Subject Name: **Front End Engineering**

Subject Code: **CS186**

Cluster : **iGamma**

Department : **DCSE**

Group:**G19**

Project Name: ToDo list Application

Submitted By:

Nawed Alam

2110992080

G19

Submitted To:

Ms. Pritpal Kaur

INTRODUCTION

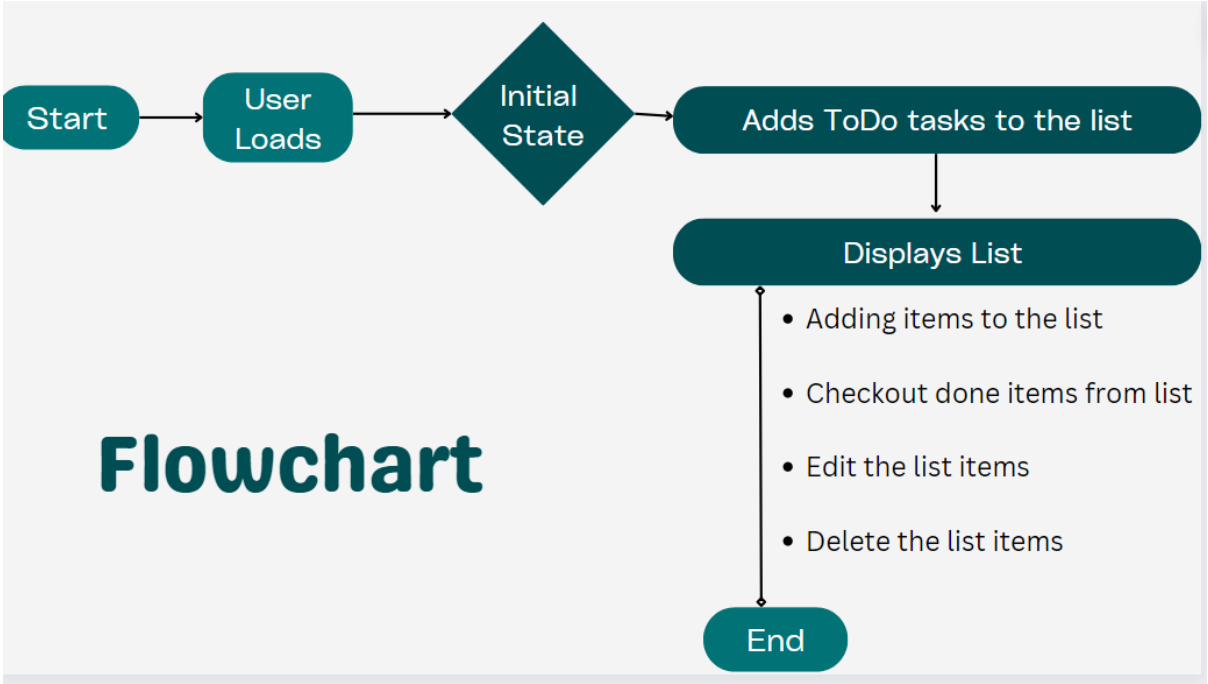
This ToDo Application is a web-based task management system built using the ReactJS framework. It helps users organize and prioritize their tasks, set deadlines, and ensure that nothing important slips through the cracks. ReactJS, known for its component-based architecture and performance optimizations, provides a solid foundation for creating a responsive and user-friendly ToDo application.

Key features

- Task Management: Users can add, edit, and delete tasks. Each task can have a title, description, and due date.
- Task Categories: Users can organize tasks into categories or projects, making it easy to group related tasks together.
- Priority Settings: Tasks can be assigned different priority levels, such as high, medium, or low, helping users identify their most important commitments.
- Task Status: Tasks can be marked as 'completed,' allowing users to keep track of their progress and accomplishments.
- Responsive Design: The application is designed to work seamlessly on various devices, including desktops, tablets, and mobile phones.

Technologies used

- React Js
- HTML
- CSS
- JavaScript



Usage

1. User starts the application, there is an input field to write the items of the list, and then it gets add in the list.
2. User can check out the done items from the list by clicking on it and it gets cut from the list.
3. User can permanently delete the items from the list, by clicking the bin icon in the side of the list.
4. User can edit the items in the list by clicking on the item.

Index.js

```
import React from "react";
import ReactDOM from "react-dom/client";
import App from "./App";

const root = ReactDOM.createRoot(document.getElementById("root"));
root.render(
<React.StrictMode>
<App />
</React.StrictMode>
);
```

Import React: The React variable is imported from the "react" library. This import is necessary to use JSX and React features in your application.

Import ReactDOM: The ReactDOM variable is imported from the "react-dom/client" module. This is used for rendering your React application into the DOM.

Import App Component: The App component is imported from a file named "App." This is the main component of your React application, where your application's UI is defined.

Create a Root: ReactDOM.createRoot(document.getElementById("root")) is used to create a root element in the DOM. The document.getElementById("root") finds the HTML element with the "root" id, which is where your React application will be rendered.

Render the App: The root.render(...) method is called to render your application. It renders the App component within a <React.StrictMode>. <React.StrictMode> is used for highlighting potential problems in your application during the development phase. It does not affect the production build.

App.js

```
import './App.css';
import { TodoWrapper } from './components/TodoWrapper';

function App() {
  return (
    <div className="App">
      <TodoWrapper  />
    </div>
  );
}

export default App;
```

Todo.js

```
import React from 'react'
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome'
import { faPenToSquare } from '@fortawesome/free-solid-svg-icons'
import { faTrash } from '@fortawesome/free-solid-svg-icons'

export const Todo = ({task, deleteTodo, editTodo, toggleComplete}) => {
  return (
    <div className="Todo">
        <p className={`${task.completed ? "completed" : "incompleted"}`} onClick={() =>
toggleComplete(task.id)}>{task.task}</p>
        <div>
        <FontAwesomeIcon className="edit-icon" icon={faPenToSquare} onClick={() => editTodo(task.id)} />
        <FontAwesomeIcon className="delete-icon" icon={faTrash} onClick={() => deleteTodo(task.id)} />
        </div>
    </div>
  )
}
```

Imports: It imports the necessary dependencies, including React, FontAwesome icons for editing and deleting tasks, and functions passed as props (deleteTodo, editTodo, toggleComplete).

Functional Component: The Todo component is a functional component that takes props as an argument.

Rendering Task: It displays the task name, and the p element's class depends on whether the task is completed or not, applying the "completed" or "incompleted" class accordingly.

Icons: It includes two FontAwesome icons, one for editing and one for deleting a task. These icons trigger the respective functions (e.g., editTodo and deleteTodo) when clicked, passing the task's id to these functions.

TodoForm.js

```
import React, {useState} from 'react'

export const TodoForm = ({addTodo}) => {
    const [value, setValue] = useState('');

    const handleSubmit = (e) => {
      // prevent default action
        e.preventDefault();
        if (value) {
          // add todo
          addTodo(value);
          // clear form after submission
          setValue('');
        }
      };
  return (
    <form onSubmit={handleSubmit} className="TodoForm">
    <input type="text" value={value} onChange={(e) => setValue(e.target.value)} className="todo-input"
placeholder='What is the task today?' />
    <button type="submit" className='todo-btn'>Add Task</button>
  </form>
  )
}
```

Imports: It imports React and the useState hook for managing the component's state.

Functional Component: The TodoForm component is a functional component that takes a prop, addTodo, which is a function to add a new task.

State Management: It uses the useState hook to create a state variable, value, which represents the text input value for the new task.

Form Submission: When the form is submitted, the handleSubmit function is called, which:

Prevents the default form submission behavior.

Checks if the value is not empty.

If the value is not empty, it calls the addTodo function, passing the value as the task to add.

It then clears the input field by setting value to an empty string.

Form Elements: The component renders a form with an input field for entering task text and a "Add Task" button. The value of the input field is controlled by the state, and its changes are tracked by the onChange handler, updating the value state as the user types.

TodoWrapper.js

```
import React, { useState } from "react";
import { Todo } from "./Todo";
import { TodoForm } from "./TodoForm";
import { v4 as uuidv4 } from "uuid";
import { EditTodoForm } from "./EditTodoForm";

export const TodoWrapper = () => {
  const [todos, setTodos] = useState([]);

  const addTodo = (todo) => {
    setTodos([
      ...todos,
      { id: uuidv4(), task: todo, completed: false, isEditing: false },
    ]);
  }

  const deleteTodo = (id) => setTodos(todos.filter((todo) => todo.id !== id));

  const toggleComplete = (id) => {
    setTodos(
      todos.map((todo) =>
        todo.id === id ? { ...todo, completed: !todo.completed } : todo
      )
    );
  }

  const editTodo = (id) => {
    setTodos(
      todos.map((todo) =>
        todo.id === id ? { ...todo, isEditing: !todo.isEditing } : todo
      )
    );
  }

  const editTask = (task, id) => {
    setTodos(
      todos.map((todo) =>
        todo.id === id ? { ...todo, task, isEditing: !todo.isEditing } : todo
      )
```

```
    );
  };

  return (
    <div className="TodoWrapper">
      <h1>Get Things Done !</h1>
      <TodoForm addTodo={addTodo} />
      {/* display todos */}
      {todos.map((todo) =>
        todo.isEditing ? (
          <EditTodoForm editTodo={editTask} task={todo} />
        ) : (
          <Todo
            key={todo.id}
            task={todo}
            deleteTodo={deleteTodo}
            editTodo={editTodo}
            toggleComplete={toggleComplete}
          />
        )
      )}
    </div>
  );
};
```

Imports: It imports React and various other components and libraries, including Todo, TodoForm, EditTodoForm, and uuidv4 for generating unique IDs.

Functional Component: The TodoWrapper component is a functional component that holds the state of the ToDo tasks and manages their manipulation.

State Management: It uses the useState hook to create a state variable, todos, which stores an array of task objects. Each task object has properties like id, task, completed, and isEditing.

Task Management Functions:

addTodo: Adds a new task to the todos array with a unique ID, the task content, and initial completion status.

deleteTodo: Removes a task from the todos array based on its id.

toggleComplete: Toggles the completion status of a task.

editTodo: Toggles the editing mode for a task.

editTask: Updates the task content when a task is being edited.

Rendering Components:

Displays a title "Get Things Done!" at the top of the component.

Renders the TodoForm component for adding new tasks.

Maps through the todos array and conditionally renders either the EditTodoForm or Todo component based on the isEditing property of each task object.

TodoWrapperLocalStorage.js

```
import React, {useState, useEffect} from 'react'
import { TodoForm } from './TodoForm'
import { v4 as uuidv4 } from 'uuid';
import { Todo } from './Todo';
import { EditTodoForm } from './EditTodoForm';
uuidv4();

export const TodoWrapperLocalStorage = () => {
    const [todos, setTodos] = useState([])

    useEffect(() => {
        const savedTodos = JSON.parse(localStorage.getItem('todos')) || [];
        setTodos(savedTodos);
    }, []);

    const addTodo = todo => {
        const newTodos = [...todos, {id: uuidv4(), task: todo, completed: false, isEditing: false}];
        setTodos(newTodos);
        localStorage.setItem('todos', JSON.stringify(newTodos));
    }

    const toggleComplete = id => {
        const newTodos = todos.map(todo => todo.id === id ? {...todo, completed: !todo.completed} : todo);
        setTodos(newTodos);
        localStorage.setItem('todos', JSON.stringify(newTodos));
    }

    const deleteTodo = id => {
        const newTodos = todos.filter(todo => todo.id !== id);
        setTodos(newTodos);
        localStorage.setItem('todos', JSON.stringify(newTodos));
    }

    const editTodo = id => {
        setTodos(todos.map(todo => todo.id === id ? {...todo, isEditing: !todo.isEditing} : todo))
    }

    const editTask = (task, id) => {
        const newTodos = todos.map(todo => todo.id === id ? {...todo, task, isEditing: !todo.isEditing} : todo);
        setTodos(newTodos);
        localStorage.setItem('todos', JSON.stringify(newTodos));
    }
  return (
    <div className='TodoWrapper'>
        <h1>Get Things Done!</h1>
        <TodoForm addTodo={addTodo} />
        {todos.map((todo, index) => (
            todo.isEditing ? (
                <EditTodoForm editTodo={editTask} task={todo} />
            ) : (
                <Todo task={todo} key={index} toggleComplete={toggleComplete} deleteTodo={deleteTodo}
editTodo={editTodo} />
            )

        ))}

    </div>
  )
}
```

Imports and Setup: It imports React, various components (TodoForm, Todo, EditTodoForm), and the uuidv4 library for generating unique IDs. It also initializes the todos state with useState.

Local Storage and State Initialization: It uses the useEffect hook to load saved ToDo tasks from local storage (localStorage) when the component mounts. If there are no saved tasks, an empty array is used as the default value.

Task Management Functions: It includes similar task management functions as the previous component (addTodo, toggleComplete, deleteTodo, editTodo, editTask). These functions update the state and local storage when tasks are added, edited, completed, or deleted.

Rendering Components: It renders a title, "Get Things Done!", and the TodoForm component for adding new tasks. It then maps through the todos array and conditionally renders either the EditTodoForm or Todo component based on the isEditing property of each task object.

Local Storage Sync: After any modification to the todos array, it updates the local storage to ensure data persistence.

EditTodoForm.js

```
import React, {useState} from 'react'

export const EditTodoForm = ({editTodo, task}) => {
    const [value, setValue] = useState(task.task);

    const handleSubmit = (e) => {
      // prevent default action
        e.preventDefault();
        // edit todo
        editTodo(value, task.id);
      };
  return (
    <form onSubmit={handleSubmit} className="TodoForm">
    <input type="text" value={value} onChange={(e) => setValue(e.target.value)} className="todo-input"
placeholder='Update task' />
    <button type="submit" className='todo-btn'>Add Task</button>
  </form>
  )
}
```

Imports: It imports React and uses the useState hook for managing component state.

Functional Component: The EditTodoForm component is a functional component that receives two props: editTodo (a function to edit a task) and task (the task being edited).

State Management: It initializes the value state with the current task's content, ensuring that the input field displays the existing task content initially.

Form Submission: When the form is submitted, the handleSubmit function is called, which:

Prevents the default form submission behavior.Calls the editTodo function, passing the updated value and the task's id.Form Elements: The component renders a form with an input field for updating the task content and a "Update Task" button. The value of the input field is controlled by the state, and its changes are tracked by the onChange handler, allowing users to edit the task content.

CONCLUSION

In summary, our ToDo List application is a testament to our commitment to delivering solutions that improve the lives of our users. With a solid foundation in ReactJS, state management, and user experience design, we're well-prepared to take on future challenges and continue enhancing this application. Thank you for your time, and we look forward to your feedback and suggestions for making our ToDo List app even better.