Mojak App: Documentation & Setup Guide

1. Project Overview

Mojak is an entertainment and "future-telling" application built with Flutter. It guides users through a series of five "stations," each offering a unique interactive experience to generate a fictional glimpse into their future life.

- Station 1: Partner Traits: Users play a mini-game to discover the physical traits and quirks of their future soulmate.
- Station 2: Love Story: Users select preferences (like their zodiac sign) to generate a short, romantic story about how they meet, get engaged, and marry their soulmate.
- Station 3: Baby Profile: Users complete a challenge to generate a profile for their future baby, including a name, loves, and hates.
- Station 4: AI Chat: Users chat with their AI-powered soulmate, whose personality is shaped by the user's previous inputs and generated results.
- Station 5: Final Reveal: A final puzzle reveals a "perfect match" avatar and a compatibility score, concluding the user's journey.

The application is localized for English and Arabic, uses Riverpod for state management, and integrates Firebase for analytics, AdMob for monetization, and Google's Gemini AI for the chat feature.

2. Core Technologies

- Framework: Flutter
- State Management: Flutter Riverpod
- Backend & Analytics: Firebase (Analytics)
- AI Chat: Google Gemini (via `google_generative_ai`)
- Monetization: Google AdMob
- Local Storage: `shared_preferences`
- Audio: `audioplayers`
- UI: `google_fonts`, `google_nav_bar`

3. Pre-requisites

Before you begin, ensure you have the following installed on your system:

1. Flutter SDK: The core framework for building the app.
2. A Code Editor: Visual Studio Code (recommended) with the Flutter extension, or Android Studio.
3. An Emulator or Physical Device:
    - Android: An Android emulator set up via Android Studio, or a physical Android device with Developer Mode enabled.
    - iOS: A physical iPhone (required for testing on a non-Mac) or the iOS Simulator (requires a macOS machine with Xcode).

---

4. Step-by-Step Setup Guide

Follow these steps carefully to get the project running.

Step 1: Install Flutter

If you don't have Flutter installed, follow the official guide for your operating system:

- Flutter Official Installation Guide

After installation, run the following command in your terminal to verify that everything is set up correctly:

```
flutter doctor
```

Address any issues that `flutter doctor` reports.

Step 2: Get the Code

1. Create a new folder for your project (e.g., `mojak_app`).
2. Inside this folder, create a `lib` directory.
3. Place all the provided `.dart` files from the codebase dump into the `lib` folder, maintaining their directory structure (e.g., `lib/src/screens/welcome_screen.dart`).

Step 3: Configure `pubspec.yaml`

This file is the heart of your project's configuration. It was not included in the dump, but based on the code, you can create it with the following content.

Create a file named `pubspec.yaml` in the root of your `mojak_app` folder:

```yaml
name: futu # Name from firebase_options.dart
description: A new Flutter project.
publish_to: 'none'

version: 1.0.0+1

environment:
  sdk: '>=3.2.3 <4.0.0'

dependencies:
  flutter:
    sdk: flutter
  flutter_localizations:
    sdk: flutter

  # Core Packages
  flutter_riverpod: ^2.4.9
  shared_preferences: ^2.2.2

  # Firebase
  firebase_core: ^2.25.4
  firebase_analytics: ^10.8.5

  # AI & Ads
  google_generative_ai: ^0.2.2
  flutter_dotenv: ^5.1.0 # For API Key
  google_mobile_ads: ^4.0.0

  # UI & Utilities
  google_fonts: ^6.1.0
  google_nav_bar: ^5.0.6
  audioplayers: ^5.2.1
  share_plus: ^7.2.2
  path_provider: ^2.1.2
  intl: ^0.19.0

  cupertino_icons: ^1.0.2

dev_dependencies:
  flutter_test:
    sdk: flutter
  flutter_lints: ^2.0.0

flutter:
  uses-material-design: true

  # Enable l10n
  generate: true

  # Assets
  # Ensure you have these folders and files in your project's root directory
  assets:
    - assets/images/
    - assets/images/avatars/
    - assets/images/station_cards/
    - assets/audio/
    - .env # For the Gemini API Key
```

Step 4: Set Up Firebase

The app uses Firebase for analytics.

1. Create a Firebase Project: Go to the Firebase Console and create a new project.
2. Install the Firebase CLI: If you haven't already, install the Firebase command-line tools by following this guide.

Install the FlutterFire CLI: Run this command in your terminal:
```
dart pub global activate flutterfire_cli
```
3.

Configure the App: Navigate to your project's root directory (`mojak_app`) in the terminal and run:
```
flutterfire configure
```
4. This will guide you through connecting your app to your Firebase project for both Android and iOS. It will automatically:
    - Create Firebase apps for Android/iOS in your project.
    - Download the necessary configuration files (`google-services.json` for Android and `GoogleService-Info.plist` for iOS).
    - Generate a new `lib/firebase_options.dart` file. This file will contain your project's actual Firebase keys and will replace the placeholder one from the code dump.

Step 5: Set Up Gemini AI API Key

The AI chat in Station 4 requires a Google Gemini API key.

1. Get an API Key: Go to the Google AI Studio and create a new API key.
2. Create a `.env` file: In the root of your project folder (`mojak_app`), create a new file named `.env`.

Add the Key to the file: Open the `.env` file and add your key like this:
```
GEMINI_API_KEY=YOUR_API_KEY_HERE
```

3. Replace `YOUR_API_KEY_HERE` with the key you just generated.

Note: The `ai_chat_service.dart` file has a hardcoded fallback key for convenience, but it's strongly recommended to use your own key in the `.env` file, as the public key may be disabled.

Step 7: Get Assets and Run the App

1. Create Asset Folders: In the root of your project, create the folders referenced in `pubspec.yaml`:
   - `assets/images/`
   - `assets/images/avatars/`
   - `assets/images/station_cards/`
   - `assets/audio/`
   - *You will need to source or create placeholder assets for the paths mentioned in the code.* For example, create a blank `Tap.mp3` file in `assets/audio/`.

Install Dependencies: Open a terminal in your project's root directory and run:
```
flutter pub get
```

2.

Run the App: Connect a device or start an emulator and run:
```
flutter run
```

3. The app should now build and launch on your device/emulator.

---

5. Project Structure Overview

The project is well-organized into several directories inside `lib/`:

- `lib/l10n/`: Contains localization files for English (`app_en.arb`) and Arabic (`app_ar.arb`).
- `lib/src/data/`: Holds static data pools (e.g., lists of names, hobbies, story fragments) used by the generation services.
- `lib/src/models/`: Defines the data structures for the app, such as `UserProfile`, `Station`, `PartnerTraits`, etc.
- `lib/src/providers/`: Contains all the Riverpod providers for managing the app's state (e.g., `user_state_provider`, `locale_provider`).
- `lib/src/screens/`: Contains all the UI screens, organized into subfolders for each station.
- `lib/src/services/`: Contains singleton services that handle specific logic, such as `AnalyticsService`, `AdService`, `AIChatService`, and the deterministic generators for each station.
- `lib/src/theme/`: Defines the application's visual theme, including colors and font styles.
- `lib/src/widgets/`: Contains reusable widgets used across multiple screens, like `BannerAdWidget`.

---

6. Key Features & How They Work

State Management (Riverpod)

The app uses `flutter_riverpod` for state management.

- `user_state_provider.dart` is the main source of truth for user progress, including which stations are complete and the user's profile data from onboarding.
- Widgets use `ConsumerWidget` or `ConsumerStatefulWidget` to listen to changes in providers.
- `ref.watch()` is used to get the current state and rebuild the widget when it changes.
- `ref.read()` is used to call methods on a provider's notifier (e.g., `ref.read(userStateProvider.notifier).completeOnboarding(...)`).

Deterministic Generation

A core feature is the ability to generate the *same results* for the *same user inputs*. This is achieved by `src/services/deterministic_generator.dart`.

- It takes a map of user choices (e.g., zodiac sign, game score).
- It converts these inputs into a unique integer seed.
- This seed is then used to initialize a `Random` number generator.
- When selecting items from data pools (like names or hobbies), it uses this seeded `Random` instance, ensuring the "random" selection is repeatable.
- Each station's generator service (e.g., `station_1_generator.dart`) is responsible for collecting the relevant inputs, creating the seed, and generating the result.

AI Chat (Station 4)

- The AI chat is powered by the Google Gemini API via the `google_generative_ai` package.
- `ai_chat_service.dart` manages all communication with the API.
- When the chat initializes, it builds a persona prompt. This prompt tells the Gemini model who it is (e.g., "Your name is Fatima, you are the user's soulmate, you met at a bookstore..."). This context is built from the results of Stations 1, 2, and 3.
- Each time the user sends a message, the service sends the entire chat history plus the persona prompt to the API, allowing the AI to respond contextually and "in character."

Guide to Signing and Publishing Your Flutter App on the Google Play Store

This guide is broken down into five main parts. Follow them in order.

Part 1: Generating a Signing Key

Before you can publish your app, you must sign it with a unique digital certificate. This key proves that you are the author of the app and ensures that updates come from you.

This key is extremely important. If you lose it, you will NOT be able to publish updates to your app. Back it up in multiple secure locations (e.g., a password manager, a secure cloud drive, an external hard drive).

You will use the `keytool` command, which is included with the Java Development Kit (JDK) that comes with Android Studio.

1. Open a terminal or command prompt.

Navigate to the `android/app` directory inside your Flutter project folder:

```
cd path/to/your/mojak_app/android/app
```

2.
3. Run the key generation command.

On macOS or Linux:

```
keytool -genkey -v -keystore my-upload-key.keystore -alias upload -keyalg RSA
-keysize 2048 -validity 10000
```

-

On Windows:

```
keytool -genkey -v -keystore my-upload-key.keystore -keyalg RSA -keysize 2048
-validity 10000 -alias upload
```

-

4. Follow the on-screen prompts:
   - Enter keystore password: Create a secure password. You will need this every time you build a release. Write it down and save it securely.
   - Re-enter new password: Confirm the password.
   - What is your first and last name? You can enter your name or your company's name.
   - What is the name of your organizational unit? (e.g., Engineering) - Optional.
   - What is the name of your organization? (e.g., Your Company Name) - Optional.
   - What is the name of your City or Locality? (e.g., San Francisco) - Optional.
   - What is the name of your State or Province? (e.g., California) - Optional.
   - What is the two-letter country code for this unit? (e.g., US) - Optional.
   - Is CN=... correct? Type `yes` and press Enter.
   - Enter key password for `<upload>`: Press Enter to use the same password as the keystore password (recommended for simplicity).

After this process, a new file named `my-upload-key.keystore` will be created inside your `android/app` directory.

Part 2: Configuring the Flutter Project to Use the Key

Now, you need to tell your Android project how to find and use this key. We will do this securely, without hardcoding passwords in your code.

1. Create a `key.properties` file:
   - Inside your `android` directory (the same level as the `app` folder), create a new file named `key.properties`.
   - Add the following content to `android/key.properties`, replacing `YOUR_KEYSTORE_PASSWORD` with the password you created in Part 1.

```
storePassword=YOUR_KEYSTORE_PASSWORD
keyPassword=YOUR_KEYSTORE_PASSWORD
keyAlias=upload
storeFile=app/my-upload-key.keystore
```

2. *Note: `storeFile` is relative to this `key.properties` file, so `app/my-upload-key.keystore` is the correct path.*
3. VERY IMPORTANT: Exclude your key files from version control.
   - Open the `.gitignore` file in the root of your Flutter project.
   - Add the following lines to the end of the file to prevent your keys and passwords from being uploaded to Git (like GitHub).

```
# Keystore files
/android/app/my-upload-key.keystore
/android/key.properties
```

4.
5. Configure Gradle to use the signing key:
   - Open the file `android/app/build.gradle`.

Add the following code at the very top of the file, before the `android { ... }` block:

```
def keystorePropertiesFile = rootProject.file('key.properties')
def keystoreProperties = new Properties()
if (keystorePropertiesFile.exists()) {
    keystoreProperties.load(new FileInputStream(keystorePropertiesFile))
}
```

-

Find the `android { ... }` block. Inside it, add a `signingConfigs` block and modify the `buildTypes` block as shown below.

```
android {
    // ... other configurations like compileSdkVersion

    signingConfigs {
        release {
            if (keystoreProperties.getProperty('storeFile') != null) {
                storeFile file(keystoreProperties.getProperty('storeFile'))
                storePassword keystoreProperties.getProperty('storePassword')
                keyAlias keystoreProperties.getProperty('keyAlias')
                keyPassword keystoreProperties.getProperty('keyPassword')
            }
        }
    }

    buildTypes {
        release {
            // ... other release settings like `shrinkResources`
            signingConfig signingConfigs.release
        }
    }
}
```

-

Your project is now fully configured for release signing.

Part 3: Building the Release App Bundle

The Google Play Store requires you to upload an Android App Bundle (`.aab` file), not an APK.

1. Open a terminal in the root directory of your Flutter project.

Run the build command:
```
flutter build appbundle
```

2.
3. Flutter will build your app in release mode, sign it with your keystore, and create the app bundle.
4. Once finished, you will find the release bundle at: `build/app/outputs/bundle/release/app-release.aab`

This is the file you will upload to the Google Play Store.

Part 4: Preparing for the Google Play Store

Before you can upload your `.aab` file, you need to set up your app's presence on the Play Store.

1. Create a Google Play Developer Account: If you don't have one, go to the Google Play Console and register. There is a one-time $25 registration fee.
2. Change the App's Package Name: The current package name is `com.example.futu`, which is a placeholder and cannot be used for publishing. You must change it to something unique, like `com.yourcompany.mojak`.
    - Easy Way: Use a package like `change_app_package_name`. Run `flutter pub add change_app_package_name` and then `flutter pub run change_app_package_name:main com.yourcompany.mojak`.
    - Manual Way: Manually edit the `applicationId` in `android/app/build.gradle` and the `package` attribute in all three `android/app/src/main/AndroidManifest.xml` files.
3. Prepare Your Store Listing Assets: You will need to create and upload these in the Play Console:
    - App Icon: 512x512 pixels, 32-bit PNG (with alpha).
    - Feature Graphic: 1024x500 pixels, JPG or 24-bit PNG (no alpha).
    - Screenshots: At least 2 screenshots are required. Take them from your app running on a device or emulator.
    - Short Description (80 characters max)
    - Full Description
4. Complete App Content Forms in the Play Console:
    - Privacy Policy: Google requires a URL to your privacy policy. You can host the text from `privacy_policy_screen.dart` on a simple website (like a GitHub Page or a blog).
    - Ads: You must declare that your app contains ads.
    - Content Rating: Fill out a questionnaire to determine the age rating for your app.
    - Data Safety: Fill out the form detailing what user data your app collects and why. Be honest about Firebase Analytics and any data saved locally.

Part 5: Uploading and Publishing Your App

1. Log in to the Google Play Console.
2. Click "Create app" and fill in your app's name, default language, etc.
3. Navigate to the Dashboard for your new app and follow the "Set up your app" checklist. This will guide you through all the forms mentioned in Part 4.
4. Once the initial setup is done, go to the "Testing" section in the left-hand menu. It is highly recommended to start with "Internal testing" or "Open testing" before going to Production.

5. Click "Create new release" on your chosen testing track.
6. In the "App bundles" section, upload the `app-release.aab` file you created in Part 3.
7. Google will process the file. The version name and code will be automatically populated from your `pubspec.yaml`.
8. Write some Release notes describing what's in this version.
9. Click "Save", then "Review release".
10. Address any errors or warnings that the Play Console shows you.
11. Finally, click "Start rollout to [Testing Track]".

Your app is now submitted for review! The first review for a new app can take anywhere from a few hours to several days. Once approved, testers you've invited (for internal/closed tracks) or anyone with the link (for open tracks) can download it. After you are confident with your testing, you can promote the release to Production.