Assignment 1

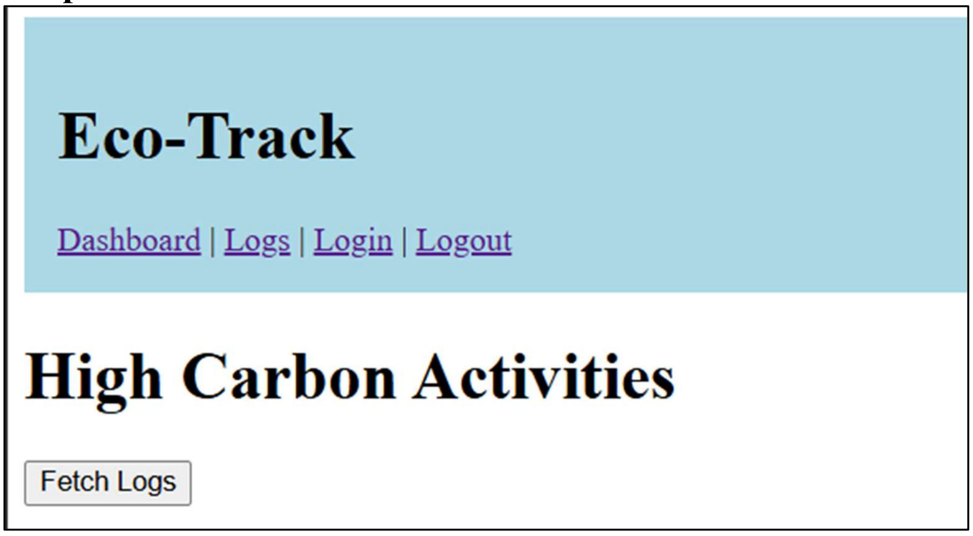
Student Name: Alisha Ameer	UID: 23BCC70017
Branch: AIT-CSE	Section: 23AIT_KRG-2 A
Semester: 6	Date of Performance: 30/01/2026
Subject Name: Full Stack II	Subject Code: 23CSH-382

- **Aim:**
  The aim of this assignment is to design and implement a Daily Water Intake Tracker feature within the EcoTrack React application. This includes incorporating routing and protected authentication, managing state, integrating an external API, enabling local storage persistence, and optimizing performance using React best practices such as React.memo, useCallback, and useMemo.
- **Objectives:**
  The primary objectives of this experiment are:
  1. To implement client-side navigation using React Router.
  2. To create protected routes with authentication handled through localStorage.
  3. To manage component state effectively using useState.
  4. To persist application data using useEffect and localStorage.
  5. To implement increment, decrement, and reset functionality.
  6. To display dynamic UI messages based on state conditions.
  7. To integrate a public API while handling loading and error states.
  8. To optimize component rendering using React.memo.
  9. To reduce unnecessary re-renders with useCallback and useMemo.

- **Implementation:**
  The **Daily Water Tracker** feature was developed using the following structured steps:
  1. **Navigation & User Access Control**
     - Set up routes: /login, /dashboard, and /dashboard/water.
     - Implemented a protected route to redirect unauthenticated users to the login page.
     - A mock login stores a token in localStorage, while logout removes it.

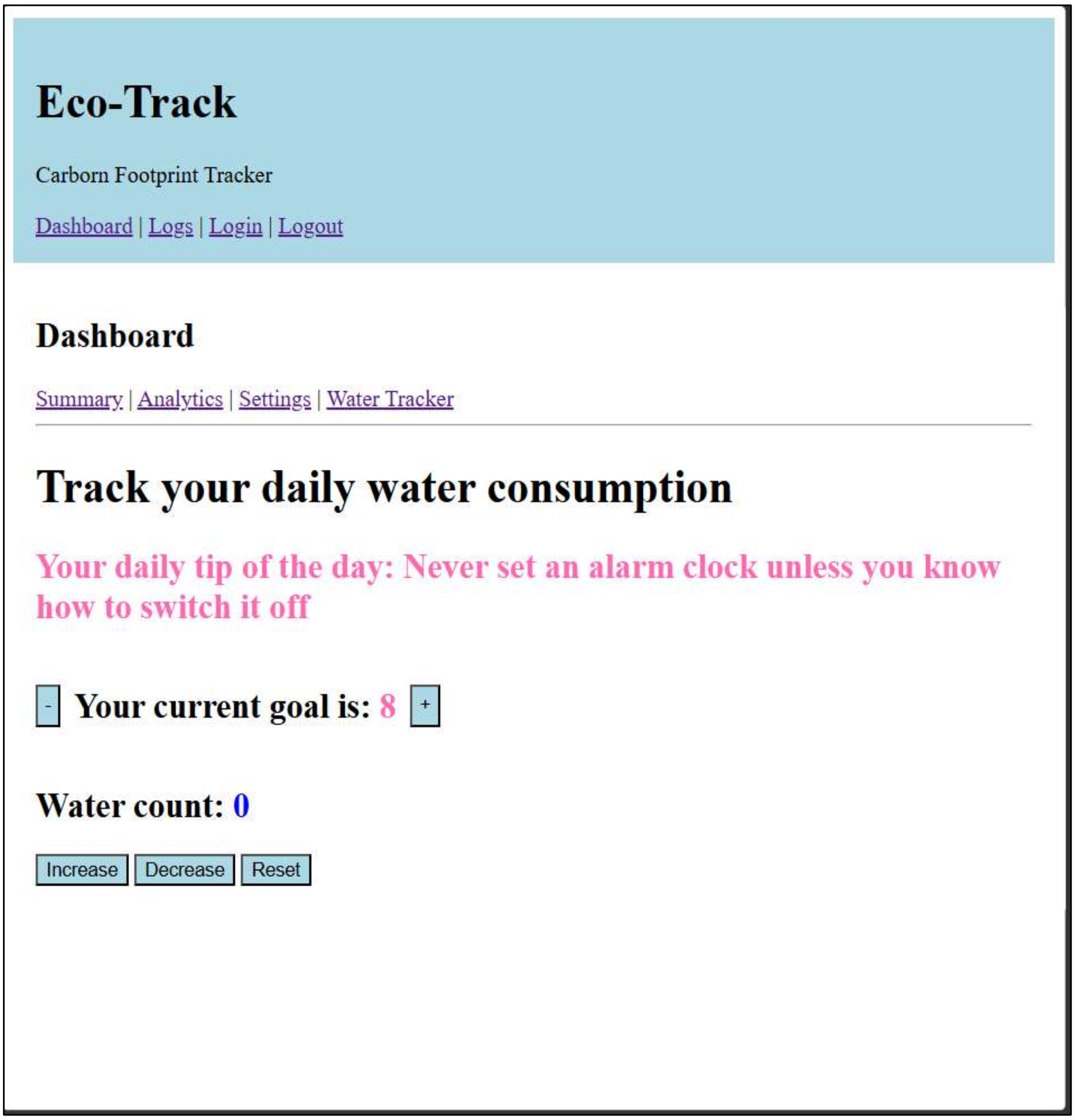
2. **Managing Tracker State**
   - Used useState to manage **count** (default: 0) and **goal** (default: 8).
   - Added functionality to **Add**, **Remove**, **Reset**, and **Save Goal**.
   - Displayed progress text and showed a **"Goal Reached"** message when the count meets or exceeds the goal.
3. **Saving Data Locally**
   - Used useEffect to save the count in localStorage.
   - Retrieved the saved value when the page refreshes.
4. **Fetching Daily Health Tips**
   - Retrieved a health tip from https://api.adviceslip.com/advice.
   - Implemented loading and error handling states.
   - Displayed the result as **"Today's Health Tip."**
5. **Improving Performance & Rendering**
   - Created a separate <CounterDisplay /> component.
   - Optimized rendering using React.memo.
   - Applied useCallback and useMemo to prevent unnecessary re-renders.

- **Output:**


Eco-Track

Dashboard | Logs | Login | Logout

High Carbon Activities



Eco-Track

Carborn Footprint Tracker

Dashboard | Logs | Login | Logout

Dashboard

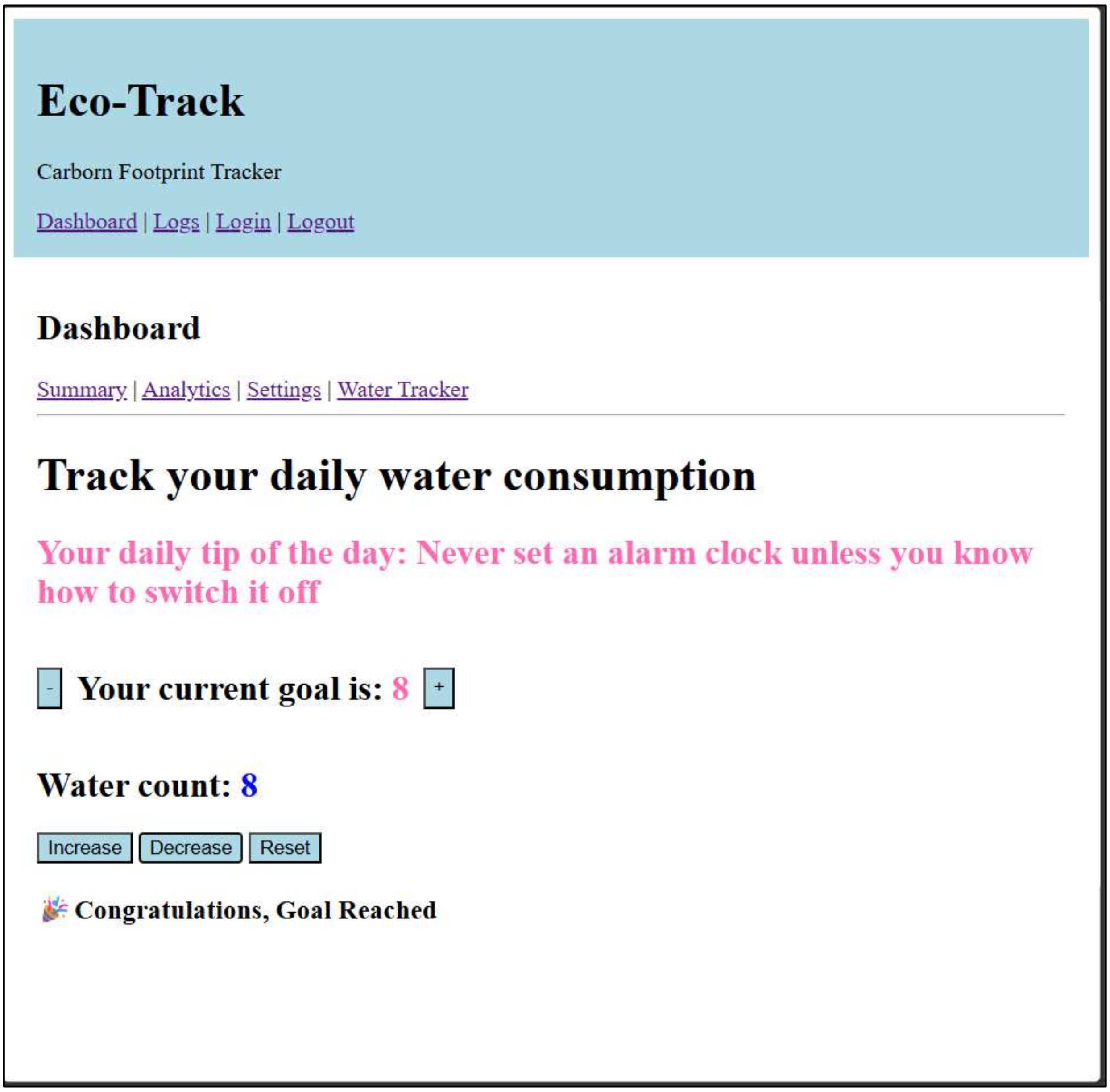
Summary | Analytics | Settings | Water Tracker

Track your daily water consumption

Your daily tip of the day: Never set an alarm clock unless you know how to switch it off

Your current goal is: 8

Water count: 0


Eco-Track

Carborn Footprint Tracker

Dashboard | Logs | Login | Logout

Dashboard

Summary | Analytics | Settings | Water Tracker

Track your daily water consumption

Your daily tip of the day: Never set an alarm clock unless you know how to switch it off

- Your current goal is: 8 +

Water count: 8

Increase Decrease Reset

🎉 Congratulations, Goal Reached

- **Results:**
  The EcoTrack Daily Water Tracker feature was successfully implemented with routing, protected authentication, state persistence, API integration, and performance optimization. The application effectively redirects unauthorized users, maintains state across page refreshes using localStorage, and displays dynamic messages based on goal progress. It also fetches and shows daily health tips with proper error handling. Additionally, performance is improved by

preventing unnecessary re-renders through the use of **React.memo** and **useCallback**. Overall, this experiment improved the application's usability, responsiveness, and efficiency while adhering to modern React development best practices.

- **Learning Outcomes:**
  After completing this experiment, I have learnt to:
  1. Implement protected routing using React Router.
  2. Manage state efficiently with useState and useEffect.
  3. Persist application data using localStorage.
  4. Render UI conditionally based on state changes.
  5. Integrate third-party APIs with proper loading and error handling.
  6. Optimize React applications using React.memo.
  7. Use useCallback and useMemo to prevent unnecessary re-renders.