Design Document

Members: Lauren Yu (ly39), Sunny Sun (ss303), Hannah Watkins (hw76)
GitHub IDs: laurennyu, ss303, watkinshannah05

Design Description

The architectural design of our project organizes the codebase into multiple folders to ensure modularity and clarity.

- The *src* folder is divided into *model*, *component*, and *view* folders.
  - The *model* folder contains files `authModel.ts` for handling user authentication and `model.ts` for managing database interactions with typed schemas.
  - The *component* folder defines reusable web components like `postTextComponent.ts` for text input, `postComponent.ts` for individual posts, `channelComponent.ts` for channels, and `workspaceComponent.ts` for workspaces.
  - The *view* folder manages the interface and interactions for specific elements like posts in `postView.ts`, channels in `channelView.ts`, and workspaces in `workspaceView.ts`, with `errorView.ts` for displaying error messages and `authView.ts` for displaying login modal.
  - Central to the system is `main.ts`, which acts as the controller, importing and orchestrating the other modules, supported by a structured logging module in `slog.ts`.
- Outside of the *src* folder that contains all typeScript files, the *html* folder holds the `index.html`, which defines the page structure and connects elements using `document.querySelector`.
- Finally, the *styles* folder includes `styles.css` to define global styles, linked to `index.html`. This modular design supports our m3ssag1n8 application structure, enabling efficient development and scalability.

Design Principles

1. **UI Simplicity Principle:** We minimized potential distractions in our messaging web application to guide users toward their intended actions. While maintaining appropriate color contrast, we adhered to a singular visual theme to avoid user confusion. In addition,

all associated features are grouped together (e.g., authorization, workspaces, channels, posts), ensuring that users do not need to search extensively for features. Only relevant features are displayed to the user; for instance, when a user is not in a workspace, features for channels and posts are hidden. When a user is in a workspace but not in a channel, they see channel-specific features, while post-specific features remain hidden.

2. **Open-Closed Principle:** We organized the application into distinct folders, each with specialized responsibilities, aligning with the open-closed principle. For instance, the component folder contains modular web components like postComponent.ts and channelComponent.ts, which are easily extensible to support new features or modified behavior without altering existing components. The central main.ts file serves as the entry point, orchestrating these modules while keeping their implementations encapsulated. This modular and extensible structure facilitates future enhancements, allowing developers to extend functionality by adding or modifying files without impacting the core logic.
3. **Single-Responsibility Principle:** We adhered to the single-responsibility principle by creating separate files dedicated to specific responsibilities. For example, in the view folder, authView.ts handles the login/logout view, workspaceView.ts manages workspaces, channelView.ts handles channels, and postView.ts is responsible for posts. Similarly, each web component class encapsulates the functionality of a single component, ensuring code clarity and maintainability.

Accessibility Principles

1. **Operable:** If a user faces an error in our system, they do not necessarily have to close out the error pop-up manually, but can carry out another task on the web application successfully and hide it that way. A user can also operate our web page without the need for “complete” motor abilities, but can solely use the keyboard to navigate around the page by tabbing. All elements that can be clicked with a mouse are focusable and reachable by just tabbing and can be activated with the keyboard, for example, by pressing enter. When wanting to create/reply to a post, the user can also easily access all available features simply with tabbing and the shift and enter keys. We use clear signposts on our page such as the workspace/channel delete buttons and pin buttons with

clear trash can and pin icons to represent the functionality. With all of these considerations in our application, our system follows the operable accessibility principle.

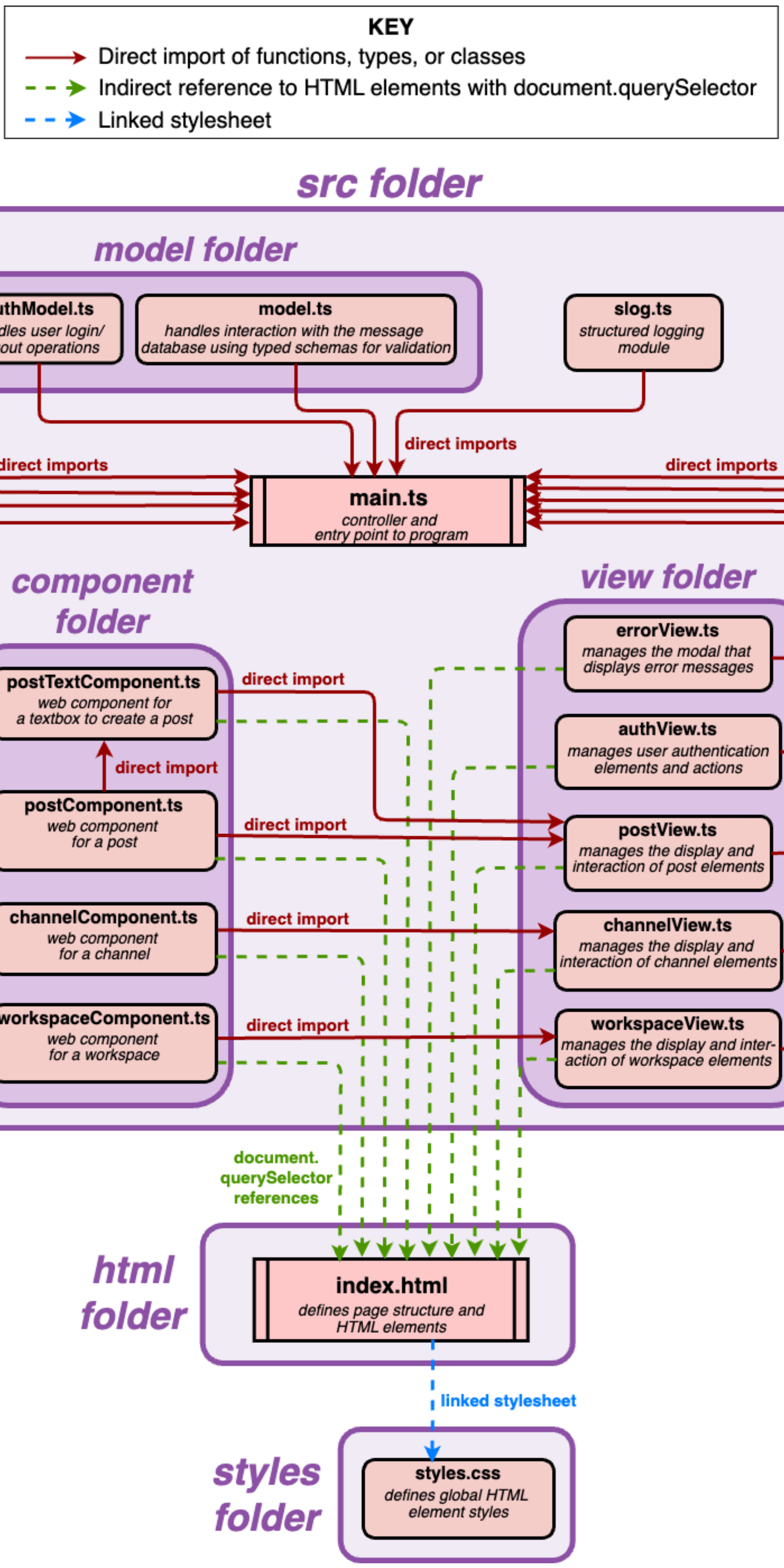
2. **Understandable:** Our messaging web application is understandable to users all-around due to the ease of knowing how to do a desired task in the application and the helpful system messages that alert the user of something gone wrong. We define the language of the site as English in our HTML file, informing screen readers of the page language so that they can correctly pronounce and read the page. Navigation elements are always in the same place regardless of the current database or workspace, with all workspaces being located in the workspaces modal and all channels being located in the side navigation. For all inputs, we notify the user if their input is valid and why, if so. We also included text along with our buttons when feasible, so a user does not have to guess what a button's symbol means to them, such as with our add workspace and add channel buttons. For inputs that don't have labels displayed on the page, we used ARIA labels to describe them. When there are errors, we promptly and efficiently inform the user of the specific error that occurred and how to avoid experiencing that error again.

Extension

The functionality we implemented for extension is **pin**. In our m3ssag1n8 application, the user can pin a workspace, channel, or post by clicking the little pin button on the side of each item. If a user chooses to pin a workspace, the workspace will appear at the top of the workspace list, making it easy for the user to find it next time. Pinning a channel will make the channel stay at the top of the channel list of that workspace. We similarly allow for pinning top-level posts in channels. If a post is pinned with replies, its replies will appear with it at the top of the page. The user may also choose to unpin a pinned workspace/channel/post. If a workspace, channel, or post is unpinned, it will not stay at the top of the list anymore but go back to its original sequence by chronological order of creation.

Concurrency

We handle concurrency in our system by being meticulous about the actions available to a user while something else is occurring. For all buttons, we ensure that they are disabled after a user first interacts with them until the desired action has been carried out or we allow the user to retry that action again. Upon notice of an action being successfully carried out or not, we reactivate the button and inform the user of any errors that occurred. In regards to posts, there were a lot of concurrency issues to be considered, such as posts arriving at the same time or receiving local and server-sent events (SSE) at the same time. To handle this, we make sure to sort every new post that comes in as it does and ensure that it goes to the correct place. We also handle all local events directly in our MVC, while only handling all outside events with SSE. This also helped us better monitor the way our system was working in a smaller context before introducing other users in the system simultaneously.


KEY
Direct import of functions, types, or classes
Indirect reference to HTML elements with document.querySelector
Linked stylesheet
src folder
model folder
authModel.ts
handles user login/ logout operations
model.ts
handles interaction with the message database using typed schemas for validation
slog.ts
structured logging module
direct imports
direct imports
direct imports
main.ts
controller and entry point to program
component folder
view folder
postTextComponent.ts
web component for a textbox to create a post
direct import
postComponent.ts
web component for a post
direct import
channelComponent.ts
web component for a channel
direct import
workspaceComponent.ts
web component for a workspace
direct import
direct import
errorView.ts
manages the modal that displays error messages
authView.ts
manages user authentication elements and actions
postView.ts
manages the display and interaction of post elements
channelView.ts
manages the display and interaction of channel elements
workspaceView.ts
manages the display and interaction of workspace elements
document. querySelector references
html folder
index.html
defines page structure and HTML elements
linked stylesheet
styles folder
styles.css
defines global HTML element styles