**Batch -** T5

**Practical No. -** 6

**Title –** Study and implementation of ReactJs

**Student Name -** Sharaneshwar Bharat Punjal

**Student PRN -** 23520011

**Perform following problem statements using ReactJs**

**Problem Statement 1: To-do List Application with State Management**

* Build a simple To-do List application that allows users to add, remove, and mark tasks as completed.
* The app should have a responsive user interface that works well on both desktop and mobile devices.
* Users should be able to input a task, which will be added to the list.
* Each task should have a checkbox to mark it as completed.
* Users can remove tasks from the list.
* Use React hooks like useState for state management.
* Add options to filter tasks based on completion status (e.g., all, completed, active).
* Break down the Todo List into multiple components, like TaskList, TaskItem, AddTaskForm, etc.
* The app should display appropriate error messages.
* The candidate should use appropriate error handling and validation to ensure that the application is robust and user-friendly.

**Installation of Tailwind CSS -**

To use Tailwind CSS in your React project, follow these steps:

1. Install Tailwind via npm:

   Run the following command in the terminal inside your project folder:

   npm install -D tailwindcss postcss autoprefixer

2. Initialize Tailwind configuration:

    Create `tailwind.config.js` and `postcss.config.js` files by running

   npx tailwindcss init -p

3. Configure your template paths:

Inside `tailwind.config.js`, configure Tailwind to remove unused styles in production mode by specifying the paths to all of your components:

   module.exports = {

     content: [

       "./src//\*.{js,jsx,ts,tsx}",

     ],

     theme: {

       extend: {},

     },

     plugins: [],

   }

4. Add Tailwind directives to your CSS:

   Open your `src/index.css` (or `App.css`), and add the following Tailwind CSS directives:

   css

   @tailwind base;

   @tailwind components;

   @tailwind utilities;

5. Start using Tailwind in your React components:

   Now you can use Tailwind CSS classes in your JSX files. For example:

   jsx

   <div className="bg-blue-500 text-white p-4 rounded">

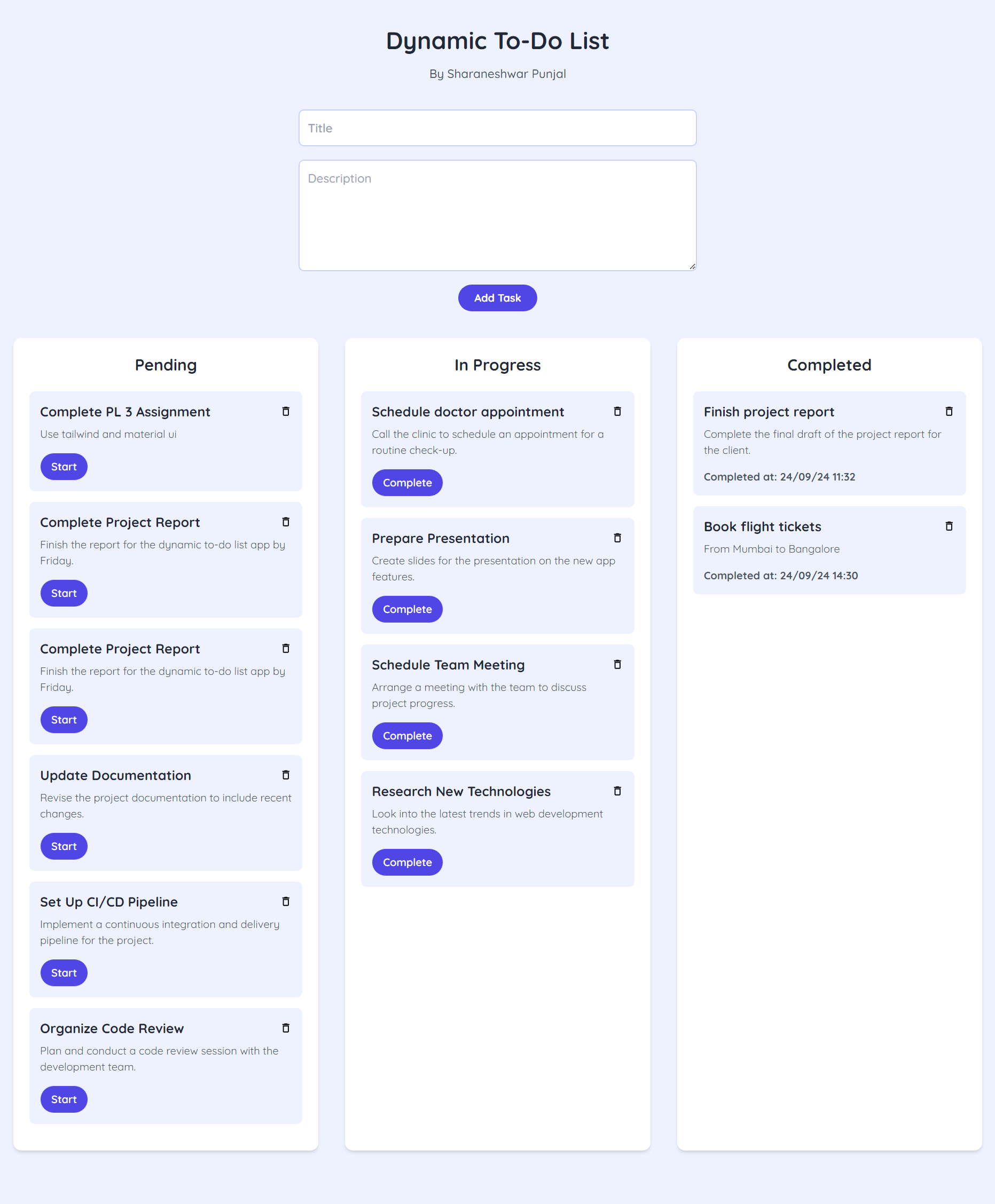
     This is a styled component.

   </div>

6. Run your React app using:

    npm run start

**Screenshots –**

****

**Problem Statement 2: Simple E-commerce Cart System**

* Develop a small e-commerce application where users can browse products and add them to a shopping cart.
* Display a list of products with details like price, name, and image.
* Users can add products to a shopping cart.
* The cart page shows all selected products and their total price.
* Users can remove items from the cart or adjust the quantity.
* Add filtering options (e.g., by price, category) and sorting (e.g., low-to-high, high-to-low).
* Add routing to navigate between different pages, like the product list, cart, and checkout pages.
* On clicking a product, navigate to a detailed view of that product.

**Setup and Installation of Material UI**

1. Create a React Application

If you haven't already set up a React project, you can create one using Vite or Create React App. Here’s how you can do it with Vite:

npm create vite@latest ecommerce-cart --template react

cd ecommerce-cart

npm install

2. Install Material UI

To use Material UI in your React application, you need to install the following packages:

@mui/material: Core components of Material UI.

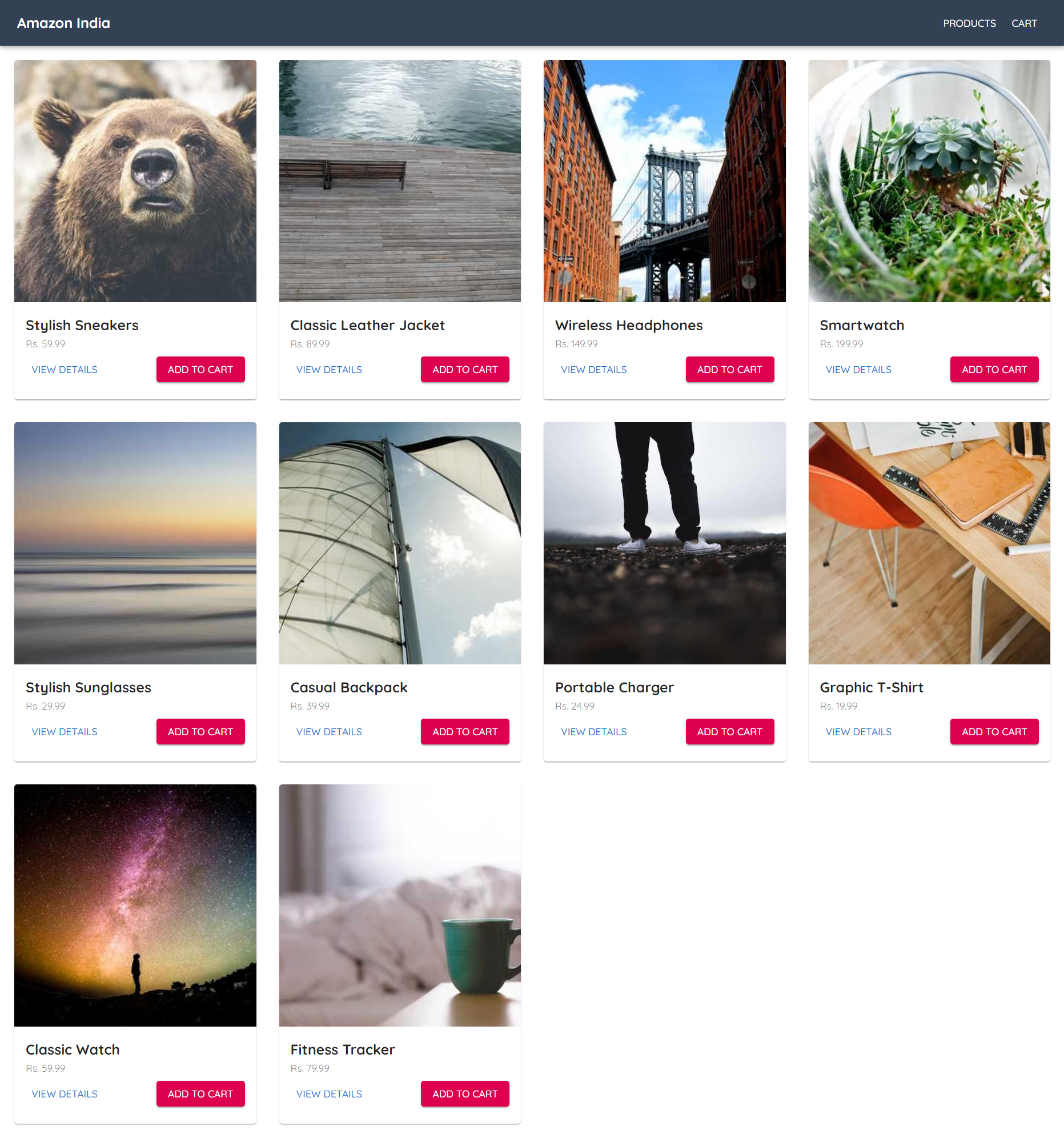
@mui/icons-material: Material UI icons (optional but often useful for cart or product icons).

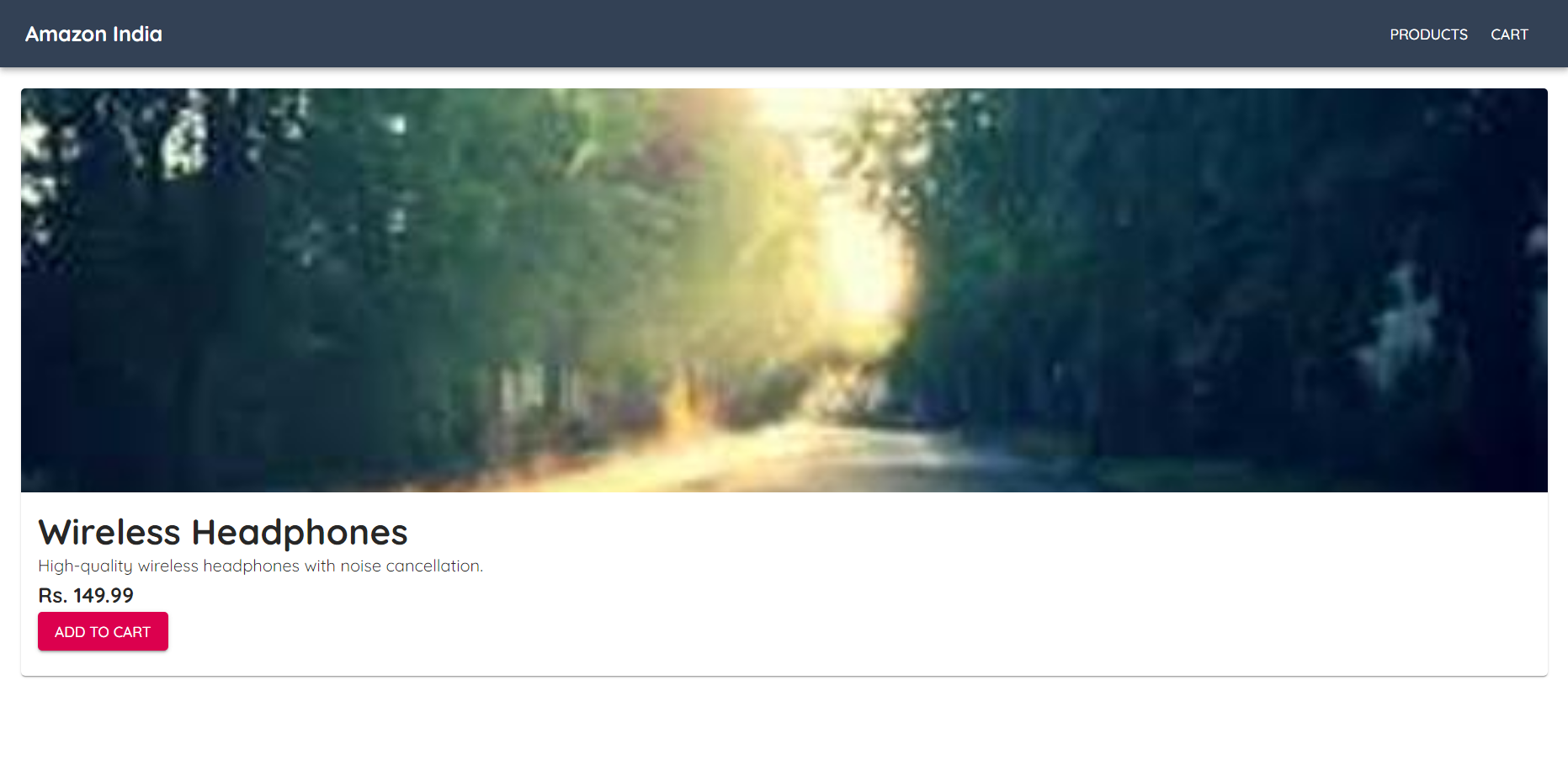
3. Run the following commands:

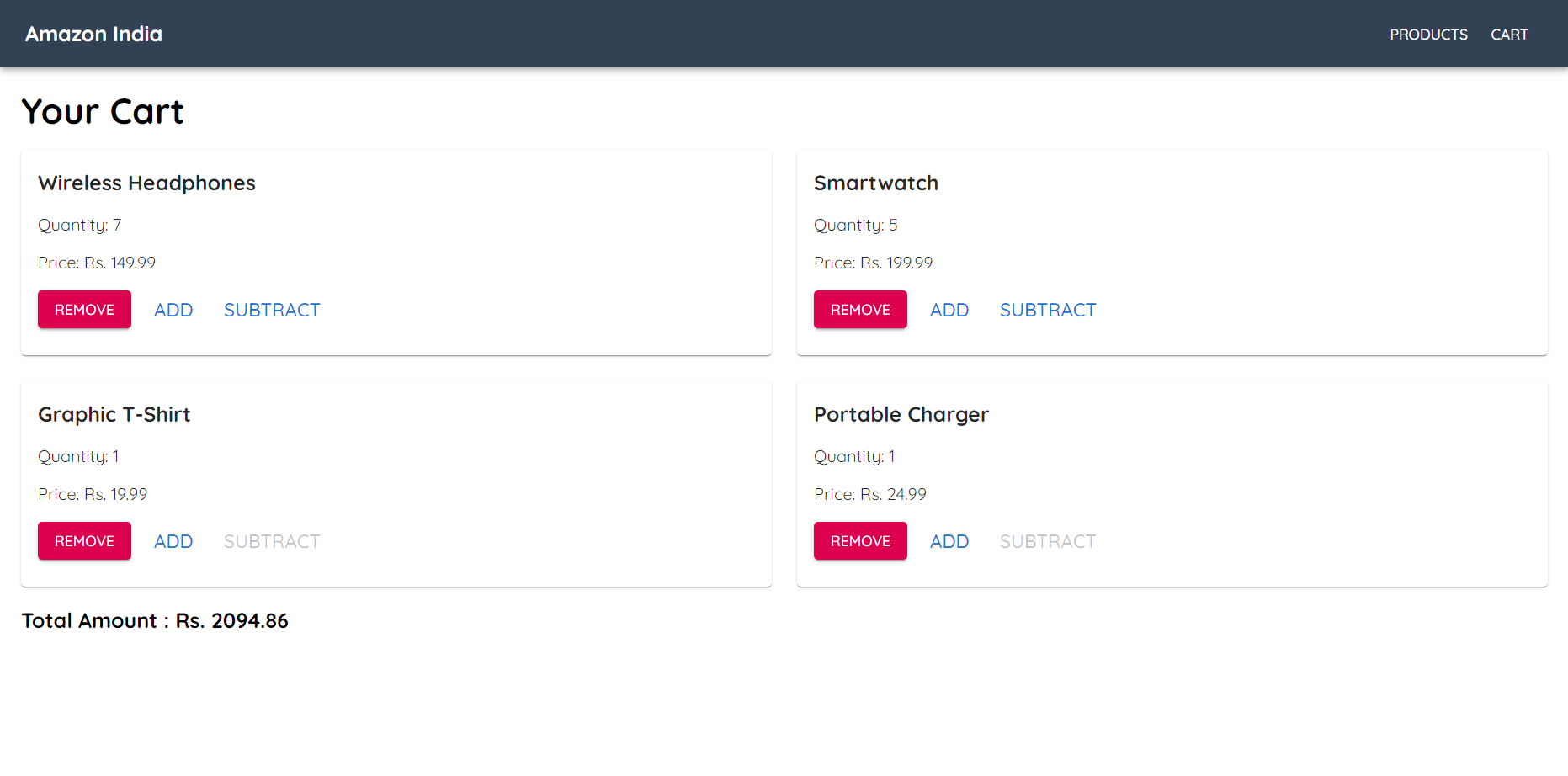
npm install @mui/material @emotion/react @emotion/styled

npm install @mui/icons-material

**Screenshots -**

****

****

****

**Problem Statement 3: User Authentication with React Context API**

Requirements

* Create a login form that checks for hardcoded user credentials.
* If authenticated, the user can see a personalized dashboard.
* Allow the user to log out, which resets the context state.
* Use React Router for navigation between login and dashboard pages.
* Protect certain routes (e.g., dashboard) so that only authenticated users can access them.

**Screenshots –**

