Final Documentation: Online Journal Local

PROJECT DESCRIPTION

Online Journal Local is a secure, local-first mobile application built with Flutter that allows users to record their daily thoughts, track moods, and get feedback from AI-psychologist. Designed with privacy in mind, all data is stored locally on the device using Hive, ensuring no data leaves the user's phone.

The application features a robust authentication system with SHA-256 password cashing, a multi-step wizard for creating journal entries, and a personalized profile system including profile picture.

INTEGRATIONS

The project leverages the following key integrations and packages:

- **Hive (NoSQL Database):** Used for high-performance local storage of User Profiles, Sessions, and Journal Entries.
- **Image Picker (Platform Channel):** integrates with Android/iOS for (Gallery/Camera) and MacOS (Gallery) to allow users to select and persist profile pictures.
- **Path Provider:** Used to access the application's secure documents directory for persisting image files.
- **Flutter Bloc:** Used for state management, draws the UI based on the Cubit's state
- **GetIt:** Used for Dependency Injection.
- **GeminiService:**: Used for API-calls to Gemini 2.5 flash model.

IMPLEMENTED OPTIONAL REQUIREMENTS

The project successfully implements the following optional requirements:

- **Platform Channel:** Implemented via `image_picker` package to access native gallery/camera.
- **Animations:**
  - **Implicit:** `Hero` widgets for smooth transitions of Mood Emojis between Home and Details screens.
  - **Implicit:** `TweenAnimationBuilder` for fading in text content.
- **Tests:**
  - **Unit Tests:** Verified `JournalCubit` state transitions (Loading → Loaded).
  - **Widget Tests:** Verified rendering of `JournalDetailsScreen` UI elements.
  - **Integration Flow:** Verified the full "Add Entry" user flow using `testWidgets` and simulated user interaction.
- **Platform Support:** Application logic and UI optimized and verified for iOS, and MacOS targets.
- **Local Persistence:** Full offline data storage implemented using Hive for user data and journal entries.
- **Authentication & Caching:** Custom backend authentication flow with Hive, password caching and auto-login session management.
- **Complex Forms:** Implemented multi-step wizards with input validation for the "Sign Up" and "Add Journal Entry" screens.
- **CI/CD:** Automated pipeline configured for static code analysis and execution of Flutter tests.

INSTRUCTION TO RUN

1. **Prerequisites:** Ensure Flutter SDK (> 3.0) is installed.
2. **Environment Setup:** Create a file named `.env` in the root directory of the repository and add your API key:
   `GEMINI_API_KEY=<your_api_key_here>`

3. **Install Dependencies:**
   `flutter pub get`

4. **Generate Adapters (Hive):**
   `dart run build_runner build --delete-conflicting-outputs`

5. **Run the App:**
   `flutter run`

   Note: For testing the Camera on iOS, a physical device is required. The Gallery works on simulators or Desktop (macOS).

TEST ACCOUNT

Since the application uses a local database, there are no pre-seeded cloud accounts.

- **To Test:** Simply launch the app and click the "Person" icon in the AppBar (or "Sign Up" button) to register a new user.
- **Note:** The app supports multiple local users. You must register to log in.

DATABASE SCHEMA (HIVE)

The application uses three distinct Hive boxes:

1. Users Box (`users_db`)

Stores registered user information.

- **Model:** `UserProfileModel` (HiveType: 1)
- **Fields:**
  - `firstName` (String), `lastName` (String)
  - `email` (String), `password` (String)
  - `street`, `city`, `zipCode` (Strings)
  - `profileImagePath` (String) - Path to locally saved avatar

2. Journal Box (`journal_box`)

Stores all journal entries.

- **Model:** `JournalEntryModel` (HiveType: 0)
- **Fields:**
  - `id` (String) - Unique Entry ID
  - `userId` (String) - Foreign Key (Email) linking entry to a user
  - `title` (String), `content` (String)
  - `mood` (String) - e.g., "Happy", "Sad"
  - `date` (DateTime)
  - `aiAnalysis` (String)
  - `imagePath` (String) Not implemented for now

CI/CD DESCRIPTION

The project includes a Continuous Integration workflow file (`.github/workflows/flutter_ci.yml`) or local script configuration.

- **Trigger:** Runs on every push to the `main` branch or Pull Requests.
- **Jobs:**
  1. **Setup:** Installs Flutter environment.
  2. **Linting:** Runs `flutter analyze`, `dart format` to check for code style violations.
  3. **Testing:** Runs `flutter test` to execute all Unit, Widget, and Integration tests.