LUBASI MILUPI lubasimilupi@gmail.com

PROJECT SUMMARY

Project Overview and Core Functionality

This project is a full-stack React Native mobile board built with Expo Router that connect service seekers with pet and house sitters. The application features two distinct, role-based navigation flows protected by layout-level navigation. Service Seekers manage their experience through full CRUD capabilities. This includes creating, editing and deleting listings with detailed forms. Whereas Sitters utilize filtering to browse and save listings via a dedicated interface. Both user flows are optimized for performance and usability, including robust loading states, error handling and real-time data synchronization.

Architecture and Database Design

The backend relies on Supabase PostgreSQL, using an ISA generalization model to handle User supertypes and Seeker/Sitter subtypes. The database is normalized to 3NF, featuring ENUM types for consistency and establishing clear relationships. Relationships present are a one-to-many link for seekers creating listings and many-to-many relationships via a junction table for sitters saving listings. The project repository also includes an SQL file detailing the schema definition for the entire database.

On the frontend, the architecture leverages the React Context API for role-based state management and Expo Router's file-based routing with route groups to strictly separate user concerns (Sitter or Service Seeker) and maintain a clean project structure.

State Management and Code Quality

Data fetching is managed via TanStack React Query, which provides automatic cache invalidation, optimistic updates, and background re-fetching. This logic is abstracted into a dedicated service layer and custom hooks, ensuring that UI components remain decoupled from backend logic. The codebase prioritizes type safety through TypeScript interfaces and maintains data integrity using comprehensive form validation and native date pickers.

Trade-offs and Execution

To meet strict time constraints, specific architectural simplifications were chosen. Firstly, the app utilizes a combination of server functions and custom hooks for CRUD operations rather than implementing distinct API routes with server endpoints. Second, useState was selected for form state management over more complex libraries like Zustand or React's useReducer hook to prioritize development speed. Finally, Supabase was adopted for its hosted PostgresSQL solution, bypassing the need for local database setup and migration. Despite these strategic simplifications, the resulting application successfully met all defined functional requirements.