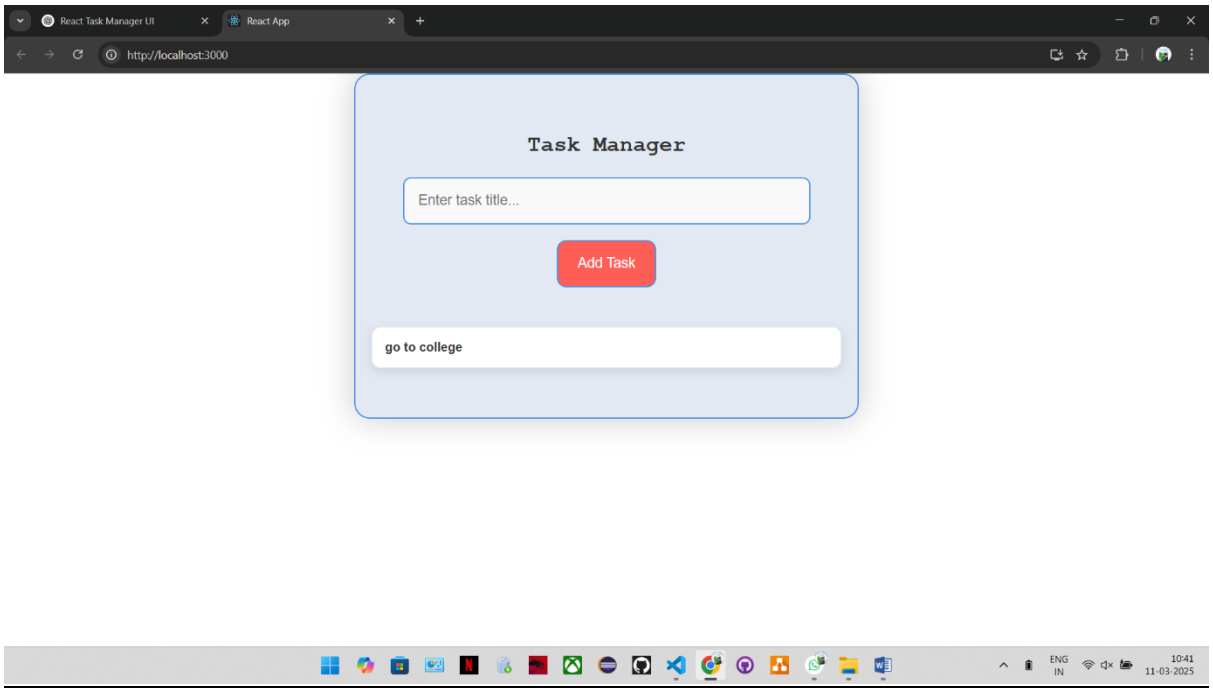
React Hooks Assignment

Name: - Pruthvi Gadhiya

Id no.:-500235241

Git-hub link:- https://github.com/Pruthvi1881/React-Hooks

Step-1:- useState



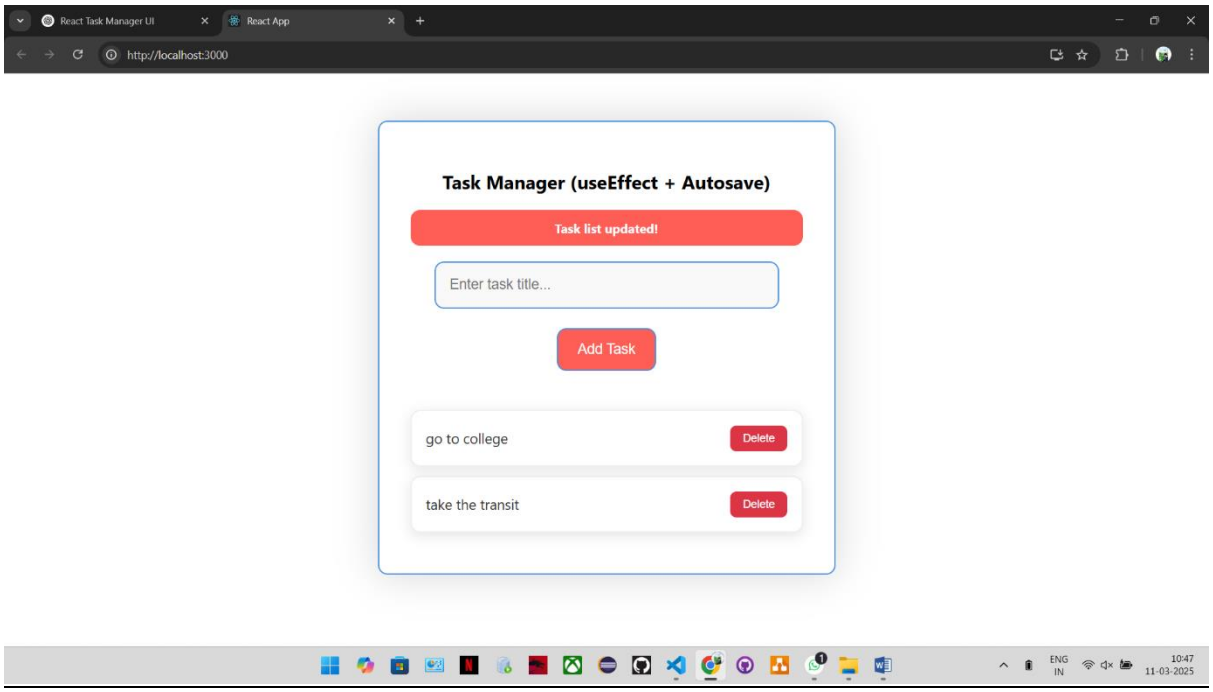
Explanation:

We need to create a form where users can enter a task with a title and description. We'll use useState to store the list of tasks.

Code Snippet:

```
src > JS App.js > TaskManager
1   import { useState } from "react";
2
3   function TaskManager() {
4     const [tasks, setTasks] = useState([]);
5
6     const addTask = (title, description) => {
7       setTasks([...tasks, { id: Date.now(), title, description, completed: false }]);
8     };
9
10    return (
11      <div>
12        <button onClick={() => addTask("New Task", "Task Description")}>Add Task</button>
13      </div>
14    );
15  }
16
17  export default TaskManager;
```

Step-2:- useEffect



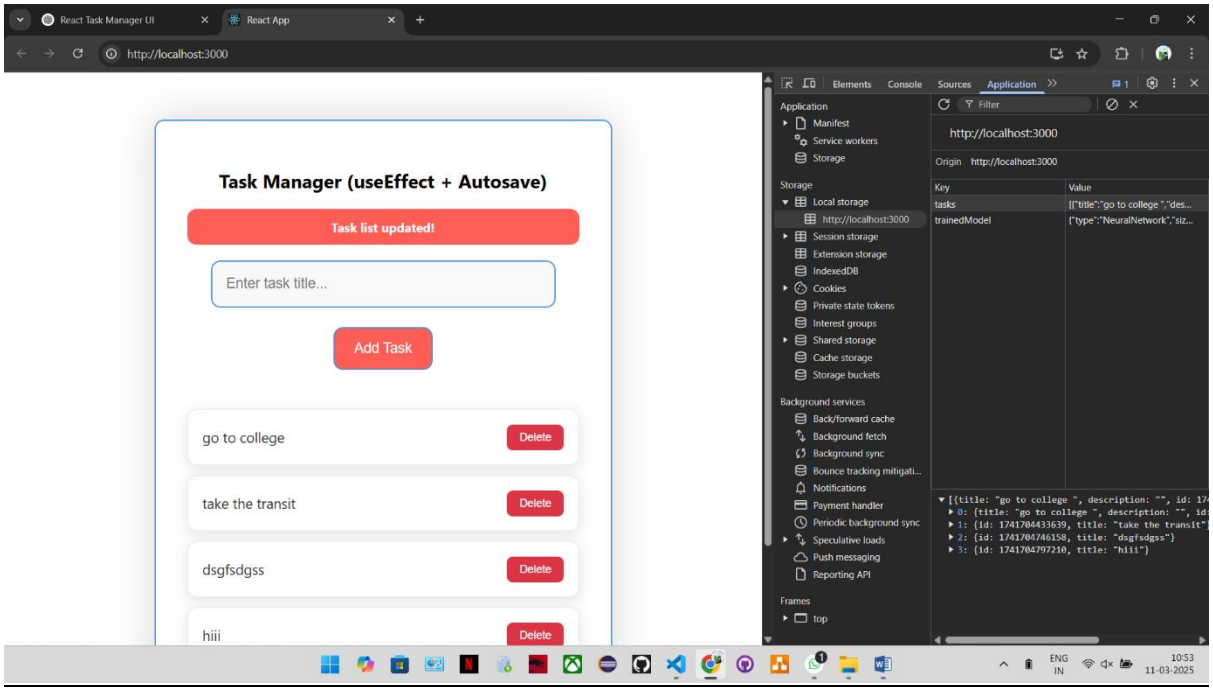
Explanation:

Whenever a task is added or removed, we should save the updated list to localStorage. Also, we'll show an alert whenever the task list changes.

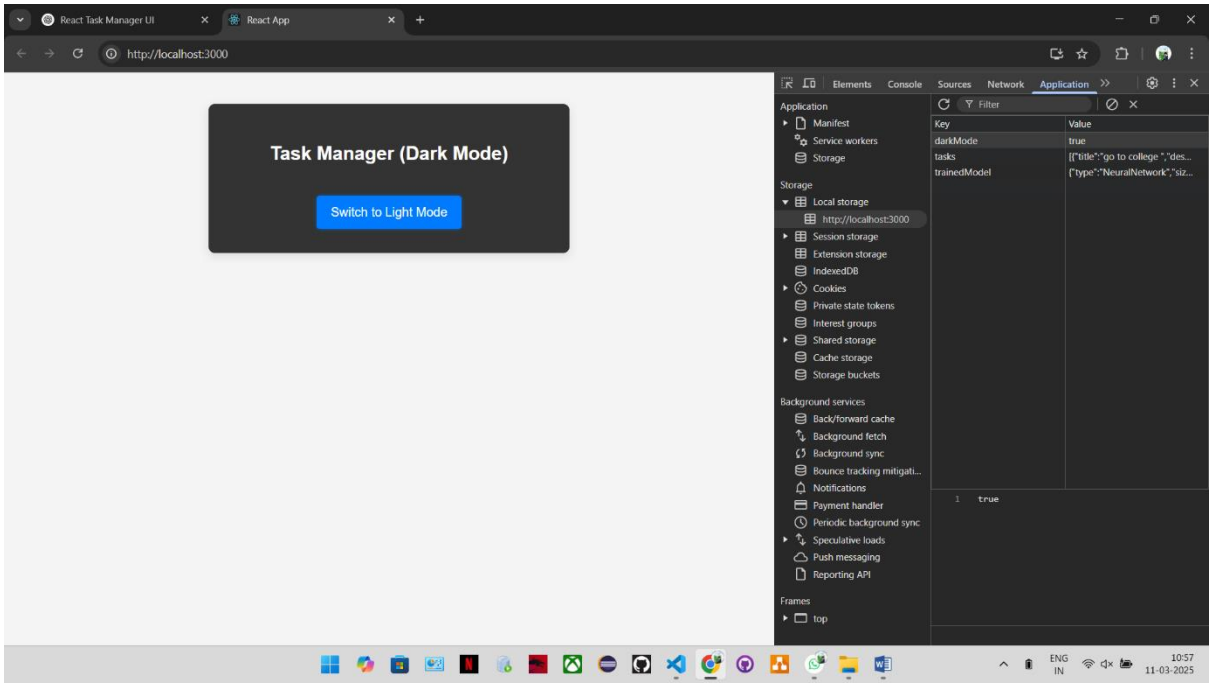
Code Snippet:

```
import { useState, useEffect } from "react";

function TaskManager() {
  const [tasks, setTasks] = useState(() => {
    return JSON.parse(localStorage.getItem("tasks")) || [];
  });

  useEffect(() => {
    localStorage.setItem("tasks", JSON.stringify(tasks));
    alert("Task list updated!");
  }, [tasks]);

  return <div>{tasks.length} tasks saved.</div>;
}

export default TaskManager;
```

Local storage:-



Step-3:- useContext



Explanation:

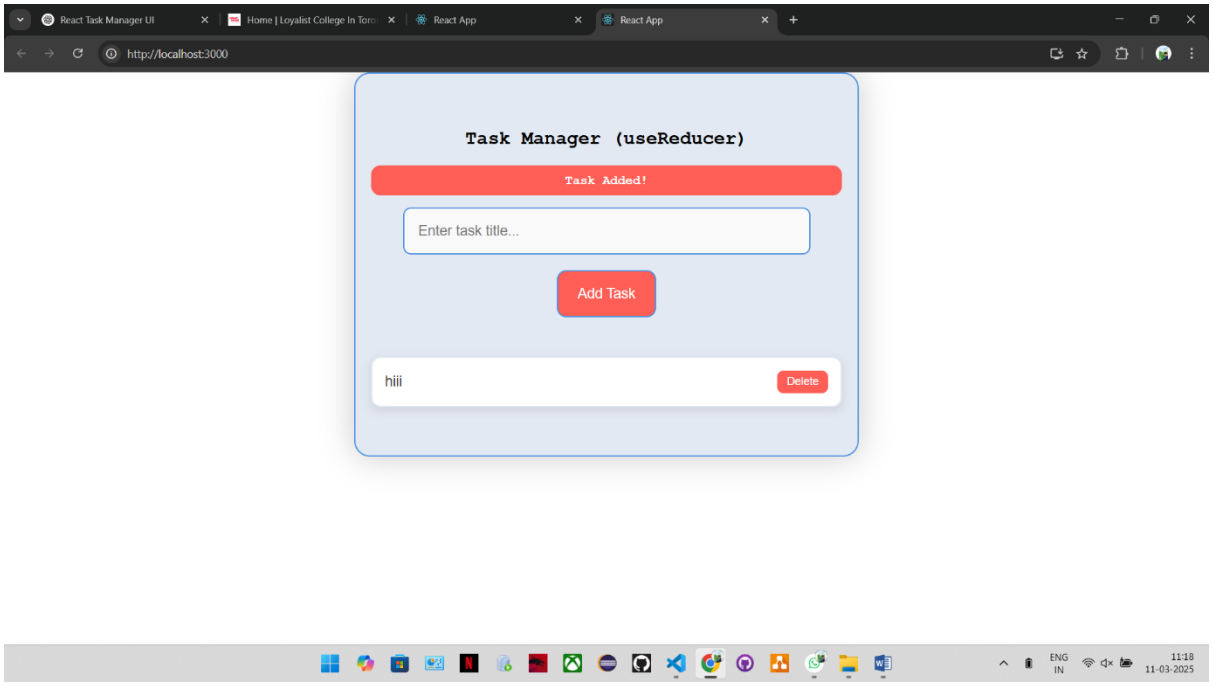
We'll create a ThemeContext to switch between dark and light mode. The theme should be remembered even after the page reloads.

Code Snippet:

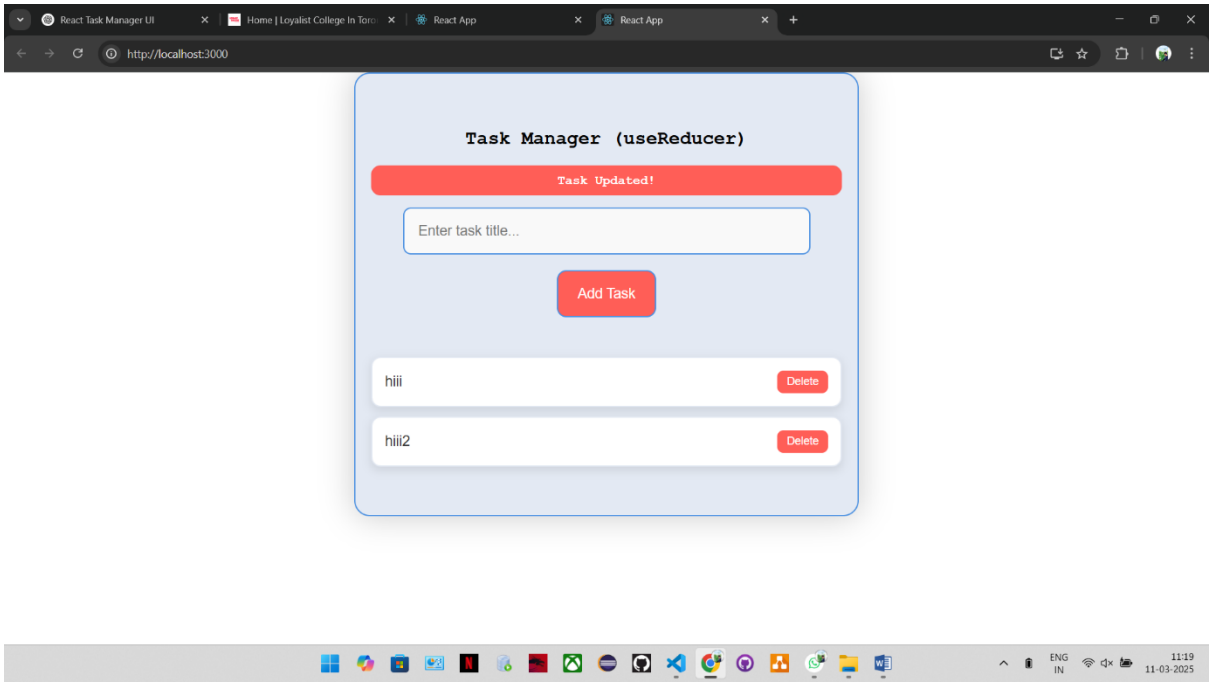
```jsx
import { createContext, useState, useContext } from "react";

const ThemeContext = createContext();

function ThemeProvider({ children }) {
  const [darkMode, setDarkMode] = useState(() => {
    return JSON.parse(localStorage.getItem("darkMode")) || false;
  });

  const toggleTheme = () => {
    setDarkMode((prev) => {
      localStorage.setItem("darkMode", JSON.stringify(!prev));
      return !prev;
    });
  };

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

function ThemeToggleButton() {
  const { darkMode, toggleTheme } = useContext(ThemeContext);
  return <button onClick={toggleTheme}>{darkMode ? "Light Mode" : "Dark Mode"}</button>;
}

export { ThemeProvider, ThemeToggleButton };
```

Step-4:- useReducer

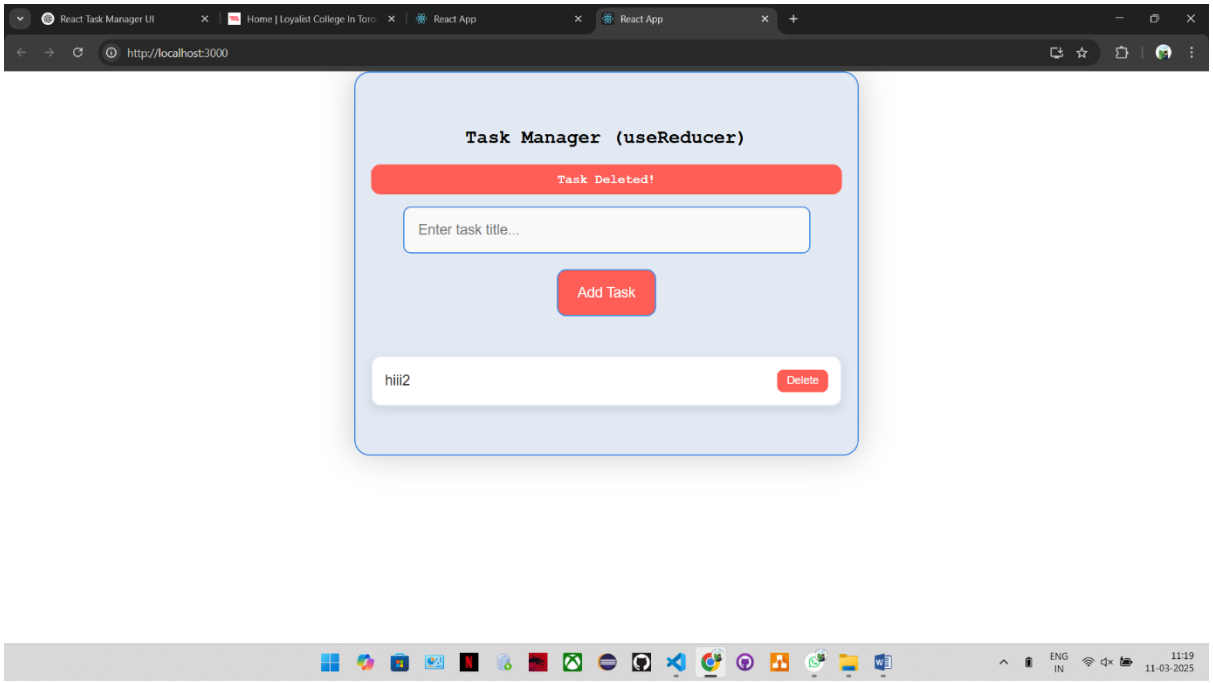
1-task_added



2-task_updated



3-task_deleted



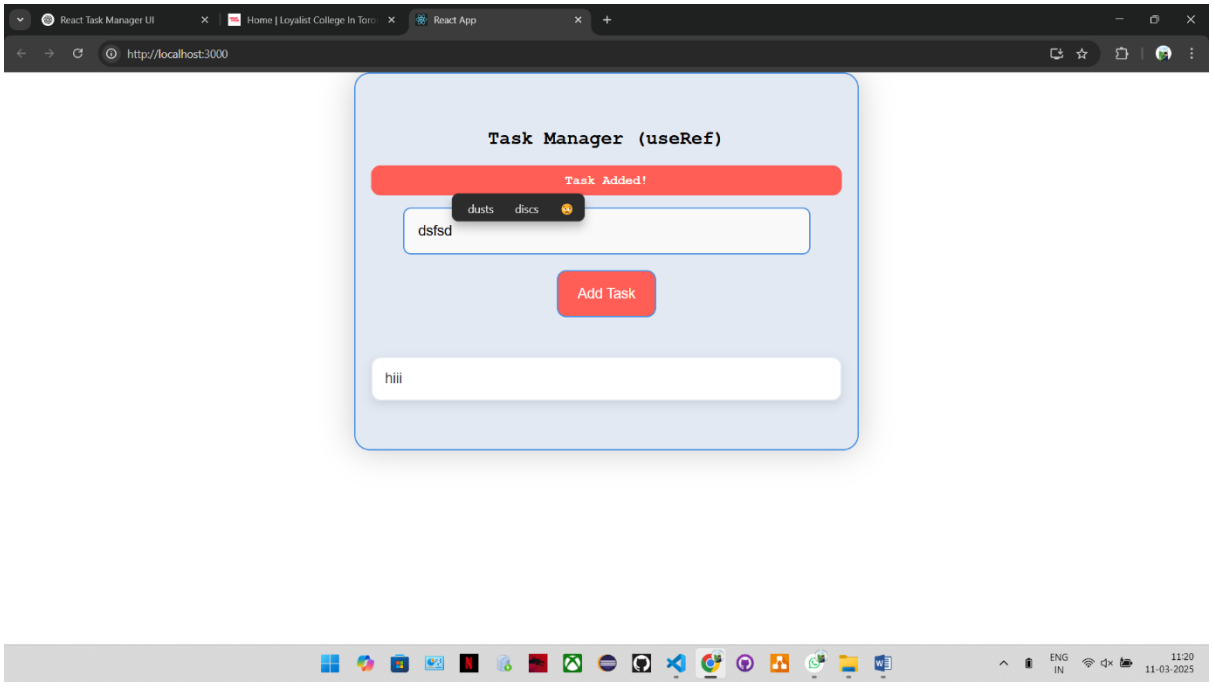
Explanation:

Instead of useState, we'll use useReducer to manage adding, updating, and deleting tasks in a structured way.

Code Snippet:

```jsx
import { useReducer } from "react";

const taskReducer = (state, action) => {
  switch (action.type) {
    case "ADD_TASK":
      return [...state, { id: Date.now(), title: action.title, completed: false }];
    case "DELETE_TASK":
      return state.filter((task) => task.id !== action.id);
    default:
      return state;
  }
};

function TaskManager() {
  const [tasks, dispatch] = useReducer(taskReducer, []);

  return (
    <div>
      <button onClick={() => dispatch({ type: "ADD_TASK", title: "New Task" })}>Add Task</button>
      {tasks.map((task) => (
        <div key={task.id}>
          {task.title} <button onClick={() => dispatch({ type: "DELETE_TASK", id: task.id })}>Delete</button>
        </div>
      ))}
    </div>
  );
}
```

Step-5:- useRef



Explanation:

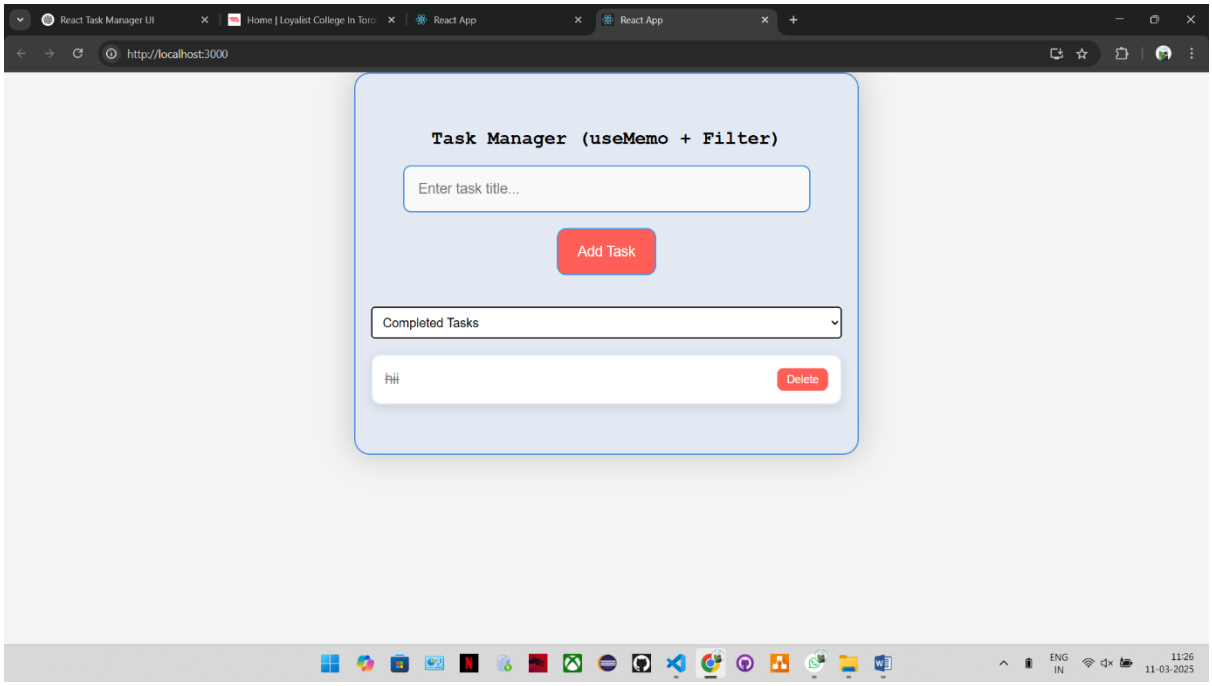
We'll use useRef to store a reference to the task input field so it gets auto-focused when the component loads.

Code Snippet:

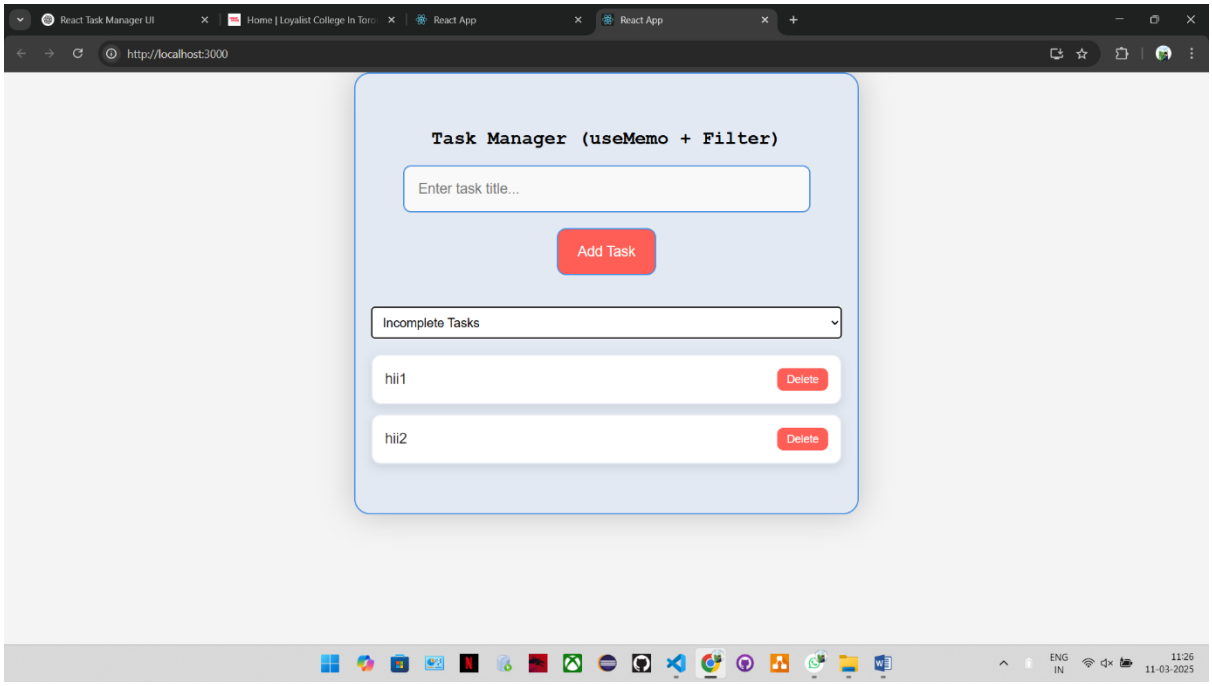
```jsx
import { useRef, useEffect } from "react";

function TaskInput() {
  const inputRef = useRef(null);

  useEffect(() => {
    inputRef.current.focus();
  }, []);

  return <input ref={inputRef} type="text" placeholder="Enter task..." />;
}

export default TaskInput;
```

Step-6:- useMemo

1-completed_task_filter



2-incompleted_task_filter



Explanation:

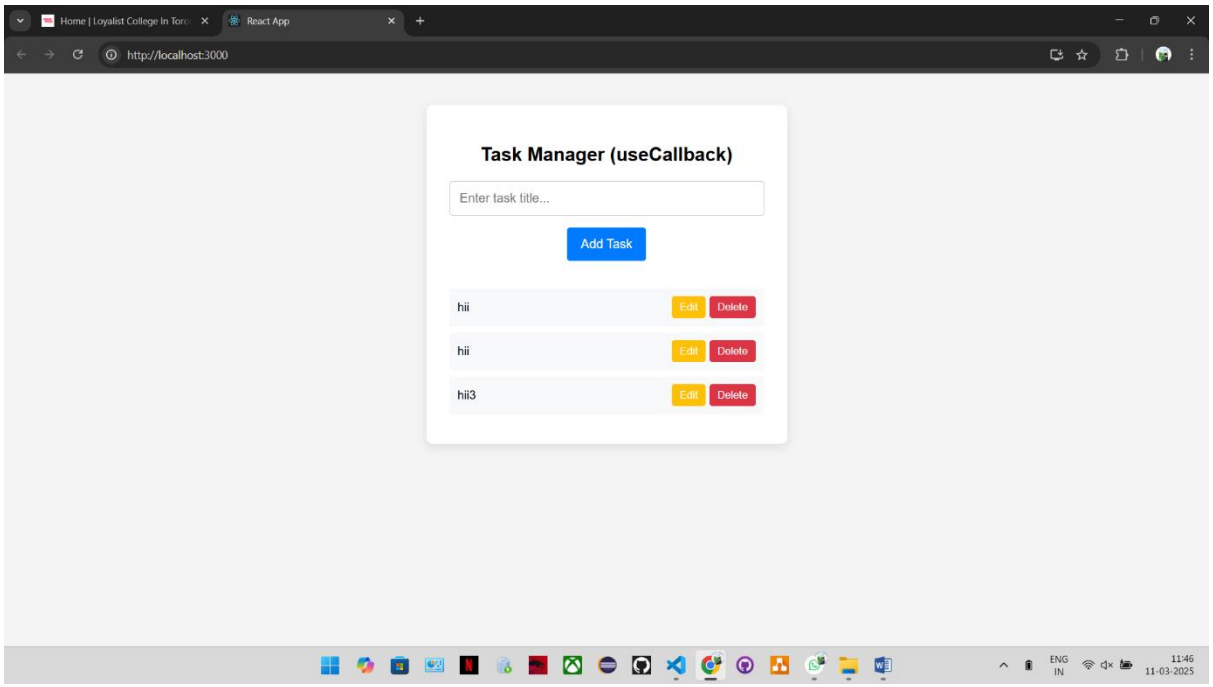
Filtering tasks can be slow if there are too many. We'll use useMemo to optimize it and prevent unnecessary calculations.

Code Snippet:

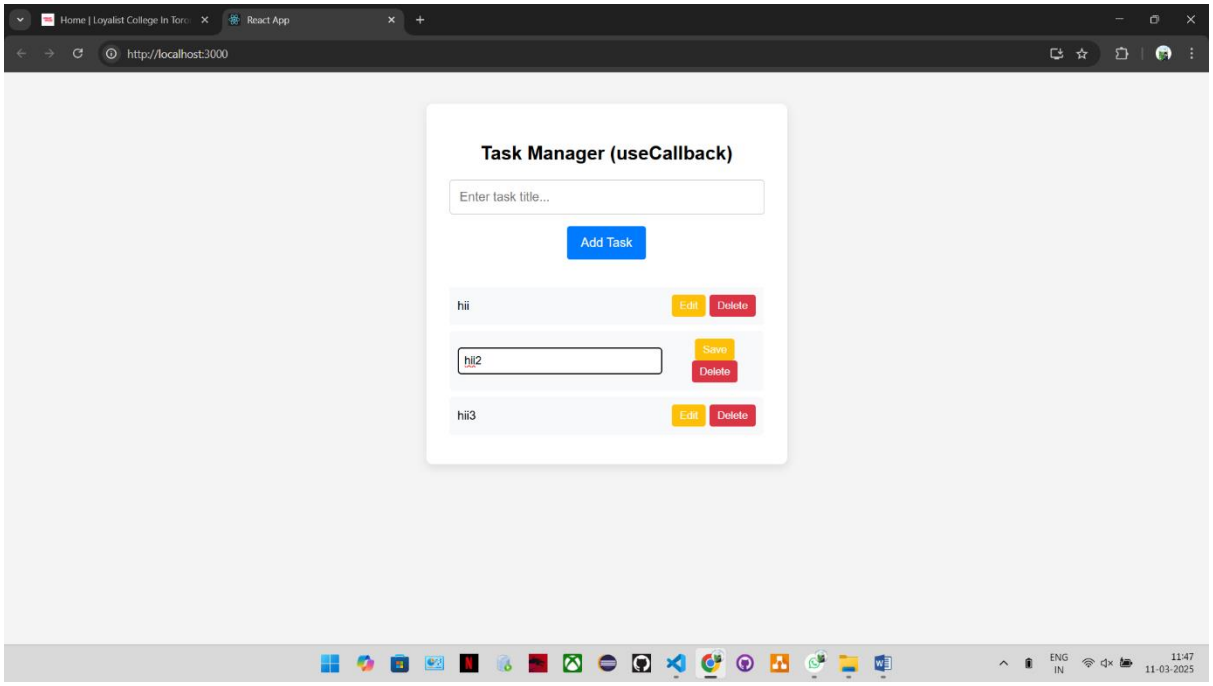
```
import { useState, useMemo } from "react";

function TaskManager() {
  const [tasks] = useState([
    { id: 1, title: "Task 1", completed: true },
    { id: 2, title: "Task 2", completed: false },
  ]);
  const [filter, setFilter] = useState("all");

  const filteredTasks = useMemo(() => {
    return tasks.filter((task) => (filter === "completed" ? task.completed : task));
  }, [tasks, filter]);

  return (
    <div>
      <button onClick={() => setFilter("completed")}>Show Completed</button>
      {filteredTasks.map((task) => (
        <div key={task.id}>{task.title}</div>
      ))}
    </div>
  );
}

export default TaskManager;
```

Step-7:- useCallback

1-added_tasks



2-edited_task



Explanation:

If a function is created every time the component re-renders, it can slow down performance. We'll use useCallback to optimize event handlers.

Code Snippet:

```
import { useState, useCallback } from "react";

function TaskManager() {
  const [tasks, setTasks] = useState([{ id: 1, title: "Sample Task", completed: false }])

  const deleteTask = useCallback(
    (id) => {
      setTasks((prevTasks) => prevTasks.filter((task) => task.id !== id));
    },
    [setTasks]
  );

  return (
    <div>
      {tasks.map((task) => (
        <div key={task.id}>
          {task.title} <button onClick={() => deleteTask(task.id)}>Delete</button>
        </div>
      ))}
    </div>
  );
}
```