Subject Name: **Front End Engineering**

Subject Code: CS186

Cluster: iGammaGroup: **G19**

Department: **DCSE**



|  |  |  |
| --- | --- | --- |
| **Submitted By:**  Vanshika Bhatia  2110992260  G19 |  | **Submitted To:**  Ms. Pritpal Kaur |

Project Name: Task Tracker

**React To-Do List App: Task Tracker**

**Introduction**

Welcome to the documentation of our React-based App. The provided code and reports describe a set of React components that, when combined, form the basis for a simple to-do list web application.

Our App is designed to help users capture and organize their thoughts, ideas, and important notes in a simple and intuitive manner. It offers a user-friendly interface and a range of features for managing notes efficiently.

**Technologies Used:**

* HTML
* CSS
* JavaScript
* React

**Key Components**

**1**. **Todo Component:** This component is responsible for rendering individual tasks in the to-do list. It allows users to view, complete, edit, and delete tasks.

**2. TodoForm Component:** The `TodoForm` component provides a form for adding new tasks to the to-do list. Users can input task descriptions and click the "Add Task" button to add them to the list.

**3. EditTodoForm Component:** This component is used to edit the descriptions of existing tasks. When a user decides to edit a task, this form allows them to make changes and update the task description.

**4. TodoWrapper Component:** The `TodoWrapper` component serves as a central hub for managing tasks. It includes the ability to add, delete, complete, and edit tasks. It displays the list of tasks and facilitates interactions with them.

**5. App Component**: The `App` component is the root component of the application. It initializes the application's structure and renders the `TodoWrapper` component, which manages the to-do list. The `App` component is essentially the starting point for the entire application.

**Usage**

In summary, this set of components, when used together, creates a basic to-do list web application. Users can add, delete, complete, and edit tasks within this application, making it a useful tool for task management.

-------------------------------**Components Documentation**---------------------------------

**Todo.js**

**I. Introduction**

This report aims to provide a comprehensive overview of the React Todo component, which is designed to manage tasks within a web application. The component is an integral part of the application's user interface and is responsible for rendering and managing individual tasks, offering functionalities such as completing, editing, and deleting tasks. Below, we will break down the key aspects of the component's implementation.

**II. Component Structure**

The React Todo component is structured as follows:



**III. Component Features**

The React Todo component provides the following features:

1. Rendering Tasks: The component receives a `task` object as a prop and displays the task's description. The task's completion status is reflected in the CSS class used for styling.

2. Complete Task: By clicking on a task, the user can toggle its completion status. This interaction is achieved by calling the `toggleComplete` function provided as a prop, passing the `task.id` as an argument.

3. Edit Task: Users can initiate the task editing process by clicking on the edit icon. The `editTodo` function is invoked with the `task.id` as an argument, enabling task modification.

4. Delete Task: The delete icon allows users to remove a task. When clicked, the `deleteTodo` function is called with the `task.id` as a parameter to facilitate task deletion.

**IV. Component Styling**

The component applies dynamic styling to represent completed and incomplete tasks, as well as the associated icons for editing and deleting. The use of Font Awesome icons enhances the user interface.

**V. Usage**

To use the React Todo component, it must be imported and included within the larger application's render structure. It should be passed the necessary props, including `task`, `deleteTodo`, `editTodo`, and `toggleComplete` functions.

Example usage:



**TodoForm.js**

**I. Introduction**

This report provides an in-depth analysis of the React TodoForm component. The TodoForm component is an essential element of a React application, responsible for capturing and adding new tasks to the todo list. The report will break down its structure, functionality, and usage.

**II. Component Structure**

The React TodoForm component is structured as follows:

****

**III. Component Features**

The React TodoForm component provides the following features:

1. Input Field: The component renders an input field where users can type the description of a new task.

2. Form Submission: When the user submits the form (by clicking the "Add Task" button or pressing Enter), the `handleSubmit` function is called. It prevents the default form submission action, checks if the input field is not empty, and then calls the `addTodo` function with the `value` of the input field as an argument. This allows the addition of a new task to the todo list.

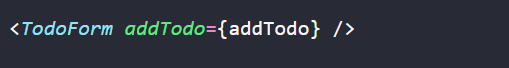
3. Clearing Input: After a task is successfully added, the input field is cleared, ready for the user to add another task.

**IV. Component Styling**

The component is styled with CSS classes to ensure a visually appealing and user-friendly interface. It features an input field and a button, both with designated CSS classes for customization and consistency.

**V. Usage**

To use the React TodoForm component, it must be imported and integrated into the application where tasks need to be added. The `addTodo` function should be passed as a prop to the component.



**TodoWrapper.js**

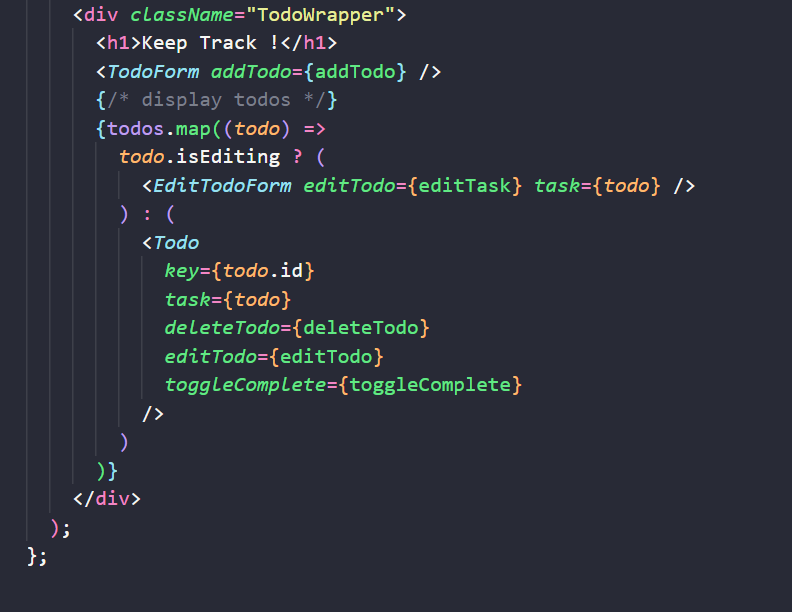
**I. Introduction**

This report provides a detailed analysis of the React TodoWrapper component, which serves as a central hub for managing tasks within a React-based todo list application. The component orchestrates the addition, deletion, completion, editing, and rendering of tasks, making it a crucial component of the application. The report will break down its structure, functionality, and usage.

**II. Component Structure**

The React TodoWrapper component is structured as follows:





**III. Component Features**

The React TodoWrapper component offers the following features:

1. Task Management: It maintains a list of tasks (`todos`) in its state, allowing users to add, delete, complete, and edit tasks.

2. Adding Tasks: The `addTodo` function is responsible for adding new tasks to the `todos` array. It generates a unique `id` for each task using the `uuidv4` library, sets the task's description, initializes it as incomplete, and marks it as not being edited.

3. Deleting Tasks: The `deleteTodo` function filters the `todos` array to remove the task with the specified `id`.

4. Completing Tasks: The `toggleComplete` function toggles the completion status of a task by mapping through the `todos` array and changing the `completed` property of the corresponding task.

5. Editing Tasks: The `editTodo` function allows users to enter edit mode for a task. When in edit mode, the task description can be changed.

6. Editing Task Description: The `editTask` function saves the edited task description, and the task is no longer in edit mode.

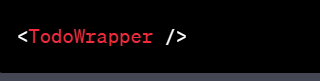
**IV. Component Styling**

The component is styled with CSS classes to ensure a visually appealing and user-friendly interface. It features a header, a form for adding tasks, and a list of tasks that can be in either display or edit mode.

**V. Usage**

The React TodoWrapper component is typically used as a container for managing tasks in a todo list application. It should be included within the larger application's structure, and the necessary functions (`addTodo`, `deleteTodo`, `toggleComplete`, `editTodo`, and `editTask`) should be passed as props to enable task management.

Example usage:



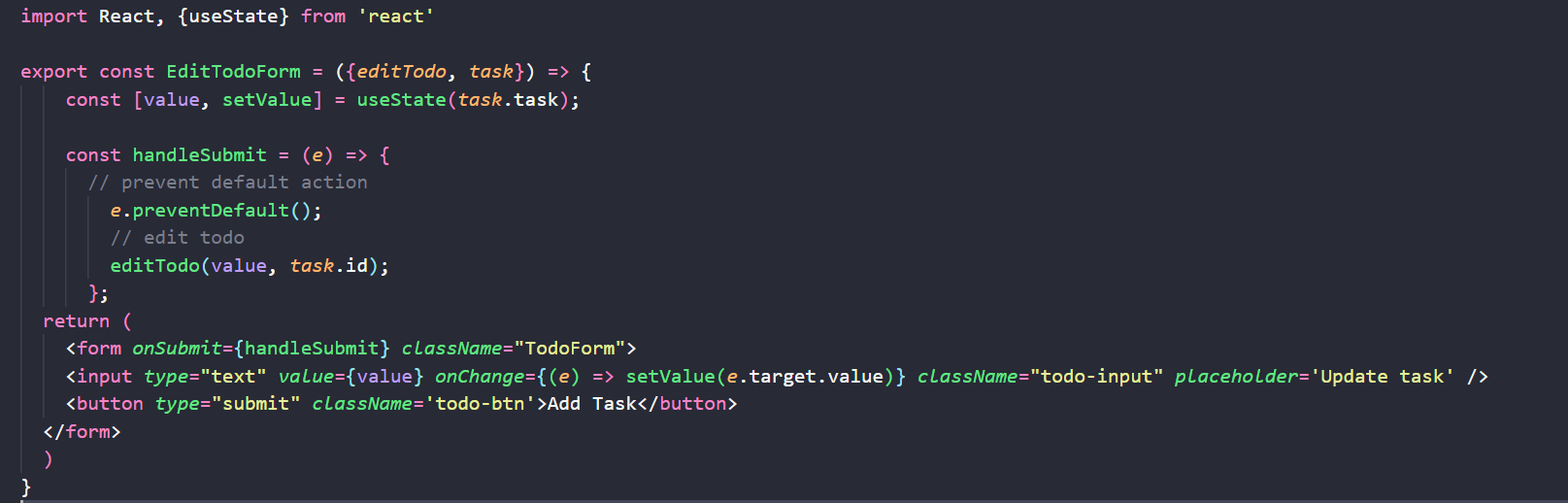
**EditTodoForm.js**

**I. Introduction**

This report provides an in-depth analysis of the React EditTodoForm component. The EditTodoForm component is a crucial part of a React-based todo list application, allowing users to edit the description of existing tasks. The report will break down its structure, functionality, and usage.

**II. Component Structure**

The React EditTodoForm component is structured as follows:

****

**III. Component Features**

The React EditTodoForm component provides the following features:

1. Input Field : The component renders an input field where users can edit the description of an existing task. The initial value of the input field is set to the current task description.

2. Form Submission : When the user submits the form (by clicking the "Update Task" button or pressing Enter), the `handleSubmit` function is called. It prevents the default form submission action and invokes the `editTodo` function with the updated `value` of the input field and the `task.id`. This allows users to update the task description.

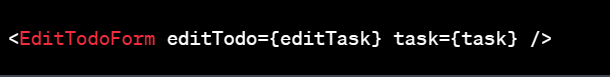
**IV. Component Styling**

The component is styled with CSS classes to ensure a visually appealing and user-friendly interface. It features an input field and a button, both with designated CSS classes for customization and consistency.

**V. Usage**

The React EditTodoForm component should be used when a user decides to edit an existing task within the application. It should be imported and rendered within the application's structure. The `editTodo` function should be passed as a prop to enable task editing.

Example usage:



**App.js**

**1. Component Structure :** The `App` component is structured as follows:



**2. Features:**

-The `App` component imports the application's CSS styles, which are defined in the ' App.css' file.

- It renders the `TodoWrapper` component, which acts as the central hub for managing tasks in the todo list application.

**3. Usage:**

- The `App` component is the root component of the application and serves as the container for the entire todo list functionality.

- It integrates the `TodoWrapper` component, allowing users to add, delete, complete, and edit tasks.

In summary, the `App` component is responsible for rendering the top-level structure of the application and initializing the main component, `TodoWrapper`, which handles task management. It sets the stage for users to interact with the todo list features provided by the application.

**Index.js**

**1. Imports :**

- The application imports the necessary modules: `React`, `ReactDOM`, and the main application component, `App`.

**2. Create Root for Rendering :**

- The code uses `ReactDOM.createRoot()` to create a root for rendering the React application. This method is part of React Concurrent Mode, which allows for asynchronous rendering and better performance.

**3. Rendering :**

- The `root` object is used to render the application by calling the `render` method.

- The main application component, `App`, is wrapped in `React.StrictMode` to enable certain development features and ensure the application is running correctly and efficiently in development mode.

**4. Target DOM Element :**

- The `root` is rendered into an HTML element with the ID "root" using the `document.getElementById("root")` method. This element typically represents the root of the application's DOM structure.

In summary, this code initializes a React application by creating a root for rendering and rendering the `App` component within the specified HTML element with the ID "root." The use of `React.StrictMode` helps ensure that the application follows best practices and performs optimally during development.

