DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

CHANDIGARH UNIVERSITY

Discover. Learn. Empower.

# Experiment 3

Student Name: Advitiya Sharma UID: 23BIT70015

Branch: AIT-CSE Section: 23AIT_KRG-1

Semester: 6 Date of Performance: 30/01/2026

Subject Name: Full Stack II Subject Code: 23CSH-382

- **Aim:**

  The aim of this experiment is to further enhance the "Eco-Track" application by integrating Redux Toolkit for centralized state management, enabling efficient data handling and asynchronous fetching of activity logs within a scalable architecture.

- **Objectives:**

  The main objectives of this experiment are as follows:

  1. Implementing a centralized global state using Redux Toolkit to manage application-wide data.
  2. Utilizing createSlice to define the state, reducers, and actions for the carbon logs in a modular fashion.
  3. Developing asynchronous logic using createAsyncThunk to simulate and handle API calls for fetching data.
  4. Managing complex state transitions (pending, fulfilled, rejected) using extraReducers to provide a responsive UI.
  5. Connecting the React frontend to the Redux store using useSelector and useDispatch hooks for data retrieval and action triggering.

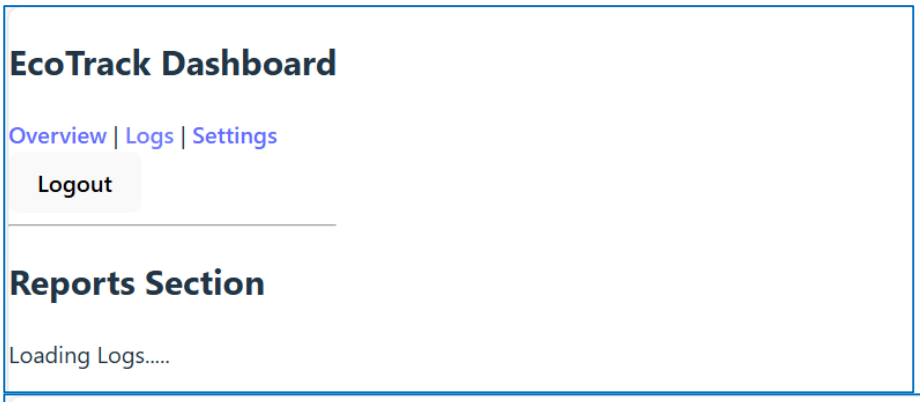
- **Implementation:**

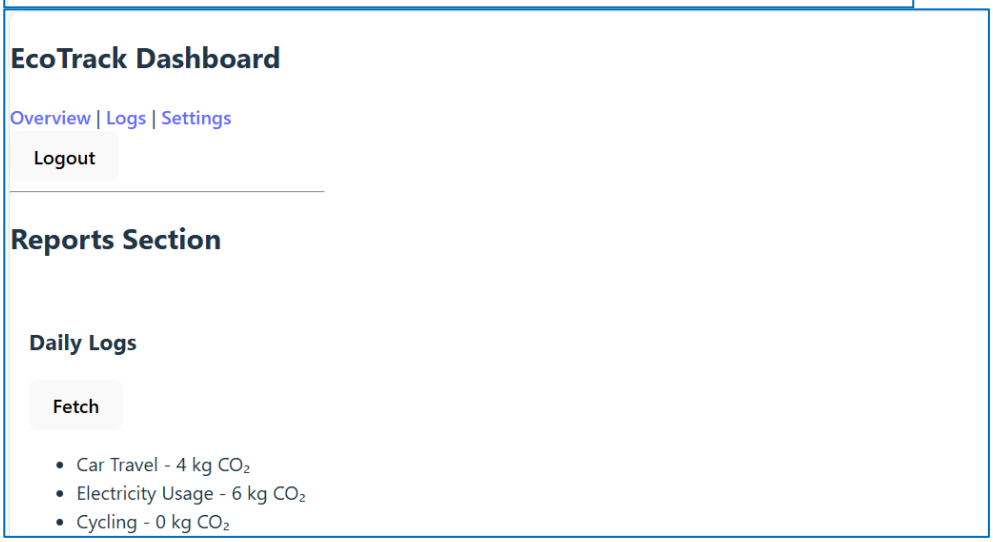
  The following general steps were followed to implement authentication and protected routing:

  1. The Redux Toolkit library was integrated into the project to replace or augment local state management.
  2. A logSlice.jsx file was created to define the initial state, including an empty data array, a status indicator ('idle', 'loading', 'success', 'failed'), and an error field.
  3. The fetchLogs async thunk was implemented to simulate a network delay using a Promise and then return data from the local logs repository.
  4. The slice was configured with extraReducers to update the application status and data based on the lifecycle of the asynchronous fetch operation.
  5. A centralized store.jsx was configured using configureStore to combine the log reducer into a single root state object.

6. The main.jsx entry point was updated to wrap the entire application in a Redux , making the store accessible to all components.
7. The Logs.jsx component was refactored to use useSelector to extract the logs data, loading status, and error messages from the global state.
8. A useDispatch hook was utilized within a button click handler to trigger the fetchLogs action programmatically.
9. Conditional rendering logic was applied in the UI to display "Loading content...." during the pending state or error messages if the fetch failed.
10. Data transformation logic was implemented within the component to filter and display only "high-carbon" activities (carbon >= 4) from the fetched global state.
11. The modular store structure ensured that the data fetching logic remained separate from the UI presentation layer.

- **Output:**



- **Results:**
  The Eco-Track application successfully moved from local state management to a robust Redux-based architecture. The integration of Redux Toolkit facilitated a predictable state container, where data fetching and filtering were handled through standardized actions and reducers. By utilizing asynchronous thunks, the application gained the ability to manage side effects gracefully, providing clear user feedback during data retrieval. This resulted in a more professional, maintainable, and debuggable codebase.

- **Learning Outcomes:**
  After completing this experiment, I have learnt to:
  1. Setup and configure a global Redux store using Redux Toolkit.
  2. Create slices to manage specific domains of application state independently.
  3. Handle asynchronous operations and side effects using createAsyncThunk.
  4. Utilize extraReducers to respond to external action types and manage loading/error states.
  5. Access global state and dispatch actions within functional components using React-Redux hooks.
  6. Decouple data fetching and business logic from UI components to improve code modularity.