMenuLabel,
  DropdownMenuSeparator,
  DropdownMenuTrigger,
} from "./ui/dropdown-menu";
import { cn } from "@/lib/utils";
import { useTranslation } from "react-i18next";
import { format } from "date-fns";
import { useNewMessage } from "@/contexts/use-new-message";
import { useModal } from "@/contexts/use-modal";
import { PT_DATE_FORMAT } from "@/global.config";

export default function SendButton({ loading }: { loading: boolean }) {
  const now = new Date();
  now.setMinutes(now.getMinutes() + 1); // Add one or two minutes margin so
that when the page loads slowly, the now date will appear to be in the past
, displaying send now on the button
  const { modal, setModal, scheduleDropdown, setScheduleDropdown } = useMod
al();
  const { message, setMessage } = useNewMessage();
  const { t } = useTranslation(["messages-page", "modals", "common"]);

  function tomorrowAt(hour: number) {
    // Create a new Date object for the current date
    const now = new Date();

    // Create a new Date object for tomorrow
    const tomorrow: Date = new Date(now);
    tomorrow.setDate(now.getDate() + 1);

    // Set the specified hour and default minutes to 0
    tomorrow.setHours(hour, 0, 0, 0);

    return tomorrow;
  }

  return (
    <div className="flex">
      <Button
```

```
        type="submit"
        className="rounded-tr-none rounded-br-none border-primary-foregroun
d border-r"
        disabled={loading}
      >
        {loading ? (
          <Loader2 className="animate-spin" />
        ) : (
          <Send className="w-4 h-4" />
        )}
        {message.scheduledDate > now
          ? `${t("submit_btn-scheduled", {
              time: "", // i18n messes up the output when passing it in lik
e this
            })} ${format(message.scheduledDate, PT_DATE_FORMAT)}`
          : t("submit_btn-normal")}
      </Button>
      <DropdownMenu open={scheduleDropdown} onOpenChange={setScheduleDropdo
wn}>
        <DropdownMenuTrigger
          className={cn("flex gap-3 items-center justify-start w-full")}
          asChild
        >
          <Button
            className={cn(
              "px-[1px] rounded-tl-none rounded-bl-none shadow-none",
              scheduleDropdown && "bg-primary/90"
            )}
            type="button"
            disabled={loading}
          >
            <ChevronDown
              className={cn(
                "h-4 w-4 transition-transform duration-300",
                scheduleDropdown && "rotate-180"
              )}
            />
          </Button>
        </DropdownMenuTrigger>
        <DropdownMenuContent align="end">
          <DropdownMenuLabel>
            <h6 className="font-bold">{t("schedule_dropdown-header")}</h6>
            <p className="text-muted-foreground font-normal">
              {t("schedule_dropdown-header_caption")}
            </p>
          </DropdownMenuLabel>
          <DropdownMenuSeparator />
          {message.scheduledDate > now && (
            <DropdownMenuItem
              onSelect={() => setMessage((m) => ({ ...m, scheduledDate: now
}))}
            >
              {t("schedule_dropdown-reset")}
```

```
            </DropdownMenuItem>
          )}
          {message.scheduledDate.getTime() !== tomorrowAt(9).getTime() && (
            <DropdownMenuItem
              onSelect={() =>
                setMessage((m) => ({
                  ...m,
                  scheduledDate: tomorrowAt(9),
                }))
              }
            >
              {t("schedule_dropdown-tomorrow_morning")}
            </DropdownMenuItem>
          )}
          {message.scheduledDate.getTime() !== tomorrowAt(15).getTime() &&
(
            <DropdownMenuItem
              onSelect={() =>
                setMessage((m) => ({
                  ...m,
                  scheduledDate: tomorrowAt(15),
                }))
              }
            >
              {t("schedule_dropdown-tomorrow_afternoon")}
            </DropdownMenuItem>
          )}
          <DropdownMenuItem
            onSelect={() => setModal((m) => ({ ...m, schedule: true }))}
          >
            <span>{t("schedule_dropdown-custom")}</span>
          </DropdownMenuItem>
        </DropdownMenuContent>
      </DropdownMenu>
    </div>
  );
}
```

/.dockerignore

```
# Ignore the .env.docker file
.env.docker

# Ignore other files and directories
node_modules
*.log
```

/.gitignore

```
# See https://help.github.com/articles/ignoring-files/ for more about ignor
ing files.

# dependencies
/node_modules
/.pnp
.pnp.*
.yarn/*
!.yarn/patches
!.yarn/plugins
!.yarn/releases
!.yarn/versions

# testing
/coverage

# next.js
/.next/
/out/

# production
/build

# misc
.DS_Store
*.pem

# debug
npm-debug.log*
yarn-debug.log*
yarn-error.log*

# env files (can opt-in for committing if needed)
.env**

# vercel
.vercel

# typescript
*.tsbuildinfo
next-env.d.ts

# Languages
/locales/
# Example data
/lib/data/*
```

/package.json

```
{
  "name": "etpzp-sms-app",
  "version": "0.1.0",
  "private": true,
  "scripts": {
    "dev": "i18nexus pull && next dev",
    "build": "i18nexus pull && next build",
    "start": "i18nexus pull && next start",
    "lint": "next lint",
    "dev-simple": "next dev"
  },
  "overrides": {
    "react-is": "^19.0.0-rc-69d4b800-20241021"
  },
  "dependencies": {
    "@hookform/resolvers": "^3.9.1",
    "@radix-ui/react-accordion": "^1.2.3",
    "@radix-ui/react-alert-dialog": "^1.1.6",
    "@radix-ui/react-avatar": "^1.1.1",
    "@radix-ui/react-checkbox": "^1.1.3",
    "@radix-ui/react-collapsible": "^1.1.1",
    "@radix-ui/react-dialog": "^1.1.2",
    "@radix-ui/react-dropdown-menu": "^2.1.2",
    "@radix-ui/react-label": "^2.1.0",
    "@radix-ui/react-popover": "^1.1.2",
    "@radix-ui/react-radio-group": "^1.2.2",
    "@radix-ui/react-scroll-area": "^1.2.1",
    "@radix-ui/react-select": "^2.1.2",
    "@radix-ui/react-separator": "^1.1.0",
    "@radix-ui/react-slot": "^1.1.0",
    "@radix-ui/react-switch": "^1.1.1",
    "@radix-ui/react-tabs": "^1.1.1",
    "@radix-ui/react-tooltip": "^1.1.4",
    "@svgr/webpack": "^8.1.0",
    "@types/pg": "^8.11.10",
    "activedirectory2": "^2.2.0",
    "class-variance-authority": "^0.7.0",
    "clsx": "^2.1.1",
    "cmdk": "1.0.0",
    "date-fns": "^4.1.0",
    "i18next": "^24.0.0",
    "i18next-resources-to-backend": "^1.2.1",
    "iron-session": "^8.0.4",
    "libphonenumber-js": "^1.11.17",
    "lucide-react": "^0.483.0",
    "next": "15.1.6",
    "next-i18n-router": "^5.5.1",
    "next-themes": "^0.4.3",
    "node": "^23.8.0",
    "pg": "^8.13.1",
    "react": "19.0.0",
    "react-day-picker": "8.10.1",
```

```
    "react-dom": "19.0.0",
    "react-hook-form": "^7.54.1",
    "react-i18next": "^15.1.1",
    "react-loading-skeleton": "^3.5.0",
    "react-resizable-panels": "^2.1.7",
    "recharts": "^2.15.1",
    "sonner": "^1.7.1",
    "tailwind-merge": "^2.5.4",
    "tailwindcss-animate": "^1.0.7",
    "zod": "^3.24.1"
  },
  "devDependencies": {
    "@tailwindcss/aspect-ratio": "^0.4.2",
    "@types/activedirectory2": "^1.2.6",
    "@types/node": "^20",
    "@types/react": "19.0.8",
    "@types/react-dom": "19.0.3",
    "@types/validator": "^13.12.2",
    "eslint": "^8",
    "eslint-config-next": "15.1.6",
    "i18nexus-cli": "^3.5.0",
    "postcss": "^8",
    "tailwindcss": "^3.4.1",
    "typescript": "^5"
  }
}
```

/hooks/use-mounted.ts

```
"use client";

import { useState, useEffect } from "react";

export default function useIsMounted() {
  const [isMounted, setIsMounted] = useState(false);

  useEffect(() => {
    setIsMounted(true);

    return () => {
      setIsMounted(true);
    };
  }, []);

  return isMounted;
}
```

/hooks/use-mobile.tsx

```
import * as React from "react"

const MOBILE_BREAKPOINT = 768

export function useIsMobile() {
  const [isMobile, setIsMobile] = React.useState<boolean | undefined>(undef
ined)

  React.useEffect(() => {
    const mql = window.matchMedia(`(max-width: ${MOBILE_BREAKPOINT - 1}px)`
)
    const onChange = () => {
      setIsMobile(window.innerWidth < MOBILE_BREAKPOINT)
    }
    mql.addEventListener("change", onChange)
    setIsMobile(window.innerWidth < MOBILE_BREAKPOINT)
    return () => mql.removeEventListener("change", onChange)
  }, [])

  return !!isMobile
}
```

/hooks/use-session.ts

```
"use client";

import { SessionData } from "@/lib/auth/config";
import { getSessionOnClient } from "@/lib/auth/sessions";
import { useState, useEffect } from "react";

export function useSession() {
  const [session, setSession] = useState<SessionData | null>(null);
  const [loading, setLoading] = useState(true);

  useEffect(() => {
    async function fetchSession() {
      try {
        const data = await getSessionOnClient();

        setSession(data);
      } catch (error) {
        console.error("Failed to fetch session:", error);
      } finally {
        setLoading(false);
      }
    }

    fetchSession();
```

```
  }, []);

  return { session, loading };
}
```

/hooks/use-debounce.ts

```
"use client";

import { useEffect, useState } from "react";

// You can pass in any value or a function and the time in milliseconds tha
t you want it to updated/debounced after
export default function useDebounce(value: any, delay: number) {
  const [debouncedValue, setDebouncedValue] = useState(value);

  useEffect(() => {
    const handler = setTimeout(() => {
      setDebouncedValue(value);
    }, delay);

    return () => {
      clearTimeout(handler);
    };
  }, [value, delay]);

  return debouncedValue;
}
```

/lib/theme.colors.ts

```
import { Theme, ThemeColors, ThemeProperties, Themes } from "@/types/theme"
;

export const themes: Themes = {
  Zinc: {
    light: {
      background: "0 0% 100%",
      foreground: "240 10% 3.9%",
      card: "0 0% 100%",
      cardForeground: "240 10% 3.9%",
      popover: "0 0% 100%",
      popoverForeground: "240 10% 3.9%",
      primary: "240 5.9% 10%",
      primaryForeground: "0 0% 98%",
      secondary: "240 4.8% 95.9%",
      secondaryForeground: "240 5.9% 10%",
```

```
      muted: "240 4.8% 95.9%",
      mutedForeground: "240 3.8% 46.1%",
      accent: "240 4.8% 95.9%",
      accentForeground: "240 5.9% 10%",
      destructive: "0 84.2% 60.2%",
      destructiveForeground: "0 0% 98%",
      border: "240 5.9% 90%",
      input: "240 5.9% 90%",
      ring: "240 5.9% 10%",
      radius: "0.5rem",
    },
    dark: {
      background: "240 10% 3.9%",
      foreground: "0 0% 98%",
      card: "24 9.8% 10%",
      cardForeground: "0 0% 98%",
      popover: "240 10% 3.9%",
      popoverForeground: "0 0% 98%",
      primary: "0 0% 98%",
      primaryForeground: "240 5.9% 10%",
      secondary: "240 3.7% 15.9%",
      secondaryForeground: "0 0% 98%",
      muted: "240 3.7% 15.9%",
      mutedForeground: "240 5% 64.9%",
      accent: "240 3.7% 15.9%",
      accentForeground: "0 0% 98%",
      destructive: "0 62.8% 30.6%",
      destructiveForeground: "0 0% 98%",
      border: "240 3.7% 15.9%",
      input: "240 3.7% 15.9%",
      ring: "240 4.9% 83.9%",
      radius: "0.5rem",
    },
  },
  Rose: {
    light: {
      background: "0 0% 100%",
      foreground: "240 10% 3.9%",
      card: "0 0% 100%",
      cardForeground: "240 10% 3.9%",
      popover: "0 0% 100%",
      popoverForeground: "240 10% 3.9%",
      primary: "346.8 77.2% 49.8%",
      primaryForeground: "355.7 100% 97.3%",
      secondary: "240 4.8% 95.9%",
      secondaryForeground: "240 5.9% 10%",
      muted: "240 4.8% 95.9%",
      mutedForeground: "240 3.8% 46.1%",
      accent: "240 4.8% 95.9%",
      accentForeground: "240 5.9% 10%",
      destructive: "0 84.2% 60.2%",
      destructiveForeground: "0 0% 98%",
      border: "240 5.9% 90%",
```

```
      input: "240 5.9% 90%",
      ring: "346.8 77.2% 49.8%",
      radius: "0.5rem",
    },
    dark: {
      background: "20 14.3% 4.1%",
      foreground: "0 0% 95%",
      card: "24 9.8% 10%",
      cardForeground: "0 0% 95%",
      popover: "0 0% 9%",
      popoverForeground: "0 0% 95%",
      primary: "346.8 77.2% 49.8%",
      primaryForeground: "355.7 100% 97.3%",
      secondary: "240 3.7% 15.9%",
      secondaryForeground: "0 0% 98%",
      muted: "0 0% 15%",
      mutedForeground: "240 5% 64.9%",
      accent: "12 6.5% 15.1%",
      accentForeground: "0 0% 98%",
      destructive: "0 62.8% 30.6%",
      destructiveForeground: "0 85.7% 97.3%",
      border: "240 3.7% 15.9%",
      input: "240 3.7% 15.9%",
      ring: "346.8 77.2% 49.8%",
      radius: "0.5rem",
    },
  },
  Blue: {
    light: {
      background: "0 0% 100%",
      foreground: "222.2 84% 4.9%",
      card: "0 0% 100%",
      cardForeground: "222.2 84% 4.9%",
      popover: "0 0% 100%",
      popoverForeground: "222.2 84% 4.9%",
      primary: "221.2 83.2% 53.3%",
      primaryForeground: "210 40% 98%",
      secondary: "210 40% 96.1%",
      secondaryForeground: "222.2 47.4% 11.2%",
      muted: "210 40% 96.1%",
      mutedForeground: "215.4 16.3% 46.9%",
      accent: "210 40% 96.1%",
      accentForeground: "222.2 47.4% 11.2%",
      destructive: "0 84.2% 60.2%",
      destructiveForeground: "210 40% 98%",
      border: "214.3 31.8% 91.4%",
      input: "214.3 31.8% 91.4%",
      ring: "221.2 83.2% 53.3%",
      radius: "0.5rem",
    },
    dark: {
      background: "222.2 84% 4.9%",
      foreground: "210 40% 98%",
```

```
      card: "24 9.8% 10%",
      cardForeground: "210 40% 98%",
      popover: "222.2 84% 4.9%",
      popoverForeground: "210 40% 98%",
      primary: "217.2 91.2% 59.8%",
      primaryForeground: "222.2 47.4% 11.2%",
      secondary: "217.2 32.6% 17.5%",
      secondaryForeground: "210 40% 98%",
      muted: "217.2 32.6% 17.5%",
      mutedForeground: "215 20.2% 65.1%",
      accent: "217.2 32.6% 17.5%",
      accentForeground: "210 40% 98%",
      destructive: "0 62.8% 30.6%",
      destructiveForeground: "210 40% 98%",
      border: "217.2 32.6% 17.5%",
      input: "217.2 32.6% 17.5%",
      ring: "224.3 76.3% 48%",
      radius: "0.5rem",
    },
  },
  Green: {
    light: {
      background: "0 0% 100%",
      foreground: "240 10% 3.9%",
      card: "0 0% 100%",
      cardForeground: "240 10% 3.9%",
      popover: "0 0% 100%",
      popoverForeground: "240 10% 3.9%",
      primary: "142.1 76.2% 36.3%",
      primaryForeground: "128 83% 97%",
      secondary: "240 4.8% 95.9%",
      secondaryForeground: "240 5.9% 10%",
      muted: "240 4.8% 95.9%",
      mutedForeground: "240 3.8% 46.1%",
      accent: "240 4.8% 95.9%",
      accentForeground: "240 5.9% 10%",
      destructive: "0 84.2% 60.2%",
      destructiveForeground: "0 0% 98%",
      border: "240 5.9% 90%",
      input: "240 5.9% 90%",
      ring: "142.1 76.2% 36.3%",
      radius: "0.5rem",
    },
    dark: {
      background: "20 14.3% 4.1%",
      foreground: "0 0% 95%",
      card: "24 9.8% 10%",
      cardForeground: "0 0% 95%",
      popover: "0 0% 9%",
      popoverForeground: "0 0% 95%",
      primary: "142.1 70.6% 45.3%",
      primaryForeground: "144.9 80.4% 10%",
      secondary: "240 3.7% 15.9%",
```

```
      secondaryForeground: "0 0% 98%",
      muted: "0 0% 15%",
      mutedForeground: "240 5% 64.9%",
      accent: "12 6.5% 15.1%",
      accentForeground: "0 0% 98%",
      destructive: "0 62.8% 30.6%",
      destructiveForeground: "0 85.7% 97.3%",
      border: "240 3.7% 15.9%",
      input: "240 3.7% 15.9%",
      ring: "142.4 71.8% 29.2%",
      radius: "0.5rem",
    },
  },
  Orange: {
    light: {
      background: "0 0% 100%",
      foreground: "20 14.3% 4.1%",
      card: "0 0% 100%",
      cardForeground: "20 14.3% 4.1%",
      popover: "0 0% 100%",
      popoverForeground: "20 14.3% 4.1%",
      primary: "24.6 95% 53.1%",
      primaryForeground: "60 9.1% 97.8%",
      secondary: "60 4.8% 95.9%",
      secondaryForeground: "24 9.8% 10%",
      muted: "60 4.8% 95.9%",
      mutedForeground: "25 5.3% 44.7%",
      accent: "60 4.8% 95.9%",
      accentForeground: "24 9.8% 10%",
      destructive: "0 84.2% 60.2%",
      destructiveForeground: "60 9.1% 97.8%",
      border: "20 5.9% 90%",
      input: "20 5.9% 90%",
      ring: "24.6 95% 53.1%",
      radius: "0.5rem",
    },
    dark: {
      background: "20 14.3% 4.1%",
      foreground: "60 9.1% 97.8%",
      card: "24 9.8% 10%",
      cardForeground: "60 9.1% 97.8%",
      popover: "20 14.3% 4.1%",
      popoverForeground: "60 9.1% 97.8%",
      primary: "20.5 90.2% 48.2%",
      primaryForeground: "60 9.1% 97.8%",
      secondary: "12 6.5% 15.1%",
      secondaryForeground: "60 9.1% 97.8%",
      muted: "12 6.5% 15.1%",
      mutedForeground: "24 5.4% 63.9%",
      accent: "12 6.5% 15.1%",
      accentForeground: "60 9.1% 97.8%",
      destructive: "0 72.2% 50.6%",
      destructiveForeground: "60 9.1% 97.8%",
```

```
      border: "12 6.5% 15.1%",
      input: "12 6.5% 15.1%",
      ring: "20.5 90.2% 48.2%",
      radius: "0.5rem",
    },
  },
  Yellow: {
    light: {
      background: "0 0% 100%",
      foreground: "48 14.3% 4.1%",
      card: "0 0% 100%",
      cardForeground: "48 14.3% 4.1%",
      popover: "0 0% 100%",
      popoverForeground: "48 14.3% 4.1%",
      primary: "51 100% 50%",
      primaryForeground: "0 0% 98%", // Maybe make this dark, or make the
primary color darker to increase contrast
      secondary: "60 4.8% 95.9%",
      secondaryForeground: "48 9.8% 10%",
      muted: "60 4.8% 95.9%",
      mutedForeground: "50 5.3% 44.7%",
      accent: "60 4.8% 95.9%",
      accentForeground: "48 9.8% 10%",
      destructive: "0 84.2% 60.2%",
      destructiveForeground: "60 9.1% 97.8%",
      border: "48 5.9% 90%",
      input: "48 5.9% 90%",
      ring: "51 100% 50%",
      radius: "0.5rem",
    },
    dark: {
      background: "48 14.3% 4.1%",
      foreground: "60 9.1% 97.8%",
      card: "48 9.8% 10%",
      cardForeground: "60 9.1% 97.8%",
      popover: "48 14.3% 4.1%",
      popoverForeground: "60 9.1% 97.8%",
      primary: "51 100% 50%",
      primaryForeground: "240 5.9% 10%",
      secondary: "48 6.5% 15.1%",
      secondaryForeground: "60 9.1% 97.8%",
      muted: "48 6.5% 15.1%",
      mutedForeground: "50 5.4% 63.9%",
      accent: "48 6.5% 15.1%",
      accentForeground: "60 9.1% 97.8%",
      destructive: "0 72.2% 50.6%",
      destructiveForeground: "60 9.1% 97.8%",
      border: "48 6.5% 15.1%",
      input: "48 6.5% 15.1%",
      ring: "51 100% 50%",
      radius: "0.5rem",
    },
  },
```

```
};

// Function to get theme by index starting at 1, not 0
export function getThemeByIndex(
  index: number,
  themeMode: "light" | "dark" | undefined = "light"
) {
  const theme = themesArray.find((theme) => theme.index === index);

  return theme?.value[themeMode] as ThemeProperties | undefined; // Return
the theme value or undefined if not found
}

export default function setGlobalColorTheme(
  themeMode: "light" | "dark",
  colorIndex: number
) {
  const theme = getThemeByIndex(colorIndex, themeMode);
  if (theme === undefined)
    throw new Error("Theme not found. The theme color is probably invalid")
;
  for (const key in theme) {
    // Use type assertion to specify that key is a key of ThemeProperties
    document.documentElement.style.setProperty(
      `--${key}`,
      theme[key as keyof ThemeProperties]
    );
  }
}

// Create a new array to hold the themes in a 1-based index format
export const themesArray = Object.keys(themes).map((key, index) => {
  return { index: index + 1, name: key, value: themes[key] };
});

export const PROFILE_COLOR_CSS_NAMES = [
  "salmon",
  "dodgerblue",
  "gold",
  "mediumorchid",
];
```

/lib/form.schemas.ts

```
import { z } from "zod";
import { parsePhoneNumberFromString } from "libphonenumber-js";
import { appearanceLayoutValues } from "@/types/user";
import { parseISO, isValid } from "date-fns";

const CustomString = (options?: { message: string }) => {
```

```
  return z.string({
    message: options?.message || "common:error-not_string",
  });
};
const CustomPhone = () => {
  return CustomString().refine(
    // this returns a boolean telling zod whether the phone data is valid o
r not
    (input: string) => {
      const parsedPhone = parsePhoneNumberFromString(input);

      return (parsedPhone && parsedPhone.isValid()) || false;
    },
    {
      message: "common:error-invalid_phone",
    }
  );
};
// Our one source of truth is the form schema. When you create a new field,
add it here.
export const MessageSchema = z.object({
  sender: CustomString().optional(),
  // recipients are handled internally for more thorough error messages
  subject: CustomString().optional(),
  body: CustomString().min(1, "new-message-page:zod_error-body_empty"),
  secondsUntilSend: z
    .number({ message: "new-message-page:zod_error-invalid_schedule_date" }
)
    // .positive({ message: "new-message-page:zod_error-negative_schedule_d
ate" })
    .optional(),
});

export const LoginSchema = z.object({
  email: CustomString().email({
    message: "login-page:zod_error-invalid_email",
  }),
  password: CustomString(),
});

export const ContactSchema = z.object({
  // id: z.string(),
  name: z
    .string()
    .min(2, { message: "modals:zod_error-short_name" })
    .max(50, { message: "modals:zod_error-long_name" }),
  phone: CustomPhone(),
  description: CustomString()
    .max(255, { message: "modals:zod_error-long_contact_description" })
    .optional(),
});
```

```
// Create a schema that handles each setting separately
export const UpdateSettingSchema = z.discriminatedUnion("name", [
  // For the language setting ("lang") we expect a 2-character string (ISO 6
39-1 code)
  z.object({
    name: z.literal("lang"),
    value: z
      .string()
      .min(2, "Language code must be exactly 2 characters")
      .max(2, "Language code must be exactly 2 characters"),
  }),
  // For profile_color_id, convert the incoming string to a number and requ
ire an integer.
  z.object({
    name: z.literal("profile_color_id"),
    value: z.preprocess(
      (val) => Number(val),
      z
        .number({
          invalid_type_error: "Profile color id must be a number",
        })
        .int("Profile color id must be an integer")
    ),
  }),
  // For primary_color_id, use similar logic as profile_color_id.
  z.object({
    name: z.literal("primary_color_id"),
    value: z.preprocess(
      (val) => Number(val),
      z
        .number({
          invalid_type_error: "Primary color id must be a number",
        })
        .int("Primary color id must be an integer")
    ),
  }),
  // For display_name, require a non-empty string with a max length of 50 c
haracters.
  z.object({
    name: z.literal("display_name"),
    value: z
      .string()
      .nonempty("Display name cannot be empty")
      .max(50, "Display name cannot exceed 50 characters"),
  }),
  // For Application layout we have a string enum defined in a type file
  z.object({
    name: z.literal("appearance_layout"),
    value: z.enum(appearanceLayoutValues),
  }),
  // For dark_mode, convert the string "true"/"false" to a boolean.
  z.object({
    name: z.literal("dark_mode"),
```

```
    value: z.preprocess((val) => {
      // Convert strings "true" and "false" to actual booleans.
      if (val === "dark") return true;
      if (val === "light") return false;
      return val;
    }, z.boolean({ invalid_type_error: "Dark mode must be a boolean value"
}})),
  }),
]);

// Admin dashboard page schemas
export const zodISODate = z.string().refine(
  (date) => {
    // Check if the date is a valid ISO 8601 date string
    const parsedDate = parseISO(date);
    return isValid(parsedDate);
  },
  {
    message: "Invalid date format. Expected ISO 8601 format.",
  }
);

export const DateRangeSchema = z.object({
  startDate: zodISODate.optional(),
  endDate: zodISODate.optional(),
});
```

/lib/.DS_Store

```
Bud1% @ @ @ @E%DSDB` @ @ @
```

/lib/auth/activedirectory/authenticate.ts

```
"use server";

import ActiveDirectory from "activedirectory2";
import { activeDirectoryConfig, SessionData } from "@/lib/auth/config";
import userExists from "./user";
import userInGroup from "./group";
import saveUser, { dummySaveUser } from "@/lib/actions/user.actions";
import type { DBUser } from "@/types/user";

export default async function authenticate({
  email,
  password,
}: {
  email: string;
```

```
  password: string;
}): Promise<SessionData & { errors: string[] }> {
  const ad = new ActiveDirectory(activeDirectoryConfig);

  // 1. Check if user even exists in the active directory server
  const exists = await userExists(ad, email, password);
  if (!exists.success) {
    return {
      isAuthenticated: false,
      isAdmin: false,
      errors: [exists.error ? exists.error : ""],
    };
  }
  // 2. Check if user is allowed to use the app
  const userGroup = "Utilizadores-SMS";
  const hasAppPermission = await userInGroup(ad, email, userGroup);

  // 3. Check if user has admin privileges
  const adminGroup = "Administradores-SMS";
  const hasAdminPermission = await userInGroup(ad, email, adminGroup);

  // 4. Sync all of this with the database
  const userResult = await saveUser(ad, email, hasAdminPermission.success);

  return {
    user: userResult.success ? userResult.data : undefined,
    isAuthenticated: hasAppPermission.success,
    isAdmin: hasAdminPermission.success,
    errors: [
      exists.error !== null
        ? exists.error
        : "An error occurred while checking if user exists",
      hasAppPermission.error !== null
        ? hasAppPermission.error
        : "An error occurred while checking if user is allowed to use the a
pp",
      hasAdminPermission.error !== null
        ? hasAdminPermission.error
        : "An error occurred while checking if user is an admin",
    ],
  };
}

export async function dummyAuthenticate({
  email,
  password,
}: {
  email: string;
  password: string;
}): Promise<SessionData> {
  const dummyUser: SessionData & { user: DBUser } = {
    user: {
```

```
      id: "1",
      email: "dummy@user.com",
      name: "Dummy User",
      first_name: "Dummy",
      last_name: "User",
      role: "ADMIN",

      lang: "pt",

      profile_color_id: 1,
      display_name: "Dummy User",

      primary_color_id: 1,
      dark_mode: false,
      appearance_layout: "MODERN",
    },
    isAuthenticated: true,
    isAdmin: true,
  };
  const userResult = await dummySaveUser(dummyUser.user as DBUser);

  return {
    user: userResult.success ? userResult.data : undefined,
    isAuthenticated: userResult.success,
    isAdmin: userResult.success,
  };
}
```

/lib/auth/activedirectory/group.ts

```
"use server";
import type ActiveDirectory from "activedirectory2";

// Check if user is apart of a specific group
export default async function userInGroup(
  ad: ActiveDirectory,
  username: string,
  group: string
): Promise<{ success: boolean; error: string | null }> {
  return new Promise((resolve) => {
    ad.isUserMemberOf(
      username,
      group,
      (err: object | null, isMember: boolean) => {
        if (err) {
          resolve({ success: false, error: JSON.stringify(err) });
        } else {
          resolve({ success: isMember, error: null });
        }
      }
```

```
    );
  });
}
```

/lib/auth/activedirectory/user.ts

```
"use server";
import type ActiveDirectory from "activedirectory2";

// Check if user even exists on the server
export default async function userExists(
  ad: ActiveDirectory,
  username: string,
  password: string
): Promise<{ success: boolean; error: string | null }> {
  return new Promise((resolve) => {
    ad.authenticate(
      username,
      password,
      (err: string | null, authenticated: boolean) => {
        if (err || !authenticated) {
          resolve({ success: false, error: err });
        } else {
          resolve({ success: authenticated, error: null });
        }
      }
    );
  });
}
```

/lib/auth/index.ts

```
"use server";

import authenticate, {
  dummyAuthenticate,
} from "./activedirectory/authenticate";
import { createSession, getSession } from "./sessions";
import { LoginSchema } from "@/lib/form.schemas";
import { Login, SessionData } from "./config";
import { ActionResponse } from "@/types/action";

export async function login(
  formData: FormData
): Promise<ActionResponse<Login>> {
  // 1. Type validation
  const email = formData.get("email") as string;
  const password = formData.get("password") as string;
```

```
  const validatedData = LoginSchema.safeParse({ email, password });
  if (!validatedData.success) {
    return {
      success: false,
      message: ["common:fix_zod_errors"],
      inputs: { email, password },
      errors: validatedData.error.flatten().fieldErrors,
    };
  }

  // 2. Authenticate user using AD and save to db
  const user: SessionData = await dummyAuthenticate({
    email,
    password,
  });

  if (!user.isAuthenticated) {
    return {
      success: false,
      message: ["server-wrong_credentials"],
      inputs: { email, password },
    };
  }

  // 3. Create new session cookie
  await createSession(user);
  return {
    success: true,
    message: [
      "server-auth_success_header",
      "server-auth_success_header_caption",
    ],
  };
}

export async function logout() {
  try {
    const session = await getSession();
    session.destroy();
    return { success: true };
  } catch (error) {
    console.log("LOGOUT FAILED:", error);

    return { success: false };
  }
}
```

/lib/auth/config.ts

```
import { User } from "@/types/user";
import { SessionOptions } from "iron-session";

export type SessionData = {
  user?: User;
  isAuthenticated: boolean;
  isAdmin: boolean;
};
export type Login = {
  email: string;
  password: string;
};
export const defaultSession: SessionData = {
  isAuthenticated: false,
  isAdmin: false,
};

// Iron session config object
export const sessionOptions: SessionOptions = {
  cookieName: "my-etpzp-app-session", // anything you want
  password: process.env.SESSION_SECRET!, // TypeScript non-null assertion o
perator

  // Optional fields
  ttl: 60 * 60 * 24, // cookie expiration from now in seconds (we want 24h)
  cookieOptions: {
    // prevent client side js from accessing the cookie
    httpOnly: true,
    // Secure only works in `https` environments. So if the environment is
`https`, it'll return true.
    secure: process.env.NODE_ENV === "production",
  },
};

export const activeDirectoryConfig = {
  url: process.env.AD_URL!,
  baseDN: process.env.AD_BASE_DN!,
  username: process.env.AD_EMAIL!, // we store emails in the username field
  password: process.env.AD_PASSWORD!,
};
```

/lib/auth/sessions.ts

```
"use server";

import { getIronSession } from "iron-session";
import { cookies } from "next/headers";
import { SessionData, sessionOptions } from "@/lib/auth/config";
import { NextRequest, NextResponse } from "next/server";
```

```
// helper function for getting the current session
export async function getSession(req?: NextRequest, res?: NextResponse) {
  const session =
    req && res
      ? await getIronSession<SessionData>(req, res, sessionOptions)
      : await getIronSession<SessionData>(await cookies(), sessionOptions);

  // For security, you can double-check the user's existence in the databas
e or AD server, but this slows down the app.
  return session;
}

export async function createSession(user: SessionData) {
  const session = await getSession();

  // Store user data in the cookie by mapping over each of the object's pro
perty
  Object.entries(user).forEach(([key, value]) => {
    if (!(key in session)) {
      (session as any)[key] = value;
    }
  });

  await session.save();
}

export async function getSessionOnClient(): Promise<SessionData> {
  const { user, isAuthenticated, isAdmin } = await getIronSession<SessionDa
ta>(
    await cookies(),
    sessionOptions
  );

  return {
    user,
    isAuthenticated,
    isAdmin,
  };
}
```

/lib/utils.ts

```
import { DBContact } from "./../types/contact";
import parsePhoneNumber, {
  CountryCode,
  parsePhoneNumberFromString,
} from "libphonenumber-js";
import { clsx, type ClassValue } from "clsx";
import { twMerge } from "tailwind-merge";
```

```
import {
  DBRecipient,
  FetchedRecipient,
  NewRecipient,
  RankedRecipient,
  WithContact,
} from "@/types/recipient";
import { DBMessage } from "@/types";
import { ActionResponse } from "@/types/action";
import { toast } from "sonner";
import { enUS, pt, de } from "date-fns/locale";

export function cn(...inputs: ClassValue[]) {
  return twMerge(clsx(inputs));
}

export function sleep(ms: number) {
  return new Promise((resolve) => setTimeout(resolve, ms));
}

export function generateUniqueId() {
  return "xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx".replace(/[xy]/g, function (
c) {
    const r = (Math.random() * 16) | 0;
    const v = c === "x" ? r : (r & 0x3) | 0x8;
    return v.toString(16);
  });
}

export function validatePhoneNumber(phone: string): NewRecipient {
  const countryCode: CountryCode = "PT";

  const phoneNumber = parsePhoneNumber(phone, countryCode);

  let properties: {
    isValid: boolean;
    formattedPhone?: string;
    error?: {
      type: "error" | "warning";
      message: string;
    };
  } = {
    isValid: false,
    formattedPhone: phoneNumber?.formatInternational(),
  };

  if (phoneNumber?.isValid()) {
    if (phoneNumber.country === countryCode) {
      properties = {
        isValid: true,
      };
    } else {
```

```
      properties.isValid = true;
      properties.error = {
        type: "warning",
        message: "tooltip-not_portuguese_number",
      };
    }
  } else {
    properties.isValid = false;
    properties.error = {
      type: "error",
      message: "tooltip-invalid_phone_number",
    };
  }
  return { ...properties, phone, proneForDeletion: false };
}

export function searchMessages(
  messages: DBMessage[],
  searchTerm: string,
  currentPage?: number
) {
  // Convert searchTerm to lowercase for case-insensitive comparison
  const lowerCaseSearchTerm = searchTerm.toLowerCase();

  // Filter messages based on userId and search term
  const filteredMessages = messages.filter(
    (message) =>
      message.subject?.toLowerCase().includes(lowerCaseSearchTerm) ||
      message.body.toLowerCase().includes(lowerCaseSearchTerm) ||
      message.status.toLowerCase() === lowerCaseSearchTerm // Assuming stat
us is also part of the search
  );

  return filteredMessages;
}

export function searchContacts(
  contacts: DBContact[],
  searchTerm: string | null,
  currentPage?: number
) {
  if (!searchTerm) return contacts;
  // Convert searchTerm to lowercase for case-insensitive comparison
  const lowerCaseSearchTerm = searchTerm.toLowerCase().trim();

  // Filter contacts based on userId and search term
  const filteredContacts = contacts.filter(
    (contact) =>
      (contact.name &&
        contact.name.toLowerCase().includes(lowerCaseSearchTerm)) ||
      contact.phone.toLowerCase().includes(lowerCaseSearchTerm)
  );
```

```
  return filteredContacts;
}

export function formatPhone(phone: string): string | undefined {
  const parsedPhone = parsePhoneNumberFromString(phone);
  if (parsedPhone && parsedPhone.isValid()) {
    return parsedPhone.number;
  } else {
    return undefined;
  }
}

export function getNameInitials(fullName: string | null | undefined) {
  // Split the full name into parts
  if (!fullName) return "";

  const nameParts = fullName.trim().split(/\s+/);

  // Get the first letter of the first name
  const firstInitial = nameParts[0][0].toUpperCase();

  // If there's only one name, return just that initial
  if (nameParts.length === 1) {
    return firstInitial;
  }

  // Get the first letter of the last name
  const lastInitial = nameParts[nameParts.length - 1][0].toUpperCase();

  // Return the initials
  return firstInitial + lastInitial;
}

// Convert contact -> recipient, because `addRecipient` function expects a
recipient type of NewRecipient not of contact type.
export function convertToRecipient(contact: DBContact): NewRecipient {
  const { id, name, phone, description } = contact;
  const validatedRecipient = validatePhoneNumber(phone);
  return {
    ...validatedRecipient,
    contact: {
      id,
      name,
      phone,
      description,
    },
  };
}
export function getUniques(
  currentRecipients: NewRecipient[],
  newRecipients: WithContact[]
```

```
): WithContact[] {
  return newRecipients.filter(
    (recipient) => !currentRecipients.some((r) => r.phone === recipient.pho
ne)
  );
}

export function toastActionResult(
  result: ActionResponse<any>,
  translate?: (translationKey: string) => string
) {
  if (!Array.isArray(result.message) || !result.message)
    throw new Error("Toast message must be an array of strings.");
  if (!result.message.length)
    return console.log("FAILED TOAST_ACTION_RESULT: message array is empty"
);

  // thankfully, this doesn't throw an error
  if (translate) {
    if (result.success) {
      toast.success(translate(result.message[0]), {
        description: translate(result.message[1]),
      });
    } else {
      toast.error(translate(result.message[0]), {
        description: translate(result.message[1]),
      });
    }
  } else {
    if (result.success) {
      toast.success(result.message[0], { description: result.message[1] });
    } else {
      toast.error(result.message[0], { description: result.message[1] });
    }
  }
}

export function capitalize(string: string) {
  return string.charAt(0).toUpperCase() + string.slice(1);
}

export function getDateFnsLocale(i18nLocale: string) {
  let dateFnsLocale;
  switch (i18nLocale) {
    case "pt":
      dateFnsLocale = pt;
      break;
    case "en":
      dateFnsLocale = enUS;
      break;
    case "de":
      dateFnsLocale = de;
```

```
      break;
    default:
      dateFnsLocale = pt;
  }
  if (!dateFnsLocale) throw new Error("Invalid locale passed in");
  return dateFnsLocale;
}

export function matchContactsToRecipients(
  rawRecipients: DBRecipient[],
  contacts: DBContact[]
) {
  // Return recipients if no data to filter
  if (!rawRecipients.length || !contacts.length)
    return rawRecipients as WithContact[];

  return rawRecipients.map((recipient) => ({
    ...recipient,
    contact: contacts.find((contact) => contact.phone === recipient.phone),
  })) as WithContact[];
}

export function rankRecipients(data: FetchedRecipient[]): RankedRecipient[]
{
  // Step 1: Create a unique array of recipients with their usage count
  const oneWeekAgo = new Date();
  oneWeekAgo.setDate(oneWeekAgo.getDate() - 7);

  const processedData: RankedRecipient[] = []; // Initialize an array for p
rocessed recipients
  const recipientMap = new Map<string, RankedRecipient>(); // Use a map to
track unique recipients

  data.forEach((recipient) => {
    // Filter out invalid data before processing to avoid unnecessary work.
    const { isValid } = validatePhoneNumber(recipient.phone);
    if (!isValid) {
      // Exit the current iteration if the recipient is invalid
      return;
    }

    // Check if the recipient already exists in the map
    if (!recipientMap.has(recipient.phone)) {
      recipientMap.set(recipient.phone, {
        id: recipient.id,
        phone: recipient.phone,
        usageCount: 0, // Initialize usage count
      });
    }

    // Increment usage count if the last_used date is within the last week
    if (new Date(recipient.last_used) >= oneWeekAgo) {
```

```
      recipientMap.get(recipient.phone)!.usageCount++; // Increment usage c
ount
    }
  });

  // Convert the map values to an array
  processedData.push(...recipientMap.values());

  // Step 2: Sort the recipients based on usage count
  return processedData.sort((a, b) => {
    // Sort by usage count (descending)
    return b.usageCount - a.usageCount;
  });
}

// Shuffle the array using Fisher-Yates algorithm
export function shuffleArray(arr: any[]) {
  for (let i = arr.length - 1; i > 0; i--) {
    const j = Math.floor(Math.random() * (i + 1));
    [arr[i], arr[j]] = [arr[j], arr[i]]; // Swap elements
  }
}

export function getPercentageChange(newValue: number, oldValue: number) {
  if (oldValue === 0) {
    // Old value is zero so we will have a 100% change if the newValue is n
ot zero
    return newValue === 0 ? 0 : newValue > 0 ? 100 : -100;
  }
  return Math.floor(((newValue - oldValue) / oldValue) * 100);
}

export function extractFirstWord(sentence: string) {
  // Split the sentence into words
  const words = sentence.split(" ");
  // Return the first word if it exists, otherwise return null
  return words.length > 0 ? words[0] : null;
}

export function getScrollAreaHeightStyles(additionalHeightPx: number) {
  return `h-[calc(100vh - var(--simple-header-height) - ${additionalHeightP
x}px)] md:h-[calc(100vh-var(--header-height)-${additionalHeightPx}px)]`;
}
```

/lib/actions/message.create.ts

```
"use server";

import db from "@/lib/db";
import { MessageSchema } from "../form.schemas";
```

```
import { Message } from "@/types";
import { getSession } from "../auth/sessions";
import { formatPhone } from "../utils";
import { NewRecipient } from "@/types/recipient";
import { ActionResponse } from "@/types/action";
import { revalidatePath } from "next/cache";

export async function sendMessage(
  existingDraftId: string | null,
  data: Message
): Promise<
  ActionResponse<Message> & {
    sendDate?: Date;
    invalidRecipients?: NewRecipient[];
    clearForm?: boolean;
  }
> {
  // 1. Check authentication
  const { isAuthenticated, user } = await getSession();
  const userId = user?.id;
  if (!isAuthenticated || !userId) {
    return {
      success: false,
      message: ["common:error-authentication"],
    };
  }

  // 2. Validate field types
  const validatedData = MessageSchema.safeParse(data);
  if (!validatedData.success) {
    return {
      success: false,
      message: ["common:fix_zod_errors"],
      errors: validatedData.error.flatten().fieldErrors,
    };
  }

  // 3. Validate recipients - these are not part of the zod schema as I nee
d to the validation myself
  if (!data.recipients.length) {
    return {
      success: false,
      message: ["new-message-page:server-no_recipients_error"],
    };
  }

  const { validRecipients, invalidRecipients } = analyzeRawRecipients(
    data.recipients
  );
  // The recipient error handling is not handled in the zod validation, so
we do validate them ourselves
  if (!validRecipients.length) {
```

```
    return {
      success: false,
      message: [`new-message-page:server-invalid_phone_numbers_error`],
      invalidRecipients,
    };
  }

  let scheduledUnixSeconds: number = 0;
  if (
    validatedData.data.secondsUntilSend &&
    validatedData.data.secondsUntilSend > 2
  ) {
    // JavaScript's Date object uses milliseconds, so we divide by 1000 to
the timestamp into seconds.
    scheduledUnixSeconds =
      Date.now() / 1000 + validatedData.data.secondsUntilSend;
  }

  const isScheduled =
    !!validatedData.data.secondsUntilSend &&
    validatedData.data.secondsUntilSend > 2; // api requires a minimum of 2
seconds in the future
  try {
    const payload = {
      // This shit can only be one full word with no special characters or
spaces
      sender: /**validatedData.data.sender */ "ETPZP", // Hardcode this for
now

      message: validatedData.data.body, // this can be any string

      recipients: validRecipients.map(({ phone }) => ({
        msisdn: phone,
      })),

      destaddr: "DISPLAY", // Flash SMS

      // The API is case-sensitive - `sendtime` has to be spelled exactly l
ike this
      sendtime: isScheduled ? scheduledUnixSeconds : undefined, // Insert t
he UNIX timestamp if the message is scheduled
    };

    const res = await fetch(`${process.env.GATEWAYAPI_URL}/rest/mtsms`, {
      method: "POST",
      headers: {
        Authorization: `Token ${process.env.GATEWAYAPI_TOKEN}`,
        "Content-Type": "application/json",
      },
      body: JSON.stringify(payload),
    });
    const resJson = await res.json();
```

```
    // -------- BEGIN DATABASE LOGIC -------- //
    if (typeof existingDraftId === "undefined" || !existingDraftId) {
      // Insert new message and recipients
      await db(
        `
          WITH insert_message AS (
            INSERT INTO message (
              subject,
              body,
              status,
              send_time,
              sms_reference_id,
              api_error_code,
              api_error_details_json,
              cost,
              cost_currency,
              user_id
            )
            VALUES ($1, $2, $3, $4, $5, $6, $7, $8, $9 $10)
            RETURNING id
          )
          INSERT INTO recipient (message_id, phone, index)
          SELECT
            insert_message.id,
            unnest($11::text[]) as phone,
            unnest($12::int[]) as index
          FROM insert_message;
        `,
        [
          // Message data
          validatedData.data.subject, // subject
          validatedData.data.body, // body
          res.ok // status
            ? scheduledUnixSeconds
              ? "SCHEDULED"
              : "SENT"
            : "FAILED",
          scheduledUnixSeconds // sendtime
            ? new Date(scheduledUnixSeconds * 1000)
            : new Date(Date.now()),
          resJson?.ids?.length ? resJson?.ids[0] : null, // sms_reference_i
d

          // Api errors
          res.ok ? null : res.status, // api_error_code
          res.ok ? null : JSON.stringify(resJson), // api_error_details_jso
n

          resJson.usage.total_cost,
          resJson.usage.currency,
```

```
          userId, // user_id

          // Recipients
          validRecipients.map((recipient) => recipient.phone), // phone num
ber array
          validRecipients.map((_, index) => index),
        ]
      );
    } else {
      // 1. Update message data
      const result = await db(
        `
          UPDATE message
          SET subject = $1,
              body = $2,
              status = $3,
              send_time = $4,
              sms_reference_id = $5,
              api_error_code = $6,
              api_error_details_json = $7,
              cost = $8,
              cost_currency = $9
          WHERE user_id = $10 AND id = $11
          RETURNING id;
        `,
        [
          // Message data
          validatedData.data.subject, // subject
          validatedData.data.body, // body
          res.ok // status
            ? scheduledUnixSeconds
              ? "SCHEDULED"
              : "SENT"
            : "FAILED",
          scheduledUnixSeconds // sendtime
            ? new Date(scheduledUnixSeconds * 1000)
            : new Date(Date.now()),
          resJson?.ids?.length ? resJson?.ids[0] : null, // sms_reference_i
d

          // Api errors
          res.ok ? null : res.status, // api_error_code
          res.ok ? null : JSON.stringify(resJson), // api_error_details_jso
n

          resJson.usage.total_cost,
          resJson.usage.currency,

          // Other
          userId, // user_id
          existingDraftId, // id of the database draft to update
        ]
```

```
      );

      // In case update did not match any rows - invalid message id
      if (result.rowCount === 0) {
        throw new Error("Invalid message id provided");
      }

      // 2. Delete old recipients
      await db(`DELETE FROM recipient WHERE message_id = $1`, [
        existingDraftId,
      ]);
      // 3. Then insert new recipients
      await db(
        `
        INSERT INTO recipient (message_id, phone, index)
        SELECT $1,
          unnest($2::text[]),
          unnest($3::int[])
        `, // check if for this query I can use VALUES instead of SELECT
        [
          existingDraftId,
          validRecipients.map((r) => r.phone),
          validRecipients.map((_, index) => index),
        ]
      );
    }
    // -------- END DATABASE LOGIC -------- //

    // Update the amount indicators in the nav panel
    revalidatePath("/new-message");

    if (!res.ok) {
      return {
        success: false,
        message: ["server-some_api_error"],
        clearForm: true,
      };
    }

    return {
      success: true,
      message: [
        isScheduled
          ? "new-message-page:server-schedule_success"
          : "new-message-page:server-send_success",
      ],
      sendDate: isScheduled ? new Date(scheduledUnixSeconds * 1000) : undef
ined,
      clearForm: true,
    };
  } catch (error) {
    console.log("Error got caught in catch block:", error);
```

```
    return {
      success: false,
      message: ["new-message-page:server-unknown_error"],
    };
  }
}

function analyzeRawRecipients(recipients: NewRecipient[]) {
  const validRecipients: NewRecipient[] = [];
  const invalidRecipients: NewRecipient[] = [];

  recipients.forEach((recipient) => {
    const parsedPhone = formatPhone(recipient.phone);
    if (parsedPhone) {
      validRecipients.push({
        ...recipient,
        phone: parsedPhone as string,
      });
    } else {
      invalidRecipients.push(recipient);
    }
  });

  return { validRecipients, invalidRecipients };
}
```

/lib/actions/_testing/responses

```
// a console.log(res) successful response will give something like this
const successResponse = /**Response */ {
  status: 200,
  statusText: "OK",
  headers: /**Headers */ {
    "content-length": "88",
    "content-type": "application/json",
    date: "Sun, 22 Dec 2024 09:17:28 GMT",
    "strict-transport-security": "max-age=31536000",
    "x-server": "GatewayAPI",
  },
  body: /**ReadableStream */ {
    locked: false,
    state: "readable",
    supportsBYOB: true,
  },
  bodyUsed: false,
  ok: true,
  redirected: false,
  type: "basic",
  url: "process.env.GATEWAYAPI_URL/rest/mtsms",
```

```
};

export const errorResponse = /**Response */ {
  status: 422,
  statusText: "Unprocessable Entity",
  headers: /**Headers */ {
    "content-length": "83",
    "content-type": "application/json",
    date: "Sun, 22 Dec 2024 09:10:28 GMT",
    "strict-transport-security": "max-age=31536000",
    "x-server": "GatewayAPI",
  },
  body: /**ReadableStream */ {
    locked: false,
    state: "readable",
    supportsBYOB: true,
  },
  bodyUsed: false,
  ok: false,
  redirected: false,
  type: "basic",
  url: "process.env.GATEWAYAPI_URL/rest/mtsms",
};

const errorResponse2 = /**Response */ {
  status: 403,
  statusText: "Forbidden",
  headers: /**Headers */ {
    "content-length": "122",
    "content-type": "application/json",
    date: "Sun, 22 Dec 2024 09:26:43 GMT",
    "strict-transport-security": "max-age=31536000",
    "x-server": "GatewayAPI",
  },
  body: /**ReadableStream */ {
    locked: false,
    state: "readable",
    supportsBYOB: true,
  },
  bodyUsed: false,
  ok: false,
  redirected: false,
  type: "basic",
  url: "process.env.GATEWAYAPI_URL/rest/mtsms",
};
```

/lib/actions/_testing/default-response.js

```
export const SuccessResponse = {
  status: 200,
```

```
  statusText: "OK",
  headers: {
    "content-length": "89",
    "content-type": "application/json",
    date: "Sun, 02 Feb 2025 13:03:24 GMT",
    "strict-transport-security": "max-age=31536000",
    "x-server": "GatewayAPI",
  },
  body: { locked: false, state: "readable", supportsBYOB: true },
  bodyUsed: false,
  ok: true,
  redirected: false,
  type: "basic",
  url: "process.env.GATEWAYAPI_URL/rest/mtsms",
};

export const SuccessResponseJson = {
  ids: [4382703917],
  usage: { countries: { DE: 1 }, currency: "EUR", total_cost: 0.0642 },
};

/**
 * Cancel scheduled responses
 Response {
  status: 410,
  statusText: 'Gone',
  headers: Headers {
    'content-length': '3',
    'content-type': 'application/json',
    date: 'Wed, 05 Feb 2025 12:10:48 GMT',
    'strict-transport-security': 'max-age=31536000',
    'x-server': 'GatewayAPI'
  },
  body: ReadableStream { locked: false, state: 'readable', supportsBYOB: tr
ue },
  bodyUsed: false,
  ok: false,
  redirected: false,
  type: 'basic',
  url: 'process.env.GATEWAYAPI_URL/rest/mtsms/4382980628'
}
 */
```

/lib/actions/message.actions.ts

```
"use server";

import {
  DraftActionResponse,
  ActionResponse,
```

```
  DataActionResponse,
} from "@/types/action";
import { getSession } from "../auth/sessions";
import db from "../db";
import { revalidatePath } from "next/cache";
import { DBMessage, Message } from "@/types";
import { sleep } from "../utils";

export async function toggleTrash(
  id: string,
  inTrash: boolean
): Promise<ActionResponse<null>> {
  const session = await getSession();
  const userId = session?.user?.id;

  try {
    if (!userId) throw new Error("Invalid user id.");
    await db(
      `
        UPDATE message
        SET in_trash = $1
        WHERE user_id = $2 AND id = $3;
      `,
      [inTrash, userId, id]
    );

    revalidatePath("/failed"); // we need this

    // Don't know why it works without the following lines. We need to test
this in production and if necessary, uncomment these lines
    // revalidatePath("/sent");
    // revalidatePath("/trash");

    return {
      success: true,
      message: [
        inTrash
          ? "messages-page:server-move_trash_success"
          : "messages-page:server-restore_success",
      ],
    };
  } catch (error) {
    return {
      success: false,
      message: [
        inTrash
          ? "messages-page:server-move_trash_unknown_error"
          : "messages-page:server-restore_unknown_error",
      ],
    };
  }
}
```

```
export async function deleteMessage(
  id: string,
  pathname?: string
): Promise<ActionResponse<null>> {
  const session = await getSession();
  const userId = session?.user?.id;

  try {
    if (!userId) throw new Error("Invalid user id.");
    await db(`DELETE FROM message WHERE user_id = $1 AND id = $2`, [
      userId,
      id,
    ]);

    if (pathname) revalidatePath(pathname);

    return {
      success: true,
      message: ["common:server-delete_message_success"],
    };
  } catch (error) {
    return {
      success: false,
      message: ["common:server-delete_message_unknown_error"],
    };
  }
}

export async function cancelCurrentlyScheduled(
  sms_reference_id: number
): Promise<DataActionResponse<DBMessage | undefined>> {
  const session = await getSession();
  const userId = session?.user?.id;

  try {
    if (!userId) throw new Error("Invalid user id.");

    const res = await fetch(
      `${process.env.GATEWAYAPI_URL}/rest/mtsms/${sms_reference_id}`,
      {
        method: "DELETE",
        headers: {
          Authorization: `Token ${process.env.GATEWAYAPI_TOKEN}`,
          "Content-Type": "application/json",
        },
      }
    );

    if (!res.ok) {
      return {
        success: false,
```

```
        message: [`api_error_${res.status}`],
      };
    }

    // TEST_PRODUCTION: This did not work with the 2 WHERE conditions on th
e dev server on Windows
    const result = await db(
      `
        UPDATE message
        SET status = 'FAILED', api_error_code = 409
        WHERE user_id = $1 AND sms_reference_id = $2;
      `,
      [userId, sms_reference_id]
    );

    revalidatePath("/scheduled");
    return {
      success: true,
      message: ["messages-page:server-cancel_scheduled_success"],
      data: result.rows[0],
    };
  } catch (error) {
    console.log(error);

    return {
      success: false,
      message: ["messages-page:server-cancel_scheduled_unknown_error"],
      data: undefined,
    };
  }
}

export async function saveDraft(
  draftId: string | undefined,
  data: Message,
  pathname?: string
): Promise<DraftActionResponse<string>> {
  const session = await getSession();
  const userId = session?.user?.id;
  let draft;

  try {
    if (!userId) throw new Error("Invalid user id.");

    if (draftId) {
      // 1. Delete old recipients first
      await db(`DELETE FROM recipient WHERE message_id = $1`, [draftId]);

      // 2. Insert the new recipients after that
      // We are await these separately so that we can be sure that there ar
e no duplicate recipients
      draft = await db(
```

```
        `
            WITH insert_message AS (
              UPDATE message SET subject = $3, body = $4, sender = $5 WHERE
id = $2 AND user_id = $1
              RETURNING id
            ),
            insert_recipients AS (
              INSERT INTO recipient (message_id,  phone, index)
              SELECT
                insert_message.id,
                unnest($6::text[]) as phone,
                unnest($7::int[]) as index
              FROM insert_message
            )
            SELECT * FROM insert_message
          `,
          [
            userId,
            draftId,
            data.subject,
            data.body,
            data.sender,

            // Recipients
            data.recipients.map((recipient) => recipient.phone), // phone num
ber array
            data.recipients.map((_, index) => index), // for the ordering of
the recipient
          ]
        );
      } else {
        // Create new draft
        draft = await db(
          `
            WITH insert_message AS (
              INSERT INTO message (user_id, subject, body, sender, status)
              VALUES ($1, $2, $3, $4, $5)
              RETURNING id
            ),
            insert_recipients AS (
              INSERT INTO recipient (message_id, phone, index)
              SELECT
                insert_message.id,
                unnest($6::text[]) as phone,
                unnest($7::int[]) as index
              FROM insert_message
            )
            SELECT id FROM insert_message
          `,
          [
            userId,
            data.subject,
            data.body,
```

```
          data.sender,
          "DRAFTED",

          // Recipients
          data.recipients.map((recipient) => recipient.phone), // phone num
ber array
          data.recipients.map((_, index) => index), // for persisting the u
ser specified recipient order
        ]
      );
    }

    if (pathname) revalidatePath(pathname);

    return {
      success: true,
      message: ["common:server-save_draft_success"],
      draftId: draftId || draft.rows[0].id,
    };
  } catch (error) {
    return {
      success: false,
      message: ["common:server-save_draft_unknown_error"],
    };
  }
}
```

/lib/actions/contact.actions.ts

```
"use server";
// !!If you are using the contacts context, refetch contacts on client afte
r each server action instead of revalidating!!
import db from "../db";
import { ContactSchema } from "../form.schemas";
import { DBContact } from "@/types/contact";
import { getSession } from "../auth/sessions";
import { formatPhone } from "../utils";
import { revalidatePath } from "next/cache";
import { DatabaseError } from "pg";
import { ActionResponse, CreateContactResponse } from "@/types/action";
import { z } from "zod";

// Binding pathname is unnecessary since we re-fetch the context, and conta
cts won't be re-fetched on revalidation.
export async function createContact(
  _: CreateContactResponse | null,
  formData: FormData
): Promise<CreateContactResponse> {
  const session = await getSession();
  const userId = session?.user?.id;
```

```
  const rawData = {
    name: formData.get("name") as string,
    phone: formData.get("phone") as string,
    description: formData.get("description") as string,
  };
  const validatedData = ContactSchema.safeParse(rawData);
  if (!validatedData.success) {
    return {
      success: false,
      message: ["common:fix_zod_errors"],
      errors: validatedData.error.flatten().fieldErrors,
      inputs: rawData,
    };
  }

  try {
    if (!userId) throw new Error("Invalid user id.");

    const { name, phone, description } = validatedData.data;
    const validatedPhone = formatPhone(phone);
    if (!validatedPhone)
      throw new Error("Phone number is unexpectedly invalid!");

    const result = await db(
      `INSERT INTO contact (user_id, name, phone, description) VALUES ($1,
$2, $3, $4) RETURNING *`,
      [userId, name, validatedPhone, description || null]
    );
    console.log(result.rows[0]);

    return {
      success: true,
      message: ["modals:create_contact-success"],
      data: result.rows[0],
    };
  } catch (error) {
    let message = "";
    if (error instanceof DatabaseError && error.code === "23505") {
      // check if it is a duplicate key error by comparing it with the erro
r code
      message = "modals:zod_error-duplicate_phone";
    } else {
      message = "modals:create_contact-unknown_error";
    }

    return {
      success: false,
      message: [message],
      inputs: rawData,
    };
  }
```

```
}

export async function updateContact(
  id: string,
  _: ActionResponse<DBContact> | null,
  formData: FormData
): Promise<ActionResponse<z.infer<typeof ContactSchema>>> {
  const session = await getSession();
  const userId = session?.user?.id;

  const rawData = {
    name: formData.get("name") as string,
    phone: formData.get("phone") as string,
    description: formData.get("description") as string,
  };
  const validatedData = ContactSchema.safeParse(rawData);
  if (!validatedData.success) {
    return {
      success: false,
      message: ["common:fix_zod_errors"],
      errors: validatedData.error.flatten().fieldErrors,
      inputs: rawData,
    };
  }
  try {
    if (!userId) throw new Error("Invalid user id.");

    const { name, phone, description } = validatedData.data;
    const validatedPhone = formatPhone(phone);

    await db(
      "UPDATE contact SET name = $1, phone = $2, description = $3 WHERE use
r_id = $4 AND id = $5",
      [name, validatedPhone, description || null, userId, id]
    );

    return { success: true, message: ["modals:edit_contact-success"] };
  } catch (error) {
    let message;
    if (error instanceof DatabaseError && error.code === "23505") {
      // check if it is a duplicate key error by comparing it with the erro
r code
      message = "modals:zod_error-duplicate_phone";
    } else {
      message = "modals:create_contact-unknown_error";
    }

    return {
      success: false,
      message: [message],
      inputs: rawData,
    };
```

```
  }
}

export async function deleteContact(
  id: string
): Promise<ActionResponse<undefined>> {
  const session = await getSession();
  const userId = session?.user?.id;

  try {
    if (!userId) throw new Error("Invalid user id.");
    await db("DELETE FROM contact WHERE user_id = $1 AND id = $2", [
      userId,
      id,
    ]);

    return {
      success: true,
      message: ["contacts-page:server-delete_success"],
    };
  } catch (error) {
    return {
      success: false,
      message: ["contacts-page:server-delete_unknown_error"],
    };
  }
}
```

/lib/actions/user.actions.ts

```
"use server";

import type { DBUser, SettingName, User } from "@/types/user";
import db from "../db";
import ActiveDirectory from "activedirectory2";
import { z } from "zod";
import { UpdateSettingSchema } from "../form.schemas";
import {
  ActionResponse,
  DataActionResponse,
  UpdateSettingResponse,
} from "@/types/action";
import { getSession } from "../auth/sessions";
import { validSettingNames } from "@/types/user";

// These are guaranteed properties when you find the user using A.D.
type userResult = {
  displayName: string; // display name

  givenName: string; // first name
```

```
  sn: string; // surname

  cn: string; // full name
};

export default async function saveUser(
  ad: ActiveDirectory,
  email: string,
  isAdmin: boolean
): Promise<DataActionResponse<User>> {
  try {
    const selectResult = await db(
      "SELECT * FROM public.user WHERE email = $1;",
      [email]
    );
    if (selectResult.rows.length) {
      return {
        success: true,
        message: ["Authentication successful!", "User already exists"],
        data: selectResult.rows[0],
      };
    } else {
      // User has never signed up before

      return new Promise((resolve) => {
        ad.findUser(email, async (err, user: any) => {
          if (err || !user) {
            resolve({ success: false, message: ["User not found."] });
            return;
          }

          const { cn, displayName, givenName, sn } = user;
          try {
            const insertResult = await db(
              "INSERT INTO public.user (email, name, role, first_name, last
_name, display_name) VALUES ($1, $2, $3, $4, $5, $6) RETURNING *;",
              [
                email, // email
                cn, // complete name
                isAdmin ? "ADMIN" : "USER",
                givenName, // first name
                sn, // surname
                displayName,
              ]
            );

            resolve({
              success: true,
              message: ["Authentication successful!", "New user created"],
              data: insertResult.rows[0],
            });
          } catch (error) {
```

```
            resolve({
              success: false,
              message: ["Error occurred", "Failed to create user in databas
e."],
            });
          }
        });
      });
    }
  } catch (error) {
    return {
      success: false,
      message: ["Error occurred", "Failed to create or fetch user."],
    };
  }
}

export async function dummySaveUser(
  user: DBUser
): Promise<DataActionResponse<User>> {
  try {
    const selectResult = await db(
      "SELECT * FROM public.user WHERE email = $1;",
      [user.email]
    );
    if (selectResult.rows.length) {
      return {
        success: true,
        message: ["Authentication successful!", "User already exists"],
        data: selectResult.rows[0],
      };
    } else {
      // User has never signed up before
      try {
        const insertResult = await db(
          "INSERT INTO public.user (email, name, role, first_name, last_nam
e, display_name) VALUES ($1, $2, $3, $4, $5, $6) RETURNING id, name, email,
role, first_name, last_name;",
          [
            user.email,
            user.name,
            user.role,
            user.first_name,
            user.last_name,
            `${user.first_name} ${user.last_name}`,
          ]
        );

        return {
          success: true,
          message: ["Authentication successful!", "New user created"],
          data: insertResult.rows[0],
        };
```

```
      } catch (error) {
        return {
          success: false,
          message: ["Error occurred", "Failed to create user in database."]
,
        };
      }
    }
  } catch (error) {
    console.log("Dummy save user error:", error);

    return {
      success: false,
      message: ["Error occurred", "Failed to create or fetch user."],
    };
  }
}

// Settings page calls this function to update one setting at a time
export async function updateSetting(
  formData: FormData
): Promise<UpdateSettingResponse> {
  const session = await getSession();
  const userId = session?.user?.id;

  // Extract raw data from the form
  const rawData = {
    name: formData.get("name") as SettingName,
    value: formData.get("value") as string,
  };

  if (!validSettingNames.includes(rawData.name)) {
    return {
      success: false,
      error: "Invalid setting",
      input: rawData.value,
    };
  }
  try {
    if (!userId) throw new Error("Invalid user id.");
    // Try to validate and parse the raw data.
    const parsedData = UpdateSettingSchema.parse(rawData);

    // If validation passed, you can proceed to update the database accordi
ngly.
    const { rows } = await db(
      `UPDATE public.user SET ${parsedData.name} = $2, updated_at = NOW() W
HERE id = $1 RETURNING *;`,
      [userId, parsedData.value]
    );

    return {
```

```
      success: true,
      input: rawData.value,
      data: rows[0][parsedData.name],
    };
  } catch (error) {
    // If the error is produced by zod, extract and send back the error det
ails.
    if (error instanceof z.ZodError) {
      const { fieldErrors } = error.flatten();

      // One option: join all errors from all fields
      const errorString = Object.values(fieldErrors)
        .flat()
        .filter(Boolean)
        .join(", ");
      return {
        success: false,
        error: errorString,
        input: rawData.value,
      };
    }

    // For any other kind of error, return a generic error message.
    return {
      success: false,
      input: rawData.value,
      error: "Something went wrong while saving this input",
    };
  }
}
```

/lib/db/general.ts

```
"use server";

import { AmountIndicators } from "@/types";
import { getSession } from "../auth/sessions";
import db from ".";
import { UserSettings } from "@/types/user";

export async function fetchUserSettings() {
  const session = await getSession();
  const userId = session?.user?.id;

  try {
    if (!userId) throw new Error("Invalid user id.");
    const { rows } = await db(
      `
      SELECT lang, profile_color_id, display_name, dark_mode, primary_color
_id, appearance_layout
```

```
      FROM public.user WHERE id = $1;
    `,
      [userId]
    );
    return rows[0] as UserSettings;
  } catch (error) {}
}

export async function fetchAmountIndicators() {
  const session = await getSession();
  const userId = session?.user?.id;

  try {
    if (!userId) throw new Error("Invalid user id.");
    // We need to do two separate queries, because I got issues when trying
to merge it into one. Maybe come back to this later and create one query to
all of them.
    const messageCount = await db(
      `
        SELECT
            COALESCE(SUM(CASE
              WHEN
                user_id = $1 AND
                in_trash = false AND
                status NOT IN ('FAILED', 'DRAFTED') AND
                send_time <= NOW()
              THEN 1 END), 0)::INTEGER AS sent,
            COALESCE(SUM(CASE
              WHEN
                user_id = $1 AND
                in_trash = false AND
                status NOT IN ('FAILED', 'DRAFTED') AND
                send_time > NOW()
              THEN 1 END), 0)::INTEGER AS scheduled,
            COALESCE(SUM(CASE WHEN status = 'FAILED' AND in_trash = false T
HEN 1 END), 0)::INTEGER AS failed,
            COALESCE(SUM(CASE WHEN status = 'DRAFTED' AND in_trash = false
THEN 1 END), 0)::INTEGER AS drafts,
            COALESCE(SUM(CASE WHEN in_trash = true THEN 1 END), 0)::INTEGER
AS trash
        FROM
            message
        WHERE
            user_id = $1;
      `,
      [userId]
    );
    const contactsCount = await db(
      `
      SELECT
          CAST(COUNT(c.id) AS INTEGER)
      FROM
          contact c
```

```
      WHERE
          c.user_id = $1;
    `,
      [userId]
    );

    return {
      ...messageCount.rows[0],
      contacts: contactsCount.rows[0].count,
    } as AmountIndicators;
  } catch (error) {}
}
```

/lib/db/contact.ts

```
"use server";

import { DBContact } from "@/types/contact";
import db from ".";
import { getSession } from "../auth/sessions";

export async function fetchContacts() {
  const session = await getSession();
  const userId = session?.user?.id;

  try {
    if (!userId) throw new Error("Invalid user id.");
    const result = await db(
      `
        SELECT * FROM contact
        WHERE user_id = $1
      `,
      [userId]
    );

    return result.rows as DBContact[];
  } catch (error) {}
}
```

/lib/db/seed.sql

```
-- Create user table
CREATE TABLE "user" (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) NOT NULL,
    email VARCHAR(255) UNIQUE NOT NULL,
    role VARCHAR(20) CHECK (role IN ('USER', 'ADMIN')) NOT NULL,
```

```
    created_at TIMESTAMP NOT NULL DEFAULT NOW(),
    updated_at TIMESTAMP NOT NULL DEFAULT NOW(),
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    -- User settings:
    -- All have defaults except display name, which defaults to the user's
AD name when they first sign up.
    lang VARCHAR(2) NOT NULL DEFAULT 'pt', -- ISO 639-1 language code
    profile_color_id SMALLINT NOT NULL DEFAULT 2,
    display_name VARCHAR(50) NOT NULL,
    primary_color_id SMALLINT NOT NULL DEFAULT 1,
    appearance_layout VARCHAR(20) CHECK (appearance_layout IN ('MODERN', 'S
IMPLE')) NOT NULL DEFAULT 'MODERN',
    dark_mode BOOLEAN NOT NULL DEFAULT false
);

-- Create message table
CREATE TABLE "message" (
    id SERIAL PRIMARY KEY,
    user_id INTEGER NOT NULL REFERENCES "user"(id) ON DELETE CASCADE,
    sender VARCHAR(255),
    subject VARCHAR(255),
    body TEXT NOT NULL,
    created_at TIMESTAMP NOT NULL DEFAULT NOW(),
    send_time TIMESTAMP DEFAULT NOW() NOT NULL, -- can be null if the messa
ge is a draft
    sms_reference_id BIGINT, -- can be null if the message is not scheduled
, failed, or a draft
    status VARCHAR(20) NOT NULL CHECK (status IN ('SENT', 'SCHEDULED', 'FAI
LED', 'DRAFTED')), -- scheduled messages will remain with status "SCHEDULED
", even when their delivery date is reached
    in_trash BOOLEAN NOT NULL DEFAULT false,
    api_error_code SMALLINT, -- This is the http status code which is saved
when an error occurs
    api_error_details_json TEXT,
    cost NUMERIC(6, 4), -- 6 total digits, 4 digits after the decimal
    cost_currency VARCHAR(10) -- assumed to be in EUR
);

-- Create contacts table
CREATE TABLE "contact" (
    id SERIAL PRIMARY KEY,
    user_id INTEGER NOT NULL REFERENCES "user"(id) ON DELETE CASCADE,
    name VARCHAR(255) NOT NULL,
    phone VARCHAR(50) NOT NULL,
    description VARCHAR(255),
    created_at TIMESTAMP NOT NULL DEFAULT NOW(),
    updated_at TIMESTAMP NOT NULL DEFAULT NOW(),
    UNIQUE (user_id, phone) -- The same phone number may exist between diff
erent user, but there cannot be contacts with the same phone number for one
user.
);
```

```
-- Create recipient table
CREATE TABLE recipient (
    id SERIAL PRIMARY KEY,
    message_id INTEGER REFERENCES message(id) ON DELETE CASCADE,
    phone VARCHAR(50) NOT NULL, -- Store phone numbers as VARCHAR to accomm
odate various formats
    index SMALLINT NOT NULL, -- This is the used for persisting the order o
f the recipients of a message
    UNIQUE (message_id, phone) -- Ensure a phone number can only be added o
nce per message. This is not an actual field in the table, but it will make
sure that there are no recipients with duplicate links
);

-- Insert a scheduled message for testing:
-- INSERT INTO "message" (
--     user_id,
--     sender,
--     subject,
--     body,
--     send_time,
--     status,
--     in_trash,
--     api_error_code,
--     api_error_details_json
-- ) VALUES (
--     1,
--     'john.doe@example.com',
--     'Meeting Reminder at 2pm',
--     'Don"t forget about the meeting tomorrow at 14 AM.',
--     NOW(),
--     'SENT',
--     false,
--     NULL,
--     NULL
-- );
```

/lib/db/Dockerfile

```
FROM postgres:17.4-alpine3.21

COPY seed.sql /docker-entrypoint-initdb.d/
```

/lib/db/dashboard.ts

```
import { getSession } from "../auth/sessions";
import db from ".";
import { DBUser } from "@/types/user";
import { format } from "date-fns";
```

```
import { CountryStat } from "@/app/[locale]/dashboard/page";
import { ISO8601_DATE_FORMAT as API_DATE_FORMAT } from "@/global.config";
import { LightDBMessage } from "@/types/dashboard";
import { DateRangeSchema } from "../form.schemas";

export async function fetchMessagesInDateRange(input: {
  startDate: string;
  endDate: string;
}) {
  const session = await getSession();

  try {
    if (!session?.isAdmin || !session?.isAuthenticated)
      throw new Error("User is not an admin or not authenticated.");

    // Validate input using Zod
    const validatedDates = DateRangeSchema.parse(input);
    const { startDate, endDate } = validatedDates;

    const result = await db(
      `
          SELECT id, user_id, send_time, cost FROM message m
          WHERE
            m.in_trash = false AND
            m.status NOT IN ('FAILED', 'DRAFTED') AND
            m.send_time BETWEEN $1 AND $2
          ORDER BY send_time ASC;
        `,
      [startDate, endDate]
    );

    return result.rows as LightDBMessage[];
  } catch (error) {}
}

export async function fetchUsers() {
  const session = await getSession();

  try {
    if (!session?.isAdmin || !session?.isAuthenticated)
      throw new Error("User is not an admin or not authenticated.");
    const result = await db(`SELECT * FROM public.user;`);

    return result.rows as DBUser[];
  } catch (error) {}
}

export async function fetchCountryStats(input: {
  startDate: string;
  endDate: string;
}): Promise<CountryStat[] | undefined> {
  if (!input.startDate) return undefined;
```

```
  const session = await getSession();

  try {
    if (!session?.isAdmin || !session?.isAuthenticated)
      throw new Error("User is not an admin or not authenticated.");

    // Validate input using Zod
    const validatedDates = DateRangeSchema.safeParse(input);
    if (!validatedDates.success || validatedDates.data.startDate == undefin
ed)
      throw new Error("Invalid input.");
    const { startDate, endDate } = validatedDates.data;

    const res = await fetch(`${process.env.GATEWAYAPI_URL}/api/usage/labels
`, {
      method: "POST",
      headers: {
        Authorization: `Token ${process.env.GATEWAYAPI_TOKEN}`,
        Accept: "application/json, text/javascript",
        "Content-Type": "application/json",
      },
      body: JSON.stringify({
        from: format(startDate, API_DATE_FORMAT),
        to: format(endDate || new Date(), API_DATE_FORMAT),
      }),
    });
    if (!res.ok) {
      throw new Error("Network response was not ok");
    }
    const resJson = await res.json();

    return resJson
      .filter((country: { label: string | null }) => country.label === null
)
      .map(
        (item: {
          amount: number;
          cost: number;
          country: string;
          currency: string;
          label: null;
        }) => ({
          country: item.country,
          cost: item.cost,
          amount: item.amount,
        })
      );
  } catch (error) {
    console.log(error);
  }
}
```

/lib/db/_seed-data.sql

```
INSERT INTO "user" (name, email, role, created_at, updated_at, first_name, last_name, lang, profile_color_id, display_name, dark_mode, primary_color_id)
VALUES
    ('Alice Johnson', 'alice@example.com', 'USER', NOW(), NOW(), 'Alice', 'Johnson', 'en', 1, 'Alice J.', false, 1),
    ('Bob Smith', 'bob@example.com', 'USER', NOW(), NOW(), 'Bob', 'Smith', 'en', 1, 'Bob S.', false, 1),
    ('Charlie Brown', 'charlie@example.com', 'ADMIN', NOW(), NOW(), 'Charlie', 'Brown', 'en', 1, 'Charlie B.', false, 1),
    ('David Wilson', 'david@example.com', 'USER', NOW(), NOW(), 'David', 'Wilson', 'pt', 1, 'David W.', false, 1),
    ('Eve Davis', 'eve@example.com', 'ADMIN', NOW(), NOW(), 'Eve', 'Davis', 'pt', 1, 'Eve D.', true, 1),
    ('Frank Miller', 'frank@example.com', 'USER', NOW(), NOW(), 'Frank', 'Miller', 'en', 1, 'Frank M.', false, 1),
    ('Grace Lee', 'grace@example.com', 'USER', NOW(), NOW(), 'Grace', 'Lee', 'en', 1, 'Grace L.', false, 1),
    ('Hank Green', 'hank@example.com', 'USER', NOW(), NOW(), 'Hank', 'Green', 'pt', 1, 'Hank G.', true, 1),
    ('Irene Taylor', 'irene@example.com', 'ADMIN', NOW(), NOW(), 'Irene', 'Taylor', 'en', 1, 'Irene T.', false, 1),
    ('Jack White', 'jack@example.com', 'USER', NOW(), NOW(), 'Jack', 'White', 'pt', 1, 'Jack W.', false, 1);
```

/lib/db/message.ts

```
"use server";

import db from ".";
import { DBMessage, StatusEnums } from "@/types";
import { getSession } from "../auth/sessions";
import { NewRecipient } from "@/types/recipient";

export async function fetchMessagesByStatus(status: StatusEnums) {
  const session = await getSession();
  const userId = session?.user?.id;

  try {
    if (!userId) throw new Error("Invalid user id.");
    const result = await db(
      `
        SELECT m.*,
              COALESCE(
                json_agg(
                  json_build_object(
                      'id', r.id,
```

```
                    'phone', r.phone
                  ) ORDER BY r.phone -- Order by phone number numerically
                ) FILTER (WHERE r.id IS NOT NULL), '[]'::json
              ) AS recipients
        FROM message m
        LEFT JOIN recipient r ON m.id = r.message_id
        WHERE m.user_id = $1 AND m.status = $2 AND m.in_trash = false
        GROUP BY m.id
        ORDER BY m.created_at DESC;
      `,
      [userId, status]
    );

    return result.rows as DBMessage[];
  } catch (error) {}
}

export async function fetchTrashedMessages() {
  const session = await getSession();
  const userId = session?.user?.id;

  try {
    if (!userId) throw new Error("Invalid user id.");
    const result = await db(
      `
        SELECT m.*,
              COALESCE(
                json_agg(
                  json_build_object(
                    'id', r.id,
                    'phone', r.phone
                  ) ORDER BY r.phone -- Order by phone number numerically
                ) FILTER (WHERE r.id IS NOT NULL), '[]'::json
              ) AS recipients
        FROM message m
        LEFT JOIN recipient r ON m.id = r.message_id
        WHERE m.user_id = $1 AND m.in_trash = true
        GROUP BY m.id
        ORDER BY m.created_at DESC;
      `,
      [userId]
    );

    return result.rows as DBMessage[];
  } catch (error) {}
}

export async function fetchSentIn(time: "FUTURE" | "PAST") {
  const session = await getSession();
  const userId = session?.user?.id;

  try {
```

```
    if (!userId) throw new Error("Invalid user id.");
    const result = await db(
      `
        SELECT m.*,
               COALESCE(
                 json_agg(
                   json_build_object(
                       'id', r.id,
                       'phone', r.phone
                   ) ORDER BY r.phone -- Order by phone number numerically
                 ) FILTER (WHERE r.id IS NOT NULL), '[]'::json
               ) AS recipients
        FROM message m
        LEFT JOIN recipient r ON m.id = r.message_id
        WHERE
          m.user_id = $1 AND
          m.in_trash = false AND
          m.status NOT IN ('FAILED', 'DRAFTED') AND
          m.send_time ${time === "PAST" ? "<=" : ">"} NOW()
        GROUP BY m.id
        ORDER BY m.send_time ${time === "FUTURE" ? "ASC" : "DESC"};
      `,
      [userId]
    );

    return result.rows as DBMessage[];
  } catch (error) {}
}

export async function fetchDraft(id?: string) {
  const session = await getSession();
  const userId = session?.user?.id;

  try {
    if (!id) throw new Error("Invalid draft ID");
    if (!userId) throw new Error("Invalid user id");

    const result = await db(
      `
       SELECT m.*,
              COALESCE(
                json_agg(
                  json_build_object(
                      'id', r.id,
                      'phone', r.phone
                  ) ORDER BY r.index -- This determines in which order the
recipient chips are on new-message
                ) FILTER (WHERE r.id IS NOT NULL), '[]'::json
              ) AS recipients
        FROM message m
        LEFT JOIN recipient r ON m.id = r.message_id
        WHERE m.user_id = $1 AND m.id = $2 AND (m.status = 'DRAFTED' OR m.s
```

```
tatus = 'FAILED') -- Add the ability to edit FAILED messages from the new-m
essage-page later on
        GROUP BY m.id;
      `,
      [userId, id]
    );

    return result.rows[0] as DBMessage & { recipients: NewRecipient[] };
  } catch (error) {}
}
```

/lib/db/recipients.ts

```
"use server";

import db from ".";
import { getSession } from "../auth/sessions";
import { DBRecipient } from "@/types/recipient";

export async function fetchRecipients() {
  const session = await getSession();
  const userId = session?.user?.id;

  try {
    if (!userId) throw new Error("Invalid user id.");
    const { rows } = await db(
      `
        SELECT
          r.id,
          r.phone,
          m.created_at AS last_used
        FROM recipient r
        JOIN message m ON r.message_id = m.id
        WHERE m.user_id = $1;
      `,
      [userId]
    );

    return rows as (DBRecipient & { last_used: Date })[];
  } catch (error) {}
}
```

/lib/db/index.ts

```
import { Pool, QueryResult } from "pg";

const pool = new Pool({
```

```
  host: process.env.POSTGRES_HOST,
  port: Number(process.env.POSTGRES_PORT),
  user: process.env.POSTGRES_USER,
  password: process.env.POSTGRES_PASSWORD,
  database: process.env.POSTGRES_DB,
});

async function db(query: string, params?: any[]): Promise<QueryResult> {
  const client = await pool.connect();
  try {
    const res = await client.query(query, params);
    return res;
  } catch (err) {
    console.error("Database query error", err);
    throw err; // Rethrow the error for handling in the calling function
  } finally {
    client.release(); // Always release the client back to the pool
  }
}

export default db;

// For testing database connections
// db("SELECT $1::text as message", ["Hello world!"])
//   .then(() => console.log("Connected to Postgres!"))
//   .catch((err) => console.error("Error connecting to Postgres!", err));
```

/components.json

```
{
  "$schema": "https://ui.shadcn.com/schema.json",
  "style": "new-york",
  "rsc": true,
  "tsx": true,
  "tailwind": {
    "config": "tailwind.config.ts",
    "css": "app/globals.css",
    "baseColor": "slate",
    "cssVariables": false,
    "prefix": ""
  },
  "aliases": {
    "components": "@/components",
    "utils": "@/lib/utils",
    "ui": "@/components/ui",
    "lib": "@/lib",
    "hooks": "@/hooks"
  },
  "iconLibrary": "lucide"
}
```

/tsconfig.json

```
{
  "compilerOptions": {
    "target": "ES2017",
    "lib": ["dom", "dom.iterable", "esnext"],
    "allowJs": true,
    "skipLibCheck": true,
    "strict": true,
    "noEmit": true,
    "esModuleInterop": true,
    "module": "esnext",
    "moduleResolution": "bundler",
    "resolveJsonModule": true,
    "isolatedModules": true,
    "jsx": "preserve",
    "incremental": true,

    // added manually:
    "allowImportingTsExtensions": true,

    "plugins": [
      {
        "name": "next"
      }
    ],
    "paths": {
      "@/*": ["./*"]
    }
  },
  "include": [
    "next-env.d.ts",
    "**/*.ts",
    "**/*.tsx",
    ".next/types/**/*.ts",
    "app/[locale]/login/page.tsx",
    "app/[locale]/(app)/(other)/new-message/not-found.js",
    "app/[locale]/(root)/(message-layout)/error.tsx"
  ],
  "exclude": ["node_modules"]
}
```

/nginx.conf

```
# NOTE: Nginx doesn't read from this file, I am just committing to have the
config available when I need it
# Main context (this is the global configuration)
```

```
worker_processes  1;

events {
    worker_connections  1024;
}

http {
    include mime.types;

    # Optional server block for HTTP to HTTPS redirection
    server {
        listen 80;
        server_name localhost;

        # Redirect all HTTP requests to HTTPS
        return 301 https://\$host\$request_uri;
    }

    # Main server block
    server {
        listen 443 ssl; # Listen on port 443 for HTTPS
        server_name localhost;

        # Here are my self signed certs. In actual production you would let
these be signed by a organization
        ssl_certificate /Users/<your_user>/nginx-certs/nginx-selfsigned.crt
;
        ssl_certificate_key /Users/<your_user>/nginx-certs/nginx-selfsigned
.key;

        # Proxying requests to the Docker container (assuming it is running
on port 3000)
        location / {
            # Tell nginx to act as a reverse proxy to forward requests to t
he node servers
            proxy_pass http://localhost:3000;
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header X-Forwarded-Proto $scheme;
            proxy_set_header X-Forwarded-Host $host;
            proxy_set_header X-Forwarded-Port $server_port;
            proxy_set_header Cookie $http_cookie;  # Forward cookies
        }
    }
}
```

/.env.example

```
APP_NAME="ETPZP SMS"
```

```
# Translations
I18NEXUS_API_KEY="<nexus_key>"

# Active Directory (AD)
AD_URL="ldap://<ip>"
AD_BASE_DN="dc=<dc_name>,dc=<dc_type>"
AD_EMAIL="<ad_email>"
AD_PASSWORD="<secret_password>"

# GATEWAYAPI Api
GATEWAYAPI_URL="https://gatewayapi.com"
GATEWAYAPI_TOKEN="<smsapi_token>"

# postgres database
POSTGRES_USER="<dbuser>"
POSTGRES_PASSWORD="<secret_password>"
POSTGRES_HOST="<database.server.com>"
POSTGRES_PORT="<3211>"
POSTGRES_DB="<mydb>"

# Auth encryption key
SESSION_SECRET="<secret_password>"
```

/eslint.config.mjs

```
export default tseslint.config({
  rules: {
    // Note: you must disable the base rule as it can report incorrect erro
rs
    // "no-unused-vars": "on",

  },
});
```

/.eslintrc.json

```
{
  "extends": ["next/core-web-vitals", "next/typescript"]
}
```

/next.config.ts

```
import type { NextConfig } from "next";
// import path from 'path';

const nextConfig: NextConfig = {
  // Recommended: this will reduce output
  // Docker image size by 80%+
  output: "standalone",
  // Optional: bring your own cache handler
  // cacheHandler: path.resolve('./cache-handler.mjs'),
  // cacheMaxMemorySize: 0, // Disable default in-memory caching
  // images: {
  //   // Optional: use a different optimization service
  //   // loader: 'custom',
  //   // loaderFile: './image-loader.ts',
  //   //
  //   // We're defaulting to optimizing images with
  //   // Sharp, which is built-into `next start`
  //   remotePatterns: [
  //     {
  //       protocol: "https",
  //       hostname: "images.unsplash.com",
  //       port: "",
  //       pathname: "/**",
  //       search: "",
  //     },
  //   ],
  // },
  // Nginx will do gzip compression. We disable
  // compression here so we can prevent buffering
  // streaming responses
  compress: false,
  // Optional: override the default (1 year) `stale-while-revalidate`
  // header time for static pages
  // swrDelta: 3600 // seconds

  // Add the ability to dynamically alter the props of local SVGs
  webpack(config) {
    config.module.rules.push({
      test: /\.svg$/,
      use: ['@svgr/webpack'],
    });
    return config;
  },

};

export default nextConfig;
```