ESCOLA TECNOLÓGICA E PROFISSIONAL DA ZONA DO PINHAL

Web-based SMS system

LUIGI MATTEO GIRKE

Final Report on the Professional Aptitude Test for the Professional Course in Computer Equipment Management Technician

Pedrógão Grande | May 2025

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

ESCOLA TECNOLÓGICA E PROFISSIONAL DA ZONA DO PINHAL

PROFESSIONAL COURSE IN COMPUTER EQUIPMENT MANAGEMENT TECHNICIANS

Web-based SMS system

LUIGI MATTEO GIRKE

STUDENT Nº 2812
CLASS E-22/25

Final Report on the Professional Aptitude Test

Supervising Professor Engineer Rui Veríssimo

Director of the Engineering Course Vítor Monteiro

Pedrógão Grande | May 2025

ACKNOWLEDGMENTS

First, I would like to thank Rui Veríssimo, the supervisor of the PAP project, and Vítor Monteiro, the director of the Engineering course. Their guidance, patience, support, and constant availability were fundamental throughout the three years of training and during this project's development.

Special thanks to my friends and family for their unconditional support and encouragement throughout my academic career, especially during this project. Finally, I would like to thank ETPZP for providing all the essential tools to carry out this project.

TABLE OF CONTENTS

INTRODUCTION

This application was created to replace the high cost of text messages for the school, providing a quick, easy, and low-cost communication solution via SMS. Being on the web made it accessible to everyone, regardless of their operating system.

The app allowed users to send messages to multiple recipients, schedule sends for future delivery, cancel scheduled messages, and manage sent messages through a user-friendly interface inspired by email clients. Authentication was done locally using the school's local Active Directory (AD) server.

It was built with Next.js, taking advantage of its App Router, along with a Postgres database, ShadCN components, and other packages. SMS sending was made possible through GatewayAPI's Representational State Transfer (REST) Application Programming Interface (API). During deployment, the app was run in a Docker container, with Nginx configured to route traffic from the router to the proper container's exposed port.

> **Tip**: Use the **Find feature** for easier navigation in this PDF. You can access it by pressing Ctrl + F (on Windows), Command + F (on macOS), or / shortcut in most applications.

TOOLS USED

Applications & external services

- **Visual Studio Code (VSCode)**: Integrated Development Environment (IDE) used for writing all the code of project. On top of it, the following plugins were used:
  - **JavaScript EJS Support**: Provides support for EJS (Embedded JavaScript) templates in Visual Studio Code.
  - **Prettier - Code formatter**: An opinionated code formatter for many languages and integrates with VSCode.
  - **ESLint**: A tool for identifying and reporting on patterns found in ECMAScript/JavaScript code, helping to maintain code quality.
  - **ES7 React/Redux/React-Native snippets**: Provides JavaScript and React snippets for faster development.
  - **Auto Import**: Automatically finds and imports React components, functions, and other modules in your code.
  - **Multi Cursor Case Preserve**: Preserves the case of text when using multi-cursor editing in VSCode.
  - **Pretty TypeScript Errors**: Enhances TypeScript error messages to be more readable and informative.
  - **VSCode Icons**: Adds icons to files and folders in the VSCode explorer for better visual organization.
  - **Docker**: Provides support for developing and managing Docker containers directly within VSCode.
  - **Code Spell Checker**: A basic spell checker for code and comments, helping to catch typos.
  - **Tailwind CSS IntelliSense**: Provides intelligent suggestions and autocompletion for Tailwind CSS classes in your code.
- **Gateway API's REST API**: Used for sending, scheduling, and canceling scheduled SMS messages and getting statistics on sent SMSs.
- **PostgreSQL**: Relational database management system used for storing and managing application data.
  - On macOS, **Postgres.app** was used
  - On Windows, **PostgreSQL** was downloaded from the official website
- **Figma**: Design program used for creating app design prototypes, wireframes, and brainstorming user interfaces.
- **Obsidian**: Markdown-based note-taking application used for writing the reports as well as taking notes throughout the project.
- **Microsoft Word**: Word processing software used for formatting and finalizing the reports.
- **Git**: Version control system used tracking code changes over time. On UNIX based operating systems, this came pre-installed. On Windows however, it needed to be installed separately.

- **GitHub**: Web-based platform for hosting and collaborating on Git repositories, used to synchronize code between different devices.
- **Bun**: Package manager used for installing project dependencies and ShadCN components
- **Docker**: Containerization platform used for creating, deploying, and managing applications in isolated environments.
- **Nginx**: High-performance web server and reverse proxy server used for serving web applications and handling load balancing.
- **dbdiagram.io**: Web-based database diagrams generator used for visualizing database schemas.
- **ChatGPT**: AI language model used on a web-based interface to help find and fix code errors.
- **DeepL**: AI-powered translation tool used on web-based interface for translating reports to Portuguese.
  On macOS, the applications were installed using **homebrew** if the respective cask was available.

Dependencies

Dependencies

- @hookform/resolvers: Resolver integration for react-hook-form.
- @radix-ui/react-*@^1: Accessible, customizable, and unstyled UI components.
- @svgr/webpack: Transform SVGs into React components.
- activedirectory2: Active Directory client library.
- class-variance-authority: Utility for managing CSS class names.
- clsx@^2: Utility for conditionally applying CSS class names.
- cmdk: Accessible command menu component.
- date-fns@^4: Comprehensive date utility library.
- i18next@^24: Internationalization framework for browser and Node.js.
- i18next-resources-to-backend: Backend adapter for i18next.
- iron-session@^8: Secure session management for Next.js applications.
- libphonenumber-js: JavaScript library for parsing, formatting, and validating phone numbers.
- lucide-react: React icons library.
- next-i18n-router: Internationalized routing for Next.js.
- next-themes: Theming support for Next.js.
- next@15: React framework for building server-rendered applications.
- node: JavaScript runtime.
- pg: PostgreSQL client for Node.js.
- react-day-picker: Accessible date picker component.
- react-hook-form@^7: Performant and extensible forms with easy validation.
- react-i18next: Internationalization for React.

- react-loading-skeleton: Skeleton loaders for React.
- react-resizable-panels: Resizable panel layout for React.
- react@19: JavaScript library for building user interfaces.
- recharts@^2: Composable charting library built on React components.
- sonner: Notification system for React.
- tailwind-merge: Utility for merging Tailwind CSS classes.
- tailwindcss-animate: Utility for adding animations to Tailwind CSS classes.
- zod@^3: Typescript-first schema validation with static type inference.

Dev dependencies

- typescript@^5: Superset of JavaScript for optional static typing.
- tailwindcss@^3: Utility-first CSS framework for rapidly building custom designs.
- eslint@^8: Pluggable JavaScript linter.
- postcss@^8: Tool for transforming CSS with JavaScript.
- @types/react@19: TypeScript definitions for React.
- @types/node@^20: TypeScript definitions for Node.js.
- eslint-config-next@15: ESLint configuration for Next.js projects.
- @types/react-dom@19: TypeScript definitions for React DOM.
- @types/validator@^13: TypeScript definitions for the validator.js library.
- i18nexus-cli@^3: CLI tool for managing i18n resources.

A list of all the dependencies and their exact versions can be found in the package.json.

Typescript

TypeScript was a superset of JavaScript with **static typing**, which helped developers catch errors early and improve code quality. It enhanced the development experience with features like autocompletion and type inference, making it ideal for the project.

TypeScript was barely altered, but a couple of rules were modified. The settings could be viewed and modified in the tsconfig.json located in /.

During build time, the following command was useful. It used the TypeScript compiler to scan the entire project for type errors, which needed to be fixed to run a build.

tsc --noEmit

For more information on TypeScript, the official **documentation** was referenced. For its use in the context of Next.js, **this documentation** was referenced.

FILE STRUCTURE

A lot of the existing file structure was chosen because it was either mandatory in Next.js or a common convention. Some people put /lib, /components, /contexts, and /hooks inside the /app directory. However, to keep the app directory as clean as possible, these directories were placed outside.

- /app/ held all the pages and styles of the app, as well as fonts and an internationalization function for loading translations server-side.
- /components/ held all the React components.
- /contexts/ held all the React contexts.
- /hooks/ held all the custom React hooks.
- /lib/ held all utility functions, zod schemas, and most of the server-side code.
- /locales/ held all the i18next translations.
- /node_modules/ held all the node modules (this folder was never touched).
- /public/ was a Next.js file convention for static assets like images and icons.
- /types/ held all the TypeScript types.
- / held all the configuration files.
- /.next was a hidden folder generated by Next.js whenever a build or dev server was spun up.
- /.vscode: was specific to Visual Studio Code and contained some workspace settings for a spellchecker plugin.

Next.js file based routing

In the "/app directory" section, many Next.js file conventions were mentioned with links to the Next.js documentation, but the basics and reasons for the chosen file structure were explained here.

- Next.js folder conventions:
  - Directories wrapped in **square brackets** like /app/[locale] represented dynamic route segments, allowing the pages and components inside to retrieve their values (in this case the current locale). All pages were located within /app/[locale], as the entire app required access to the current language for internationalization to function.
  - Directories wrapped in **round brackets** like /app/(root) served as route groups and were invisible to the end user. They functioned like normal directories to group different pages together. In this project, they were utilized to group pages with the same layout, ensuring the layout.tsx in that directory applied to all other pages **without creating an actual route segment** like etpzp-sms.com/(root).
  - Directories with **standard names** containing a page.tsx represented the names of the route segments. For example, /app/contacts/page.tsx was accessible at etpzp-sms.com/contacts.

- Directories starting with an **underscore** indicated disabled routes (not accessible to the end user). They functioned like code comments. In this project, /_seed/page.tsx was used solely during development.

- Next.js file conventions:
  - page.tsx files represented pages accessible by the name of the directory above.
  - layout.tsx files served as layouts applied to all pages in the same directory and nested directories.
  - loading.tsx utilized React Suspense behind the scenes to display a fallback UI while the page was loading.
  - error.tsx files implemented error boundaries that caught unexpected errors in the same directory, handling unexpected errors by providing a fallback UI.
  - not-found.tsx was displayed whenever a 404 error occurred.
    Most of these features were chosen as they dramatically improved user experience.

/app directory

- /app/favicon.ico (**Next.js file convention**): Image file to set the app icon in the browser tab
- /app/globals.css: CSS file for globally used css variables

- /app/i18n.js (i18next file convention): i18next config file for loading translations server-side. Holds the code from this **tutorial**
- /app/layout.tsx (**Next.js file convention**): Root layout of the app
- /app/not-found.tsx (**Next.js file convention**): Global 404 not found page
- /app/scattered-profiles.module.css: CSS modules used in message-display.tsx component
- /app/[locale] (**Next.js file convention**): Dynamic route segment for the current language
    - /app/[locale]/(root) (**Next.js file convention**): Route group for pages that use the resizable nav-panel (see /app/[locale]/(root)/layout.tsx).
        - /app/[locale]/(root)/(message-layout) (**Next.js file convention**): Route group for similar pages that used the same translations and needed access to the contacts (see /app/[locale]/(root)/(message-layout)/layout.tsx), sharing the same error page. As of now, all these pages used the **three-column-layout** of resizable panels.
        - /app/[locale]/(root)/(other) (**Next.js file convention**): Route group for special pages that **did not** share the same characteristics. Since this directory did not have a layout, the files adhered to the layout above (see /app/[locale]/(root)/layout.tsx).
- /app/fonts: Directory for local fonts, imported in the root layout (/app/layout.tsx). For information on local fonts, the Next.js **documentation** was referenced.
  The normal directories not mentioned here are pages, which were explained in the "PAGES" chapter.

/lib directory

The lib folder stored reusable utility functions, and most server-side code. This includes the database seed file and fetching functions, authentication configuration, auth server actions, and server actions for mutating data and making API calls, shared across different components and pages. Although the authentication code contained server actions, it was placed in its own directory (/lib/auth) to separate the topic and keep the actions directory (/lib/actions) less cluttered.

- /lib/:
    - form.schemas for zod schemas

    - theme.colors for Tailwind and Next.js theme configuration
    - utils.ts for utility functions usable anywhere
- /lib/actions had server actions and a testing directory for development testing purposes to simulate API calls.
- /lib/auth had authentication code and a directory called activedirectory, which included functions wrapping around the activedirectory2 package.
- /lib/db had a file to seed the database (/lib/db/seed.sql) with the initial database schema, database fetching functions, and a Dockerfile for seeding the Postgres database run inside the Docker container during production.

/components directory

This directory held all the React components organized into subdirectories and the ShadCN components.

- /components/admin-dashboard: Held components used on the admin dashboard and were put into a separate directory to separate the topic
- /components/modals: Held the modal/popup components and were put into a separate directory to separate the topic
- /components/shared: Held shared components that were very commonly used
- /components/ui: Held ShadCN components: The files inside remained mostly unchanged, except for small color adjustments, which involved substituting hardcoded colors with CSS variables.

/ directory (config files)

- components.json was used for ShadCN configuration to add components in the same style whenever a new one was added from the CLI.
- tsconfig.json was the TypeScript configuration file.
- tailwind.config.ts was for Tailwind CSS.
- .dockerignore was utilized to specify files to ignore during Docker deployments.
- .env was the environment variable file.
- .env.docker was the Docker-specific environment variable file.
- .env.example served as an example of environment variables.
- eslintrc.json was the ESLint configuration file.
- .gitignore was used to specify files to ignore in Git.
- .prettierignore was utilized to specify files to ignore in Prettier.
- .bun.lock was the lock file for the Bun package manager.
- docker-compose.yaml was used for Docker Compose configuration.
- Dockerfile was the file for building Docker images.
- eslint.config.mjs was the configuration file for ESLint in module format.
- global.config.ts was used for global configuration settings; it was added for JavaScript constants used in multiple components.
- 118n.config.ts was the configuration file for internationalization.

- middleware.ts was used for middleware functions.
- next.config.ts was the configuration file for Next.js with three modifications:
    - output: standalone was used to optimize the size of the build, filtering out unnecessary files.
    - compress: false was set for compression settings.
    - A webpack configuration was added to load local SVGs for dynamic styling.
- nginx.conf was the configuration file for Nginx.
- package.json was the file for managing project dependencies.
- README.md was the documentation file.
- postcss.config.mjs was included for PostCSS configuration and was not modified.
- next-env.d.ts was the TypeScript definition file for Next.js and was not modified.

FRONT-END

Styling

In addition to the pre-styled ShadCN components, Tailwind CSS—a utility-first CSS framework that allowed for rapid UI development—was utilized alongside standard CSS. Inline styles were also used in some cases, particularly because dynamically generated Tailwind classes, such as bg-${chosenColor}, would not work due to Tailwind purging unused classes in production and not recognizing dynamically created class names.

- **Standard CSS**
  - globals.css contained globally used CSS classes and CSS variables.
  - scattered-profiles.module.css were CSS modules used on the message display panel on the message visualization pages. More information was provided in the "PAGES" chapter.
- **TailwindCSS** was used throughout the project. It served as the primary source of truth, and it was recommended to use Tailwind whenever possible.
- **Inline-CSS** was used minimally in cases where no other choice existed, or where the styling was very specific and would be overridden if done using standard CSS.

ShadCN with dynamic themes

ShadCN was a UI component library designed for building modern web applications with customizable themes and a focus on user experience.

1. **Initialize Project**:
   - A new Next.js project with Shad CN UI was created.
2. **Install Dependencies**:
   - next-themes was installed for light/dark mode toggle.
   - Lucide React was installed for icons.
   - Tailwind and Prettier VSCode plugins were added for formatting.
3. **Setup Global CSS** (/app/globals.css):
   - CSS variables from Shad CN UI's themes page were copied into the Global CSS file.
4. **Define Theme Colors** (theme.colors.ts):
   - An interface for theme colors was created, and available colors were set up based on Shad CN UI's themes.
5. **Convert CSS Variables** (/lib/theme.colors.ts):
   - Shad CN UI's theme color CSS variables were converted into a JavaScript object.
6. **Create Theme Function** (/lib/theme.colors.ts):
   - A function was developed to override global CSS variables for real-time theme color changes.
7. **Theme Data Context and Provider** (theme-data-provider.tsx):

- A state was implemented to prevent flickering between default and saved colors on initial load.
- A helper function was exported to access theme context throughout the component tree.
- A Theme Data Provider component was created to manage theme state and local storage.

8. **Wrap with Next Theme Provider** (theme-data-provider.tsx):
   - The Theme Data Provider was encapsulated inside a Next Theme Provider in the top-level layout.

With the basic configuration in place, a button was created to toggle between light and dark modes, linking it to the setTheme function. A dropdown menu for selecting theme colors was developed, using the defined colors from the themes.

More info on these front-end-setting changers can be found on the settings page in the "PAGES" chapter.

React resizable panels

This library was for React components for resizable panel groups/layouts. It was used to achieve a layout with 2 or 3 horizontal panels, and it was chosen for its ease of use and nice integration with ShadCN.

Calculations for persisting the sizes of different columns were difficult but necessary. It was done by storing the percentage contribution of each column as an array in the cookies and retrieving it in the root layout to pass it to the components. The most-left panel of the 3 was the hardest to configure as it also had the functionality to collapse when a certain width was reached.

Relevant custom components included:

- resizable-panel-wrapper, which wrapped all the ResizablePanel ShadCN components, was only used once in the app-layout.tsx component.
- children-panel handled its sizing logic itself and was used in almost all pages that utilized the multiple panel-based layout.

There was also the option to do a 2-column layout without resizable panels, but it wanted to take on the challenge of this unique layout, which was rarely seen on the web.

PAGES

Login (/login)

The login page was easy and fast to implement. It was placed outside the main layout groups but inside the /app/[locale]/, which was necessary for it to have the current locale.

The page contained a simple form handled mainly by one component (/components/login-form.tsx). It used the ShadCN card component and included 2 fields: one for the email and the other for the password. The password input contained a button to toggle its type between "text" and "password," providing the classic behavior of show password buttons in web forms.

Form submission on client
Since redirecting on success from the server side caused issues, it was decided to use client-side router redirection, leading to the implementation of "Scenario 2" found in the "Form setups" section of the "RULES FOR CONSISTENCY" chapter. It first displayed errors through toasts. Then, if the server response was successful, it synchronized the local storage with the user's database settings and redirected programmatically to /. It also maintained a pending state to disable elements during submission.

Form submission on server
The form called the login server action (/lib/auth/index.ts), the logic of which was explained in the "Authentication flow" section of the "AUTHENTICATION & AUTHORIZATION" chapter.

New message (/new-message)

The new message page was by far the most complicated page to build. It could be navigated to by clicking on the big primary colored button in the left sidebar. The form itself on the page was handled in /components/new-message-form.tsx and /components/recipients-input.tsx. However, the main logic and states were stored in a dedicated context in /contexts/use-new-message.tsx for separations.

The new-message-form consisted of four main visible fields:

- Sender field: Static disabled field which was hardcoded to be "ETPZP".
- Recipients field: complex custom input component.
- Subject field: this field also changed the title when it changed.
- Message field: Text area used to hold the content of the message.
  The page also contained some other buttons:
- **Save draft** button was used for displaying the current state of the draft (saved or not with related errors in the tooltip) and it also allowed the user to save the draft manually.
- **Fullscreen** available on desktop allowed the user to hide other elements around the page.
- **Close** was a link to /sent.

- **Discard** (at the bottom left) was a link to /sent which also deleted the draft when clicked.
- **Send** (at the bottom right) displayed if the message was scheduled or to be sent now and submitted the form. It also had a little menu on the side that allowed the user to schedule the send of the message.

Form submission on client
It adhered to "Scenario 2" found in the "Form setups" section in the "RULES FOR CONSISTENCY" chapter. The entire client-side validation and the error displaying from the server were handled in the handleSubmit function with toasts for error messages. The server returned various flags, translation strings, and data that decided on how the errors were displayed and translated on the front-end. In the case of zod errors, the errors were looped over and displayed as separate toast messages. Each input also had certain animations like red blinking or just a red underline or red placeholder built into them to show users what caused the error. It also keeps a pending state to disable elements during form submission.

Form submission on server
There existed one server action for sending messages called sendMessage located in lib/actions/message.create.ts. The function first did a couple of security checks which exited the function early when they failed:

- User authentication was checked (line 22 - 30)
- Field validation was handled using zod (line 32 - 48)
- More in-depth custom validation was done for recipients (line 50 - 60 & line 259 - 276)
- Moving on, the data was prepared for the API call, and the API was called using fetch (line 62 - 100).
- After that, the database logic began.
    - One main check was done to see if the message had already been saved to the database as a draft, in which case the draft was updated (line 103 - 157). On the contrary, a new message was inserted (158 - 200). As much information about the message was saved, including API errors if they existed.
    - After inserting the message, the recipients were handled separately (202 - 225). Any old existing recipients were first deleted, and then new ones were inserted.
- Responses were sent back to the client with case-specific translation strings, which were translated on the client-side (line 228 - 257).

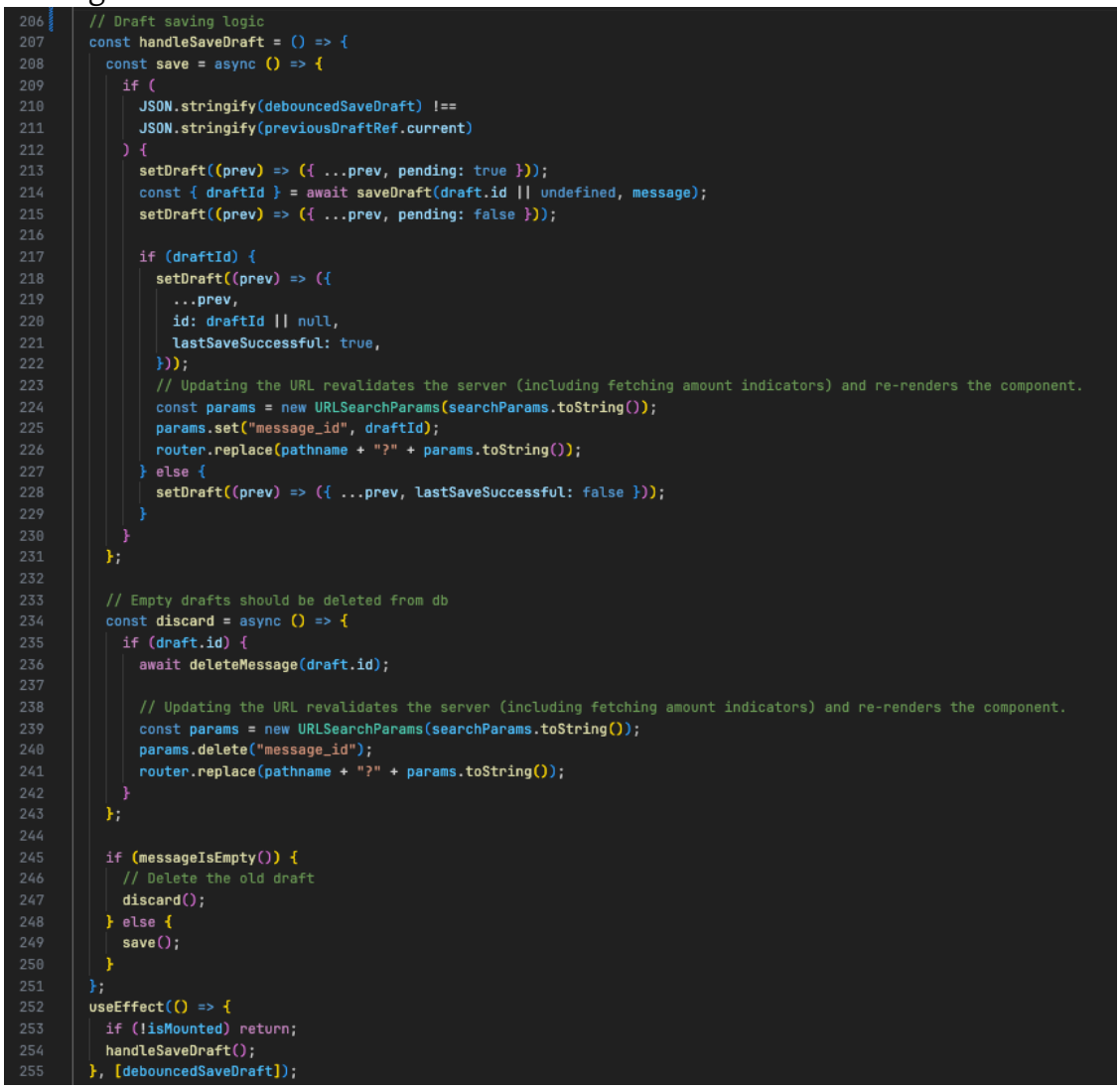
Custom recipients input
The component in /components/recipients-input.tsx was more than just an input, it was a custom component. As a first functionality, it allowed the user to type in any string and press enter or tab to add it as a new recipient. The system then early on did some client-side validation to detect what could be wrong with the number.

As a second big feature, a window appeared when the user started typing, showing recipients that they could insert. This absolutely positioned custom element behaved the following way:

- It was not even displayed if there were no existing recipients or contacts.
- If the input was empty but focused, the window contained "recommended recipients" which were calculated based on usage in the last week, as well as if they were saved as contacts or not. If there were not enough recipients that were used in messages, the rest was filled with unused contacts if they existed.
- If the input was not empty and focused, the window contained the "search results" which were the filtered out recipients and contacts based that contained the value the user put in the input.
- The user could add these shown recipients/contacts by from their keyboard by navigating with up and down arrows and insert them using enter or tab. Or they could just click on the recipient of their choice.
- Upon adding one, it was removed from the concurrent search results or recommendations, as the user should not be able to add the same recipient 2 times. However, if they tried to type a phone number that already existed in the recipients, this case was also handled and an error message was shown as a toast.

Automatic drafting system
It was chosen that after a cooldown, the draft was automatically saved assuming at least one field held a value. If the fields were all empty, the existing draft was deleted from the database again.
Draft saving logic was handled in the handleSaveDraft function in /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]);
```

- If the component was mounted, it was called from a useEffect which got triggered by a constant which received changes after the debounce of no changes triggering the useEffect only after that useDebounce. A custom hook was created for this behaviour in /hooks/use-debounce.tsx (line 252 - 255).
- The function then checked if the message was empty and called the correct function accordingly (line 245 - 250).
- The save function checked if the current draft had changed compared to the previous draft, saved it if it had, updated the draft's ID and status based on the save result, and modified the URL to reflect the new draft ID while revalidating the server (208 - 231).
- The discard function deleted the current draft from the database if it had an ID and updated the URL to remove the draft ID, which revalidated the server and re-rendered the component (234 - 243).

Modals

This page used the following modals:

- schedule-modals.tsx contained one modal for selecting a schedule date and another for warning the user that the date was invalid.
- recipient-info.tsx was shown when a user clicked a recipient chip, displaying additional information about the selected recipient (or contact).

Challenges
First, there was an issue with the draft saving. Whenever the URL got updated (even just with URL search params), it caused all the components of that page to re-render since the top-level server component retrieved the message_id parameter from the URL to fetch the draft data. This re-rendering led to all the fields losing their values, including previously open popups or popover menus which would also get hidden. To fix this, a context was created which persisted all the values during the re-renders.

Additionally, building the suggested recipients window with all its functionalities and ensuring it was bug-free took a significant amount of time. It was difficult to find a setup that would always have the most up-to-date values, and as the new-message-context grew larger, it became increasingly challenging to work with.

Settings (/settings)

Determining the code architecture for the settings was challenging due to a lack of clear guidance. Preferring automatic updates whenever a setting was modified, save buttons were avoided. The settings page featured a custom setup where some settings were managed by libraries and others with a custom implementation.

While most settings were saved in local storage, theme data and the current language were stored in cookies due to how the libraries handled them. However, directly updating local storage did not refresh the React components. To resolve this, a settings context was created (/contexts/use-settings.tsx), which managed the settings state and included various helper functions.

A server action called updateSetting that updated individual settings one at a time was manufactured (/lib/actions/user.actions.ts), leading to the creation of multiple forms. This approach, while resulting in more forms, allowed for easier management through centralized logic (/components/settings-item.tsx).

Re-usable components
Due to the repetitive nature of settings, reusable components were developed: a SectionHeader for setting categories and SettingsItem for individual settings.

The SectionHeader component (/components/headers.tsx) was simpler as it required passing in the title and caption to display, along with the anchor-tag name.

The SettingsItem component (/components/settings-item.tsx) was more complex as it contained all the error handling and pending logic. It was made customizable by adding a renderInput prop, which allowed passing of completely custom HTML for input while still providing access to the database submit handlers and other important data. Each of these forms adhered to "Scenario 2" found in the "Form setups" section of the "RULES FOR CONSISTENCY" chapter.

It was worth mentioning that the language changer used the updateLanguageCookie function, which could not be implemented without using the Next.js router for replacing and refreshing internally. Due to this internal refresh, it caused a reset, necessitating its own component due to the increased complexity of changing the language.

Considerations
Initially, using a single form for all settings was considered but was rejected due to performance and readability concerns. A single form would complicate handling, requiring the entire settings set to be sent to the server for each modification, which would hinder validation and error handling.

Admin dashboard (/dashboard)

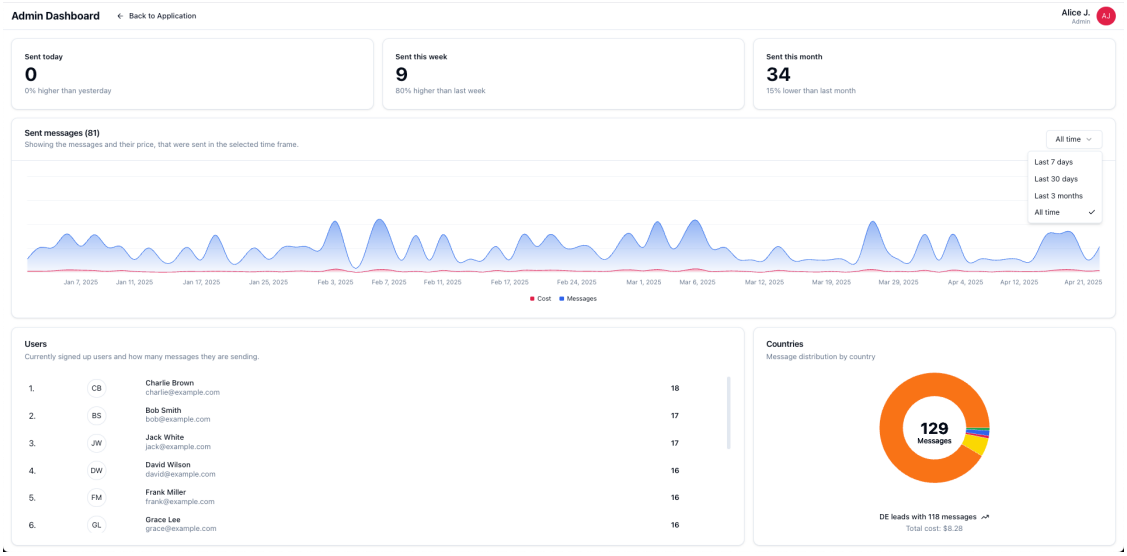
The admin dashboard was built last and included statistical information. It was placed outside the main layout groups but inside the /app/[locale]/, which was necessary for it to have the current locale.

The page was only accessible to admins, as explained in the "Authentication flow" section of the "AUTHENTICATION & AUTHORIZATION" chapter. The ReCharts library was used for the responsive area and pie charts. For coloring the area chart, it retrieved the primary theme color and the profile color. For coloring the pie chart, it used the primary color of each theme. The order was randomized, and the colors were saved to a state so that they would change during component re-renders caused by users modifying the date.

The page included:

- 3 cards at the top displaying the number of messages sent compared to the past.
- An area chart showing the messages and cost since a specified time.
- A toggle that changed the start date for the other charts.
- A users table ranking the signed-up users based on sent messages since the selected start date. The end_date search parameter could also be injected into the URL, and the application would apply the filter for an end date.

- A pie chart displaying information about the countries of the recipient phone numbers, retrieved from the **label statistics API**.



Date filtering

The start date toggle, found in /components/admin-dashboard/message-area-chart.tsx, was a Select dropdown that replaced the current URL with a new URL containing

updated search parameters whenever its value changed.

```
<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>
```

Data fetching

Since larger datasets were expected after some time of app deployment, "Scenario 2" of the "Conservative data fetching" section in "RULES FOR CONSISTENCY" was utilized. This meant that the data was fetched in the top-level server component, where the URL parameter was retrieved and passed to the backend fetching functions. Whenever the

URL parameters changed, it re-rendered, causing the data to be updated.

```tsx
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_FORMAT),
    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}
    />
  );
}
```

The data from the top-level server component was then passed to the AdminDashboard client component, where additional data formatting and calculations were performed.

```tsx
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">
```

Challenges

One challenge was getting the pie chart to work. Sometimes it just wouldn't display. This was later found to be due to a height that was too small, so a fixed height was added to its parent container.

Additionally, a custom tooltip for the pie chart had to be implemented, which was inspired by the tooltip in the area chart to maintain design consistency.

Other pages

These pages were very similar:

- /sent for sent messages
- /scheduled for scheduled messages with a send time in the future. Once the scheduled time was reached, it showed up in /sent
- /failed for failed messages where an error occurred on the API's side or got canceled by the user
- /drafts for drafted messages that had been saved but not sent, allowing users to edit or finalize them before sending
- /trash for trashed messages, where one could recover them or delete them permanently
- /contacts for contacts - contacts held additional information like the phone number, name and a description. This page was slightly different from the others, but it was similar enough to be put in the same layout.

Shared layout
The pages from this chapter lived in the same route group due to their similarity (/app/[locale]/(root)/(message-layout)/) which shared the same layout and error handling files. The layout wrapped the children pages with a provider for the translation context while loading in the necessary namespaces. Since the pages needed access to the contacts, the children pages were wrapped with a provider for the contacts context,

passing in some initial contacts (line 34-36).

```
1  import initTranslations from "@/app/i18n";
2  import TranslationsProvider from "@/contexts/translations-provider";
3  import { ContactsProvider } from "@/contexts/use-contacts";
4  import { fetchContacts } from "@/lib/db/contact";
5
6  type LayoutProps = Readonly<{
7    children: React.ReactNode;
8    params: Promise<{ locale: string }>;
9  }>;
10
11 export default async function TranslationLayout({
12   children,
13   params,
14 }: LayoutProps) {
15   // Internationalization (i18n) stuff
16   const i18nNamespaces = [
17     "messages-page",
18     "contacts-page",
19     "modals",
20     "common",
21     "errors",
22   ];
23   const { locale } = await params;
24   const { resources } = await initTranslations(locale, i18nNamespaces);
25
26   return (
27     /* This is a client layout component containing the translation provider for the nav panel */
28     <TranslationsProvider
29       /* Only wrap what's necessary with the TranslationsProvider */
30       resources={resources}
31       locale={locale}
32       namespaces={i18nNamespaces}
33     >
34       <ContactsProvider initialContacts={(await fetchContacts()) || []}>
35         {children}
36       </ContactsProvider>
37     </TranslationsProvider>
38   );
39 }
40
```

Page architecture

- messages-page.tsx as the wrapper for the other components
- messages-list.tsx which showed the contact search results (middle column)
- message-display.tsx which displayed the message itself and was also wrapped in the children panel component. More details about this were provided in the "React resizable panels" section of the "FRONT-END" chapter.

Search/filtering

The search.tsx component was the search bar UI used for searching through messages and contacts. It called the passed-in function (onSearch) upon input changes and persisted the user's query in the URL for bookmarkability and accidental refreshing. This URL update did not refresh the server components, as Next.js hooks were not used.

```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>
  );
}

```

The searchMessages and searchContacts functions filtered their respective arrays based on a user-provided search term, enabling client-side searching. Both functions converted the search term to lowercase for case-insensitive comparison. searchMessages checked for matches in the message's subject, body, or status, while searchContacts looked for the term in the contact's name or phone number. If no search term was provided in

searchContacts, it returned the original list of contacts.

```
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 status 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;
}
```

Message display

These pages from this chapter except contacts had these buttons on their message display:

- The **Resend** button was for taking all the fields of a message and inserting them into the new-message form again. It worked by first creating a new draft in the database and then passing the ID to the message_id parameter on the /new-message page.
- The **Move to trash** button was for moving messages to trash. On the trash page itself, the message was deleted from the database.
- The **Close** button was for deselecting the currently selected item shown in the column on the far right. Page-specific buttons:
    - The **scheduled page** also had a **cancel scheduled** message button, which canceled the SMS and obtained a refund via the API, moving the message to failed. This was useful for testing the app without costs.
    - The **trash page** also had a **recover message** button, which moved the message back to its original location, recovering it from the trash.
      It was decided that each message displayed its recipients in chips format, which, upon clicking, brought up the recipient-info.tsx modal to show more information about the recipient (or contact). By default, the

recipients were collapsed, and they could be expanded by clicking the little arrow on the right.
It was worth mentioning the effort to display the contact profiles nicely. The first five recipients/contacts were shown in a little overview element containing their profile circles. Their styling was handled in the scattered-profiles.module.css file using CSS modules. The sizes and positions were hardcoded, but the colors were randomized by storing a shuffled array in a state, and each time a new message was selected, the procedure was repeated.

Contacts page
While also being very similar, it had its own components because the data was completely different, and the code needed to be kept clean:

- contacts-page.tsx instead of messages-page.tsx.
- contacts-list.tsx instead of messages-list.tsx.
- contact-display.tsx instead of message-display.tsx.
  It was decided to have this page include one button each for creating, editing, and deleting contacts in the database.

Modals
It was decided that all of the mentioned pages wrap their display component in a modals-provider, a provider of a context for managing which modals are currently open. The contact pages use these modals:

- edit-contact.tsx contained a form for editing a contact with a useActionState setup
- create-contact.tsx contained a form for creating a contact with a useActionState setup
  The other pages use this modal:
- recipient-info.tsx displayed more information about a recipient (or contact)

RULES FOR CONSISTENCY

Use of server actions

As of Next.js 15 with the app router, it was recommended to use **server actions** for data fetching or making API requests on the backend. Server actions simplified development by allowing developers to define server-side functions invoked directly from client components, automatically making POST requests on the backend when the action gets called.

Previously, developers had to create separate API routes for data fetching, which was cumbersome and time-consuming. In most cases, it's better to use server actions over API routes. The following guide was consulted whenever there was uncertainty about which one to use.

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	✅	

Since Next.js recommended treating server actions like public API routes, it was highly recommended to **verify user authentication in every server action** for security purposes.

Use of React Contexts

Introduction: React Contexts enabled developers to manage global state and share data across components without prop drilling. This proved useful for applications where multiple components required access to the same data, such as user authentication, themes, or settings, leading to a more efficient state management approach.

How It Worked: React Context created a context object to hold shared data. A Provider component wrapped parts of the application, making the context value accessible to nested components. Components that needed the context used the useContext hook to access the data, ensuring that only those components re-rendered when the context value changed.

When It Was Applied: The rule to use React Contexts was established during the initial setup phase to create a clear state management strategy. As a guideline, contexts were created when data needed to be accessed by more than four components or when prop drilling extended beyond three layers in the component tree.

Form setups

There were 2 different scenarios for forms. Based on complexity, it was necessary to choose one of the following options to maintain a consistent codebase:

- **Scenario 1**: In simple situations, it was recommended to let the form submit directly to the server without altering the submit process.
    - **Implementation**: This setup involved using the useActionState React hook, with the action passed to the action prop of the form tag.
    - **Server response**: If the action response was needed, it was necessary to create a useEffect with the server state in the dependencies array.
    - **Example**: An example of this scenario can be found in /components/modals/create-contact.tsx.
- **Scenario 2**: If code needed to run upon form submission, interrupting the natural submit behavior, this scenario would have to be used.
    - **Implementation**: First, it was necessary to create a function (commonly named handleSubmit) to pass to the onSubmit prop of the form tag. In the handleSubmit function, special logic could be written, and the server action could be called.
    - **Server response**: If the HTML required the action response, it was necessary to create a useState that would be set in the handleSubmit function.
    - **Example**: An example of this scenario can be found in /components/login-form.tsx.
      These scenarios were developed through extensive experimenting, research, and testing, and they proved to be the most readable, effective, and efficient.

Fetching from server components

Next.js encouraged fetching data from top-level server components, a practice that was widely implemented in the application. By leveraging server components for data fetching, it took advantage of server-side rendering, which improved performance and ensured that all nested components had access to the necessary data without additional client-side fetching.

When updates to the data were required, the revalidatePath() function from the Next.js API was employed. By calling this function on the specific path, it triggered the server component to re-render, which in turn re-fetched the latest data. By always using the Next.js APIs, it was possible to prevent page refreshing (only internal refreshing), which made it feel more like an application.

Conservative data fetching

There were many ways of fetching data in Next.js. After extensive research and testing, 2 methods came into consideration:

- **Method 1**: This method involved fetching data from the server component once and then using client-side JavaScript to perform filtering. It minimized server load by executing the database query only during the initial page load, resulting in instant filtering results.
    - **Advantage**: It decreased server load due to a singular data fetch on page load and provided very fast filtering for medium-sized datasets or smaller.
    - **Disadvantage**: It could become laggy if the dataset was too large or the filtering was too complex for the client-side JavaScript, especially if the user had an old computer.
- **Method 2**: This method involved passing updated search parameters from the client component into the dynamic database queries. When the search parameters changed, the server component automatically re-rendered and re-fetched the database.
    - **Advantage**: It did not rely on the user's computer as the filtering was done on the server in the SQL query.
    - **Disadvantage**: It increased server load due to frequent data re-fetches (every time a filter was updated) as well as database query delay.

Due to the medium-sized dataset (less than 1000 messages per user), **Method 1 was the most suitable approach** for most use cases.

Page wrapper files

Each page that was created had to contain at least a loading.tsx and an error.tsx, with a layout being optional.

- error.tsx was a file in Next.js that served as a catchall for unexpected errors. It created a React error boundary that prevented the app from crashing when unexpected exceptions occurred.
- loading.tsx was a file shown while the page was loading, containing all the skeleton UI for that page.
- layout.tsx was a layout around the page. To keep page.tsx cleaner, it was advisable to place everything unnecessary in layout.tsx, including the translations provider and other providers if possible.
- page.tsx was the page itself, which was to be kept as clean as possible, with a server component for fetching data later on if needed.
  Initially, it had everything set up with a React.Suspense suspense boundary used for loading. This approach was beneficial for implementing partial pre-rendering, allowing some parts of the page to load faster than others with a separate loading indicator. However, it was quickly discovered that using only one might as well follow the Next.js file convention, which kept the files more organized.

Metadata

Basic metadata, such as tab titles, icons, and descriptions, enhanced shareability and bookmarkability. As a result, it helped users quickly identify the website.

Metadata was generated server-side using the generateMetadata function from the Next.js API, which was essential for metadata translation.

```
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  }
```

The app logo was shown by naming it favicon.ico and placing it in /app, which Next.js automatically recognized and used for metadata.

Statically exporting the metadata was also possible, but in that case, it would have been impossible to translate into different languages.

DATABASE

PostgreSQL was chosen for its reliability and feature-rich capabilities. As an open-source relational database, it offered robust data integrity and strong security features. Connection was established using pg, with the production environment running in a Docker container on port 5432. More information can be found in the "DEPLOYMENT" chapter.

At first, **Prisma**, a database toolkit and object-relational mapping (ORM) layer, was considered used along with the PostgreSQL database. It streamlined database access by providing a **type-safe API**. However, it was rejected to keep the project lightweight and minimize dependencies.

Connecting to the database

The node-postgres library was used, due to its an efficient way of executing SQL queries and retrieving results.

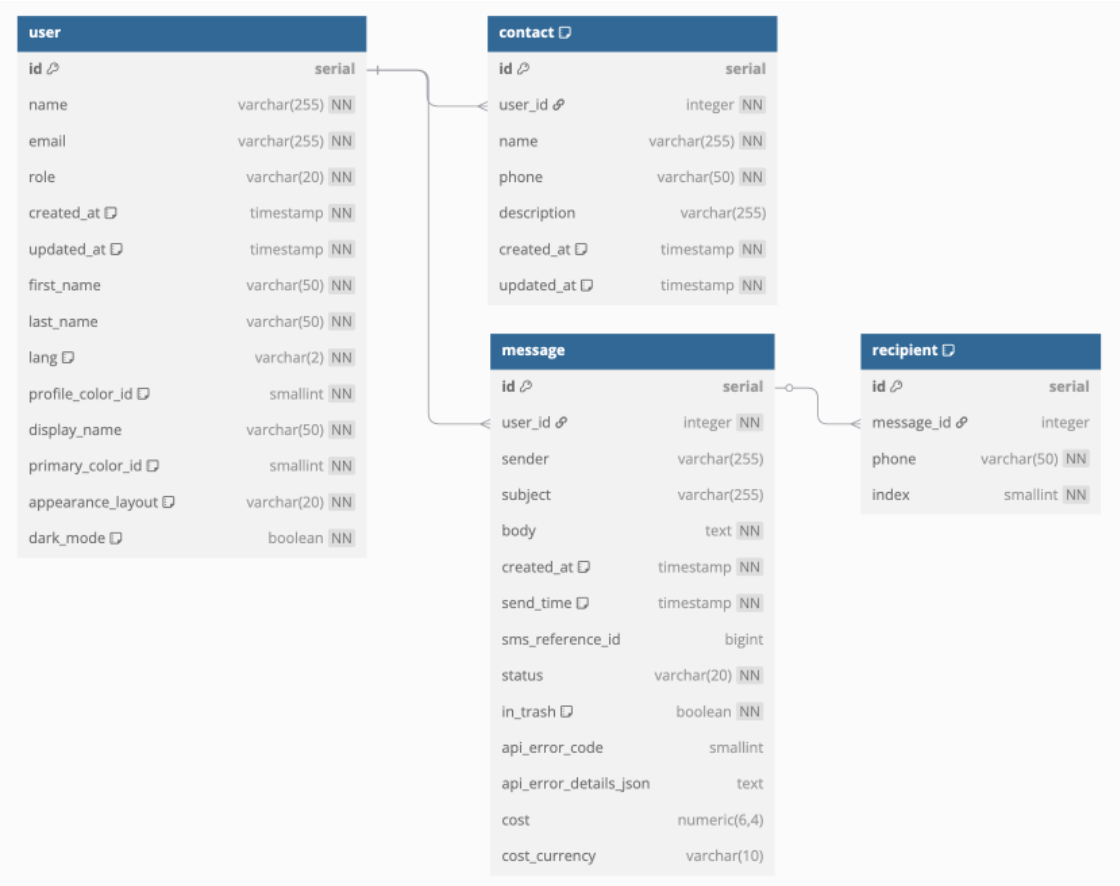
When connecting to PostgreSQL using pg, there were 2 options: **pool** or **client**. A **pool** was a group of reusable connections ideal for concurrent queries, which was utilized due to multiple queries at once. A **client**, on the other hand, represented a single connection per interaction.

To simplify querying, a helper function was created (/lib/db/index.ts) that took in the SQL query and values to insert. It started by creating a new pool, connecting to that pool to create a new client, and then querying it while catching unexpected errors, and lastly releasing the client back into the pool.

```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;
```

Now it was just as easy as importing the db helper function and passing in the SQL query and values. For information on pg, the **documentation** was referenced.

Database schema



The database schema, defined in the seed file (/lib/db/seed.sql), created four tables:

- user held all the users' data, including account data and settings.
- contact held all the contacts, including their important data like name, phone number, description, creation date, and last updated date.
- message held all the messages, with each message referencing user table's primary key. Furthermore, each message contained important data such as the sender, subject, body (content of the SMS), sent date, status (sent, scheduled, failed, or drafted), and other data returned by the API when the message was sent.
- recipient held all the message recipients, with each recipient referencing message table primary key. Additionally, each recipient consisted of a unique phone number, and an index used to display the recipient in the same user-defined order on the new-message page during component re-renders.
  Moreover, all of the mentioned tables used a serial primary key field called id utilized to distinguish the different items.

Considerations
One consideration at the beginning of the project was to have separate tables for

message types (drafts, trash, etc.), but it was realized to be unnecessarily complex. Ultimately, all messages were stored in the same table, with each message having fields like status and in_trash, which determined in which category it would be shown on the front-end.

For a long time during the app's development, contacts were linked to recipients using the primary key. However, three-quarters into the project, a migration was necessary due to querying and insert problems, as well as overall flaws in the architecture. The new solution involved checking for contacts on the front-end, looping over recipients to verify if their phone numbers matched a contact's.

Even though it worked this way, another consideration for improvement was to handle scheduled messages differently. As of that point, scheduled messages remained with the status "SCHEDULED" even when their delivery date was reached, which was not logically accurate. To resolve this logic issue, it could be suggested to rename the field to something else or update the status to "SENT" when the delivery date was reached.

AUTHENTICATION & AUTHORIZATION

The application utilized a combination of Active Directory (AD) and Iron Session for authentication and authorization purposes.

Active Directory

Since the school already used an AD server for managing students' computer accounts, it was integrated with the application. This combination made managing user access much easier later, as the user accounts were all managed in one place.

AD worked similarly to a database, storing information about all users and their data. In this case, the application had 2 specific groups set up in AD: "Utilizadores-SMS" and "Administradores-SMS." These groups were used to determine the permissions each user had within the application.

If a user was part of the "Utilizadores-SMS" group, it granted them basic access to the application, allowing them to send SMS messages and manage their own messages. Conversely, if a user belonged to the "Administradores-SMS" group, it provided all the same permissions as the first group, along with access to an admin dashboard that offered detailed statistics on all users and sent messages.

Active Directory implementation

To connect to the AD server from inside the application, 2 different packages came into consideration: activedirectory and activedirectory2. The activedirectory2 package was ultimately chosen because it was the most up to date, and the other one did not work.

This package used **Lightweight Directory Access Protocol (LDAP)** queries and provided a JavaScript (JS) wrapper where it allowed the passing of the email and password of a valid, already registered AD account along with some other arguments. An AD instance object was first created as shown below:

```
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!,
});
```

After that, methods of this instance could be used as shown below:

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

The utilized functions can be found in /lib/auth/activedirectory. For information on activedirectory2, the **documentation** was referenced.

Session management

Session management was the process of handling user sessions in web applications, where session data was typically stored as a cookie. A session management library provided tools to create, maintain, and terminate user sessions (auth cookies), simplifying authentication and state management.

Since the AD setup already handled most of the authentication, a full-blown authentication library was not necessary. In fact, a lightweight session management library did the job. The iron-session package was chosen due to its session-based nature, and its lightweight, secure, and easy-to-implement features.

Another session management library called jose was also considered. However, it was quickly rejected, as token-based authentication was not necessary for the project. Additionally, iron-session was more lightweight and easier to use. More information on authentication types can be found in the "Session-based vs. Token-based authentication" section.

Also, different browser storage options for persisting user authentication data were explored. Initially, session storage was mistakenly considered because its name suggested suitability; however, using this storage option for user authentication data was inappropriate since session storage expired when the tab was closed, unlike cookies, which were commonly used to persist session data.

Session management implementation

When a user got authenticated successfully, his information was stored in the database, and subsequently in an encrypted session id cookie generated by iron-session.

Configuration
iron-session sessions were customized by modifying a configuration object:

- Name and password could be anything, but for extra security the password was generated using openssl
- The session expired after 24 hours instead of the default 14 days.

```
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  };
```

Authentication config file located in /lib/auth/config.ts

Helper functions

A simple helper function called getSession was created to wrap the Iron Session api, which on the server-side retrieved the active session from the cookie or created a new one if none existed. The previously customized sessionOptions config object was utilized to as one of the arguments passed to the getIronSession function.

```
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  }
```

getSession helper function located in /lib/auth/sessions.ts

Another helper function was manufactured with the purpose of creating a new session. This one used the getSession function to first retrieve the current session, and subsequently it attached useful information about the user and if he was authenticated or not and an admin or not to the session. Lastly the modifications to the session were applied (line 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  }
```

createSession helper function located in /lib/auth/sessions.ts

For information on the iron-session package, the **documentation** was referenced.

Authentication flow

With the initial Active Directory and iron-session setup out of the way, the final implementation could be written.

In summary, the authentication flow involved the following steps:

1. **User Login**: Users entered their credentials (username and password) into the on the client and submitted the form to the server.
2. **AD Authentication**: The application checked these credentials with the Active Directory server.
3. **Iron Session Creation**: If successful, Iron Session created a new session cookie and user information was saved to the PostgreSQL database.
4. **Session Retrieval**: On subsequent requests, the application checked if the user's session was still valid by decrypting the cookie on the server and checking if the isAuthenticated property was set to true.

The broad logic was handled in the **login function** where it first retrieved the submitted values, validated it using zod (line 18, 19), called the **authenticate function**, and lastly returned the appropriate response to the client while creating a new session if the user got authenticated successfully.

```
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  }
```

Login function located in /lib/auth/index.ts

The **authenticate function** contained the important logic for authenticating the user with the AD server and saving the result to the database. First, the account's existence on the AD server was verified, and if negative, the according response was returned early. If it did exist, accounts privileges were inspected, and lastly the results of these queries were saved to the database and returned back to the login function.

```
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  }
```

Authenticate function located in /lib/auth/activedirectory/authenticate.ts

For checking if the user existed (line 20), the ad.authenticate() method was used and for checking the account existed in the specific AD group (line 30 and 34), the ad.isUserMemberOf() method. The detailed code for this was located in the files in lib/auth/activedirectory/.

Subsequent requests

On every request, Next.js automatically recognized the /middleware.ts file and executed the default export of that file before any pages were served. This made it the perfect place to verify user authentication. The code included redirecting authenticated users on /login to /, while unauthenticated users were being redirected to /login if they were not there already.

```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;
}
```

Since Next.js recommended treating server actions like public API routes, user authentication was also **verified in every server action**.

Additionally, checks for admin permissions were distributed across the app to display certain things for them that normal users should not see, with the most critical point being a programmatic redirection in the admin-dashboard layout.

```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 />;

```

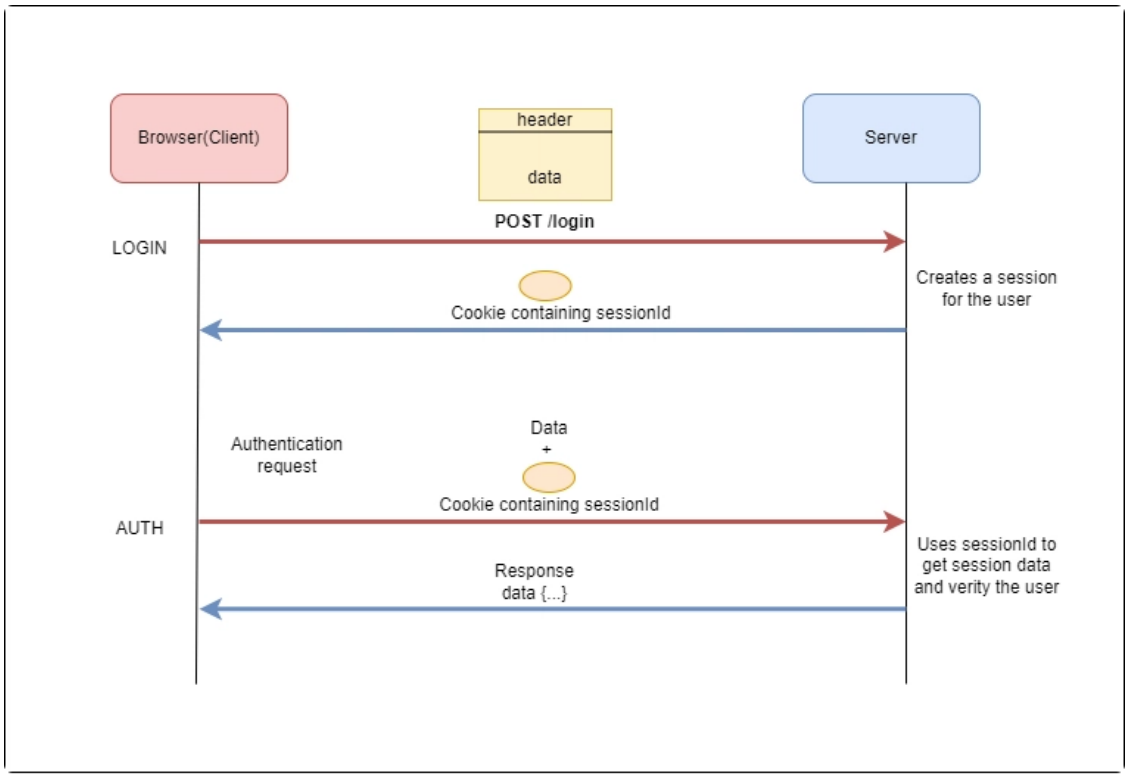
Dashboard layout located in /app/[locale]/dashboard/layout.tsx

Session-based vs. Token-based authentication

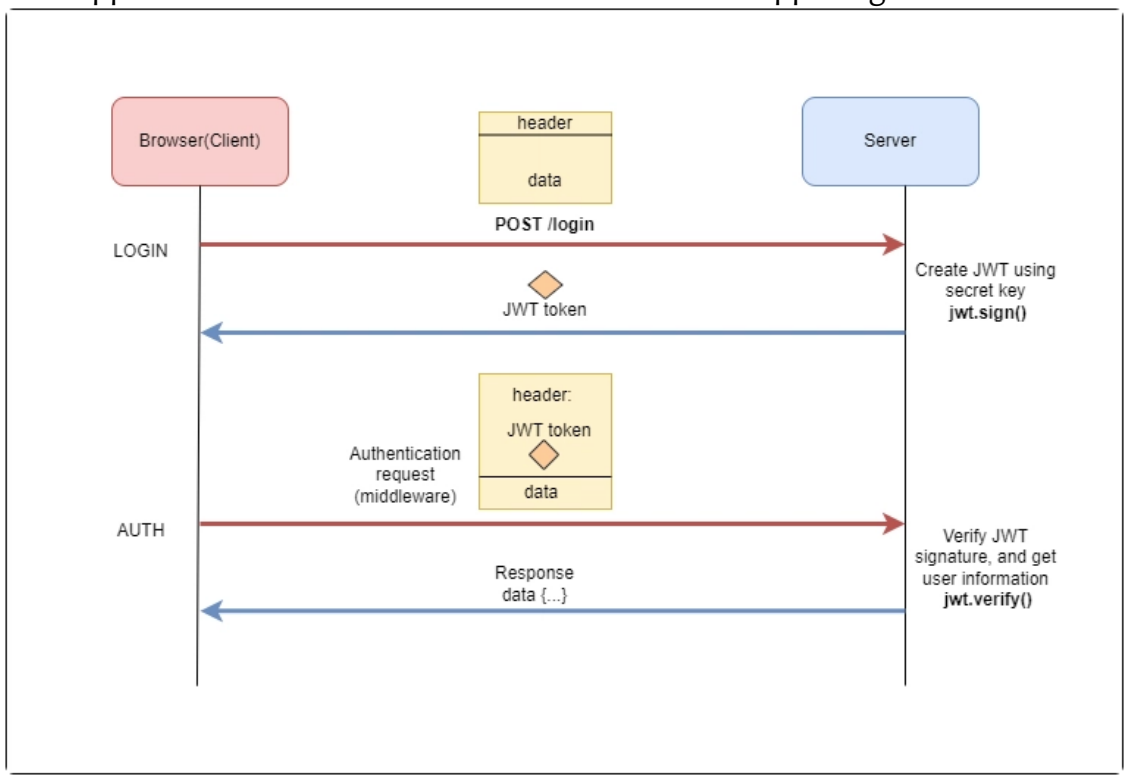
Two methods for securely persisting user sessions were examined:

1. **Session-based authentication**: A unique session ID was generated upon login, stored on the server. The session ID was sent to the client **as a cookie** to verify the user's identity in subsequent requests, with mechanisms for expiration and

invalidation.



2. **Token-based authentication**: A JSON Web Token (JWT) was generated after login, containing user information and an expiration timestamp. The client stored the token and sent it in the **Authorization header** with each request. This method allowed for **stateless authentication**, as the server did not maintain session state, and supported cross-domain authentication and mobile app integration.



Given the limited scope of the application and hosting on a singular server, session-based authentication was deemed appropriate.

Sources:

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

INTERNATIONALIZATION (i18n)

Internationalization (i18n) was the process of designing and developing software that could be easily adapted to different languages, cultural contexts, and regions without major changes to the core codebase. It included translating the user interface, handling Unicode, and separating content from code, ensuring the application was accessible and usable by a global audience.

I18next was chosen as the base library for i18n, along with additional packages. An external service called i18nexus was included, providing a Graphical User Interface (GUI) for managing translations and the ability to automatically translate strings from the base language into other languages.

Special terms in this chapter included:

- **namespace**: A way to organize translation keys into separate groups, allowing for better management and structure of translations within i18next.
- **translation-string**: A key-value pair where the key was a unique identifier for a specific text string, and the value was the actual translated text in the target language.
- **interpolator**: A feature in i18next that allowed for dynamic insertion of variables into translation strings, enabling the creation of more flexible and context-aware translations.

Implementation

At first, **React-Intl** was considered as an i18n library. However, **i18next** was determined to be the superior choice for internationalization in React applications due to its comprehensive feature set, easier integration, more intuitive API, larger and more active community, and better performance, making it the preferable solution for the project.

In addition to **i18next**, other packages were used:

- **i18next** was the core internationalization library that provided the foundational functionality for managing translations and localization.
- **react-i18next** was the package that integrated i18next with React, providing hooks and components that made it easier to work with translations in React components.
- **i18next-resources-to-backend** was the plugin that enabled the loading of translation resources from a backend server. It was particularly useful for server-side rendering (SSR), allowing the application to fetch translations dynamically based on the user's locale.
- **next-i18n-router** was specifically designed for Next.js app router projects. It implemented internationalized routing and locale detection, allowing developers to easily manage routes based on the selected language without having to build the routing logic from scratch.

Setup

1. The packages were installed.
2. A config file was created (/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,
};
```

   - It specified a locales property, which was an array of languages the app would support.
   - The defaultLocale property was the language that visitors would fall back to if the app did not support their language.

3. A dynamic segment was created inside the /app directory to contain all pages and layouts, named [locale].
4. The middleware was updated (/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).*)",
};
```

- The next-i18n-router package made it easy, as it returned the value of the **i18nRouter function**, handling all the locale routing logic.

5. The initTranslations function was created (/app/i18n.js), which used **i18next-resources-to-backend** to load translations server-side, with code copied from the tutorial.
6. The TranslationsProvider was added (/contexts/translations-provider.jsx), which wrapped the components where the t function from react-i18next was used, with code copied from the tutorial.
7. The generateStaticParams function from the Next.js API was added to the root layout to **statically generate** routes at build time instead of on-demand at request time.
8. The app was connected to the i18nexus platform, with more details available in the "i18nexus" and "i18nexus integration" sections.

For the setup, tutorials by **i18nexus** were referenced:

- **Written tutorial**
- **Video tutorial (30 min)**

Usage
In a client component, the translations function (called t) was obtained by de-structuring it from the useTranslation hook from **react-i18next**.

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

In a server component, the translations function (called t) was obtained by de-structuring it from the initTranslations function created in the setup, passing in the current locale and an array of namespaces.

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

After that, the t function could be used anywhere in the component by passing in the translation string and, if applicable, the interpolator.

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

i18next included a variety of other features, but these were the only ones used in this project.

i18nexus

i18nexus was a platform that simplified internationalization (i18n) and localization (l10n) for software applications. The old-fashioned method involved manually creating multiple JSON files for each namespace and language. However, translations were written and managed in the Graphical User Interface (GUI) provided by i18nexus. Translations were

first written in a base language (English) and then automatically translated into other languages.

One notable aspect was that the application did not depend on an external service. It was possible to pull all translation JSON files into the project using a terminal command.

Initially, only the free plan was used, but later the basic plan was purchased due to running out of translation strings. After finishing the app, this plan was canceled, and the application continued to function.

One problem that was noticed was that the platform used the Google Translate API on the free and basic plans, which only supported Brazilian Portuguese. After contacting support, it was able to quickly implement a fix where the platform used the DeepL translator for European Portuguese, even on the lower tiers.

When using the App Router with i18next, it was a good practice to "namespace" strings per page. This approach allowed it to avoid loading all strings for the entire app when viewing one page, enabling it to load only the strings for that specific page at a time.

i18nexus integration

To connect the app to i18nexus, these steps were followed:

1. **i18nexus-cli** was installed globally (bun i i18nexus-cli -g) and as a dev dependency (bun i i18nexus-cli --save-dev), which was the command line interface used to pull translation files into the project.
2. The project API key was added to the .env file with the variable named I18NEXUS_API_KEY.
3. The command i18nexus pull was executed from the terminal in the root directory of the project to pull or update the locales.
4. For convenience, this command was also added to the package.json scripts, so that the most up-to-date translations were automatically pulled whenever a server was spun up.

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

SELF-HOSTING & DEPLOYMENT

The app was deployed on a school computer in a Docker container. To simplify access, it obtained a free domain name from No-IP, a DDNS provider. The traffic flowed as follows:

1. A client requested etpzp-sms.ddns.net.
2. The router received the request and forwarded it to Nginx.
3. Nginx redirected to HTTPS if necessary and forwarded the request to the Docker container.
4. The Node.js server in the container processed the request.
   This setup allowed easy access to the app through a simple domain name while ensuring proper routing and security.

Docker

Docker was chosen as an open-source platform that allowed developers to package applications and their dependencies into lightweight, portable containers, simplifying deployment and enhancing portability across different environments.

During production, there were 2 separate Docker containers: one for the web application with the Node.js server itself, and another one for the PostgreSQL database. Either of them ran Alpine Linux, which was a very lightweight Linux operating system.

Other files related to Docker that were not explained included .env.docker and .dockerignore, which were used to manage environment variables and specify files and directories that should be excluded from the Docker build context, respectively.

Dockerfile explained

A Dockerfile was a text file that contained a series of instructions for building a Docker image, specifying the application environment, dependencies, and configuration needed to run the application. It had 2 Dockerfiles in the application.

Node.js Dockerfile

This was the most important Dockerfile located in /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"]
```

1. **Base Image**: A base image was defined using a lightweight Alpine Linux environment with Bun (line 1).
2. **Install Node.js and i18nexus**: Node.js and npm was installed, along with the i18nexus CLI for translations (lines 4-5).
3. **Dependencies Stage**: A new stage named deps was created to install application dependencies. It set the working directory, copied necessary files, and ran the installation command (lines 8-12).
4. **Build Stage**: The builder stage was initiated, where it set the working directory, copied installed dependencies from the previous stage, and built the application (lines 15-19).
5. **Production Server Stage**: The final stage, runner, set the working directory and defined the environment variable for production. Built application files from the previous stage were copied (lines 22-28).
6. **Copying Additional Files**: The code also copied the package.json and node_modules from the previous stage to ensure all necessary files were available (lines 33-34).
7. **Expose Port**: The Dockerfile exposed port 3000, allowing external access to the application (line 36).
8. **Start Command**: Finally, a command was specified to start the application (line 37).

Database Dockerfile

This was the simplest configuration located in /lib/db/Dockerfile for setting up and seeding the database:

1. **Base image**: The base image was defined using a specific version of PostgreSQL, which was based on a lightweight Alpine Linux variant (line 1).
2. **Copy seed script**: A SQL file named seed.sql was copied into a designated directory within the PostgreSQL container used for seeding the database when the container started (line 2).

docker-compose.yaml explained

A docker-compose.yaml file was a text file that defined a multi-container Docker application. It specified the services (containers) that made up the application, their configurations, and how they interacted with each other. This file allowed for the definition and management of the entire application stack, including networking, volumes, and environment variables, in a single file.

It would have been possible to achieve the same results without Docker Compose by creating and managing the individual Docker containers, networks, volumes, and other resources required for the application. However, this would have been more complex and time-consuming.

The Docker Compose file, located in /docker-compose.yaml, defined 2 services: web and database.

```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"
```

- The web service:
  - It built the Docker image using the Dockerfile in the current directory.
  - It loaded environment variables from the .env.docker file.

    - It exposed port 3000 on the host and mapped it to port 3000 in the container.
    - It depended on the database service and waited for it to be healthy before starting.
- The database service:
    - It built the Docker image using the Dockerfile in the ./lib/db directory.
    - It set the container name to postgres.
    - It loaded environment variables from the .env.docker file.
    - It exposed the POSTGRES_PORT environment variable on the host and mapped it to the same port in the container.
    - It mounted a volume named database-v to the /var/lib/postgresql/data directory in the container.
    - It defined a healthcheck that checked if PostgreSQL was ready to accept connections every 5 seconds, with a timeout of 5 seconds and a maximum of 5 retries.
- The database-v volume was defined to persist the PostgreSQL data.

No-IP & port forwarding

To simplify user access, a free domain name was obtained from No-IP, a dynamic Domain Name Service (DNS) provider. This allowed users to connect to the application using a memorable domain name instead of the router's IP address, which may have changed frequently.

No-IP automatically updated the domain name to reflect the current IP address of the router, ensuring consistent access. This feature was particularly useful in environments where dynamic IP addressing was common. In other words, it basically made the dynamic IP behave like a static one.

To set up No-IP, **this guide** was referenced which explained what was done to set up No-IP:

1. **Create an account:** A new account was created on No-IP's website and the required information was filled in.
2. **Confirm the account:** The email was checked for a confirmation link and clicked it.
3. **Log in:** It accessed the account using the email and password.
4. **Adding a hostname:** A hostname for the server was created
5. **(Optional) Creating a dynamic DNS key:** A dynamic DNS key was created for added security and compatibility.
6. **Making the host dynamic:** No-IP's Dynamic Update Client (DUC) was installed and configured the device for updates.
7. **Configuring the router:** Port forwarding was set up for necessary services (e.g., web, FTP).
8. **Running the services:** The setup was verified with a port check tool and started using the services.

Port forwarding was configured on the router to direct incoming traffic to the specific port of the Docker container running the application. This setup enabled users to reach the application easily and from networks beyond just the school's network. For the configuration, **this guide** was referenced.

Nginx

Nginx provided various features, primarily serving as a web server and functioning as a reverse proxy to redirect traffic to other servers. In this project it was used to set up SSL certificates, redirect http traffic to https, and to redirect traffic to the application running inside Docker. For learning the basics of Nginx, **this tutorial** was referenced.

Nginx was fairly easy to set up:

1. Nginx was installed.
2. It was started using the nginx command.
3. It was first generated a self-signed SSL certificate using this command: openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout nginx-selfsigned.key -out nginx-selfsigned.crt
4. Then, an attempt was made to use a command from Certbot to generate an authority-signed SSL certificate for free. However, this process encountered an issue because Certbot was not compatible with the Windows school computer.
5. The rest of the work consisted of editing the nginx.conf file, where all Nginx's behavior was defined.

nginx.conf
Even though the Nginx configuration (/nginx.conf) file was committed to the repository, it was not read from this file. It existed to ensure availability when needed. The actual config file's location could be checked by running nginx -V, allowing the path containing the config file nginx.conf to be copied. Here was the basic configuration:

```
# 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
        }
    }
}
```

- It set up Nginx with 1 worker process and 1024 connections.
- It redirected all HTTP traffic (port 80) to HTTPS (port 443).
- It used self-signed SSL certificates for HTTPS.
- It proxied requests to a backend service running on port 3000.
- It passed client information through headers to the backend.

Useful commands

- nginx -s reload reloaded the configuration without dropping connections.
- nginx -s stop gracefully stops the server.
- nginx -s quit stopped the server immediately after closing current connections.
  For more information on nginx, the **documentation** was referenced.

SSL certificates

Getting a **self-signed** SSL certificate was easily done using the following command. It generated a self-signed key, which it placed in ~/nginx-certs/ and then referenced from the Nginx config file using the absolute path. It changed into that newly created directory:

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

Getting an **authority-signed** SSL certificate was done using Certbot by completing a "challenge." It required the domain name for this, **read more here**:

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

After that, it still had to do more tasks, such as generating a symlink. It followed **this tutorial** for all of the steps.

CONCLUSION

In conclusion, it was determined that the application effectively addressed the high costs of text messaging for the school by offering an efficient and affordable SMS communication solution accessible to all users. The project was fully responsive and usable on mobile devices, with all planned features implemented and additional ones included. With capabilities such as multi-recipient messaging, scheduled sends, and a user-friendly interface, it streamlined communication and integrated seamlessly with the school's Active Directory for authentication.

The website was recognized for its rapid production, thanks to robust technologies and a strong focus on performance. This emphasis resulted in a snappy interface with minimal latency, enhancing the user experience and making it feel more like an application than a website. Built with top-tier technologies like Next.js and PostgreSQL, it showcased the potential of leveraging REST APIs for SMS functionality. Lastly, deploying it using Docker and Nginx on a local school computer was great for learning the basics of hosting and demonstrated the practical application of these technologies.

While the project presented challenges, it was particularly difficult to bring it to a close towards the end, ultimately serving as a significant learning experience. With the knowledge gained, it was anticipated that future iterations of similar apps could be developed more efficiently and effectively.

Regrets

- Only using the basic i18next features without i18nexus, avoiding plural and translation branching.
- Repeating complex scroll area calculation often instead of managing it in one place
- Complex front-end settings setup with some settings being handled by libraries
- Slow realization of incompatibility between Sheet and ScrollArea ShadCN components
- Using .safeParse instead of .parse with zod causing error handling code to live in the try block which doesn't adhere to separation of concerns
- No snippets for repetitive code
- No uniform way to handle errors in server actions
- No clear/consistent naming conventions for functions, TypeScript types, and Zod schemas

Omitted features

The most important feature not implemented was **polling the API for SMS delivery status**. It was crucial because, while immediate errors were managed on both the user's side and the gateway API's side, messages could fail to reach the end recipient due to issues such as an invalid phone number or problems with the recipient's phone. This status was to be displayed in the app. For information on how to poll the API, the **GatewayAPI documentation** was referenced.

Although the API recommended **using webhooks** rather than polling for efficiency, it required message polling due to its self-hosting setup. This approach allowed it to manage situations where the server might be turned off during holidays, ensuring that it could still retrieve SMS delivery status when the server was back online.

Even though this feature was not implemented, notes were taken:

- From a logic standpoint, the database field status of scheduled messages should not be "SCHEDULED" when it's delivery date was reached.
- On user login, messages could be checked for the confirmed_delivery flag.
- Delivery errors for individual recipients could be displayed on the message display.
- A field like was_scheduled or scheduled_send could be added to indicate how the message was sent.
- For updating amount indicators, a 5-minute refresh timer for polling scheduled message delivery statuses could be added in the root layout.

Other Features

- Links to the modified/created item in success toast messages for easy access to details
- Admins only:
    - Links to GatewayAPI sign-in page on admin dashboard
    - Auth cookie max-age setting
    - Option to back up and restore the database
    - Option to specify available select option/s for the name of the sender
- More contact information displayed on each message item in the list
- Contact profile photos
- Undefined values should be passed as null to the database

ATTACHMENT I - USER MANUAL

This chapter provides clear, step-by-step explanations of common procedures and non-intuitive processes to help new users navigate the project and access necessary tools for expansion. Additional tips and guides for specific setups can be found in other chapters.

Getting started

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

1. The user installs the Bun package manager by following the instructions on its **website**.
2. The user sets the required environment variables found in the ATTACHED section. These include .env and .env.docker, which both go into the root directory of the project.
3. The user navigates into the correct directory from a terminal application of their choice.
4. The user installs the packages by executing the bun install terminal command.
5. The user starts the development server by executing the bun dev terminal command. If an error occurs, the user can use the alternative command: bun next dev.

Note: The user can utilize any package manager they prefer, but the author recommends using Bun as it is the fastest and most efficient package manager, while also providing a nearly identical API to npm.

GitHub

Getting Started: The user creates a GitHub account and sets up Git on their local machine. They configure their username and email with git config --global user.name "Your Name" and git config --global user.email "your.email@example.com".

Cloning the Repo: The user uses the command git clone <repository-url> to copy a remote repository to their local machine, allowing them to work on the project locally.

Adding a New Branch and Setting Upstream: The user creates a new branch with git checkout -b <branch-name>, then pushes it to the remote repository for the first time using git push -u origin <branch-name>. The -u flag sets the upstream tracking reference, linking the local branch to the remote branch. Branches are created only for new features, and once a feature is completed, tested, and working, it can be merged into the main branch.

Pushing to the Repo: After making changes, the user stages them with git add ., commits with git commit -m "Your message", and pushes to the remote repository using git push origin <branch-name>.

Working in a development environment

This process can be tricky across different platforms, but the setup and some tips for developing this project are explained below.

Node.js Web Server
To start a development server, the following command is used:

bun dev

If there is no internet connection or another error occurs, the command below is utilized:

bun next dev

Debugging PostgreSQL Database
On macOS, Postgres.app must be running in the background for it to function properly. To query the database directly, the user executes the psql command in a terminal to access the psql shell, where all queries can be run.

On Windows, Postgres should always be running. The user opens the psql app, which contains the psql shell for executing queries.

> Tip: Connection issues are likely due to invalid credentials.

PostgreSQL Commands

- To seed the database, the user runs \i your_project_file_path/lib/db/seed.sql in the psql shell. This command is the same for both macOS and Windows. However, if issues arise on Windows, the user should try using backslashes (\) instead of forward slashes (/).
- **SQL Commands**:
  - To delete all tables: DROP TABLE IF EXISTS recipient, contact, message, public.user;
  - To check how many messages were sent in the last 30 days: SELECT COUNT(*) FROM message WHERE send_time >= CURRENT_DATE - INTERVAL '1 months' AND in_trash = false AND status NOT IN ('FAILED', 'DRAFTED');

Working in a Production Environment (Deployment)

1. The user ensures the Docker engine is running by opening the Docker app.
2. The user starts the Docker containers with the following command:

docker-compose up --build

3. If Nginx is not running, the user executes this command:

nginx

4. The user restarts the Nginx web server using the command:

nginx -s *reload*

Note: In the first command, the --build flag is optional. It prompts Docker to rebuild the images and should be used when there are changes that need to be applied. If omitted, Docker Compose uses existing images, speeding up the process.

Debugging Docker

Accessing a Docker container: To access a running Docker container, the user executes the following command:

docker exec -it <container_name_or_id> /bin/sh

The user replaces <container_name_or_id> with the actual name or ID of the container. Since Alpine is in use, the user accesses the sh shell instead of bash.

Accessing a PostgreSQL database in a Docker container: To access a PostgreSQL database via the psql shell running inside a Docker container, the user runs:

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

The user replaces <postgres_container_name_or_id>, <username>, and <database_name> with the appropriate values.

More commands:

- **Listing Running Containers**: docker ps
- **Stopping a Container**: docker stop <container_name_or_id>
- **Starting a Container**: docker start <container_name_or_id>
- **Removing a Container**: docker rm <container_name_or_id>
- **Viewing Container Logs**: docker logs <container_name_or_id>

Danger zone:

- **Removing stopped containers**: docker container prune
- **Removing unused images**: docker image prune
- **Removing unused volumes**: docker volume prune
- **Removing unused networks**: docker network prune
- **Removing all Docker resources**: docker system prune -a --volumes

Working with i18nexus

Disclaimer: Choosing not to use i18nexus and editing JSON files manually results in permanent loss of changes when running the pull command, as the /locales directory is not committed to git.

1. The user signs in to the **i18nexus platform** with the provided account.
2. The user makes changes on the platform. The free plan limits translation strings, preventing the addition of new ones. An "archive (not used anywhere)" namespace is available for moving and editing unused translations.

3. The user syncs changes by running:

i18nexus pull

ATTACHMENT II - CODE FILES

/middleware.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 l
ocale 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

```
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

```
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

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

/types/action.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

```
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

```
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";

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

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

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

export const appearanceLayoutValues = ["MODERN", "SIMPLE"] as const; // this is need
ed 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 new message, v
iew 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>(undefined);

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 sh
ould 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 be used on the s
erver
  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 whe
n 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.phone)
     )
     // 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 array
```

```
    .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 function only gets ru
n 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 format
    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 === phone);
```

```
    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 ch
ecks
  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) - deselect the previous p
hone
    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 error pulsing for more inva
lid recipients.
    setMessage((prev) => ({ ...prev, invalidRecipients: undefined }));
   }
  }
```

```
  // Note: Works only correctly here; won't update correctly with add/remove 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 doesn't re-render when a draft gets saved */}
   <InsertContactModal />
   <ScheduleMessageModal />
   <ScheduleAlertModal />
   {/* This should always be defined as we pass a defaultPhone and may create 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 th
ose 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) || undefined,
  });

  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 as strings bec
ause 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 iss
ues
    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

```tsx
"use client";

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

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 "react";
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

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

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

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

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

export default function Loading() {
```

```
 return <MessagesPageSkeleton category="TRASH" />;
}
```

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

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

export default function Loading() {
```

```
 return <MessagesPageSkeleton category="SENT" />;
}
```

/app/[locale]/(root)/(message-layout)/sent/page.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 provider for the nav pan
el */
    <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";
```

```tsx
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 namespace as we only pu
t 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 provider for the nav p
anel */
        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`, // Diagonal grad
ient 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(--simple
-header-height))]">
     <div className="flex-1 flex flex-col items-center justify-center gap-1
0">
      {/* <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 from 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 between 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,
status, 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, fir
st_name, last_name, lang, profile_color_id, display_name, dark_mode, prima
ry_color_id)
        VALUES
          ('Alice Johnson', 'alice@example.com', 'USER', NOW(), NOW(), 'Al
ice', 'Johnson', 'en', 1, 'Alice J.', false, 1),
          ('Bob Smith', 'bob@example.com', 'USER', NOW(), NOW(), 'Bob', 'S
mith', '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(), 'Dav
id', '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(), 'Fra
nk', '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(), 'Ir
ene', '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 provider for the nav pan
el */
    <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 scroll 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 props are used as it han
dles its own submission,
       // and setServerState is passed so elements update with 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 inter
nally (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-between"
     )}
    >
     <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 provider for the nav pan
el */
    <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 shadow-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-foreground
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-foregroun
d 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 shadow-none"
           type="button"
           disabled
          >
           <ChevronDown className={cn("h-4 w-4 transition-transform")} />
          </Button>
         </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-full
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 text-mu
ted-foreground">
       {title}
      </span>
     ) : (
      <span className="mb-1 flex items-center text-sm text-muted-foreground
 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-message";
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.scheduledDateConfirmed,
        }
      : 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 authorization
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_FORMAT),
   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 element bigge
r */
 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 only 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-sc
reen`}
   >
    <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); /* Diagonal 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:outline-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 f
ocus-visible:border-b-ring disabled:opacity-100 placeholder:text-muted-foreground;
}

.shadcn-input {
  @apply flex h-9 w-full rounded-md bg-transparent px-3 py-1 text-base shadow-sm tra
nsition-colors file:border-0 file:bg-transparent file:text-sm file:font-medium file:text-acc
ent-foreground placeholder:text-accent-foreground focus-visible:outline-none focus-vi
sible:ring-1 focus-visible:ring-ring disabled: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] rout
e 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 (als
o 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

```
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 app 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 hover
: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 s
hadow-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 s
hadow-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>
    </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-[state
=closed]:animate-out data-[state=closed]:fade-out-0 data-[state=open]:fade
-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 tra
nslate-x-[-50%] translate-y-[-50%] gap-4 border bg-background p-6 shadow-l
g duration-200 data-[state=open]:animate-in data-[state=closed]:animate-ou
t 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-
out-to-left-1/2 data-[state=closed]:slide-out-to-top-[48%] data-[state=ope
n]:slide-in-from-left-1/2 data-[state=open]:slide-in-from-top-[48%] sm:rou
nded-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 tex
t-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=a
ctive]:bg-background data-[state=active]:text-foreground data-[state=activ
e]: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-offset-
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 shadow-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=cl
osed]: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-bottom-2 dark:border-slate-8
00 bg-background dark:text-slate-50",
    className
   )}
   {...props}
  />
 </PopoverPrimitive.Portal>
));
PopoverContent.displayName = PopoverPrimitive.Content.displayName;

export { Popover, PopoverTrigger, PopoverContent, PopoverAnchor };
```

/components/ui/chart.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[stroke='
#ccc']]:stroke-border/50 [&_.recharts-curve.recharts-tooltip-cursor]:strok
e-border [&_.recharts-dot[stroke='#fff']]:stroke-transparent [&_.recharts-
layer]:outline-none [&_.recharts-polar-grid_[stroke='#ccc']]:stroke-border
 [&_.recharts-radial-bar-background-sector]:fill-muted [&_.recharts-rectan
gle.recharts-tooltip-cursor]:fill-muted [&_.recharts-reference-line_[strok
e='#ccc']]:stroke-border [&_.recharts-sector[stroke='#fff']]:stroke-transp
arent [&_.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}</div>;
 }, [
  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-slate-
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, key);
     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-foreground",
        indicator === "dot" && "items-center"
       )}
      >
       {formatter && item?.value !== undefined && item.name ? (
        formatter(item.value, item.name, item, index, item.payload)
       ) : (
        <>
         {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-transparent":
               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-9
50 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", nameKey },
  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-mute
d-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-[state
=closed]:animate-out data-[state=closed]:fade-out-0 data-[state=open]:fade
-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=op
en]: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-bottom d
ata-[state=open]:slide-in-from-bottom",
    left: "inset-y-0 left-0 h-full w-3/4 border-r data-[state=closed]:slide-o
ut-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]:slide-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 opacit
y-70 ring-offset-white transition-opacity hover:opacity-100 focus:outline-
none focus:ring-2 focus:ring-slate-950 focus:ring-offset-2 disabled:pointe
r-events-none data-[state=open]:bg-slate-100 dark:ring-offset-slate-950 da
rk: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.ScrollAreaScrollbar>
>(({ 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-ful
l 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:absolute
  after:inset-y-0 after:left-1/2 after:w-1 after:-translate-x-1/2 focus-vis
ible:outline-none focus-visible:ring-1 focus-visible:ring-ring focus-visib
le: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-
direction=vertical]:after:w-full data-[panel-group-direction=vertical]:aft
er:-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 text-
slate-50 animate-in fade-in-0 zoom-in-95 data-[state=closed]:animate-out d
ata-[state=closed]:fade-out-0 data-[state=closed]:zoom-out-95 data-[side=b
ottom]:slide-in-from-top-2 data-[side=left]:slide-in-from-right-2 data-[si
de=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-slat
e-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-5
00 dark:border-red-900/50 dark:text-red-900 dark:dark:border-red-900 dark:
[&>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 rounded-f
ull border-2 border-transparent shadow-sm transition-colors focus-visible:
outline-none focus-visible:ring-2 focus-visible:ring-slate-950 focus-visib
le:ring-offset-2 focus-visible:ring-offset-white disabled:cursor-not-allow
ed disabled:opacity-50 data-[state=checked]:bg-slate-900 data-[state=unche
cked]:bg-slate-200 dark:focus-visible:ring-primary dark:focus-visible:ring
-offset-slate-950 dark:data-[state=checked]:bg-slate-50 dark:data-[state=u
nchecked]:bg-slate-800",
   className
  )}
  {...props}
  ref={ref}
 >
  <SwitchPrimitives.Thumb
   className={cn(
    "pointer-events-none block h-4 w-4 rounded-full bg-background shadow-lg
 ring-0 transition-transform data-[state=checked]:translate-x-4 data-[stat
e=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:text-sla
te-400",
    row: "flex w-full mt-2",
    cell: cn(
     "relative p-0 text-center text-sm focus-within:relative focus-within:z
-20 [&:has([aria-selected])]:bg-slate-100 [&:has([aria-selected].day-outsi
de)]:bg-slate-100/50 [&:has([aria-selected].day-range-end)]:rounded-r-md d
ark:[&:has([aria-selected])]:bg-muted dark:[&:has([aria-selected].day-outs
ide)]:bg-muted/50",
     props.mode === "range"
      ? "[&:has(>.day-range-end)]:rounded-r-md [&:has(>.day-range-start)]:ro
unded-l-md first:[&:has([aria-selected])]:rounded-l-md last:[&:has([aria-s
elected])]: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-primar
y-foreground",
    day_today: "border border-muted",
    day_outside:
     "invisible day-outside text-slate-500 aria-selected:bg-slate-100/50 ar
ia-selected:text-slate-500 dark:text-slate-400 dark:aria-selected:bg-muted
/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:aria-sel
ected: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-slate-
900 shadow focus:outline-none focus-visible:ring-1 focus-visible:ring-slat
e-950 disabled:cursor-not-allowed disabled:opacity-50 dark:border-slate-80
0 dark:border-slate-50 dark:text-slate-50 dark:focus-visible:ring-primary",
    className
   )}
   {...props}
  >
   <RadioGroupPrimitive.Indicator className="flex items-center justify-center
">
    <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 te
xt-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-hea
ding]]:font-medium [&_[cmdk-group-heading]]:text-muted-foreground [&_[cmdk
-group]:not([hidden])_~[cmdk-group]]:pt-0 [&_[cmdk-group]]:px-2 [&_[cmdk-i
nput-wrapper]_svg]:h-5 [&_[cmdk-input-wrapper]_svg]:w-5 [&_[cmdk-input]]:h
-12 [&_[cmdk-item]]:px-2 [&_[cmdk-item]]:py-3 [&_[cmdk-item]_svg]:h-5 [&_[
cmdk-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-none p
laceholder:text-muted-foreground disabled:cursor-not-allowed disabled:opac
ity-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", className
me)}
    {...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 [&_[cm
dk-group-heading]]:py-1.5 [&_[cmdk-group-heading]]:text-xs [&_[cmdk-group-
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-none
data-[selected=true]:bg-slate-100 data-[selected=true]:text-slate-900 data
-[disabled=true]:opacity-50 [&_svg]:pointer-events-none [&_svg]:size-4 [&_
svg]:shrink-0 dark:data-[selected=true]:bg-slate-800 dark:data-[selected=t
rue]: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-10
0 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-[state
=closed]:animate-out data-[state=closed]:fade-out-0 data-[state=open]:fade
-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-[state=
closed]:fade-out-0 data-[state=open]:fade-in-0 data-[state=closed]:zoom-ou
t-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-fr
om-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 opacit
y-70 ring-offset-white transition-opacity hover:opacity-100 focus:outline-
none focus:ring-2 focus:ring-slate-950 focus:ring-offset-2 disabled:pointe
r-events-none data-[state=open]:bg-slate-100 data-[state=open]:text-muted-
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:spa
ce-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-
950 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-slate-9
00/80 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-500/8
0 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-sla
te-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]:bg-s
late-100 dark:hover:bg-slate-800/50 dark:data-[state=selected]:bg-slate-80
0",
   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 [&:h
as([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]]:tran
slate-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-
visible:ring-1 focus-visible:ring-primary disabled:pointer-events-none dis
abled: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 focus-v
isible:ring-white",
    destructive:
     "bg-red-500 text-slate-50 shadow-sm hover:bg-red-500/90 dark:bg-red-90
0 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-transparent
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 focus-v
isible:outline-none focus-visible:ring-1 focus-visible:ring-slate-950 disa
bled: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 py-1
.5 text-sm outline-none focus:bg-slate-100 data-[state=open]:bg-slate-100
[&_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 s
hadow-lg 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-in-from
-top-2 data-[side=left]:slide-in-from-right-2 data-[side=right]:slide-in-f
rom-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-[sta
te=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-from-l
eft-2 data-[side=top]:slide-in-from-bottom-2",
    className
   )}
   {...props}
  />
 </DropdownMenuPrimitive.Portal>
));
DropdownMenuContent.displayName = DropdownMenuPrimitive.Content.displayNam
e;
```

```
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-[d
isabled]: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-[di
sabled]: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-c
enter">
   <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 focus
:text-slate-900 data-[disabled]:pointer-events-none data-[disabled]:opacit
y-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-c
enter">
   <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.displayName;

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 bord
er bg-transparent px-3 py-2 text-sm shadow-sm ring-offset-white placeholde
r:text-muted-foreground focus:outline-none focus:ring-1 focus:ring-slate-9
50 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.displayName;

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 border
bg-background text-foreground 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]:zoom-out-95 data-[state=open]:zoom-in-95 d
ata-[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-bo
ttom-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(--radix-sel
ect-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 items-c
enter rounded-sm py-1.5 pl-2 pr-8 text-sm outline-none data-[disabled]:poi
nter-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 tex
t-base shadow-sm placeholder:text-muted-foreground focus-visible:outline-n
one focus-visible:ring-1 focus-visible:ring-primary disabled:cursor-not-al
lowed 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<"input"
>>(
 ({ 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-base s
hadow-sm transition-colors file:border-0 file:bg-transparent file:text-sm
file:font-medium file:text-slate-950 placeholder:text-muted-foreground foc
us-visible:outline-none focus-visible:ring-primary focus-visible:ring-1 di
sabled:cursor-not-allowed disabled:opacity-50 md:text-sm dark:file:text-sl
ate-50 dark:placeholder:text-muted-foreground",
     className
    )}
    ref={ref}
    {...props}
   />
  );
 }
);
```

```
Input.displayName = "Input";

export { Input };
```

/components/ui/form.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 mobil
e and a message is selected we only want to show the column containing the selected
message.
        // Check if the layout is a 3-column middle-bar panel. Use the previous 3-column la
yout if available; otherwise, render the fallback for different 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 the variable has a lar
ger 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 above we are
using reverse logic here. If we are on mobile, and nothing is selected, this component s
hould 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")}</span>
      </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 chan
ge 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 when user starts typ
ing
        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 marked as prone for deleti
on
        const newRecipients = prev.recipients.map((recipient, index) => {
          if (index === lastRecipientIndex) {
            return { ...recipient, proneForDeletion: true }; // Mark as prone for deletion
          }
          return recipient; // Return the other recipients unchanged
        });

        // Return the new state with the last recipient marked as prone for 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 gets 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-fu
ll 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 text-mu
ted-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 cli
cks 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 hove
r:bg-muted dark:hover:bg-muted cursor-pointer whitespace-nowrap h-full hov
er:shadow-none",
                        error && "error-border-pulse",
                        recipient.proneForDeletion && "border-destructive",
                        !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-xl pl-1.5"
                      >
                        {recipient?.contact?.name || recipient.phone}
                      </div>

                      <Button
                        variant="none"
                        className="h-full py-0 px-1.5 cursor-pointer closeX 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 focus 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 absolute elements this c
ommon parent div */
      >
       <Input
        ref={inputElement}
        // this name only used for the focus state, not for submitting any value
        name="new-recipient"
        className={cn(
         "h-min text-sm w-full p-0 ring-0 focus:ring-0 shadow-none rounded-no
ne 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 border s
hadow-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 th
e items scroll all the way up to the end of the container */
         >
          <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 border 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 -transl
ate-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

```
"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 border 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, this component
 gets re-rendered, while the new-message-context keeps it's state
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 (>) means 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 date has not been con
firmed 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 be
tween 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 error
    );

    // 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 amount indicators) a
nd 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 amount indicators) an
d 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 is a <Select>.
  if (focusedInput) {
    const inputElement = document.querySelector(
      `[name="${focusedInput}"]`
    ) as HTMLElement;

    // Move cursor to end of textarea to prevent default behavior of placing it at the begin
ning.
    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 more checks here to pr
event the user from clicking the button over and over
                if (draft.pending || messageIsEmpty()) return; // not empty and not pending mea
ns 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 result 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 result 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 shadow-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 array 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-muted-foreg
round 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 shadow-non
e resize-none placeholder:text-muted-foreground",
         message.serverStateErrors?.body &&
          "ring-red-500 placeholder:text-red-400 dark:placeholder:text-red-40
0"
        )}
        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/.DS_Store

Bud1n-dashadmin-dashboarddsclboolmodalsdsclboolshareddsclbooluidsclbool @ @ @ @E DSDB ` @ @ @

/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-nowrap">
     {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 `New 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 accordi
ngly
    setSelected(messages[0] || null);
  }
 }, [messages]);

 useEffect(() => {
  if (isMounted && onMobile) {
    // On mobile, it should show the list by default without having the first one selected li
ke on desktop.
    setSelected(null);
  }
 }, [isMounted]);

 return (
  <>
   <ResizablePanel
    className={cn(onMobile && selected !== null && "hidden")} // If we are on mobil
e and a message is selected we only want to show the column containing the selected
message.
    // Check if the layout is a 3-column middle-bar panel. Use the previous 3-column la
yout if available; otherwise, render the fallback for different 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, this component sh
ould be hidden.
     className={cn(onMobile && selected === null && "hidden")}
    >
     {/* If you need other modals somewhere else, move the provider up the 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-foregro
und";

  // isActive takes Link, compares it to the current url, and returns whether 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-collap
sed=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) && activeStyles,
   link.isNewButton && "justify-center",
   link.hidden === true && "hidden"
  );

  return isCollapsed ? ( // NavPanel is collapsed = render with tooltips
   <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 tooltips
```

```
        <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 well
 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 not 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 spreading 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 d
oesn'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)] bo
rder-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")}</span>
   </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 = undefined;

              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-${chosenColor}` won't work bec
ause Tailwind purges unused classes in production, and it doesn't recognize dynamical
ly created class names.
                 style={{
                  // Only show color for saved contacts
                  backgroundColor: foundContactName
                   ? profileColors[index]
                   : "",
                 }}
                />
               );
              }
             )}
            </div>
            <div className="flex flex-col gap-1 grow overflow-hidden">
             <div className="flex justify-between items-center relative">
              <span className="font-semibold ellipsis">
               {message.subject || t("no_subject")}
              </span>
              {message.send_time && (
               <span
                className="text-xs text-muted-foreground relative whitespace-n
owrap"
```

```
            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-ba
ckground 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 recipie
nts 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")}:</div>

           {message.recipients.map(
            (recipientWithoutContact, index) => {
             const recipient: NewRecipient = {
              ...recipientWithoutContact,
```

```
                contact: contacts.find(
                 (contact) =>
                  contact.phone ===
                  recipientWithoutContact.phone
                ),
                // This is a temporary solution. Maybe change 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-xs h-min hover:bg-muted px-[2px]"
                 >
                  {recipient.contact?.name || recipient.phone}
                 </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-words 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 botto
m 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 well 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-renders.
    setSettings((prev) => ({
     displayName: name === "display_name" ? result.data : prev.displayName,
     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-not-a
llowed 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=${cookieValue}; pat
h=${cookiePath};`;
    }}
    onResize={() => {
     setIsCollapsed(false);
     const cookieValue = JSON.stringify(false);
     const cookiePath = "/";
     document.cookie = `react-resizable-panels:collapsed=${cookieValue}; pat
h=${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 clicked is a <Link
```

```
> and close the NavPanel. This is so that we can have the nice 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 wa
y) */
   >
    <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 localstorage
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 good. Use div inste
ad of ScrollArea, because otherwise it the Sheet component 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 j
ustify-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/card";
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 simple nav-pane
l, regardless of any layout*/
        className="h-[calc(100vh-var(--simple-header-height))]"
      >
```

```
<div
 className="p-4" /* Inside looks better with rimless bottom on scroll */
>
 <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.getDate());
  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 months
 };

 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 b
```

```
ad 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 : curr));
 }, [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 t
he 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 justi
fy-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-none">
   {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-slate-
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]:text-muted-foregroun
d
       >
        <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.cost} />
          <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 dar
k: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 items-start px-4 p
y-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 items-start px-4 p
y-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 items-start px-4 p
y-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 items-start px-4 p
y-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)?.primary
  })`, // 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_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>
   </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 updates and re-renders, som
etimes the label date was not a valid date, so we need to handle the date formatting gra
cefully to prevent a thrown error from format
              const parsedDate = parseISO(dateString);
              return isNaN(parsedDate.getTime()) // check if the date is valid before trying t
o 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

```
"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:flex-ro
w h-[325px] max-w-[250px] xs:max-w-full mx-auto xs:mx-0 p-0" /** This is the ex
act 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(value)))
            }
           />
          </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 scheduleDat
eConfirmed 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

```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 using {{name}} into th
e 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 message 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-medium"
       >
        <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 te
xt-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 mobil
e and a message is selected we only want to show the column containing the selected
message.
        // Check if the layout is a 3-column middle-bar panel. Use the previous 3-column la
yout if available; otherwise, render the fallback for different 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 the variable has a lar
ger 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 above we are
using reverse logic here. If we are on mobile, and nothing is selected, this component s
hould not be displayed.
    >
     {/* If you need other modals somewhere else, move the provider up the component
 tree. And don't forget to update the skeleton too! */}
     <ModalProvider>
      <MessageDisplay message={null} reset={() => {}} category={category} />
     </ModalProvider>
    </ChildrenPanel>
   </>
  );
}

function MessageSkeleton() {
 return (
  <div className="flex flex-col items-start gap-2 rounded-lg border p-3 tex
t-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", // bo
rder-b
     className
    )}
   >
    <DropdownMenu>
     <DropdownMenuTrigger
      className={cn(
       "flex gap-3 items-center justify-start w-full focus-primary-ring",
       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 text to the right depe
nding 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") ? "/" : "/dashboard"}>
         <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, 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 dat
a 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-foreg
round" />
    <Input /** this input is not part of a form, we are just using the input element as it ha
s 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 fallb
ack
   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<"input"
>>(
 ({ className, type, ...props }, ref) => {
  return (
   <input
    type={type}
    className={cn(
     "focus-visible:ring-b-1 focus-visible:ring-ring flex h-9 w-full rounde
d-md bg-transparent px-3 py-1 text-base shadow-sm transition-colors file:b
order-0 file:bg-transparent file:text-sm file:font-medium file:text-accent
-foreground placeholder:text-muted-foreground focus-visible:outline-none d
isabled: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 ma
tter where it got saved.
      document.cookie = `react-resizable-panels:layout:app=${cookieValue}; pat
h=${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-l
eft 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-destructive text-xs"
>
```

```
              <div className="flex h-2 w-2 rounded-full bg-destructive" />
              {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 be
fore 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 relativ
e 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-[25px] opac
ity-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-[-5px] opaci
ty-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 first one selected li
ke 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 the column containing th
e selected contact.
    // Check if the layout is a 3-column middle-bar panel. Use the previous 3-column la
yout if available; otherwise, render the fallback for different 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 rounde
d-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, this component sh
ould be hidden.
     className={cn(onMobile && selected === null && "hidden")} // like above we are
using reverse logic here. If we are on mobile, and nothing is selected, this component s
hould 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-ring
"
   >
```

```
      <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 font-di
sket-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 container
          // objectFit="cover" // This will crop the image to fill the 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")}</SubmitButton>
    </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-muted-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

```
"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 alr
eady 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 justif
y-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

```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 NewRecipie
nt[]
      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)] bo
rder-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")}</TooltipContent>
</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 text-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 o
n the button
  const { modal, setModal, scheduleDropdown, setScheduleDropdown } = useModal();
  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-foreground b
order-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 like this
       })} ${format(message.scheduledDate, PT_DATE_FORMAT)}`
      : t("submit_btn-normal")}
    </Button>
    <DropdownMenu open={scheduleDropdown} onOpenChange={setScheduleDropd
own}>
     <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 ignoring 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>(undefined)

 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 that you want it t
o 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 the
me 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 or 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_date" })
  .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 639-1 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 require 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 characters.
  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 app",
      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 ActivityDirectory 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 operator

  // 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 ret
urn 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 database 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 property
  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<SessionData>(
    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 status is als
o 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 ty
pe 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.phone)
  );
}

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 processed recipi
ents
  const recipientMap = new Map<string, RankedRecipient>(); // Use a map to track uniq
ue 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 count
    }
  });

  // 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 not 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) - ${additionalHeightPx}p
x)] md:h-[calc(100vh-var(--header-height)-${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 need to the validatio
n 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 th
em 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 i
nto 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 t
he 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 like this
    sendtime: isScheduled ? scheduledUnixSeconds : undefined, // Insert the UNIX time
stamp 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_id
```

```
    // Api errors
    res.ok ? null : res.status, // api_error_code
    res.ok ? null : JSON.stringify(resJson), // api_error_details_json

    resJson.usage.total_cost,
    resJson.usage.currency,

    userId, // user_id

    // Recipients
    validRecipients.map((recipient) => recipient.phone), // phone number 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_id

    // Api errors
    res.ok ? null : res.status, // api_error_code
```

```
    res.ok ? null : JSON.stringify(resJson), // api_error_details_json

    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) : undefined,
      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: true },
  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 producti
on 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 the dev serv
```

```
er 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 are no duplicate re
cipients
   draft = await db(
```

```
    `
            WITH insert_message AS (
              UPDATE message SET subject = $3, body = $4, sender = $5 WHER
E 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 number 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 number array
        data.recipients.map((_, index) => index), // for persisting the user specified recipien
t 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 after each server act
ion 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 contacts won't b
e 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 error 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 user_i
d = $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 error 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 database."],
      });
     }
    });
   });
  }
 } 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_name, di
splay_name) VALUES ($1, $2, $3, $4, $5, $6) RETURNING id, name, email, rol
e, first_name, last_name;",
     [
      user.email,
      user.name,
      user.role,
      user.first_name,
      user.last_name,
      `${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 accordingly.
  const { rows } = await db(
   `UPDATE public.user SET ${parsedData.name} = $2, updated_at = NOW() WHE
RE 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 details.
  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_colo
r_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 int
o 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
THEN 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)::INTEGE
R 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 t
hey 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', 'SIMPLE
')) 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 message is a
draft
    sms_reference_id BIGINT, -- can be null if the message is not scheduled, failed, or a d
raft
    status VARCHAR(20) NOT NULL CHECK (status IN ('SENT', 'SCHEDULED', 'FAILED', '
DRAFTED')), -- scheduled messages will remain with status "SCHEDULED", even when t
heir 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 different use
r, 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 accommod
ate various formats
    index SMALLINT NOT NULL, -- This is the used for persisting the order of the recipient
s of a message
    UNIQUE (message_id, phone) -- Ensure a phone number can only be added once per
message. This is not an actual field in the table, but it will make sure that there are no re
cipients 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 == undefined)
   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', 'Br
own', '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.
status = 'FAILED') -- Add the ability to edit FAILED messages from the new
-message-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 availa
ble 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 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
    }
  }
}
```

/.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 errors
    // "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;
```