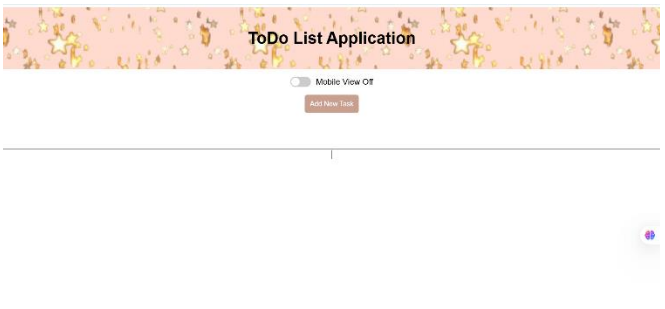
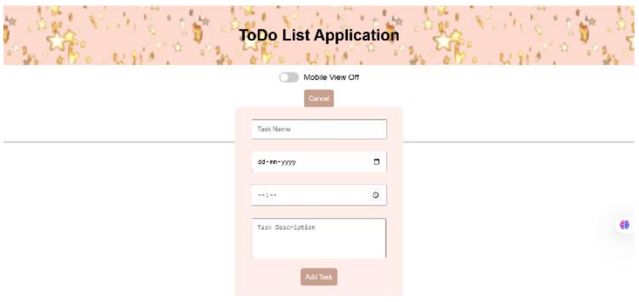
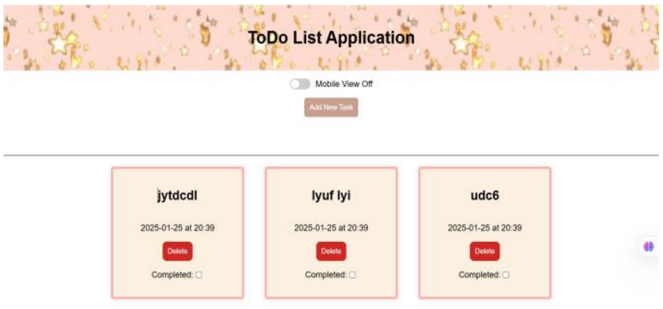
Assignment- 04

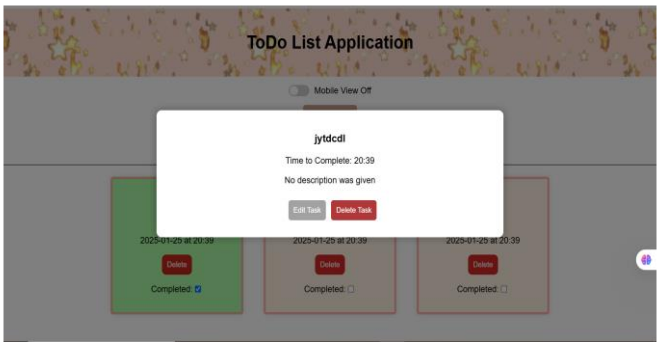
Create a ToDo list application using react:

Output:









Check the following link for the hosted application:

https://my-task-lister.web.app/

Code:

public/index.html

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <link rel="icon" href="https://img.icons8.com/plasticine/100/to-do.png" />
    <meta name="viewport" content="width=device-width, initial-scale=1" />
    <meta name="theme-color" content="#000000" />
    <meta
      name="description"
      content="Web site created using create-react-app"
    />
    <!--
      manifest.json provides metadata used when your web app is installed on a
      user's mobile device or desktop. See
https://developers.google.com/web/fundamentals/web-app-manifest/
    -->
    <!-- <link rel="manifest" href="%PUBLIC_URL%/manifest.json" /> -->
    <!--
      Notice the use of %PUBLIC_URL% in the tags above.
      It will be replaced with the URL of the `public` folder during the
build.
      Only files inside the `public` folder can be referenced from the HTML.

      Unlike "/favicon.ico" or "favicon.ico", "%PUBLIC_URL%/favicon.ico" will
      work correctly both with client-side routing and a non-root public URL.
      Learn how to configure a non-root public URL by running `npm run build`.
    -->
    <title>ToDo List</title>
  </head>
  <body>
    <noscript>You need to enable JavaScript to run this app.</noscript>
    <div id="root"></div>
    <!--
      This HTML file is a template.
      If you open it directly in the browser, you will see an empty page.

      You can add webfonts, meta tags, or analytics to this file.
      The build step will place the bundled scripts into the <body> tag.

      To begin the development, run `npm start` or `yarn start`.
      To create a production bundle, use `npm run build` or `yarn build`.
    -->
  </body>
</html>
```

src/index.js:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

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

```
    <App />
  </React.StrictMode>
);

// If you want to start measuring performance in your app, pass a function
// to log results (for example: reportWebVitals(console.log))
// or send to an analytics endpoint. Learn more: https://bit.ly/CRA-vitals
reportWebVitals();
```

src/App.js:

```
import React, { useState,useEffect } from 'react';
import './App.css';
import Header from './Header';
import TaskForm from './TaskForm';
import TaskList from './TaskList';

function App() {
  const [tasks, setTasks] = useState([]);
  const [selectedTask, setSelectedTask] = useState(null);
  const [taskBeingEdited, setTaskBeingEdited] = useState(null);
  const [isFormVisible, setIsFormVisible] = useState(false);
  const [isMobileView, setIsMobileView] = useState(false);

  useEffect(() => {
    console.log("isMobileView updated:", isMobileView);
  }, [isMobileView]);

  // Add a task to the list
  const addTask = (task) => {
    const newTask = {
      ...task,
      id: Date.now(),
    };
    setTasks([...tasks, newTask]);
  };

  const handleToggle = () => {
    setIsMobileView((prev) => !prev);
  };

  // Toggle completion status of a task
  const toggleCompletion = (taskId) => {
    const updatedTasks = tasks.map((task) =>
      task.id === taskId ? { ...task, isCompleted: !task.isCompleted } : task
    );
    setTasks(updatedTasks);
  };
  const editTask = (updatedTask) => {
    const updatedTasks = tasks.map((task) =>
      task.id === updatedTask.id ? updatedTask : task
    );
    setTasks(updatedTasks);
    setSelectedTask(null); // Close the task details overlay after saving
  };
  // Delete a task from the list
  const deleteTask = (taskId) => {
    const updatedTasks = tasks.filter((task) => task.id !== taskId);
    setTasks(updatedTasks);
  };

  // Show task details in the center of the page
  // const showTaskDetails = (task) => {
  //   setSelectedTask(task);
  // };

  // Hide task details
  const hideTaskDetails = () => {
    setSelectedTask(null);
  };

  return (
    <div className={`App ${isMobileView ? "mobile-view" : ""}`}>
      <Header />
      <div className="toggle-container">
        <label className="switch">
          <input
            type="checkbox"
            checked={isMobileView}
            onClick={handleToggle}
          />
          <span className="slider"></span>
        </label>
        <span>{isMobileView ? "Mobile View On" : "Mobile View Off"}</span>
      </div>
      <TaskForm
        addTask={addTask}
        task={taskBeingEdited || null}
        editTask={editTask}
        cancelEdit={taskBeingEdited ? () => setTaskBeingEdited(null) : null}
        isFormVisible={isFormVisible}
        setIsFormVisible={setIsFormVisible}
      />
      <TaskList
```

```
        tasks={tasks}
        toggleCompletion={toggleCompletion}
        deleteTask={deleteTask}
        setSelectedTask={setSelectedTask}
        isFormVisible={isFormVisible}
      />

      {/* Show Task Details in the Center */}
      {selectedTask && (
        <div className="task-details-overlay" onClick={hideTaskDetails}>
          <div className="task-details-box" onClick={(e) =>
e.stopPropagation()}>
            <h3>{selectedTask.taskName}</h3>
            <p>Time to Complete: {selectedTask.estimatedTime}</p>
            <p>{selectedTask.description}</p>
            <button className="edit" onClick={()
=>  {setTaskBeingEdited(selectedTask); hideTaskDetails();}}>
              Edit Task
            </button>
            <button className="del" onClick={() =>
deleteTask(selectedTask.id)}>Delete Task</button>
          </div>
        </div>
      )}
    </div>
  );
}

export default App;
```

src/Header.js

```
import React from 'react';

const Header = () => {
  return (
    <header>
      <h1>ToDo List Application</h1>
    </header>
  );
};

export default Header;
```

src/TaskForm.js:

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

function TaskForm({ addTask, task = null, editTask, cancelEdit, isFormVisible,
setIsFormVisible }) {
  const [taskName, setTaskName] = useState("");
  const [estimatedDate, setEstimatedDate] = useState("");
  const [estimatedTime, setEstimatedTime] = useState("");
  const [description, setDescription] = useState("");

  useEffect(() => {
    if (task) {
      setTaskName(task.taskName);
      setEstimatedDate(task.estimatedDate ||""); // Default to next day's date
if not provided
      setEstimatedTime(task.estimatedTime || "");  // Default to current time
if not provided
      setDescription(task.description || ""); // Default description
      setIsFormVisible(true);  // Ensure form is visible when editing
    } else {
      setIsFormVisible(false);  // Hide form when no task is being edited
    }
  }, [task, setIsFormVisible]);
  const getNextDayDate = () => {
    const nextDay = new Date();
    nextDay.setDate(nextDay.getDate() + 1); // Set to the next day
    return nextDay.toISOString().split('T')[0]; // Return date as YYYY-MM-DD
  };

  const getCurrentTime = () => {
    const currentTime = new Date();
    const hours = currentTime.getHours().toString().padStart(2, '0'); // Get
hours in 24-hour format
    const minutes = currentTime.getMinutes().toString().padStart(2, '0'); //
Get minutes, padded to two digits
    return `${hours}:${minutes}`; // Format as HH:mm
  };
  const handleSubmit = (e) => {
    e.preventDefault();
    const newTask = {
      ...task, // Keep existing task ID if editing
      taskName: taskName || "",
      estimatedDate: estimatedDate || getNextDayDate(),
      estimatedTime: estimatedTime || getCurrentTime(),
      description: description || "No description was given",
    };
    task ? editTask(newTask) : addTask(newTask);
    clearForm();
    setIsFormVisible(false);
  };

  const clearForm = () => {
    setTaskName("");
    setEstimatedDate("");
    setEstimatedTime("");
    setDescription("");
    setIsFormVisible(false); // Close form
    if (cancelEdit) cancelEdit();
  };

  return (
    <div className='di'>
      <button className="but" onClick={() => {if (isFormVisible) {
      clearForm();
    }
    setIsFormVisible(!isFormVisible);
      }}>
        {isFormVisible ? "Cancel" : "Add New Task"}
      </button>
      <div className={`form-container ${isFormVisible ? "visible" : ""}`}>
        {isFormVisible && (
          <form onSubmit={handleSubmit} className="form">
            <input
              type="text"
              placeholder="Task Name"
              value={taskName}
              onChange={(e) => setTaskName(e.target.value)}
              required
            />
            <input
              type="date"
              value={estimatedDate}
              onChange={(e) => setEstimatedDate(e.target.value)}
              //required
            />
            <input
              type="time"
              value={estimatedTime}
              onChange={(e) => setEstimatedTime(e.target.value)}
              //required
            />
            <textarea
              placeholder="Task Description"
              value={description}
              onChange={(e) => setDescription(e.target.value)}
              //required
            />
            <button type="submit">{task ? "Save Changes" : "Add
Task"}</button>
            {task && <button onClick={ ()=>{ cancelEdit(); clearForm(); }
}>Cancel</button>}
          </form>
        )}
      </div>
    </div>
  );
}

export default TaskForm;
```

src/TaskItem.js:

```
import React from 'react';

const TaskItem = ({ task, toggleCompletion, deleteTask, setSelectedTask }) =>
{
  return (
    <div
      className={`task-item ${task.isCompleted ? 'completed' : ''}`}
      onClick={() => setSelectedTask(task)} // Call showTaskDetails when task
is clicked
    >
      <div className="task-header">
        <h2>{task.taskName}</h2>
        <p>{task.estimatedDate} at {task.estimatedTime}</p>
        <button onClick={(e) => {e.stopPropagation();
            deleteTask(task.id);}
            }>Delete</button>
        <div className="Che">
          <p>Completed:</p> <input
            type="checkbox"
            checked={task.isCompleted}
            onChange={() => toggleCompletion(task.id)} // Toggle completion
status
          /></div>
      </div>
    </div>
  );
};
```

```
export default TaskItem;
```

src/TaskList.js:

```
import React from 'react';
import TaskItem from './TaskItem';

const TaskList = ({ tasks, toggleCompletion, deleteTask, setSelectedTask,
isFormVisible }) => {
  return (
    <div className={`task-list ${isFormVisible ? 'hidden' : ''}`}>
      {tasks.map((task) => (
        <TaskItem
          key={task.id}
          task={task}
          toggleCompletion={toggleCompletion}
          deleteTask={deleteTask}
          setSelectedTask={setSelectedTask}
        />
      ))}
    </div>
  );
};

export default TaskList;
```

src/App.css:

```
.App {
  text-align: center;
  font-family: Arial, sans-serif;
  display: flex;
  flex-direction: column;
  align-items: center;
  justify-content: center;
}

header {
  background-color: #fecfbdbd;
  background-image: url("th-removebg-preview.png");
  color: rgb(0, 0, 0);
  padding: 20px;
  width: 100vw;
```

```css
  justify-content: center;
  margin-top: 5px;
  height: 80px;
  position:fixed;
  top:0;
}

form {
  margin: 20px;
  padding: 20px;
}

.form-container input, textarea {
  margin: 5px;
  padding: 10px;
  width: 250px;
}

button {
  padding: 10px;
  background-color: #c99f8f;
  color: white;
  border: none;
  cursor: pointer;
  border-radius: 5px;
}

button:hover {
  background-color: #ca9783;
}

.task-list {
  display: grid;
  width: 100vw;
  grid-template-columns: repeat(3, 300px);
  justify-content: center;
  top: 280px;
  position:relative;
  transition: all 0.3s ease;
  border-top: 2px solid rgba(60, 60, 60, 0.594);
}
.task-list.hidden {
  opacity: 0.5; /* Reduced opacity */
  pointer-events: none; /* Optional: Disable interaction while dimmed */
}
.task-item {
  background: rgba(252, 238, 220, 0.918);
  padding: 15px;
  border-radius: 5px;
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
  margin: 20px;
  border: 4px solid #fb9999c5;
}

.task-item.completed {
  background: #a8f7a2;
}

.task-header {
  display: flex;
  justify-content: space-between;
}

.task-details {
  margin-top: 10px;
}

.task-item:hover .task-details {
  display: block;
}
.task-details-overlay {
  position: fixed;
  top: 0;
  left: 0;
  width: 100%;
  height: 100%;
  background-color: rgba(0, 0, 0, 0.5);
  display: flex;
  justify-content: center;
  align-items: center;
  z-index: 999;
}

.task-details-box {
  background-color: white;
  padding: 20px;
  border-radius: 10px;
  box-shadow: 0 0 15px rgba(0, 0, 0, 0.3);
  width: 50%;
  text-align: center;
}

.but{
  margin: 0 20px 0px 20px;
}
.form-container {
  max-height: 0; /* Initially hide the form by reducing height */
  opacity: 0; /* Hide the form by making it transparent */
  overflow: hidden; /* Prevent content from being displayed outside */
  transition: all 0.5s ease-in-out; /* Smooth transition effect */
}

.form-container.visible {
  max-height: 500px; /* Allow enough height for the form content */
  opacity: 1; /* Make the form fully visible */
  transition: all 0.5s ease-in-out;
}

.form {
  display: flex;
  flex-direction: column; /* Stack items vertically */
  align-items: center; /* Center items horizontally */
  justify-content: center; /* Center items vertically */
  background-color: #ffeee9; /* Light green background */
  padding: 20px; /* Add padding around the form */
  gap: 15px; /* Add space between form elements */
  border-radius: 8px; /* Rounded corners */
  width: 300px; /* Set a fixed width for the form */
  box-shadow: 0px 4px 6px rgba(0, 0, 0, 0.1);
  margin: 0;
}
textarea {
  resize: none;
  height: 60px;
}
.di{
  display: flex;
  flex-direction: column;
  align-items: center;
  position: fixed;
  top:175px;
  z-index: 10;
}
.task-header{
  display: flex;
  flex-direction: column;
  align-items: center;
}
.Che{
  display:flex;
  flex-direction: row;
  align-items: center;
  justify-items: flex-start;
  column-gap: 0px;
```

```css
}
.task-header button{
  background-color: rgba(199, 0, 0, 0.836);
  color:rgb(252, 252, 252);
  border-radius: 8px;
  grid-template-columns: repeat(3, 300px);
}
.edit{
  background-color: rgb(162,162,162) !important;
  margin: 10px;
  border-radius: 5px;
}
.del{
  background-color: rgba(152, 0, 0, 0.781) !important;
  border-radius: 5px;
}
.che input[type="checkbox" i] {
  width: 30px!important;
}

.App.mobile-view .task-list {
  display: flex;
  flex-direction: column; /* Stack tasks vertically */
  gap: 10px;
  align-items: center;
}

.App.mobile-view .task-item{
  width: 270px;
}
/* Toggle Switch */
.toggle-container {
  display: flex;
  align-items: center;
  gap: 10px;
  margin: 20px;
  margin-top: 40px;
  position: fixed;
  top: 100px;
}

.switch {
  position: relative;
  display: inline-block;
  width: 40px;
  height: 20px;
}
```

```css
.switch input {
  opacity: 0;
  width: 0;
  height: 0;
}

.slider {
  position: absolute;
  cursor: pointer;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  background-color: #ccc;
  transition: 0.4s;
  border-radius: 20px;
}

.slider:before {
  position: absolute;
  content: "";
  height: 16px;
  width: 16px;
  left: 2px;
  bottom: 2px;
  background-color: white;
  transition: 0.4s;
  border-radius: 50%;
}

input:checked + .slider {
  background-color: #4caf50;
}

input:checked + .slider:before {
  transform: translateX(20px);
}
```