Project Overview

This project implements a rich text editor using the technologies React JS and Tailwind CSS for styling, allowing users to format text with options like bold, italic, underline, strikethrough, and various heading levels. The editor supports keyboard shortcuts, content persistence via localStorage, can be copy pasted while persisting the styling and accessibility features.

Architecture Decisions

Component Structure

- **`TextEditorContainer`**: The main container component that holds the editor's state and logic.
- **`Toolbar`**: Contains formatting options and interacts with the editor.
- **`ContentBox`**: The editable area where users input and format text.

State Management

- **Local Component State**: Managed using React's `useState` hook for:
    - `content`: Stores the HTML content of the editor.
    - `clicked`: Tracks active formatting options.
    - `dropDownValue`: Stores the selected value from the dropdown menu.
- **Persistent Storage**: Utilizes `localStorage` to save and retrieve content, ensuring data persists across sessions.

Event Handling

- **Clipboard Events**: Custom paste handling to insert HTML or plain text content appropriately.

Accessibility Considerations

Ensuring accessibility is crucial for inclusivity. The following practices have been implemented:

- **Semantic HTML Elements**: Used appropriate HTML tags like `<h1>`, `<h2>`, `<p>`, etc., to ensure content structure is meaningful.

- **Keyboard Navigation**: Ensured that all interactive elements are focusable and navigable via keyboard, including toolbar buttons and dropdowns.
- Lighthouse accessibility is at 100.

ARIA Roles and Labels: Added `aria-label` attributes to toolbar buttons for screen reader support. For example:

```
<button aria-label="Bold" onClick={() => handleFormat('bold')}>
  <strong>B</strong>
</button>
```

- **Color Contrast**: Ensured sufficient contrast ratios for text and background to accommodate users with visual impairments.

- **Focus Management**: Managed focus appropriately, especially after formatting actions, to enhance keyboard navigation.

Implementation Details

Handling Formatting Options

Formatting options like bold, italic, and headings are applied using the `document.execCommand` method.Even though `document.execCommand is declared deprecated, the latest two versions of the major browsers support it as there is no direct replacement for execCommand.`

For example, to apply bold formatting:

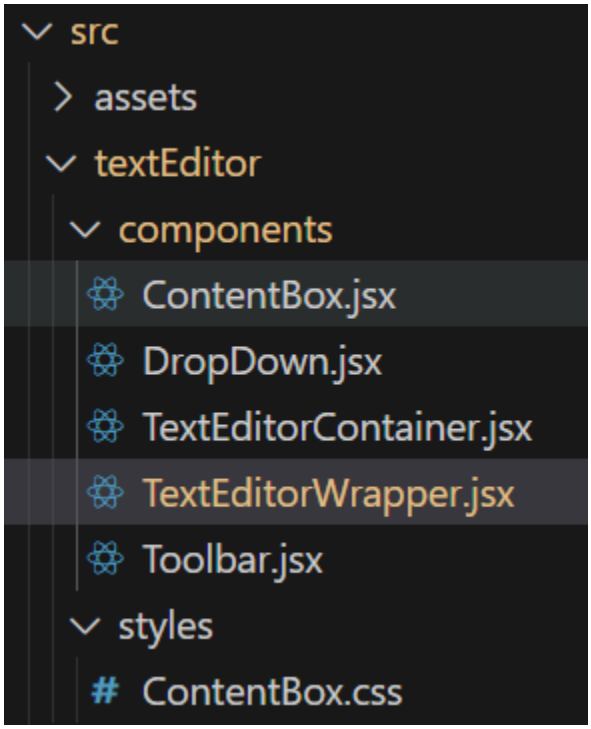
```
const handleFormat = (formatType) => {
  document.execCommand(formatType, false, null);
};
```

Dropdown for Heading Selection

A dropdown menu allows users to select heading levels. The selected value is applied to the selected text:

```
<select onChange={(e) => handleFormat(e.target.value)}>
  <option value="h1">Heading 1</option>
  <option value="h2">Heading 2</option>
  <option value="h3">Heading 3</option>
  <option value="h4">Heading 4</option>
</select>
```

Folder Structure



Setup and Installation

Clone the repository:

```
git clone https://github.com/yourusername/rich-text-editor.git
cd rich-text-editor
```

1. Install dependencies:
   `npm install`
2. Start the development server:
   `npm start`
3. Open your browser and navigate to `http://localhost:3000` to view the editor.

The application is deployed at https://text-editor-assignment-theta.vercel.app/