Taniya Nagdev **D15B / 50**

Experiment 2

Aim: Experiment based on React Hooks (useEffect, useContext, custom hooks)

Theory: React introduced **Hooks** in version 16.8 to allow state management and side effects in functional components without the need for class-based components. Hooks improve **code reusability, readability, and maintainability**.

1. **React Hooks Overview**

   - Hooks are JavaScript functions that let developers "hook into" React features such as state (`useState`), lifecycle (`useEffect`), or context (`useContext`).

   - They encourage writing applications in a **functional programming style**, reducing boilerplate and improving modularity.

2. **useEffect Hook**

   - The `useEffect` hook is used for handling **side effects** in React components.

   - Common use cases include:

     - Data fetching from APIs
     - Subscribing/unsubscribing to events
     - Interacting with browser storage (e.g., localStorage, sessionStorage)

   - In this experiment, `useEffect` monitors the **task list state** and automatically saves updates to **localStorage**, ensuring persistence across browser refreshes. This makes the To-Do List more reliable and user-friendly.

3. **useContext Hook**

   - `useContext` provides a way to **share data globally** without prop drilling (passing props manually through each level of the component tree).

   - It works with the **Context API**, where a global state (like theme, user authentication, or language preference) is defined and consumed by multiple components.

   - In this experiment, `useContext` is used to handle the **Theme Switcher**. Instead of passing theme-related props down to every child, the context allows the theme to be accessed by all components directly.

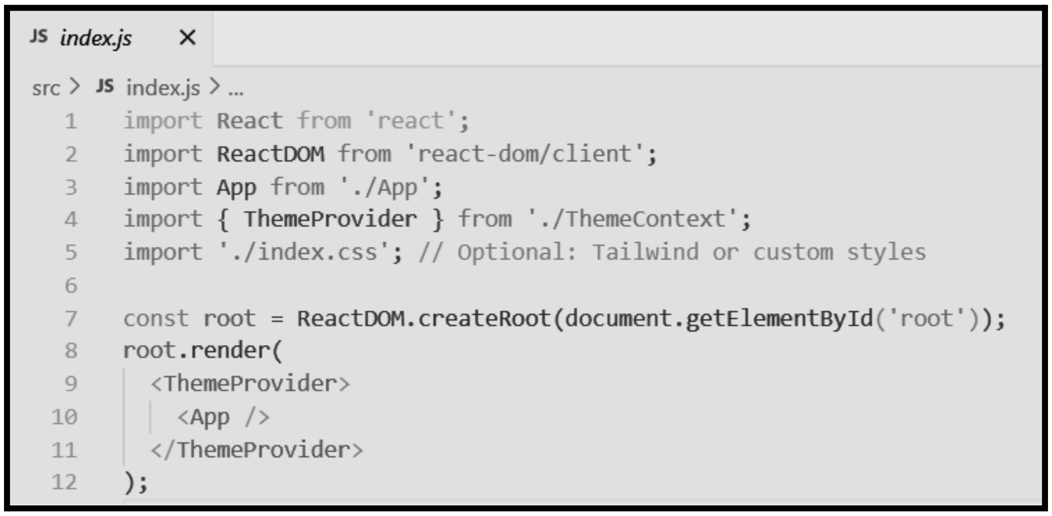
4. **Custom Hooks**
    - Custom Hooks are developer-defined functions that start with `use` and allow **encapsulation of reusable logic**.
    - They help remove repetitive code and keep components cleaner.
    - Example in this experiment:
        - A custom hook `useLocalStorage` is implemented to handle synchronization between component state and `localStorage`.
        - Instead of manually writing get/set logic every time, this hook provides a reusable abstraction.
5. **Advantages of Using Hooks in this Experiment**
    - **Cleaner Code**: Avoids class components and lifecycle methods like `componentDidMount`.
    - **Persistence**: Tasks remain even after refreshing the page.
    - **Global Theme Control**: Light/Dark theme applies instantly across the app.
    - **Reusability**: Custom hook (`useLocalStorage`) can be reused in other projects.

Extra:

- Local Storage - The website stores tasks using **localStorage**, so the data remains even after refreshing or reopening the browser.
- Toggle - A light and dark mode toggle was also added, allowing the user to switch between different themes.

Taniya Nagdev **D15B / 50**

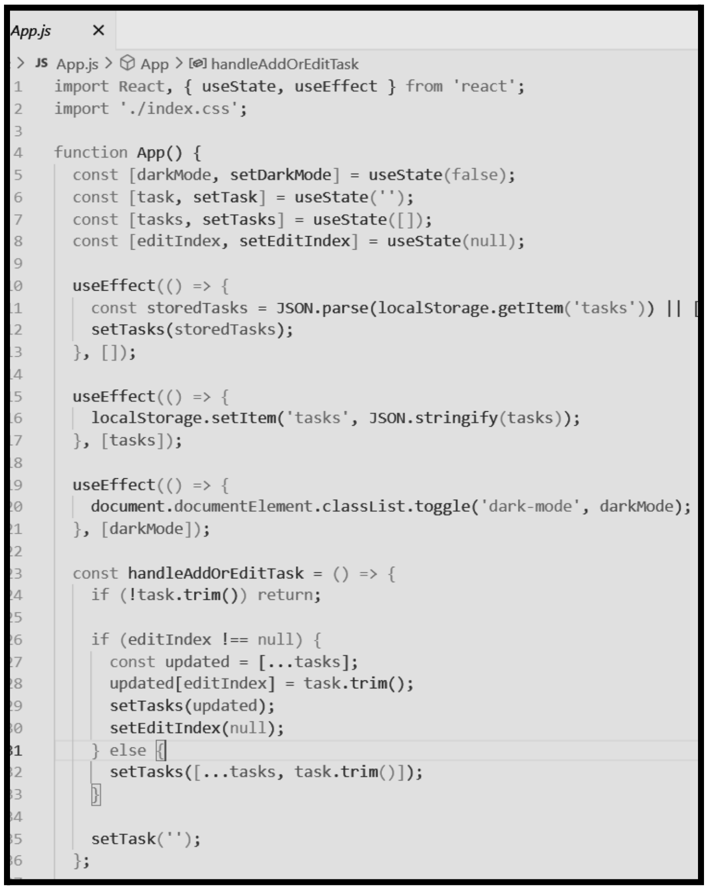
<u>Code</u>:



JS index.js ×

src > JS index.js > ...

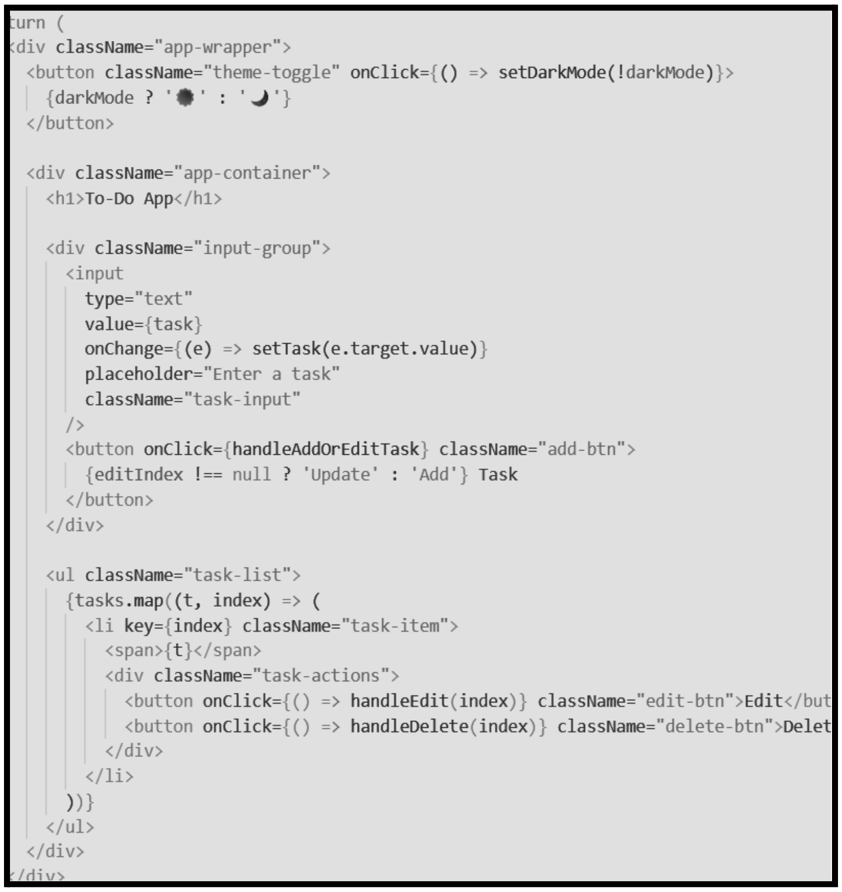
```js
import React from 'react';
import ReactDOM from 'react-dom/client';
import App from './App';
import { ThemeProvider } from './ThemeContext';
import './index.css'; // Optional: Tailwind or custom styles

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <ThemeProvider>
    <App />
  </ThemeProvider>
);
```



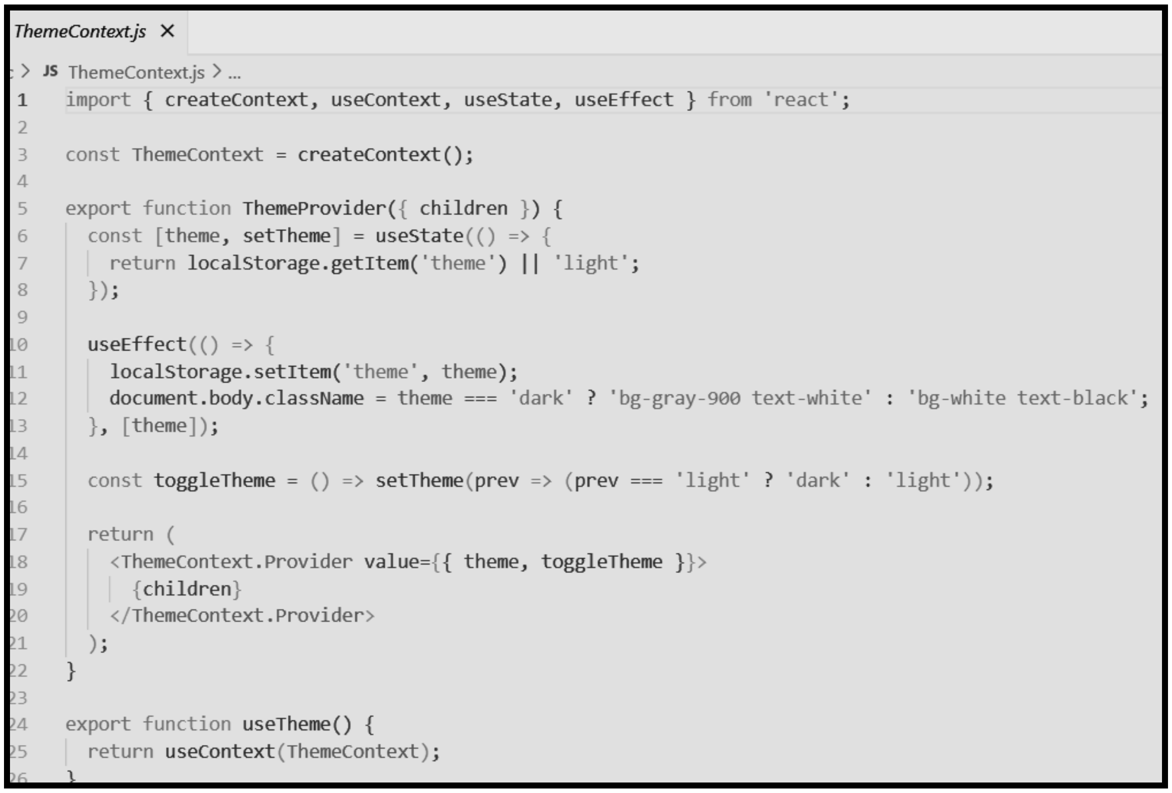
App.js ×

> JS App.js > App > handleAddOrEditTask

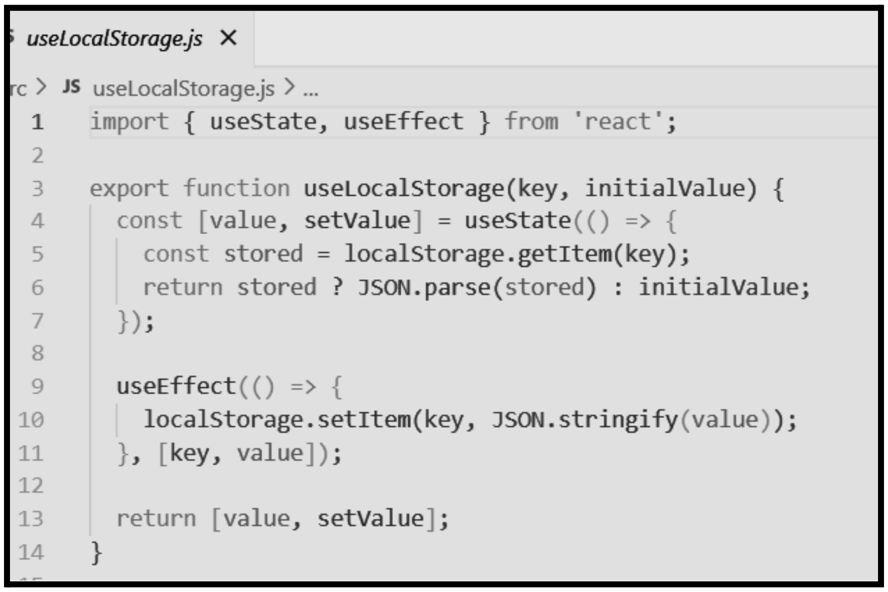
```js
import React, { useState, useEffect } from 'react';
import './index.css';

function App() {
  const [darkMode, setDarkMode] = useState(false);
  const [task, setTask] = useState('');
  const [tasks, setTasks] = useState([]);
  const [editIndex, setEditIndex] = useState(null);

  useEffect(() => {
    const storedTasks = JSON.parse(localStorage.getItem('tasks')) || [
    setTasks(storedTasks);
  }, []);

  useEffect(() => {
    localStorage.setItem('tasks', JSON.stringify(tasks));
  }, [tasks]);

  useEffect(() => {
    document.documentElement.classList.toggle('dark-mode', darkMode);
  }, [darkMode]);

  const handleAddOrEditTask = () => {
    if (!task.trim()) return;

    if (editIndex !== null) {
      const updated = [...tasks];
      updated[editIndex] = task.trim();
      setTasks(updated);
      setEditIndex(null);
    } else {
      setTasks([...tasks, task.trim()]);
    }

    setTask('');
  };
```



```jsx
turn (
<div className="app-wrapper">
  <button className="theme-toggle" onClick={() => setDarkMode(!darkMode)}>
    {darkMode ? '🌞' : '🌙'}
  </button>

  <div className="app-container">
    <h1>To-Do App</h1>

    <div className="input-group">
      <input
        type="text"
        value={task}
        onChange={(e) => setTask(e.target.value)}
        placeholder="Enter a task"
        className="task-input"
      />
      <button onClick={handleAddOrEditTask} className="add-btn">
        {editIndex !== null ? 'Update' : 'Add'} Task
      </button>
    </div>

    <ul className="task-list">
      {tasks.map((t, index) => (
        <li key={index} className="task-item">
          <span>{t}</span>
          <div className="task-actions">
            <button onClick={() => handleEdit(index)} className="edit-btn">Edit</but
            <button onClick={() => handleDelete(index)} className="delete-btn">Delet
          </div>
        </li>
      ))}
    </ul>
  </div>
</div>
```



```javascript
import { createContext, useContext, useState, useEffect } from 'react';

const ThemeContext = createContext();

export function ThemeProvider({ children }) {
  const [theme, setTheme] = useState(() => {
    return localStorage.getItem('theme') || 'light';
  });

  useEffect(() => {
    localStorage.setItem('theme', theme);
    document.body.className = theme === 'dark' ? 'bg-gray-900 text-white' : 'bg-white text-black';
  }, [theme]);

  const toggleTheme = () => setTheme(prev => (prev === 'light' ? 'dark' : 'light'));

  return (
    <ThemeContext.Provider value={{ theme, toggleTheme }}>
      {children}
    </ThemeContext.Provider>
  );
}

export function useTheme() {
  return useContext(ThemeContext);
```

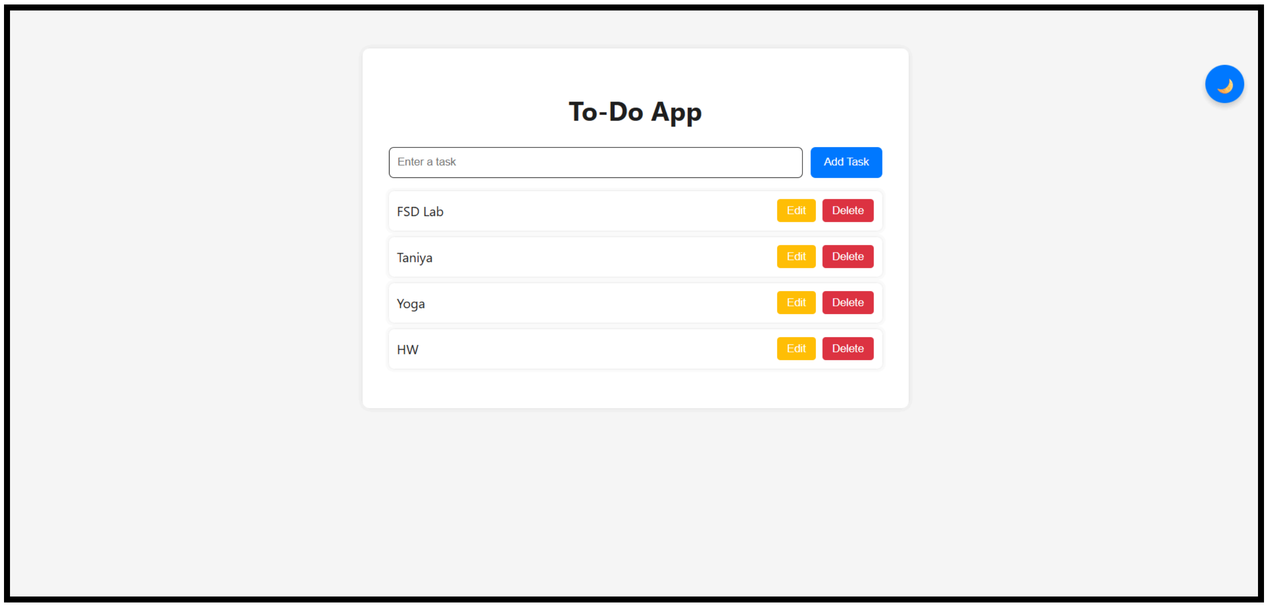


```javascript
import { useState, useEffect } from 'react';

export function useLocalStorage(key, initialValue) {
  const [value, setValue] = useState(() => {
    const stored = localStorage.getItem(key);
    return stored ? JSON.parse(stored) : initialValue;
  });

  useEffect(() => {
    localStorage.setItem(key, JSON.stringify(value));
  }, [key, value]);

  return [value, setValue];
}
```

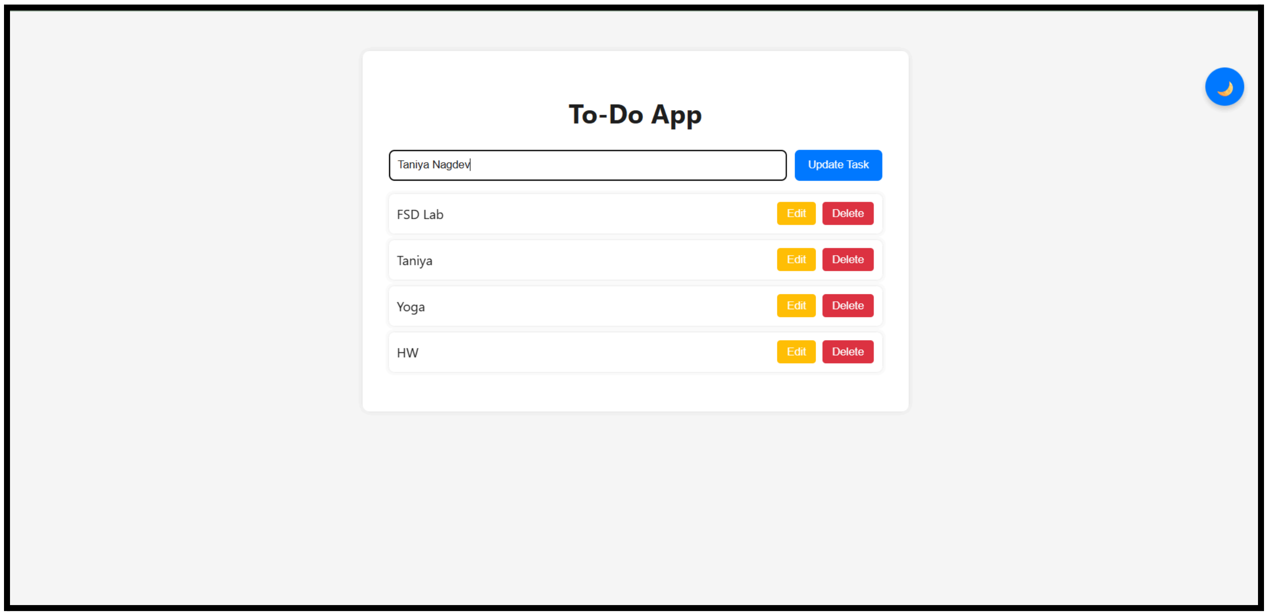
Taniya Nagdev **D15B / 50**

Implementation:

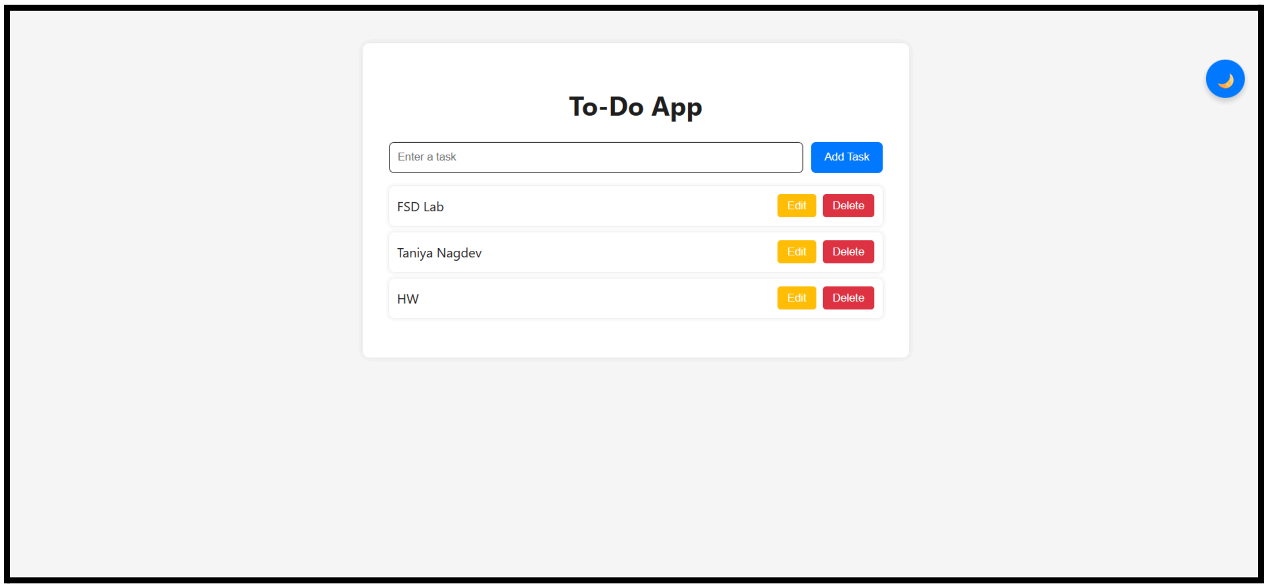
Add Task



Edit Task

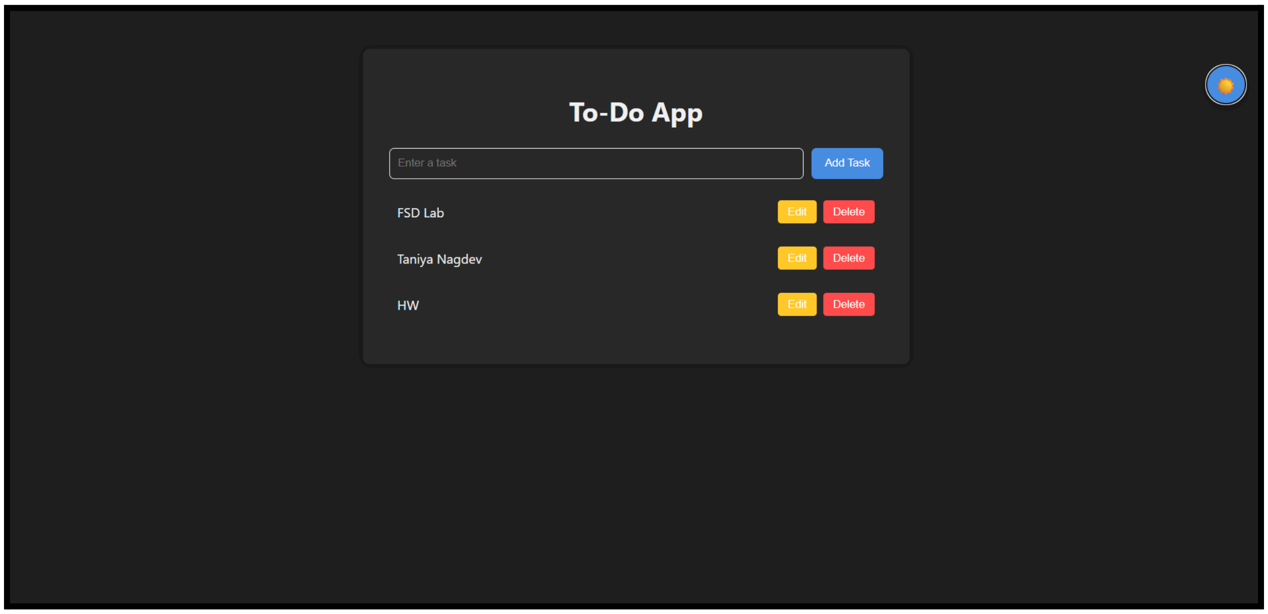


Delete Task



FSD Lab
Taniya Nagdev
HW

Dark Mode



FSD Lab
Taniya Nagdev
HW

Conclusion:

The To-Do List with Theme Switcher demonstrates the practical use of React Hooks to build modern, efficient, and user-friendly applications. By combining `useEffect`, `useContext`, and custom hooks, the app achieves persistent task storage, global theme management, and cleaner, reusable code.