3-2 Mini Project — Self Assessment application

Status	In Progress
Due Date	@April 26, 2025
Priority	High
Task List	1. prepare a basic outline of project 2. create a most suitable tech-stack 3. first initialize with frontend 4. start on backend such as language and database.
Assigned To	Ⓚ Koushik Marelly
Progress	0
Notes	project still not started and only abstract is done . After two weeks there is presentation on project.

Below is a practical, step-by-step guide crafted for someone starting from zero coding background. This plan will help you read, understand, and modify both the frontend and backend parts of your project to achieve the basic working features within a week.

---

1. Set Up Your Development Environment

a. Install Essential Tools

- **Node.js & npm:**

  Your project uses Node.js to run JavaScript. Install the latest LTS version from nodejs.org.

- **Code Editor:**

Use a beginner-friendly editor like Visual Studio Code (VS Code), which offers helpful features (syntax highlighting, autocomplete, extensions).

- **Git:**

  Although not mandatory initially, installing Git can help you track changes and revert mistakes later. Learn basics from Git documentation.

b. Run the Existing Project

1. **Open the Project Folder in VS Code:**

   Use VS Code's file explorer to navigate through your project structure.

2. **Install Dependencies:**

   In the terminal (inside VS Code), run:

   ```
   npm install
   ```

   This command installs all packages listed in `package.json`.

3. **Start the Development Server:**

   Once installation is complete, run:

   ```
   npm run dev
   ```

   This will launch your Vite development server. You should see your application in the browser.

Tip: These commands give you live feedback; every time you save your code changes, the browser reloads.

---

2. Familiarize Yourself with the Project Structure

Understanding the project's folders and files is key:

Frontend (Client-Side)

- `src/`:

  Contains the main code.

  - `components/`:

Reusable UI pieces like `LoginForm.tsx`, `DiaryEditor.tsx`, and a UI library in `components/ui/`.

How to read: Open a component file and look at JSX (HTML-like syntax) and TypeScript code. Notice how props and state are defined.

- **`pages/`:**

  These are the views or pages of your application (e.g., `Dashboard.tsx`, `Profile.tsx`).

  How to read: These components assemble various UI components and manage routing.

- **`context/`:**

  Global state management (e.g., `AuthContext.tsx` for user authentication).

- **`hooks/`:**

  Custom hooks used throughout your project; these encapsulate reusable logic.

- **`integrations/supabase/`:**

  Code for backend integration. Start here if you want to see how your app communicates with Supabase.

Backend (Server & Database Integration)

- **Supabase:**

  Your app uses Supabase as a backend service for authentication, database, and possibly file storage.

  - **`supabase/config.toml`:**

    Configuration details for connecting to Supabase.

  - **`integrations/supabase/client.ts`:**

    Initializes and configures the connection to Supabase.

Tip: Don't be intimidated by the file names and extensions (like `.tsx` for React with TypeScript). They mostly follow similar patterns to HTML/JavaScript with additional type information.

---

3. Begin Learning the Technologies Step-by-Step

a. HTML, CSS, and Basic JavaScript

- **HTML & CSS:**

  Understand the basic structure of web pages.

  - **What to Focus On:**

    HTML tags, CSS selectors, and how styles affect your UI.

  - **Resource:**

    MDN Web Docs: HTML and CSS.

- **JavaScript (ES6):**

  Learn variables, functions, and basic programming logic.

  - **Resource:**

    MDN JavaScript Guide.

Task: Build a simple static HTML page with some CSS styling to get comfortable.

b. React Basics

- **React Fundamentals:**

  Learn what components are, how to create them, and understand JSX.

  - **Key Concepts:**

    Components, props, state, and the component lifecycle.

  - **Resource:**

    React Official Tutorial.

- **Hands-On:**

  Walk through a simple React tutorial (for instance, build a counter app).

  How It Relates to Your Project:

  Files like `App.tsx` and `LandingHero.tsx` are React components that follow these concepts.

c. TypeScript Essentials

- **Why TypeScript:**

  It adds strict types to your JavaScript code to help catch errors early.

- **Learning Resources:**

  TypeScript Handbook.

- **Key Concepts:**

  Types, interfaces, and how to annotate functions/components.

Task: Update a simple React component with explicit types. For example, add type definitions to a state variable in `LoginForm.tsx`.

d. Tailwind CSS

- **Understanding Utility-First CSS:**

  Learn how Tailwind classes (like `p-4`, `bg-blue-500`, etc.) directly style elements.

  - **Resource:**

    Tailwind CSS Documentation.

Task: Make a visual change in a component (e.g., change button colors or add spacing) to see Tailwind in action.

---

4. Understand the Application's Flow

To make useful changes, know how the application works as a whole:

a. Frontend Navigation and Pages

- **Routing:**

  Even if you don't see explicit routing code, your pages (inside `src/pages/`) are likely connected using a routing library like React Router.

  How It Works:

  The navigation (possibly handled by `DiaryNav.tsx`) links to different pages (e.g., Dashboard, Profile).

b. Global State & Authentication

- **AuthContext:**

  Read `src/context/AuthContext.tsx` to see how the app manages user state and authentication.

  Key Concepts:

Using React Context to share authentication status across components.

c. Supabase Integration (Backend)

- **Connecting to Supabase:**

  Open `src/integrations/supabase/client.ts` to see how your app connects to a Supabase backend.

  What to Look For:

  The API keys, endpoints, and methods that fetch or post data.

- **User Operations:**

  Check the `LoginForm.tsx` and `SignupForm.tsx` to see how user authentication is implemented.

Tip: Look for comments in the code—these can be very useful to understand what each part does.

---

5. Plan to Make Incremental Changes

Since you have a deadline, focus on small, measurable steps:

Day 1-2: Environment & Basic Reading

- Set up your environment and run the project.
- Read through the Readme (if available) and major components (like `App.tsx`, `LandingHero.tsx`).
- Make a small change (e.g., update a headline or button text) and verify it in the browser.

Day 3-4: Learn and Experiment

- Work through a simple React tutorial to get comfortable with components, state, and props.
- Experiment with Tailwind CSS by modifying a component's style.
- Add a console log in a component to see how data flows.

Day 5: Understand Authentication & Supabase

- Study the `AuthContext.tsx` and basic Supabase integration in `client.ts`.

- Try simulating a login/logout flow by editing the login form. Use debugging (e.g., `console.log`) to track what happens when the user logs in.

Day 6-7: Build a Basic Feature

- Implement a basic diary entry feature:
  - **Creation:** Work in `DiaryEditor.tsx` to allow entering text.
  - **Display:** Use `Dashboard.tsx` or another page to fetch and display entries (you might mock the data if Supabase integration is challenging at first).
- Test every change and ensure everything works as expected.

Backend diagrams needed:

Diagram Type	Purpose	Notes
ER Diagram	Database schema & relationships	Important for Supabase DB
Component Diagram	Backend services & their interactions	Supabase services + API client
Sequence Diagram	Interaction flow over time	Signup/login, diary operations
Deployment Diagram	Physical deployment environment	Cloud + frontend hosting
Data Flow Diagram	Data movement through backend processes	API request/response flow
API Architecture	Overview of backend API endpoints	List endpoints and their roles

Tech-Stack Used:

Great! Here's a clean **Tech Stack** overview for your project `saikoushik08-ethereal-diary-keeper` based on the directory and file types you shared:

Tech Stack for Ethereal Diary Keeper

Frontend:

- **React** with **TypeScript** (tsx files)
- **Vite** (build tool)
- **Tailwind CSS** (styling)
- **React Context API** (state management - AuthContext)
- **React hooks** (custom hooks like `use-toast`, `use-mobile`)
- UI components built from scratch + third-party libs (e.g., `tailwindcss-animate`)

Backend:

- **Supabase** (PostgreSQL database + authentication + backend services)
- Supabase client and types in `src/lib/supabase/`

Database:

- PostgreSQL (managed by Supabase)
- Tables and schema configured in Supabase dashboard (not visible here)

API:

- Supabase APIs (for auth, database, storage)
- REST/GraphQL layer abstracted by Supabase SDK

Other tools/config:

- ESLint (linting)
- TypeScript config (`tsconfig.json` files)
- PostCSS (for Tailwind processing)
- Git (version control)

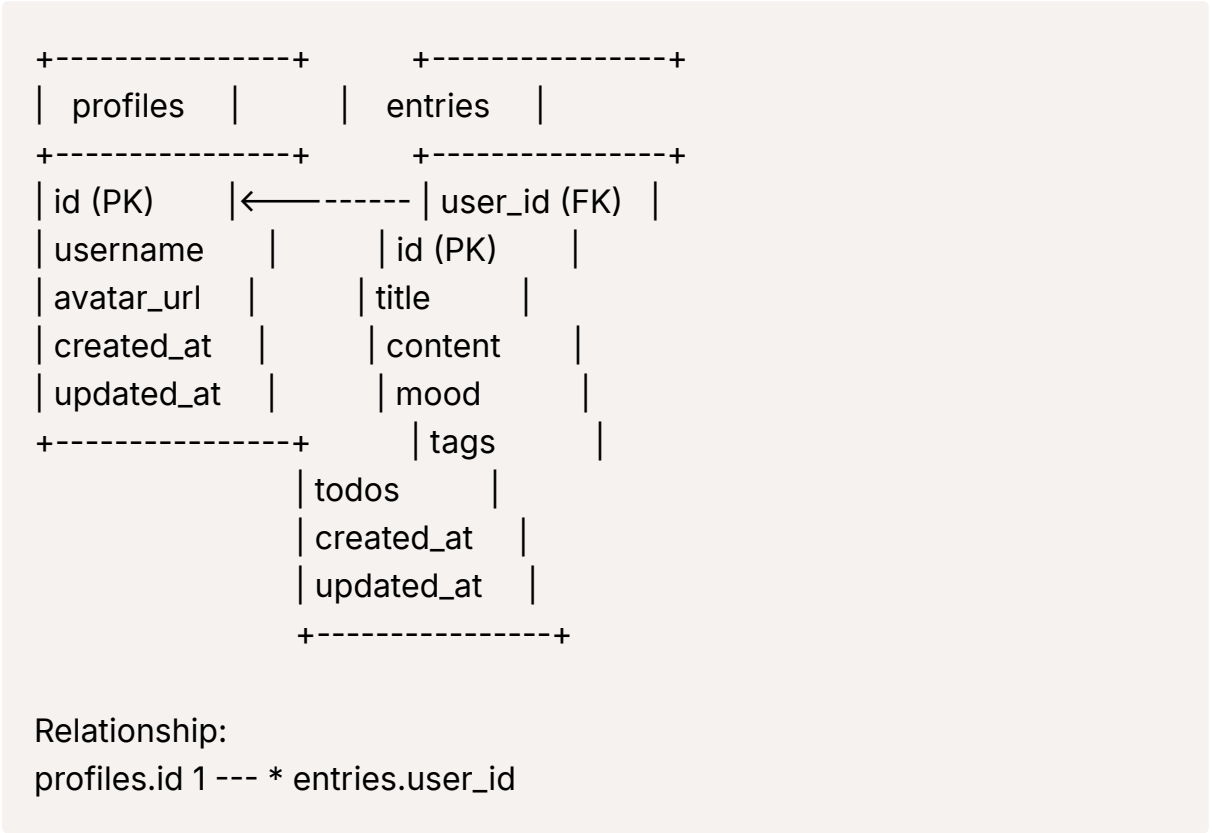
Next Step: Break down the project into

- Frontend modules & components
- Backend services & APIs
- Database schema overview

Backend Diagrams:

E-R Diagram:

Got it! Here's the ER diagram in ASCII art style based on your schema:



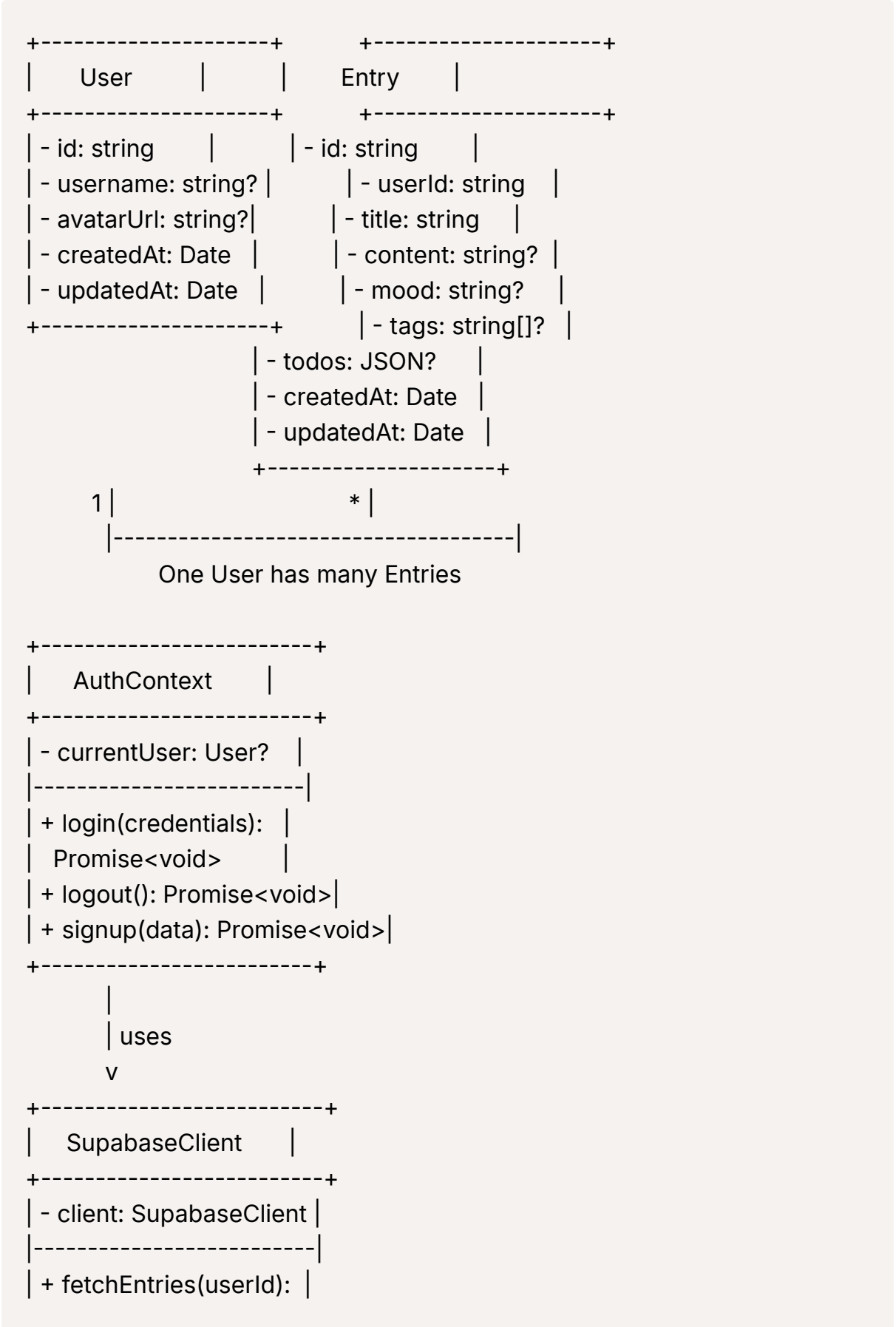
Relationship:
profiles.id 1 --- * entries.user_id

UML Diagrams:

Class Diagram:

Perfect! Here's both the **ASCII UML class diagram** and a **visual UML class diagram** based on the conceptual classes from your project:

ASCII UML Class Diagram



```
|  Promise<Entry[]>      |
| + fetchUserProfile(userId):|
|  Promise<User>         |
| + createEntry(entryData):|
|  Promise<Entry>        |
| + updateEntry(entryId,  |
|  updates): Promise<Entry>|
| + deleteEntry(entryId):  |
|  Promise<void>         |
+-------------------------+
```

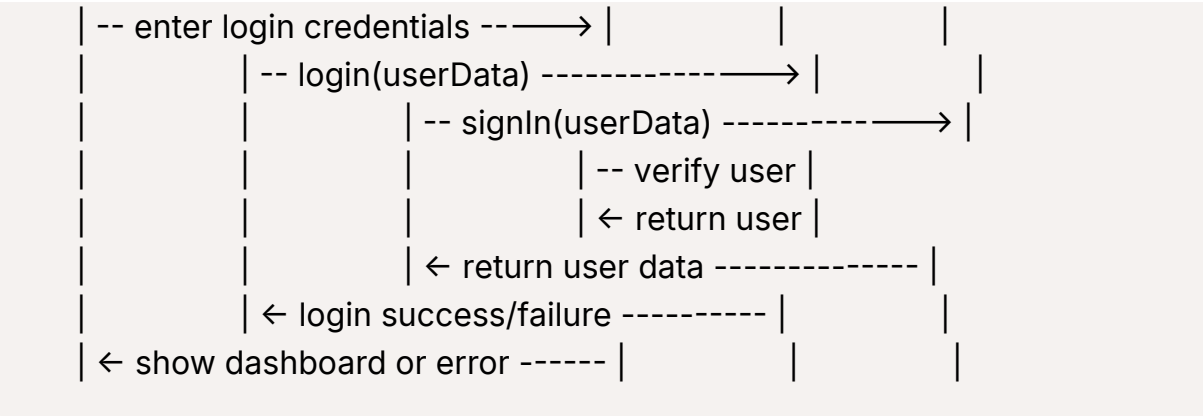
Sequence Diagrams:

Here's the **ASCII-style Sequence Diagram** for the **Signup Flow**:

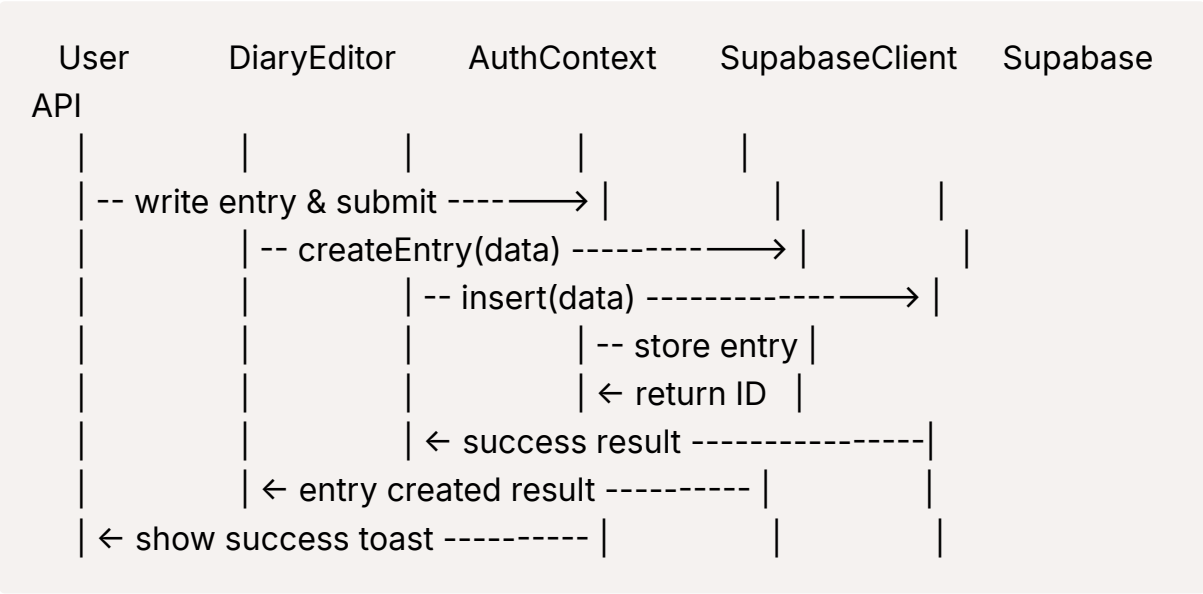
```
  User        SignupForm      AuthContext     SupabaseClient    Supabase API
   |             |              |               |                |
   | -- enter signup details ------⟶ |              |              |
   |             | -- register(userData) --------⟶ |              |
   |             |              | -- signUp(userData) ----------⟶ |
   |             |              |               | -- create user |
   |             |              |               | ← return user |
   |             |              | ← return user data -------------- |
   |             | ← registration success/failure ----|             |
   | ← show confirmation ------------ |              |              |
   |             |              |               |                |
```

1. User Login Flow

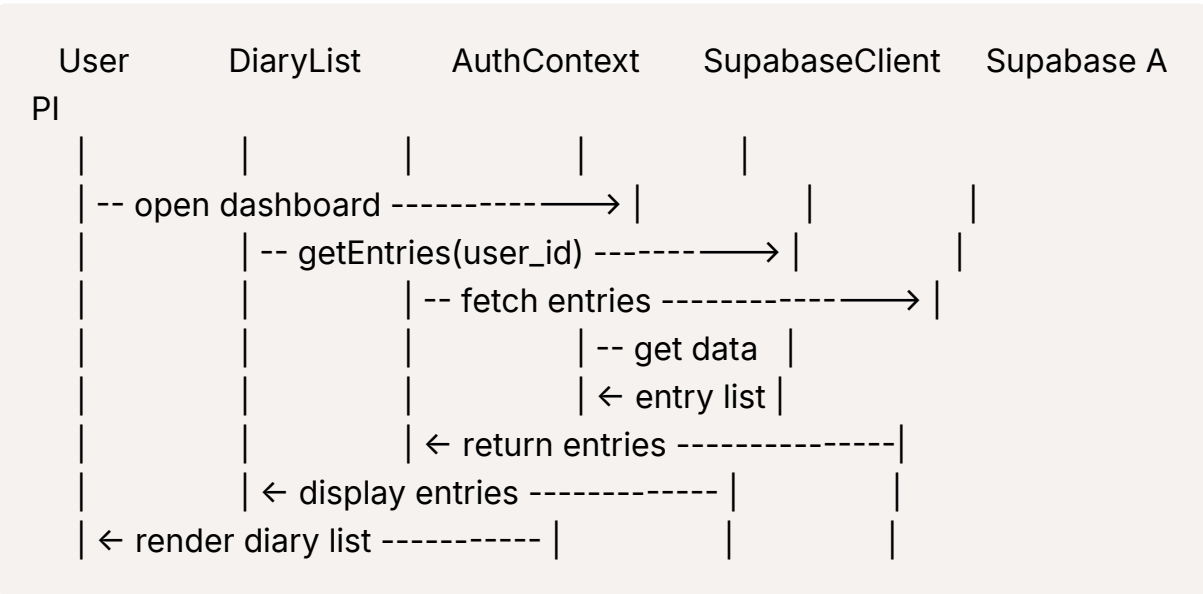
```
  User        LoginForm       AuthContext     SupabaseClient    Supabase API
   |             |              |               |                |
```



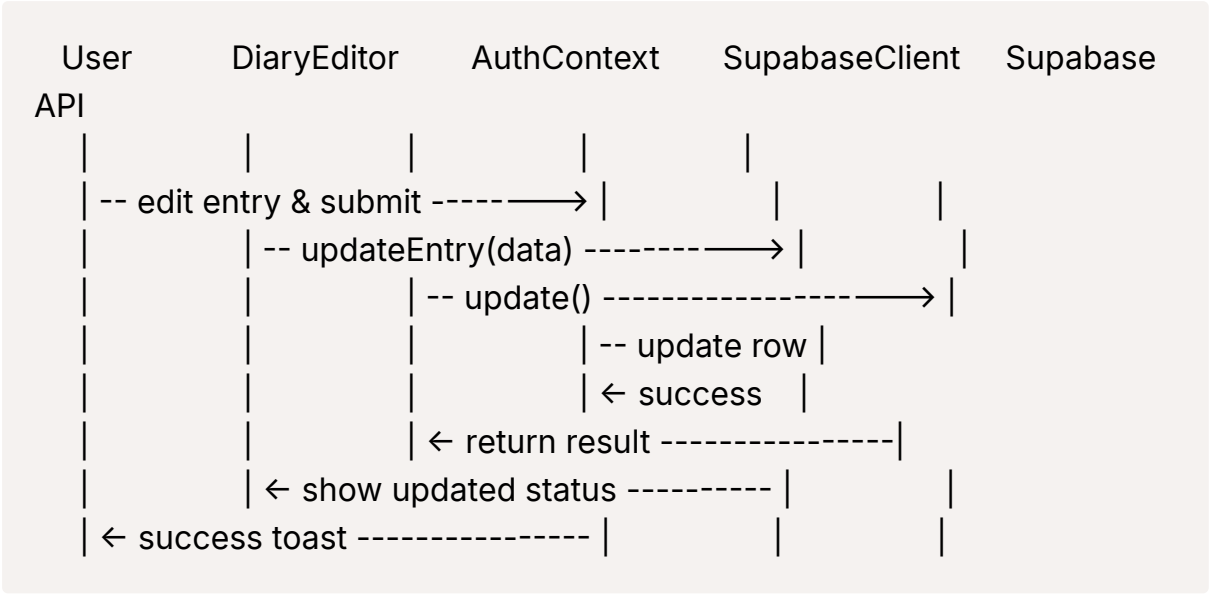
2. Create Diary Entry



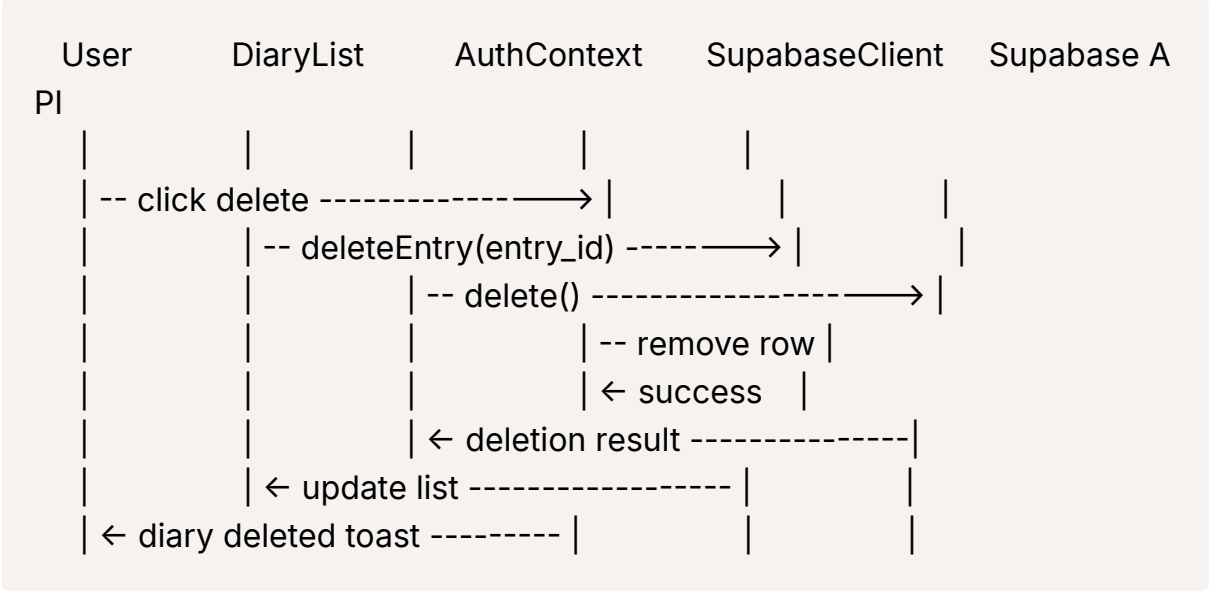
3. Fetch Diary Entries



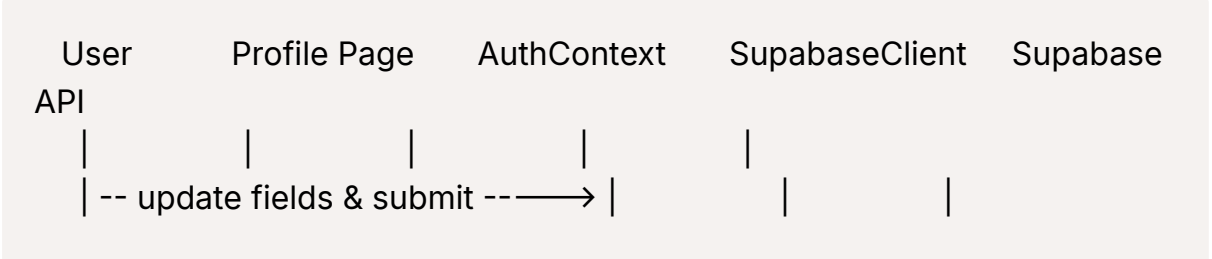
4. Edit Diary Entry



5. Delete Diary Entry



6. Update Profile Info

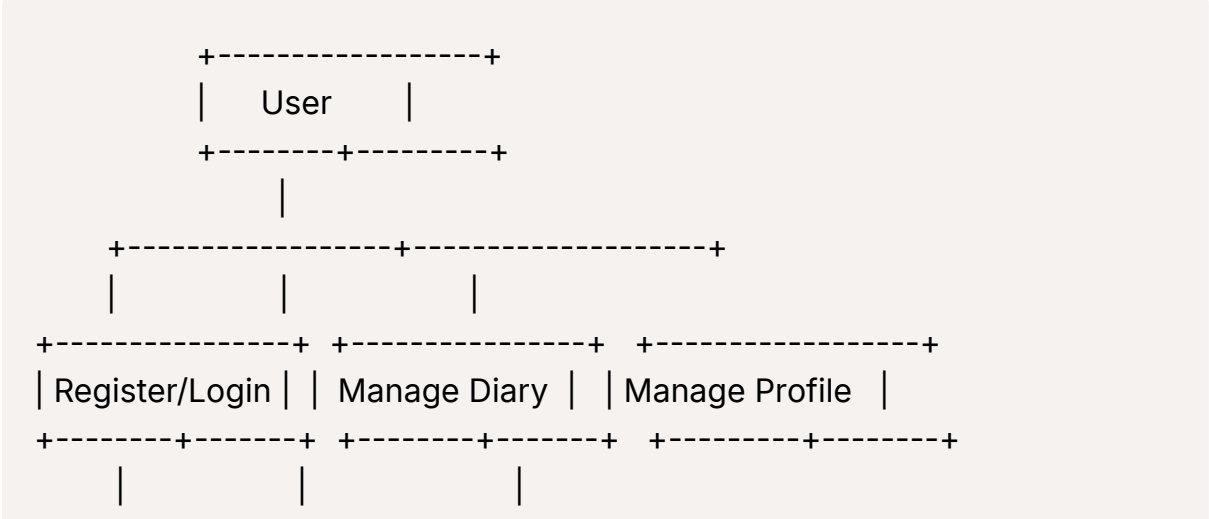


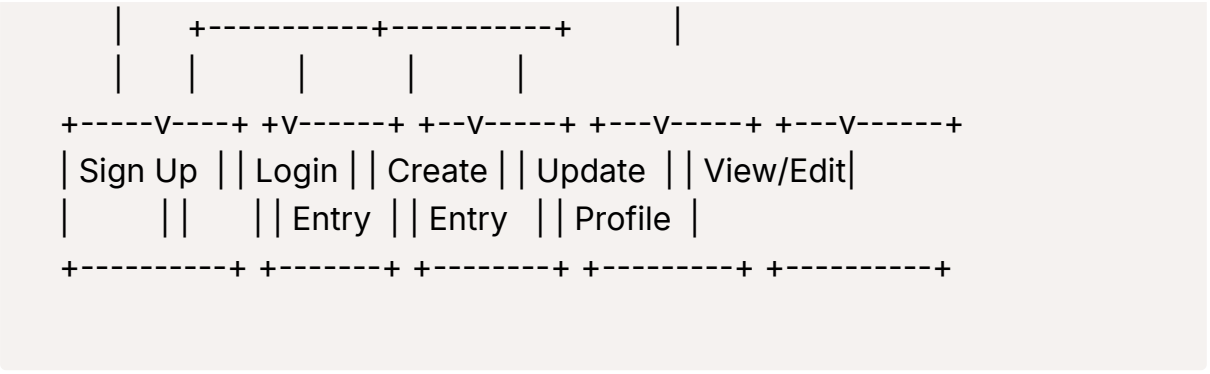
```
    |             | -- updateProfile(data) ------⟶ |             |
    |             |             | -- update() -----------------⟶ |
    |             |             |              | -- update row |
    |             |             |              | ← success     |
    |             |             | ← result ----------------------|
    |             | ← show success toast ----------- |             |
    | ← profile updated toast ------- |             |             |
```

7. Fetch Profile Info

```
  User        Profile Page      AuthContext      SupabaseClient    Supabase
API
    |             |             |              |
    | -- open profile -------------⟶ |             |             |
    |             | -- getProfile(user_id) ------⟶ |             |
    |             |             | -- select() -----------------⟶ |
    |             |             |              | -- get row    |
    |             |             |              | ← return data|
    |             |             | ← result ----------------------|
    |             | ← populate profile UI ---------- |             |
    | ← profile details shown -------- |             |             |
```

Use-Case Diagram:

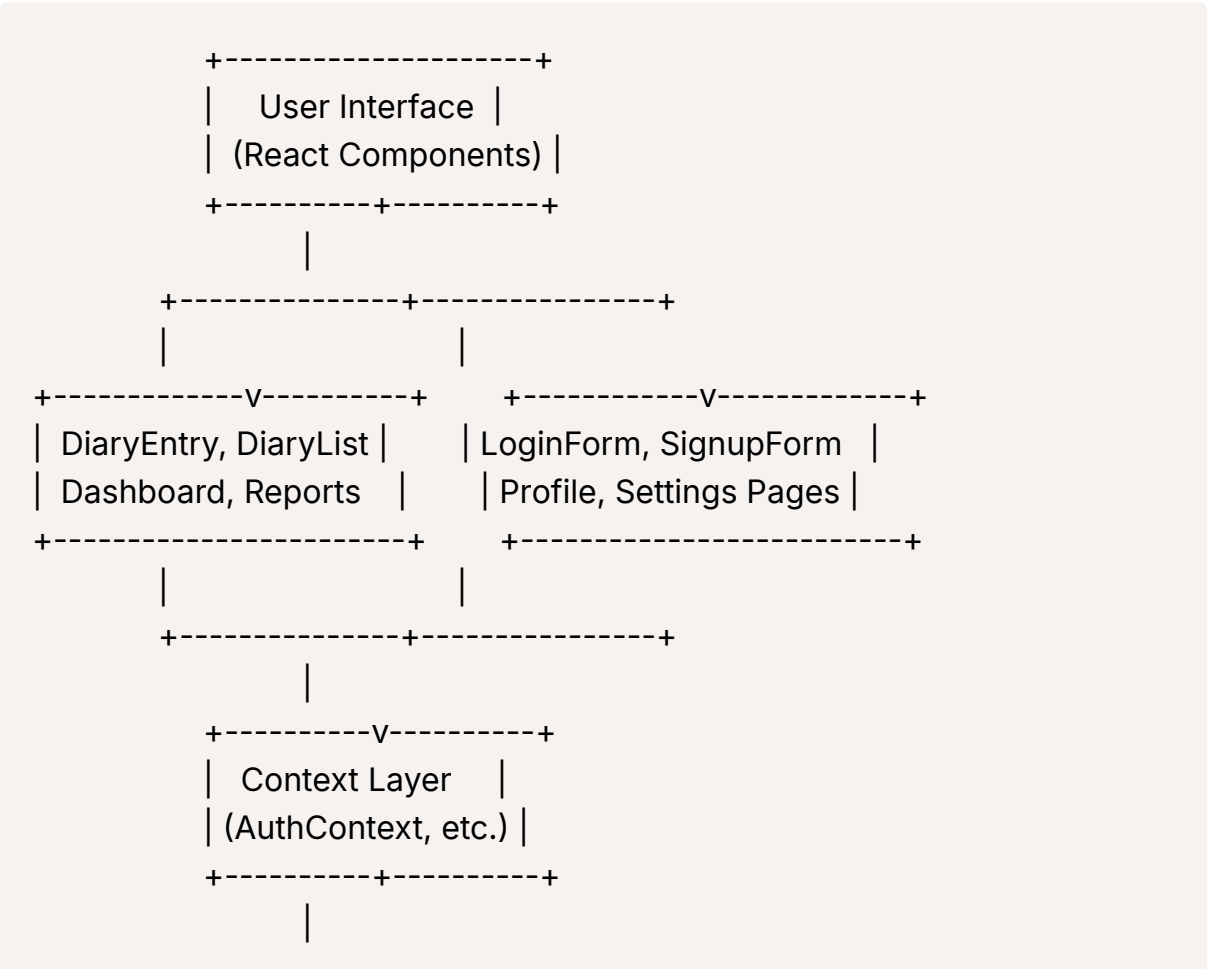


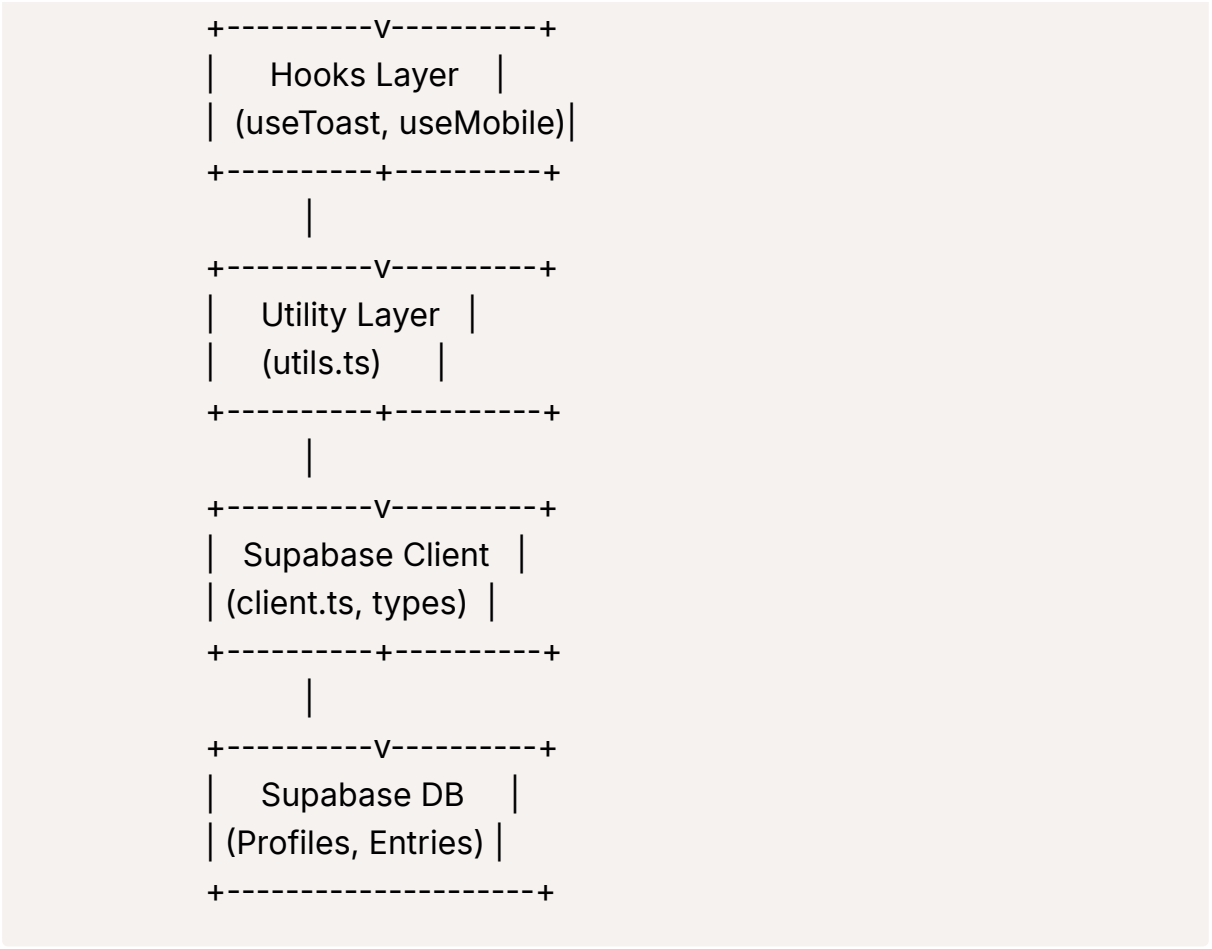


Key Use Cases:

- **Register/Login**: Sign Up, Login
- **Manage Diary**: Create Entry, Update Entry, Delete Entry
- **Manage Profile**: View/Edit Profile

Component Diagram:




Hooks Layer
(useToast, useMobile)
Utility Layer
(utils.ts)
Supabase Client
(client.ts, types)
Supabase DB
(Profiles, Entries)