
Project Documentation: PasteHub

1. Project Overview

PasteHub is a web application designed for creating, managing, and sharing text snippets, notes, or code. It functions like a personal pastebin, allowing users to save their text pastes, view them later, edit them, and share them with others. The application's state is managed using Redux Toolkit and all data is persisted in the browser's Local Storage, ensuring that user data is saved across sessions.

---

2. Technology Stack

- **Frontend Framework:** React 18
- **Build Tool:** Vite
- **Styling:** Tailwind CSS
- **Routing:** React Router DOM
- **State Management:** Redux Toolkit
- **UI Components:**
    - **Icons:** Lucide React
    - **Notifications:** Sonner (for toast messages)

---

3. Project Setup & Installation

This section details the steps taken to set up the project environment and install all necessary dependencies.

3.1. Initial Project Scaffolding

React Project Creation: The project was initialized using Vite for a fast and modern development experience.

```
npm create vite@latest pastehub -- --template react
```

```
cd pastehub
```

3.2. Styling and Routing

Tailwind CSS Installation: Tailwind was installed for utility-first styling.
Bash
1. Install Tailwind and its peer dependencies

npm install -D tailwindcss postcss autoprefixer

2. Generate config files

npx tailwindcss init -p

React Router Installation: This library was added to handle client-side navigation.
Bash
npm i react-router-dom

3.3. UI Enhancement Dependencies

UI Libraries: After the initial setup, the UI was upgraded, which required the installation of new libraries for icons and notifications.

npm install sonner lucide-react

3.4. Configuration

- **Tailwind Configuration (`tailwind.config.js`)**: The config file was updated to define a custom design system, including colors, animations, and shadows required for the new theme.
- **Global Styles (`index.css`)**: The main CSS file was updated to include Tailwind's base directives and define CSS variables for light/dark mode theming.

---

4. Architecture & Core Concepts

4.1. State Management (Redux Toolkit & Local Storage)

- **Redux Store**: A central Redux store (`store.js`) was configured to manage the application's global state.
- **Paste Slice (`pasteSlice.js`)**: All logic related to managing pastes (adding, updating, deleting) is encapsulated within this slice.
- **Local Storage Persistence**: The application's state is synchronized with the browser's Local Storage.

Initialization: On application load, the initial state for pastes is read from Local Storage. If no data exists, it defaults to an empty array.

```
JavaScript
// In pasteSlice.js

initialState: {

  pastes: localStorage.getItem("pastes")

    ? JSON.parse(localStorage.getItem("pastes"))

    : []

}
```

- **Synchronization**: After any state modification (add, update, delete), the `pasteSlice` updates Local Storage to ensure data persistence.

4.2. Routing (React Router DOM)

The application uses `createBrowserRouter` to define its routes:

- `/` → **Home Page**: Serves as the main interface for both creating new pastes and editing existing ones.
- `/pastes` → **Pastes List Page**: Displays all saved pastes with options to manage them.
- `/pastes/:id` → **View Paste Page**: A read-only page to view the full content of a specific paste.

---

5. Component Breakdown & Functionality

5.1. `Home.jsx` (Create & Edit)

- **Dual Functionality**: This component cleverly handles two different use cases based on the URL.
  - If the URL is `/`, it's in "Create" mode.
  - If the URL has a search parameter (`/?pasteID=...`), it switches to "Edit" mode.
- **Data Handling**: It uses the `useSearchParams` hook to read the `pasteID` from the URL.
- **Conditional UI**: The page title and button text change dynamically (`Create Paste` vs. `Update Paste`).
- **Form Population**: In "Edit" mode, a `useEffect` hook fetches the corresponding paste data from the Redux store and populates the input fields.
- **Dispatching Actions**: When the user clicks the button, the `createpaste` function dispatches either `addToPastes` or `updateToPastes` to the Redux store. Afterwards, it clears the form and the URL parameters.

5.2. `Paste.jsx` (List View)

- **Data Fetching**: Uses the `useSelector` hook to get the list of all pastes from the Redux store.
- **Search/Filter**: Implements client-side filtering that allows users to search for pastes by title in real-time.
- **Actions per Paste**: Each item in the list has several actions:
    - **Edit**: A `Link` that navigates to the Home page with the `pasteID` as a search parameter.
    - **View**: A `Link` that navigates to the dynamic `/pastes/:id` route.
    - **Delete**: Calls a `handleDelete` function that dispatches the `removeFromPastes` action.
    - **Copy & Share**: Use the Browser's Clipboard API (`navigator.clipboard.writeText`) to copy content or the shareable URL.

5.3. `ViewPaste.jsx` (Read-Only View)

- **Dynamic Route**: Uses the `useParams` hook to get the `:id` from the URL.
- **Data Fetching**: Finds the specific paste from the Redux store. It includes a crucial check to handle cases where the paste is not found, preventing the app from crashing.
- **Read-Only UI**: Renders the paste title and content in disabled form fields, preventing edits. It provides "Copy" and "Share" functionality for user convenience.

Interview Preparation Guide

This section contains potential interview questions about this project and how you can answer them.

Q1: Can you give me a high-level overview of your PasteHub project?

Answer: "PasteHub is a client-side CRUD application I built using React and Vite. It allows users to create, read, update, and delete text pastes. For state management, I used Redux Toolkit, which provides a centralized store for all the pastes. A key feature is data persistence—I synchronized the Redux state with the browser's Local Storage so that a user's data is saved even after they close the tab. The entire UI is modern and responsive, built with Tailwind CSS, and routing is handled by React Router DOM."

Q2: Why did you choose Redux Toolkit for state management?

Answer: "For this project, the application state (the list of pastes) needed to be accessible across multiple components—the creation form, the list view, and the single view page. Redux Toolkit provides a single source of truth, which simplifies state management compared to prop drilling. Its `createSlice` function also reduces boilerplate by automatically generating action creators and reducers. Since I needed to perform actions like adding, updating, and deleting from a central list, Redux was a natural fit."

Q3: How did you implement the "Edit" functionality?

Answer: "That was an interesting challenge. The `Home` component serves for both creating and editing. When a user clicks 'Edit' on a paste, I use React Router's `<Link>` component to navigate back to the home route but with a URL search parameter, like `/?pasteID=123`.

Inside the `Home` component, I use the `useSearchParams` hook to check for this `pasteID`. If it exists, I know the user is in 'Edit' mode. I then use a `useEffect` hook that triggers whenever the `pasteID` changes. This effect finds the corresponding paste data in the Redux store and uses `useState` setters to populate the form fields with the existing title and content. When the user saves, the same component dispatches an `updateToPastes` action instead of the `addToPastes` action."

Q4: What was a technical challenge you faced?

Answer: "A subtle but important challenge was ensuring the application was robust. Initially, if a user manually typed a URL to view a paste that didn't exist (e.g., `/pastes/invalid-id`), the application would crash because it was trying to access properties of an `undefined` object.

To solve this, in the `ViewPaste` component, after trying to find the paste in the Redux store, I added a simple but crucial conditional check: `if (!paste)`. If the paste is not found, instead of trying to render the main component, I return a user-friendly 'Paste not found' message. This makes the application more resilient to bad URLs or deleted data."

Q5: If you had more time, what features would you add?

Answer: "There are a few features I'd love to add.

1. **Backend Integration**: Currently, it's a client-side app. I would connect it to a Node.js/Express backend and a database like MongoDB or PostgreSQL to allow users to save their pastes permanently and access them from different devices.
2. **User Authentication**: With a backend, I could add user accounts. This would allow each user to have their own private set of pastes.
3. **Dark Mode Toggle**: The CSS is already set up with variables for a dark theme. I would implement a simple UI toggle to allow users to switch between light and dark modes.
4. **Syntax Highlighting**: Since this can be used for code snippets, I would integrate a library like `highlight.js` to automatically apply syntax highlighting in the view page."