**Project Report: To-Do Studio**



**Course:** Front End UI/UX Development

**Project Title:** To-Do Studio: An Immersive & Calm Productivity Application

**Submitted by:** Shreya Sunil Morajkar 2460450

**1. Abstract**

**To-Do Studio** is a fully client-side web application designed to provide a calming and immersive productivity environment. It reinvents the traditional to-do list by integrating a unique **ambient mood player** with soundscapes generated programmatically via the **Web Audio API**. The application features a complete user authentication system, full task management (CRUD) capabilities, and data persistence using the browser's localStorage. Built with **Vanilla JavaScript**, it demonstrates a strong command of core web fundamentals while prioritizing a minimalist design and a user-centric, focused experience.

**2. Introduction**

**2.1. Problem Statement**

In an era of digital saturation, many productivity tools are cluttered, complex, and contribute to the very stress they aim to alleviate. Users often face a steep learning curve or are distracted by excessive features. There is a clear need for a simple, focused, and pleasant application that helps users manage tasks without adding to their cognitive load. The "To-Do Studio" project was conceived to address this gap.

**2.2. Project Objectives**

* To develop a fully functional, client-side to-do list application using Vanilla JavaScript.
* To design and implement a unique user experience centered around a "calm and immersive" theme.
* To create a secure, multi-user environment using browser localStorage for data persistence.
* To integrate the Web Audio API to generate ambient soundscapes without relying on external audio files.

**3. System Architecture and Design**

**3.1. Architecture**

The application is architected as a **fully client-side, serverless application**. All logic, data storage, and rendering are handled directly within the user's web browser. This design choice was made to prioritize user privacy, ensure offline functionality (once the page is loaded), and simplify deployment, as no backend infrastructure is required.

**3.2. Data Persistence**

Data persistence is achieved using the browser's **localStorage API**. A single JSON object stores all registered users, and each user object contains their credentials, task list, and mood preferences.

* **Justification:** localStorage was chosen for its simplicity and for keeping all user data on their own machine, which enhances privacy. While this prevents cross-device synchronization, it was deemed an acceptable trade-off for the scope of this project.

**4. Technology Stack**

The technologies for this project were selected to build a lightweight, performant, and modern web application without reliance on heavy frameworks.

* **HTML5:** For the semantic structure of the application.
* **CSS3:** For all custom styling, including animations and the "glassmorphism" visual effect.
* **Bootstrap 5:** Used for its responsive grid system and pre-styled components to ensure a consistent and mobile-friendly layout.
* **Vanilla JavaScript (ES6+):** Chosen over frameworks like React or jQuery to demonstrate a strong command of core web fundamentals, including DOM manipulation, event handling, and manual state management. This results in a faster application with no external script dependencies.
* **Web Audio API:** A crucial component for generating the ambient soundscapes. This browser-native API provides the power to create and manipulate audio programmatically, which is central to the app's unique user experience.

**5. Implementation and Key Features**

**5.1. Design and User Experience (UX) Philosophy**

A core goal of this project was to create a standout user experience. Our UX philosophy was centered on the concept of **"calm productivity."** Every design choice was intentional:

* **Visuals:** The "glassmorphism" effect (backdrop-filter), soft shadows, and a gentle color palette were used to create a clean, modern, and non-distracting interface.
* **Microinteractions:** Subtle animations, such as the popIn effect when adding a task, provide positive feedback and make the application feel more responsive and delightful to use.

**5.2. User Authentication**

The application supports multiple users. The system handles registration and login, ensuring that each user's tasks and preferences are stored separately and securely within localStorage. A session management system keeps users logged in between visits.

**5.3. Task Management**

Full CRUD (Create, Read, Update, Delete) functionality is implemented for tasks. Users can add new tasks, edit existing ones by double-clicking, mark them as complete, and delete them. The interface also includes filters to view all, active, or completed tasks.

**5.4. Technical Deep Dive: The Ambient Mood Player**

The most unique feature is the ambient sound engine, built with the Web Audio API. Instead of using bulky audio files, sound is generated in real-time.

* **How it Works:** The engine uses an AudioContext as its foundation. Sounds like "rain" are created by generating white noise with a BufferSource, which is then passed through BiquadFilterNodes (low-pass and high-pass) to shape the sound and make it sound natural. An OscillatorNode and GainNode are used to create other tones and control volume envelopes, resulting in a dynamic and non-repetitive soundscape.

JavaScript

// Simplified example of creating filtered noise for rain

const noise = ctx.createBufferSource();

const lowpassFilter = ctx.createBiquadFilter();

lowpassFilter.type = 'lowpass';

lowpassFilter.frequency.value = 6000;

const gain = ctx.createGain();

gain.gain.value = 0.06;

noise.connect(lowpassFilter);

lowpassFilter.connect(gain);

gain.connect(masterGain);

noise.start();

**6. Challenges and Solutions**

* **Challenge:** Managing the application's state (current user, tasks, filters) in Vanilla JavaScript without a library like Redux.
* **Solution:** We implemented a simple state management pattern. A central currentUser object holds all session data. A single persistCurrentUser() function is called after any data modification, which updates localStorage. The UI is then re-rendered by a master renderTasks() function, ensuring the view is always synchronized with the stored state.

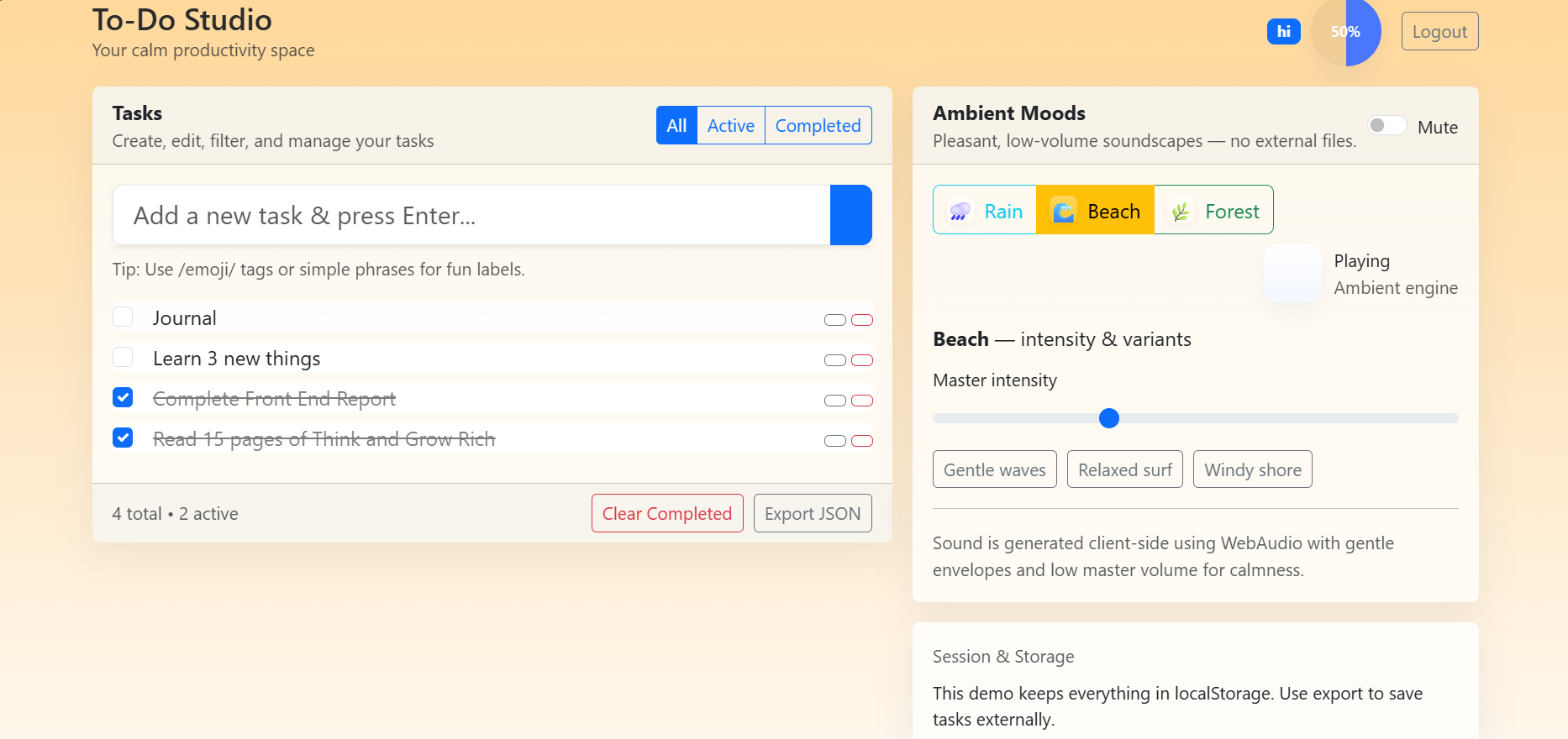
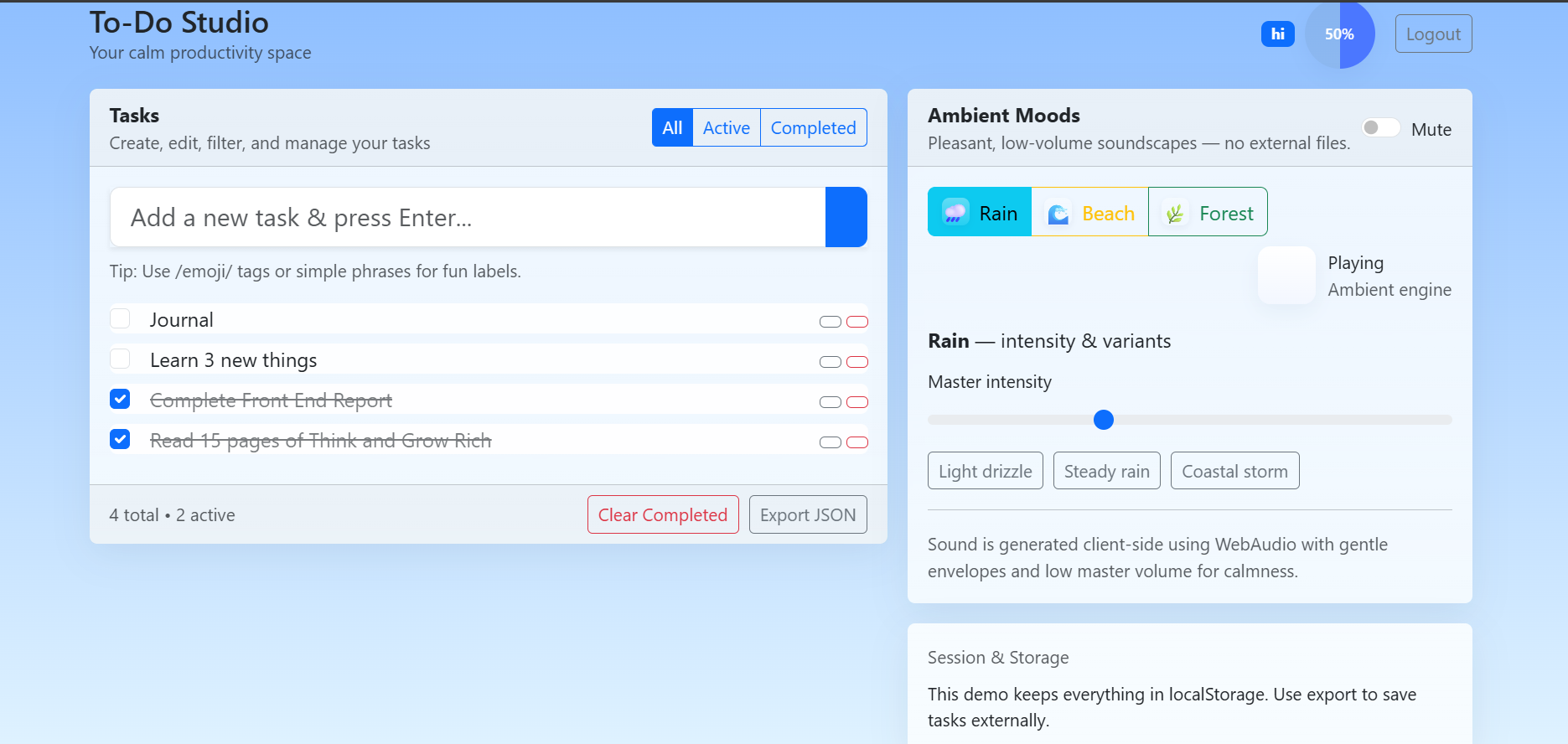
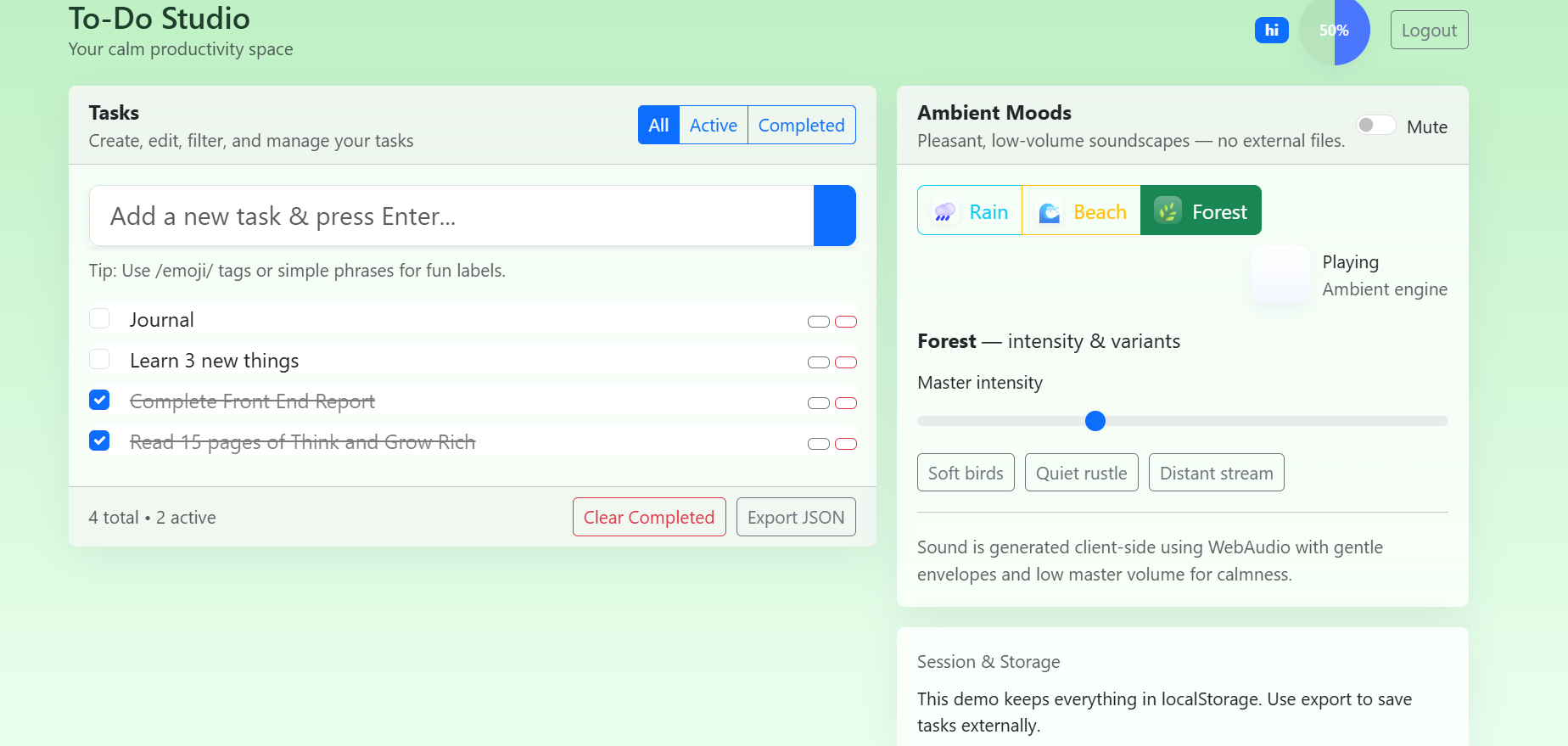
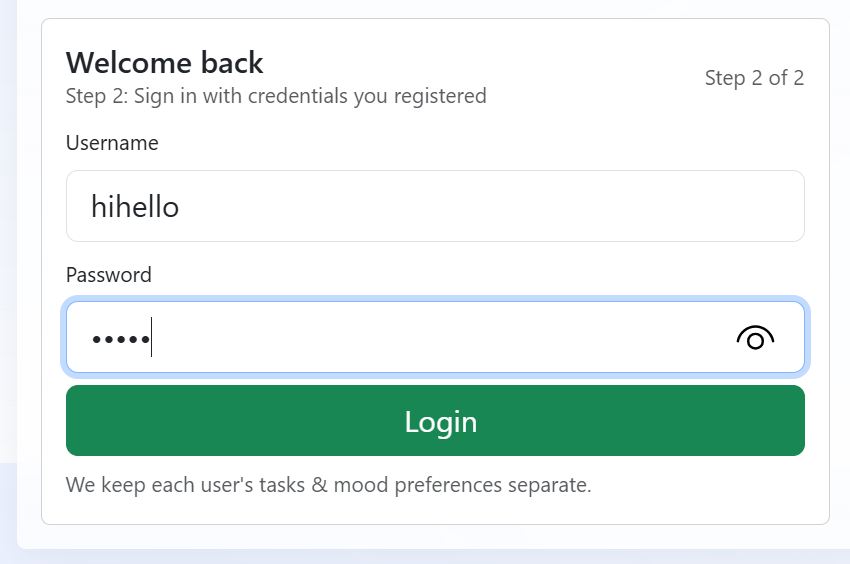
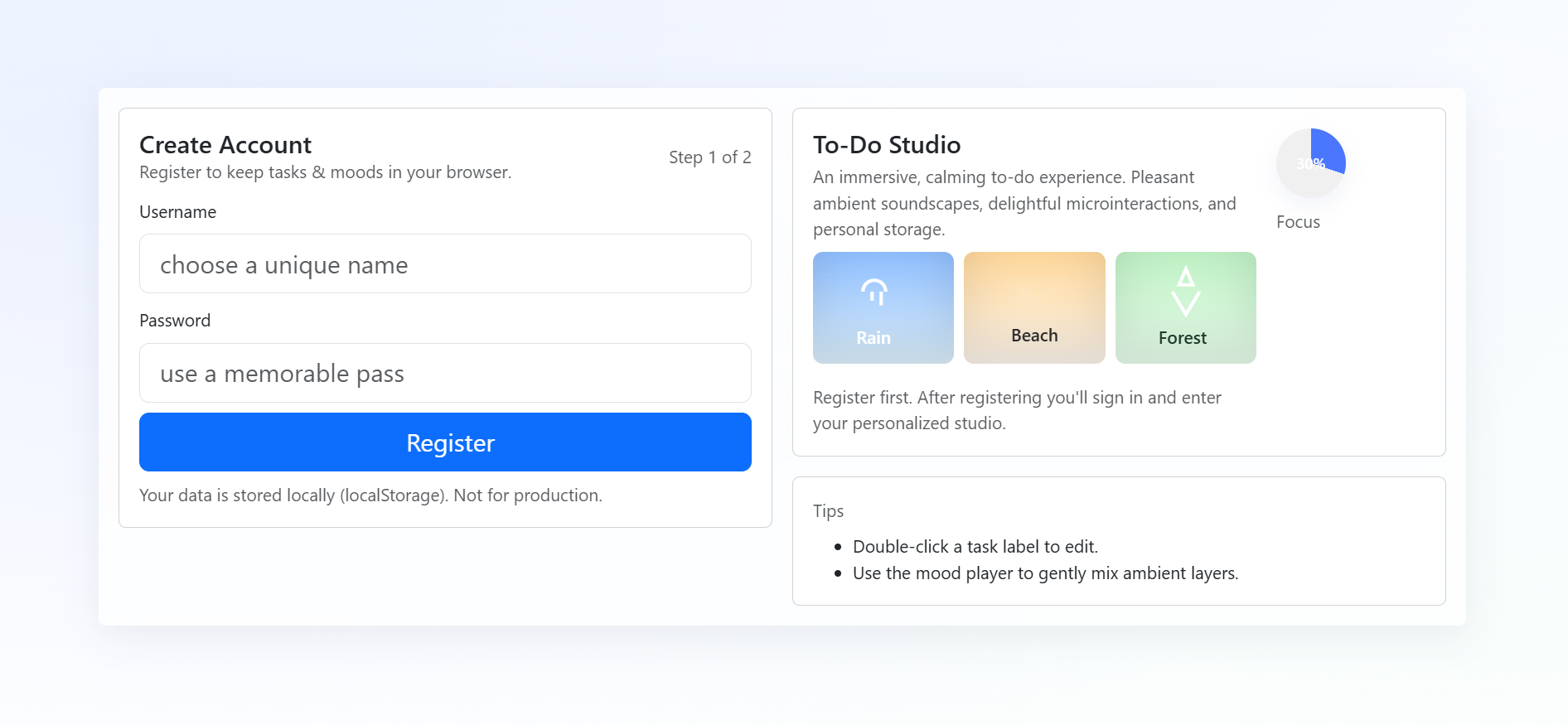
**7. Results and Screenshots**

The final application successfully meets all project objectives. It is a functional, visually pleasing, and unique to-do list application.

*(Here, you would insert annotated screenshots of your application, for example:)*

* **Screenshot 1:** The registration/login screen.
* **Screenshot 2:** The main task view, with an arrow pointing to the real-time progress ring and a note explaining its function.
* **Screenshot 3:** The mood player controls, with a callout explaining the function of the intensity slider.

8. Output Screenshots



**9. Conclusion and Future Scope**

**9.1. Conclusion**

The "To-Do Studio" project successfully demonstrates the creation of a modern, feature-rich web application using fundamental, client-side technologies. It proves that a compelling user experience can be achieved without reliance on heavy frameworks, and showcases the creative potential of browser APIs like the Web Audio API.

**9.2. Future Scope**

* **Cloud Sync:** The next logical step would be to add an optional backend service (e.g., using Node.js and a database) to allow for cross-device synchronization of tasks.
* **More Moods:** The audio engine could be expanded to include more complex and varied soundscapes.
* **Web App Manifest:** Implementing a manifest file to make the application a Progressive Web App (PWA), allowing users to "install" it on their devices.