Experiment 3

Student Name: Rakshit Chauhan **UID: 23BCS12628**
Branch: CSE **Section/Group: KRG 3-B**
Semester: 6th **Date of Performance:02/02/2026**
Subject Name: Full Stack Development – II **Subject Code: 23CSH-309**

1. **Aim**: To implement **global state management** in the EcoTrack application using **React Redux (Redux Toolkit)** for managing daily environmental activity logs, including fetching data asynchronously, adding new logs, and removing existing logs.

2. **Objective**:

- To understand the concept of **state management using Redux**
- To implement **Redux Toolkit** for simplified Redux configuration
- To manage application-wide state using a **centralized Redux store**
- To implement **async data fetching** using createAsyncThunk
- To add and delete data from the Redux store using reducers
- To integrate Redux with React using useDispatch and useSelector
- To simulate real-time environmental activity logging
- To understand the separation of UI logic and state logic

3. **Implementation / Code:**

Tools & Technologies Used:-

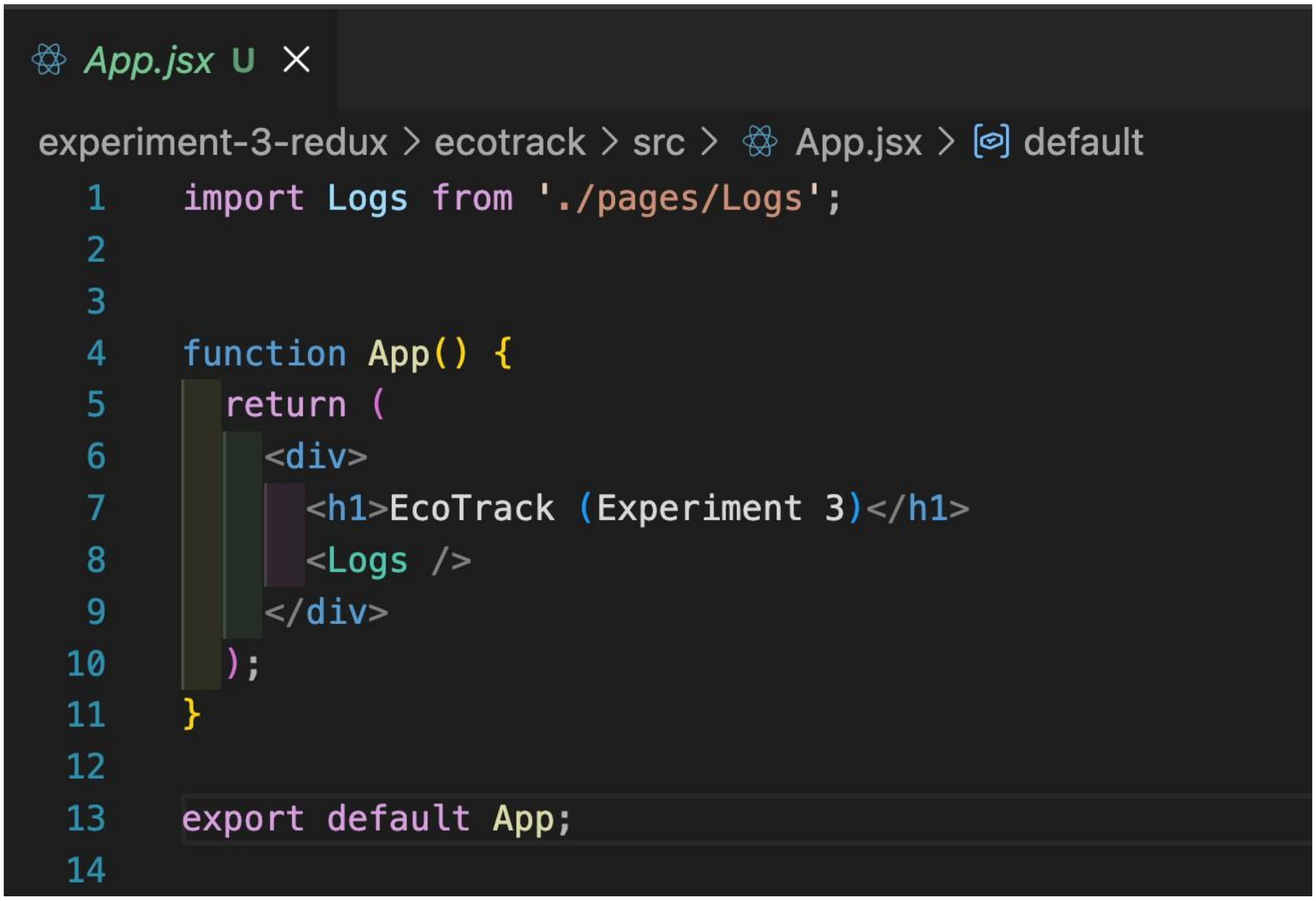
- React.js
- Redux Toolkit
- React Redux
- JavaScript (ES6)
- VS Code
- Web Browser (Google Chrome / Firefox)

Implementation Description:-

1) The EcoTrack application uses Redux Toolkit to manage daily carbon emission logs globally.
2) A Redux store is configured using configureStore, and a logs slice is created using createSlice.

3) Asynchronous data fetching is implemented using createAsyncThunk to simulate an API call that loads initial activity logs.
4) The Logs component uses:
5) useSelector to access logs data and loading status from the Redux store
6) useDispatch to dispatch actions such as fetching logs, adding logs, and removing logs
7) Users can:
8) Add new activities with carbon emission values
9) View the list of logged activities
10) Delete any activity from the list
11) This approach ensures predictable state updates, better scalability, and cleaner code organization.

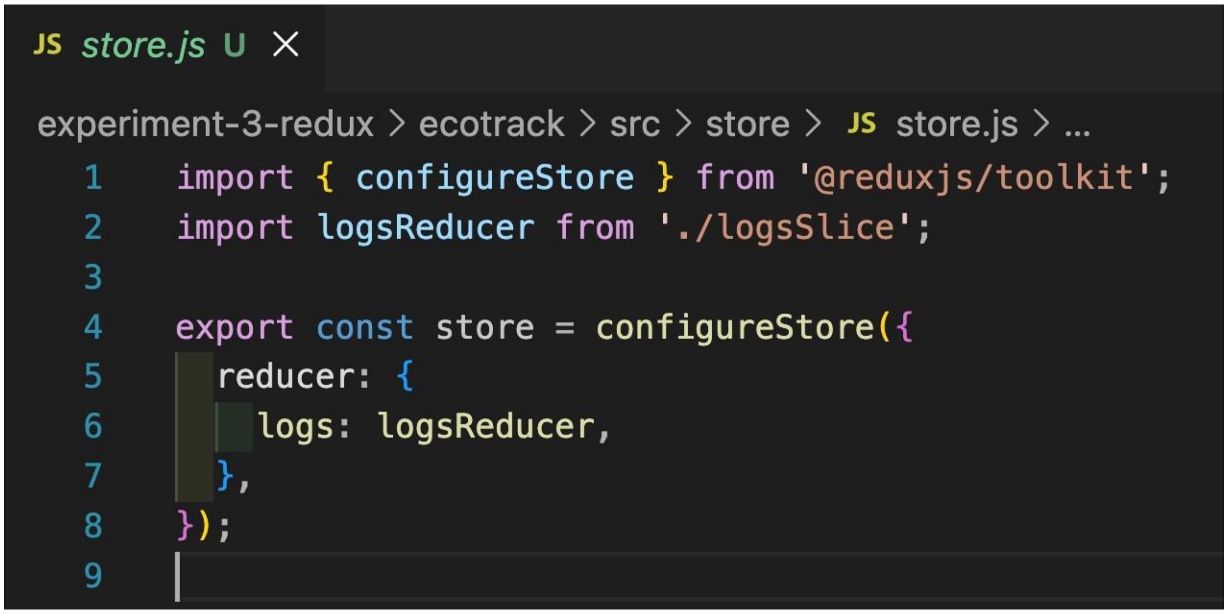
Sample Code Snippet:-



App.jsx U

experiment-3-redux > ecotrack > src > App.jsx > default

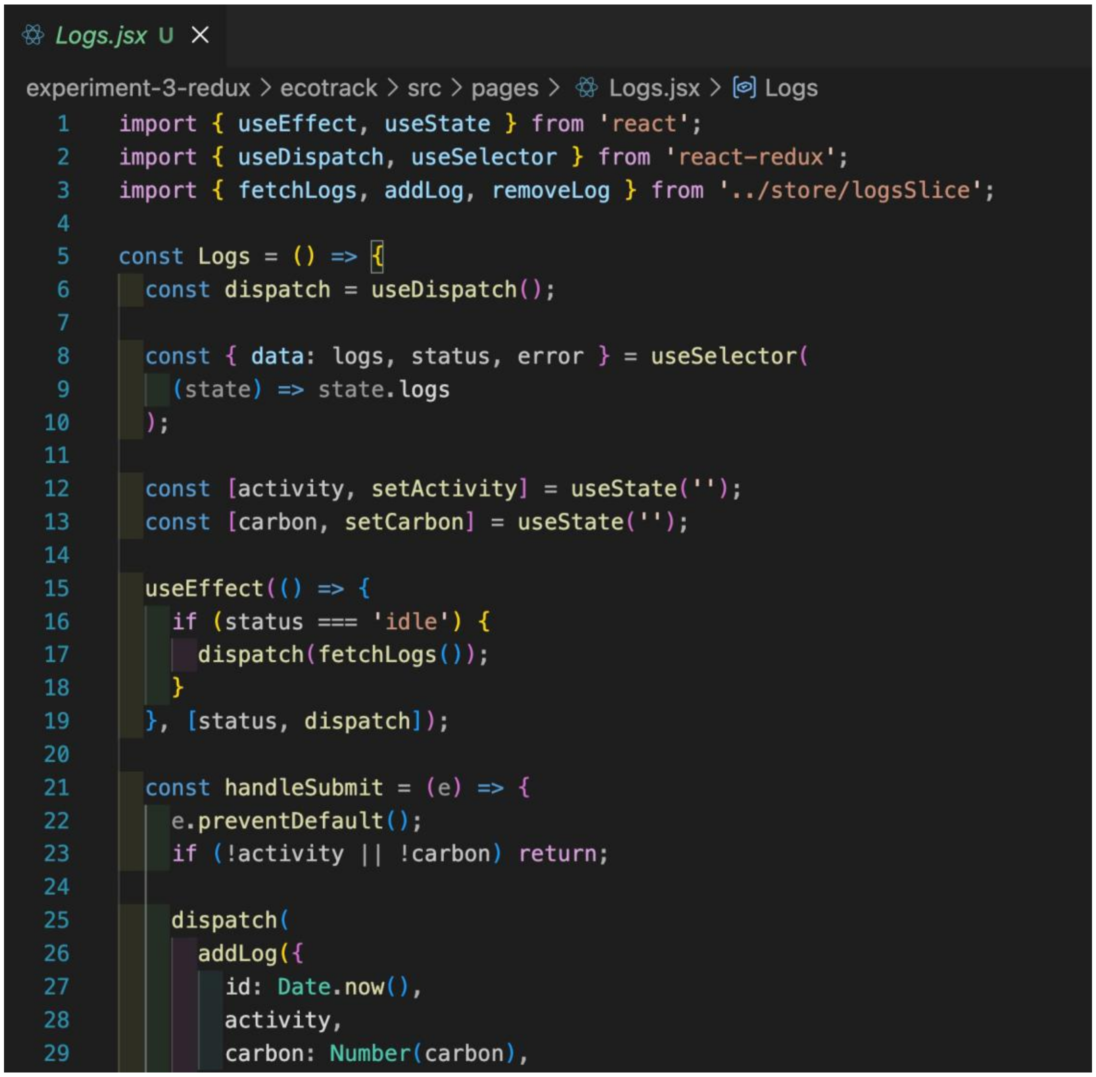
```
import Logs from './pages/Logs';


function App() {
  return (
    <div>
      <h1>EcoTrack (Experiment 3)</h1>
      <Logs />
    </div>
  );
}

export default App;

```



store.js U ×

experiment-3-redux > ecotrack > src > store > store.js > ...

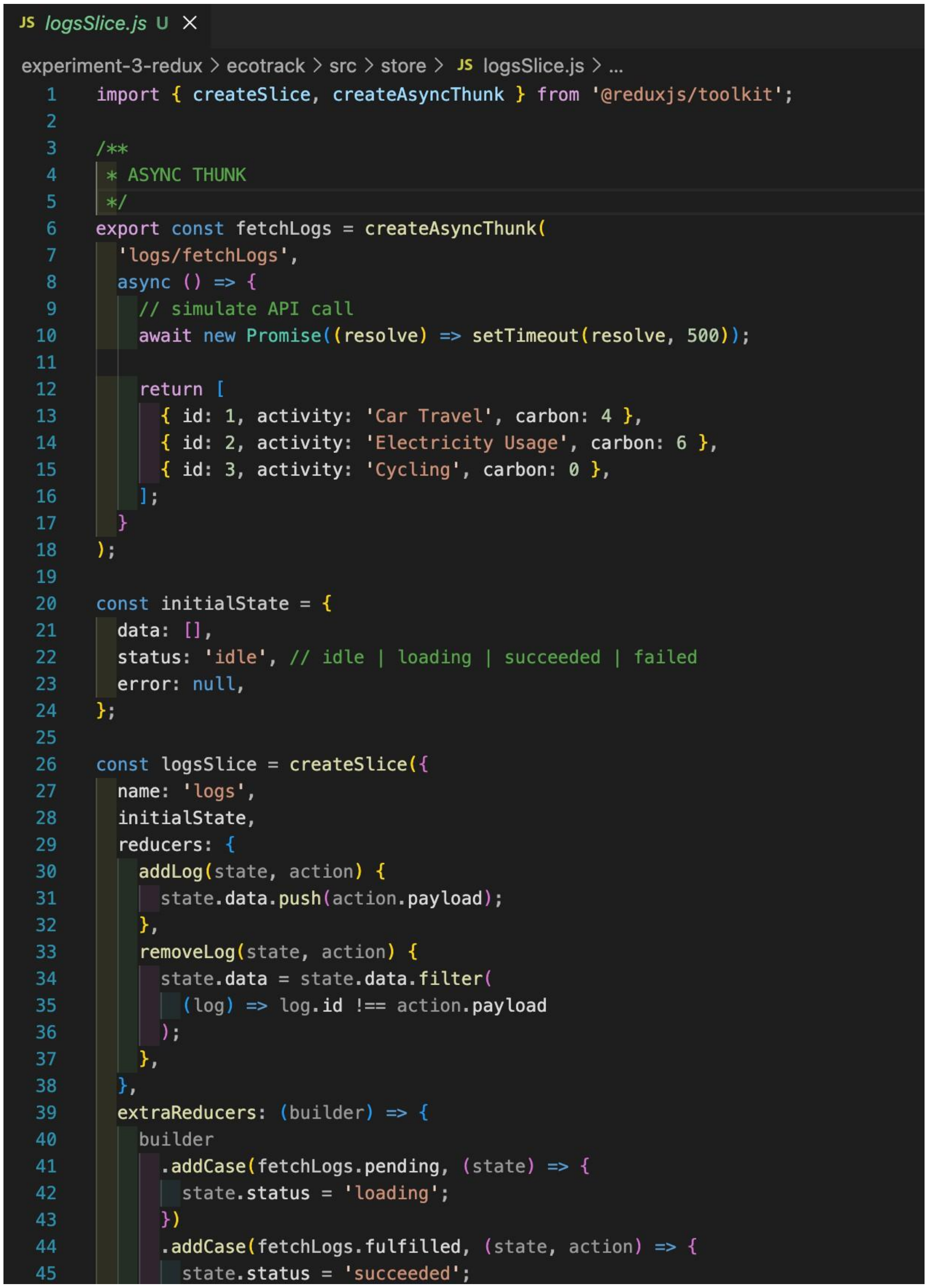
```
import { configureStore } from '@reduxjs/toolkit';
import logsReducer from './logsSlice';

export const store = configureStore({
  reducer: {
    logs: logsReducer,
  },
});

```



Logs.jsx U ×

experiment-3-redux > ecotrack > src > pages > Logs.jsx > Logs

```
import { useEffect, useState } from 'react';
import { useDispatch, useSelector } from 'react-redux';
import { fetchLogs, addLog, removeLog } from '../store/logsSlice';

const Logs = () => {
  const dispatch = useDispatch();

  const { data: logs, status, error } = useSelector(
    (state) => state.logs
  );

  const [activity, setActivity] = useState('');
  const [carbon, setCarbon] = useState('');

  useEffect(() => {
    if (status === 'idle') {
      dispatch(fetchLogs());
    }
  }, [status, dispatch]);

  const handleSubmit = (e) => {
    e.preventDefault();
    if (!activity || !carbon) return;

    dispatch(
      addLog({
        id: Date.now(),
        activity,
        carbon: Number(carbon),
```



logsSlice.js U ×

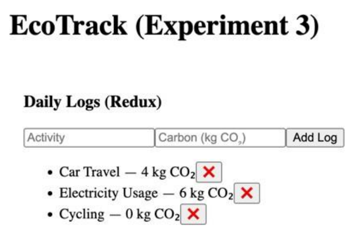
experiment-3-redux > ecotrack > src > store > logsSlice.js > ...

```
import { createSlice, createAsyncThunk } from '@reduxjs/toolkit';

/**
 * ASYNC THUNK
 */
export const fetchLogs = createAsyncThunk(
  'logs/fetchLogs',
  async () => {
    // simulate API call
    await new Promise((resolve) => setTimeout(resolve, 500));

    return [
      { id: 1, activity: 'Car Travel', carbon: 4 },
      { id: 2, activity: 'Electricity Usage', carbon: 6 },
      { id: 3, activity: 'Cycling', carbon: 0 },
    ];
  }
);

const initialState = {
  data: [],
  status: 'idle', // idle | loading | succeeded | failed
  error: null,
};

const logsSlice = createSlice({
  name: 'logs',
  initialState,
  reducers: {
    addLog(state, action) {
      state.data.push(action.payload);
    },
    removeLog(state, action) {
      state.data = state.data.filter(
        (log) => log.id !== action.payload
      );
    },
  },
  extraReducers: (builder) => {
    builder
      .addCase(fetchLogs.pending, (state) => {
        state.status = 'loading';
      })
      .addCase(fetchLogs.fulfilled, (state, action) => {
        state.status = 'succeeded';
```

4. Output:

- The EcoTrack application successfully loads daily activity logs using Redux
- Logs are fetched asynchronously and displayed dynamically
- Users can add new environmental activities with carbon values

- Users can remove existing logs instantly
- State updates occur without page reload
- Redux ensures centralized and predictable state management
- UI remains responsive and synchronized with the store



EcoTrack (Experiment 3)

Daily Logs (Redux)

Activity | Carbon (kg CO₂) | Add Log

- Car Travel — 4 kg CO_2 ❌
- Electricity Usage — 6 kg CO_2 ❌
- Cycling — 0 kg CO_2 ❌

5. Learning Outcomes (What I Have Learnt):

- Understand Redux architecture and data flow
- Implement Redux Toolkit for efficient state management
- Use createAsyncThunk for asynchronous operations
- Manage global state using Redux store and slices
- Integrate Redux with React using hooks
- Perform add and delete operations on centralized state
- Build scalable and maintainable React applications
- Differentiate between Context API and Redux usage