**Batch: T6**

**Practical No. 6**

**Title of Assignment: Study and implementation of ReactJs**

**Student Name: Parshwa Herwade**

**Student PRN: 22510064**

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.

ANS.

Here’s a detailed overview of each project, including the technologies used and their specific roles, along with components and features.

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

**Technologies/Frameworks Used:**

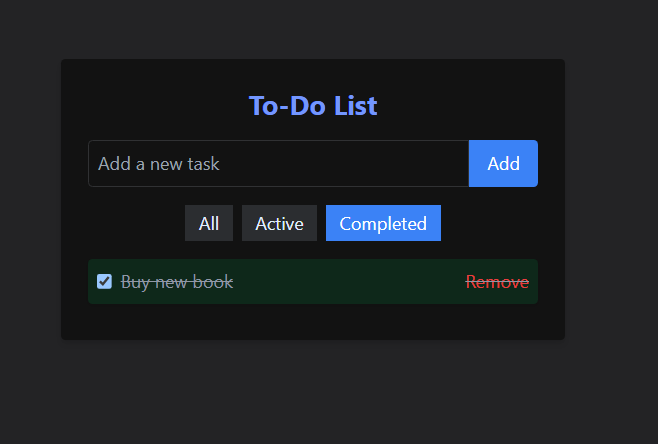
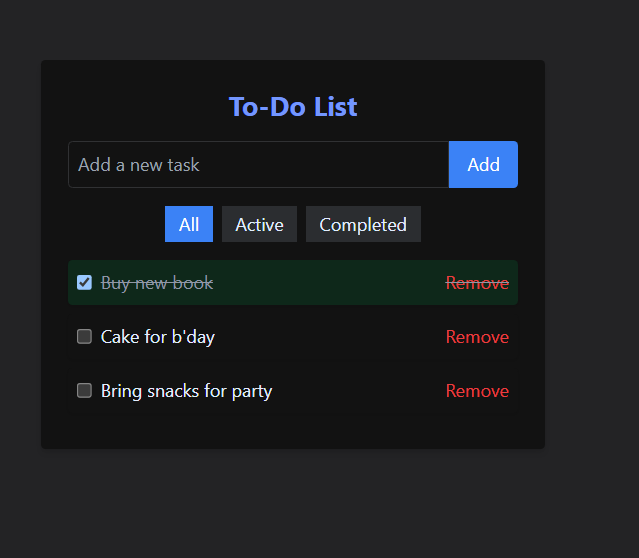
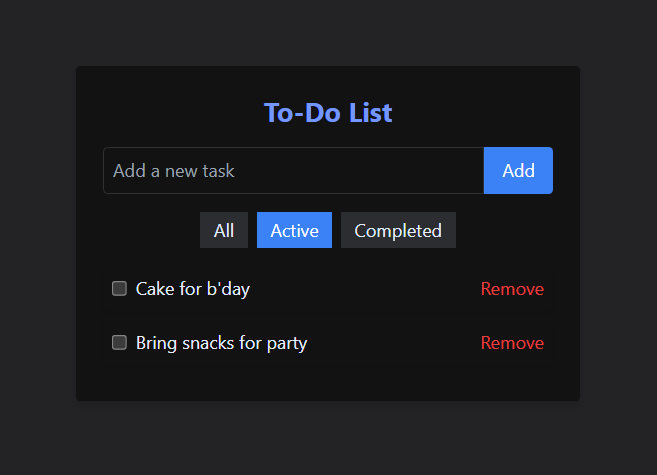
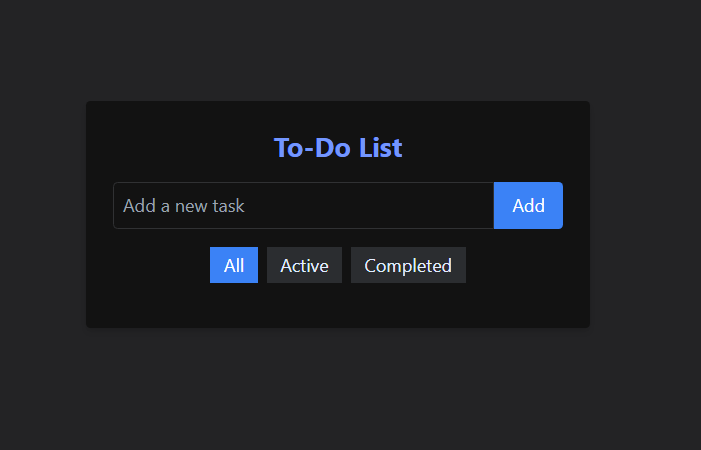
* **React**:
  + **Role**: A JavaScript library for building user interfaces.
  + **Key Features**: Component-based architecture, virtual DOM for efficient rendering, and hooks for managing state and lifecycle events.
* **Tailwind CSS**:
  + **Role**: A utility-first CSS framework for styling.
  + **Key Features**: Responsive design capabilities, utility classes for rapid development, and easy customization through configuration files.
* **React Hooks**:
  + **Role**: Functions that let you use state and other React features without writing a class.
  + **Key Hooks**:
    - useState: To manage local component state.
    - useEffect: (if needed) To perform side effects like data fetching or subscriptions.

**Components:**

1. **AddTaskForm**:
   * Handles input for new tasks.
   * Validates input to ensure no empty tasks are added.
   * Uses props to communicate with parent for adding tasks.
2. **TaskItem**:
   * Displays an individual task with a checkbox.
   * Includes buttons to mark as completed and remove.
3. **TaskList**:
   * Renders TaskItem components based on the current filter.
   * Filters tasks by completion status using state.

**Features:**

* Input validation and error handling (e.g., alerting if the task input is empty).
* Responsive design using Tailwind’s utility classes to ensure a good user experience across devices.
* Filtering options (all, completed, active) using buttons or a dropdown.



Problem Statement 2: Simple E-commerce Cart System Requirements

• 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-tolow).

• 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.

**ANS.:**

**Technologies/Frameworks Used:**

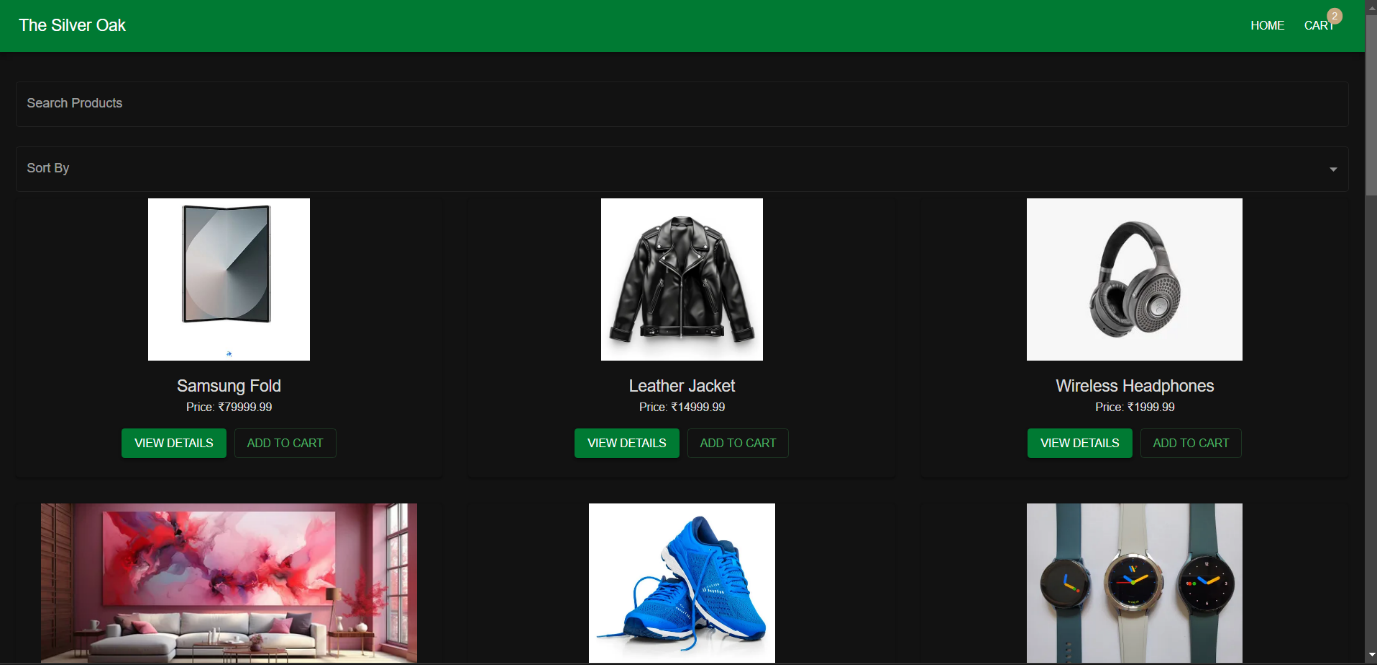
* **React**:
  + **Role**: To manage the UI and state effectively with a component-driven approach.
  + **Key Features**: Use of props and state to pass data and manage UI updates.
* **Material UI**:
  + **Role**: A popular React UI framework that provides pre-styled components.
  + **Key Features**: Responsive design, customizable components, and built-in accessibility features.
* **React Router**:
  + **Role**: A library for routing in React applications.
  + **Key Features**: Enables navigation between different views (pages) using declarative routing.

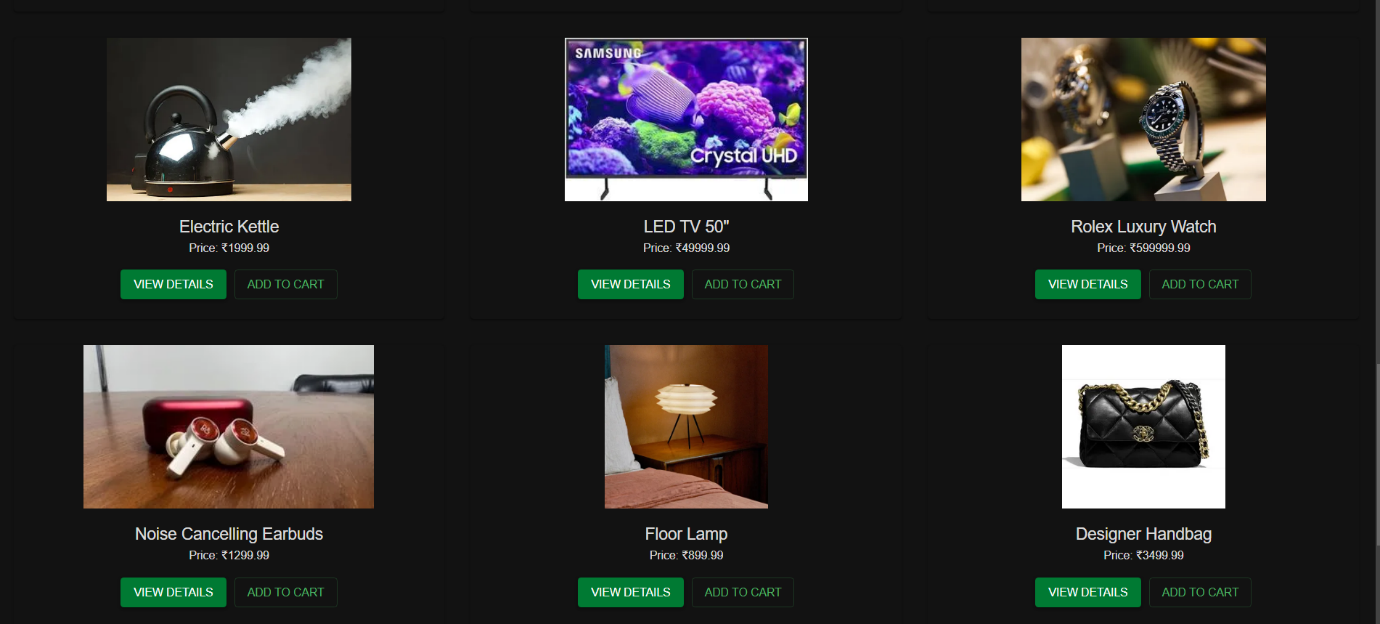
**Components:**

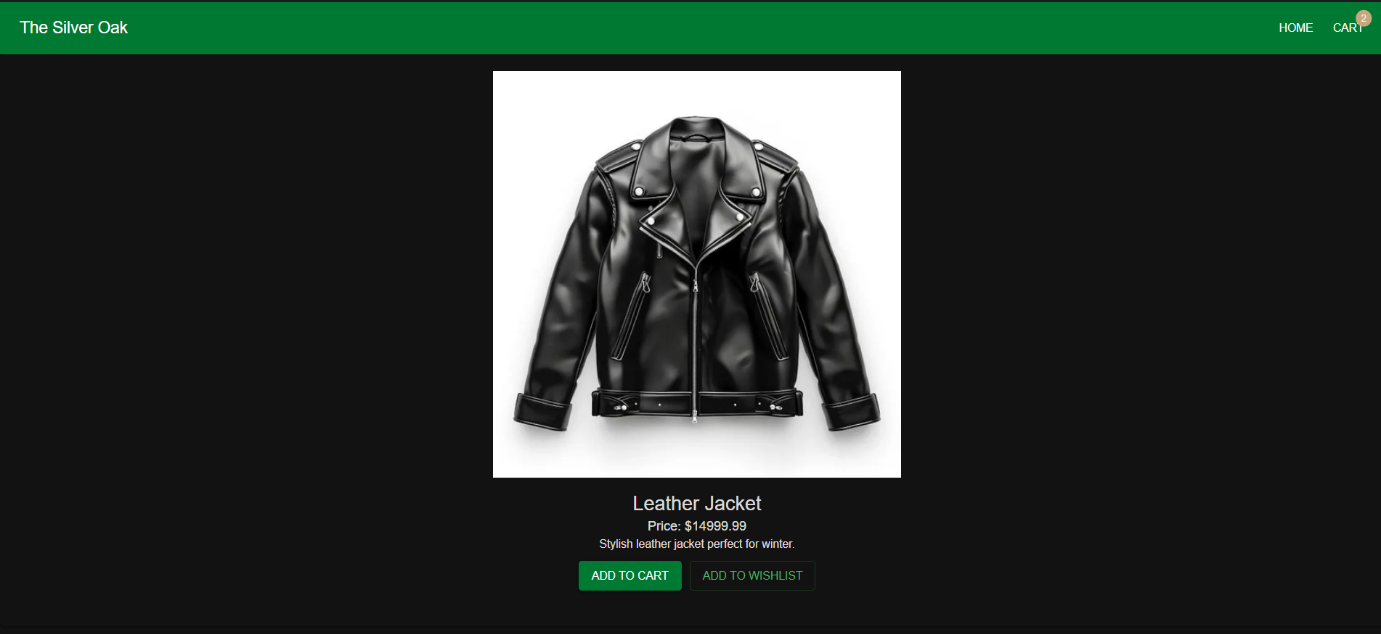
1. **ProductCard:**
   * **Displays product information (image, name, price).**
   * **Contains a button to add the product to the cart.**
2. **ProductList:**
   * **Fetches products from an API or static data.**
   * **Renders a list of ProductCard components and implements filtering/sorting.**
3. **Cart:**
   * **Shows products added to the cart with quantity controls.**
   * **Calculates and displays the total price.**
4. **Checkout:**
   * **Collects user information for the purchase process (optional).**
   * **Can handle payment processing (integration with payment gateways if required).**

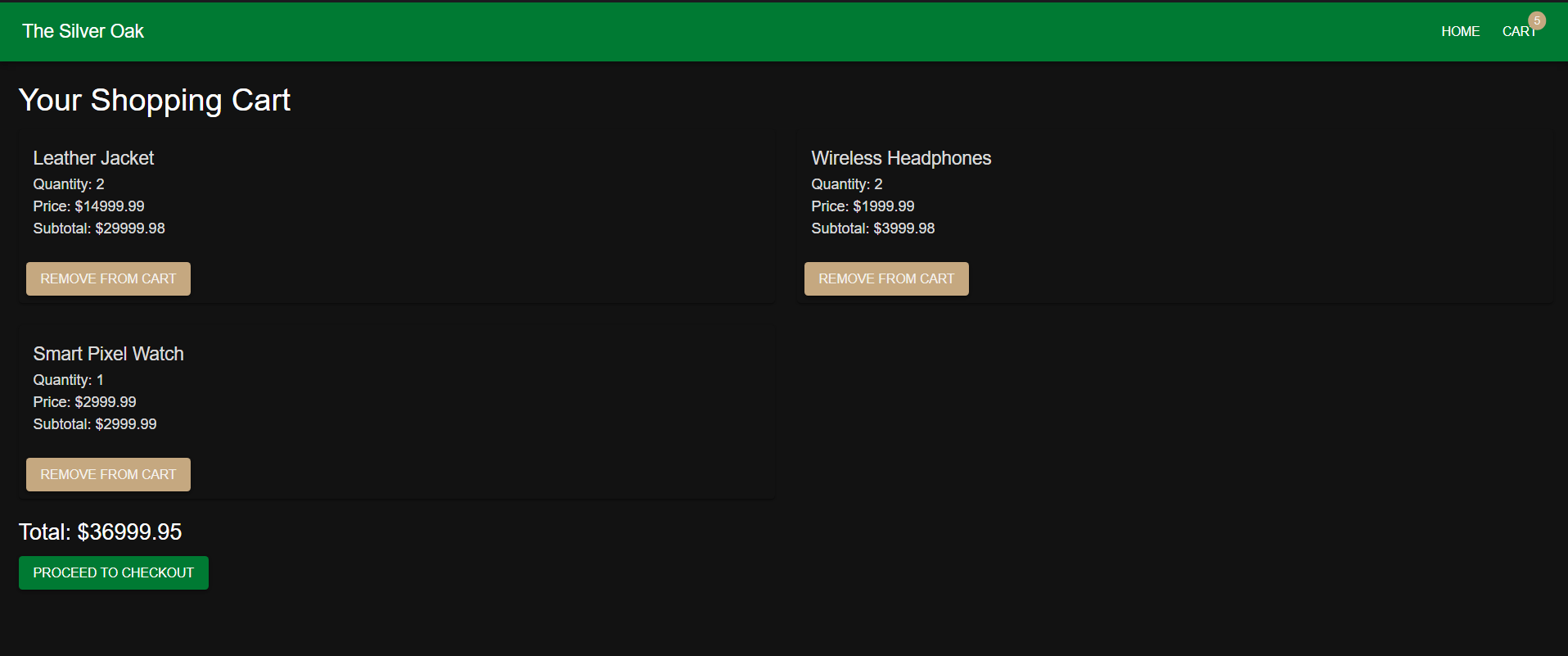
**Features:**

* **Responsive UI with Material UI components that adapt to various screen sizes.**
* **Detailed product views navigable through React Router.**
* **Persistent cart state, which can be implemented using local storage or context for multi-page persistence.**









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.

ANS.

**Technologies/Frameworks Used:**

* **React**:
  + **Role**: Provides the base for building the application interface and managing state.
  + **Key Features**: Component-based design and hooks for managing component state.
* **Material UI**:
  + **Role**: Offers ready-to-use UI components with a modern design.
  + **Key Features**: Theme customization and responsive grid layouts.
* **React Router**:
  + **Role**: Manages navigation and route protection.
  + **Key Features**: Declarative routing and nested routing support.
* **React Context API**:
  + **Role**: Allows for state management across the application without prop drilling.
  + **Key Features**: Creates a context that can be accessed by any component, useful for global state like authentication.

**Components:**

1. **Login**:
   * Contains a form for users to enter credentials.
   * Uses context to authenticate users and update state.
2. **Dashboard**:
   * Displays personalized content for authenticated users (e.g., user info, activities).
3. **ProtectedRoute**:
   * A wrapper for routes that require authentication.
   * Redirects unauthenticated users to the login page.
4. **Navbar**:
   * Shows different navigation links based on the user's authentication status.

**Features:**

* Hardcoded user credentials for authentication.
* Logout function resets authentication state via context.
* Protects certain routes to ensure only authenticated users can access them.

**Additional Technologies/Libraries:**

* **Axios or Fetch API**: For making HTTP requests to fetch products or user data (especially in the E-commerce project).
* **Redux (optional)**: For more complex state management if needed in larger applications.

