Documentation: Design Approach and Choices used in the code design

1. Skills

- Language used: JavaScript, HTML, CSS.
- Library : ReactJS

2. React Components Structure:

The application is structured using React components, enhancing a modular and reusable design approach. Here's a breakdown of the components:

- **App Component:** The main component of the application. It manages the state for columns, modal visibility, form data, and form errors. It renders the header, columns, and modal components.

- **Column Component:** Represents each column in the layout. It receives props such as column data, card data, and event handlers for drag-and-drop and card clicks.

- **Modal Component:** Responsible for rendering the modal for adding/editing cards. It receives props for modal visibility, form data, input change handlers, form submission, card deletion, and form error handling.

3. State Management:

- **useState Hook:** Utilizes the useState hook for managing state within functional components, providing a correct and efficient way to handle state changes.

- **Columns State:** Manages the state of columns using an object where each key represents a column and its value is an array of cards.

- **Modal State:** Manages the visibility of the modal and form data for adding/editing cards.

- **Form Errors State:** Tracks form validation errors for title and description fields.

4. Form Handling and Validation:

- **handleInputChange Function:** Handles input changes in the modal form fields. Performs validation based on field name and updates form data accordingly.

- **Validation Functions:** “validateTitle” and “validateDescription” functions validate title and description inputs respectively. They update the form errors state based on validation rules.

- **handleSubmit Function:** Handles form submission. It triggers form validation, and if no errors are found, it adds/edit cards accordingly.

5. Drag and Drop Functionality:

- **HTML Drag and Drop:** Utilizes native drag-and-drop functionality for moving cards between columns.
- **handleDragStart Function:** Sets the dragged card's ID in the data transfer object during drag start.
- **handleDrop Function:** Handles dropping cards into target columns. It retrieves the card ID from the data transfer object and updates the columns state accordingly.

6. Styling:

- **CSS Style file:** Uses a separate CSS style file (styles.css) for better UI.
- **Flexbox and Grid Layouts:** Used Flexbox and CSS Grid for creating responsive and visually appealing layouts.
- **Modal Styling:** The modal is styled to appear as a centered popup with an overlay for better user experience.

7. Dependency Management:

- **Package.json:** To manage project dependencies and scripts for development and production.
- **React and React-DOM:** Core libraries for building the user interface.
- **React-Scripts:** Provides development and build scripts for React applications.
- **UUID:** Used for generating unique IDs for cards.

8. Development Tools:

- **React StrictMode:** Wraps the App component with “StrictMode” to highlight potential issues and enable future-proofing.

9. Accessibility:

- **Close Button Accessibility:** The close button in the modal is accessible with keyboard navigation and screen readers, enhancing usability for all users.

10. Potential Improvements:

- **Code Optimization:** Review and optimize code for performance and readability, considering potential refactoring and code splitting where necessary.
- **Responsive Design:** Enhance responsiveness for various screen sizes and devices to provide a consistent user experience.

11. Deployment:

- **Deployment:** Used Netlify Client for deployment of website same link is provided in a submission form.

Conclusion:

The design approach and choices made in the development of this application prioritize modularity, simplicity, and user experience. By leveraging React's component-based architecture, state management hooks, the application provides a seamless interface for managing column-based lists with drag and drop functionality. Continuous improvement, testing, and deployment strategies can further enhance the application's robustness and usability.