TherapyAI - Installation and Maintenance Guide

Version: 1.0
Date: 2024-03-10

1. Introduction

- **Purpose:** This guide provides technical instructions for setting up the TherapyAI Android application development environment from a provided ZIP archive, building the application, configuring essential settings, and performing routine maintenance tasks.
- **Target Audience:** This guide is intended for developers, DevOps personnel, or technical staff responsible for building, configuring, and maintaining the TherapyAI Android application.
- **Scope:** This guide covers the process starting from the project source code provided as a ZIP file up to generating build artifacts (APK/AAB). It does *not* cover backend deployment or specific app distribution methods (e.g., Google Play Store submission details, MDM deployment).

2. Prerequisites

Before proceeding, ensure the following software is installed and configured on your development machine:

- **Java Development Kit (JDK):** Version 11 or higher is typically required for modern Android development. Verify compatibility based on the Android Gradle Plugin version used in the project (check build.gradle).
- **Android Studio:** The latest stable version is recommended (e.g., Iguana, Hedgehog, or newer). Download from the official Android Developers website.
- **Android SDK:** Install required SDK Platforms and Build-Tools via the Android Studio SDK Manager. Pay attention to the compileSdk and targetSdk versions specified in the app/build.gradle file.
- **Git:** While the project is provided as a ZIP, Git is essential for managing future code changes, branching, and potential integration with version control systems.
- **(Optional but Recommended) Version Control System:** After initial setup, it's highly recommended to initialize a Git repository within the extracted project folder to track local changes.

3. Installation from ZIP Archive

1. **Obtain the ZIP File:** Securely acquire the project ZIP archive.
2. **Choose Installation Directory:** Select a suitable location on your development machine to store the project source code (e.g., ~/dev/, C:\Projects\). Avoid paths with spaces or special characters if possible.
3. **Extract the Archive:** Extract the contents of the ZIP file into your chosen directory. This will create a folder containing the project structure (e.g., TherapyAI_Android_Source/).
4. **Open Project in Android Studio:**
   - Launch Android Studio.
   - Select "Open" (or "Open an Existing Project").
   - Navigate to and select the *extracted project folder* (the one containing the app, gradle directories, and build.gradle files).
   - Click "Open".
5. **Gradle Synchronization:**
   - Android Studio will automatically initiate a Gradle sync process. This downloads required dependencies and configures the project. This may take several minutes, especially on the first open.
   - Monitor the "Build" output window for progress and potential errors.
   - Ensure you have a stable internet connection during this process.
   - *Troubleshooting:* If Gradle sync fails, check network connectivity, proxy settings (if applicable), and ensure the correct JDK is configured in Android Studio (File -> Project Structure -> SDK Location). Try File -> Invalidate Caches / Restart.
6. **Install Missing SDK/Build Tools:**
   - Android Studio might prompt you to install missing SDK platforms or build tools versions specified in the project's build.gradle files. Follow the on-screen instructions to install them.

4. Essential Configuration

After the project is successfully opened and synced in Android Studio, several critical configurations are required:

1. **API Base URL:**
   - **Purpose:** Defines the root URL of the backend API the app communicates with.
   - **Location:** Open the file app/src/main/java/com/example/therapyai/data/remote/ApiServiceProvider.java.
   - **Action:** Locate the BASE_URL constant and update its value to the correct URL for your target environment (e.g., development, staging, production).
   - // Example inside ApiServiceProvider.java
   - private static final String BASE_URL = "https://your-actual-api-backend.com/"; // <-- CHANGE THIS
   - **Importance:** The app will not function correctly without the proper backend URL.
2. **Firebase Configuration (google-services.json):**
   - **Purpose:** Connects the app to your Firebase project, enabling features like Push Notifications (FCM).
   - **Action:**
     - Set up a Firebase project at https://console.firebase.google.com/.
     - Register this Android app within your Firebase project using the application ID found in app/build.gradle (press the android icon in the firebase console in the project overview).
       Currently this value is “com.example.therapyai” (without quotes), Of course can be changed by the facility. No need to fill in the optional fields.
     - In the next step, download the google-services.json configuration file provided by Firebase.

        - Place the downloaded google-services.json file into the app/ directory of the project structure (e.g., TherapyAI_Android_Source/app/google-services.json).
        - No need to follow the add firebase SDK step as it is already set in.
    - **Importance:** Push notifications (NotificationService) will not work without a valid google-services.json linked to an active Firebase project.

3. **Build Signing Configuration (for Release Builds) - <u>NOT MANDATORY</u>:**

   NOTE: Currently, the debug build is mocking the release in terms of build flags (using pro-guard, shrinking etc). To build in a normal debug environment, go to app\build.gradle file and remove the flags of the debug section under buildTypes or set them to false.

   Disclaimer: we recommend keeping this as is because the app's purpose is to be used as a facility app and not for distribution. We added this section in case the facility would like to distribute it outwards.

    - **Purpose:** Required to sign release builds of the application for distribution (e.g., Google Play Store, enterprise deployment). Debug builds use a temporary debug key.
    - **Action:**
        - **Generate a Keystore:** If you don't have one, use Android Studio's built-in tool (Build -> Generate Signed Bundle / APK... -> Choose APK or App Bundle -> Create new...) or the keytool command-line utility to generate a .jks or .keystore file. **Securely store this keystore file and its passwords.** Losing it prevents future updates.
        - **Configure build.gradle (app):** Open the app/build.gradle file. Locate the signingConfigs block (or add one if missing) within the android block. Configure it to reference your keystore. **Crucially, DO NOT hardcode passwords directly in build.gradle.** Use local.properties or environment variables.
            - **Create/Update local.properties:** In the project's root directory, create or edit the local.properties file (this file is typically *not* committed to version control). Add the following lines, replacing placeholders with your actual values:

- KEYSTORE_FILE=/path/to/your/release.keystore
- KEYSTORE_PASSWORD=your_keystore_password
- KEY_ALIAS=your_key_alias
- KEY_PASSWORD=your_key_password

```
Modify app/build.gradle:
      android {
    ...
    signingConfigs {
        release {
            // Check if properties exist before trying to read them
            if (project.rootProject.file('local.properties').exists()) {
                Properties props = new Properties()
                props.load(project.rootProject.file('local.properties').newDataInputStream())

                if (props.containsKey('KEYSTORE_FILE') && props.containsKey('KEYSTORE_PASSWORD') &&
                    props.containsKey('KEY_ALIAS') && props.containsKey('KEY_PASSWORD')) {

                    storeFile file(props['KEYSTORE_FILE'])
                    storePassword props['KEYSTORE_PASSWORD']
                    keyAlias props['KEY_ALIAS']
                    keyPassword props['KEY_PASSWORD']
                } else {
                    println("Warning: Release signing information missing in local.properties.")
                    // Optionally configure a dummy signingConfig or fail the build here
                    storeFile file("dummy.jks") // Example: provide dummy values
                    storePassword "password"
                    keyAlias "alias"
                    keyPassword "password"
                }
            } else {
                 println("Warning: local.properties file not found for release signing.")
                 // Optionally configure a dummy signingConfig or fail the build here
                storeFile file("dummy.jks") // Example: provide dummy values
                storePassword "password"
                keyAlias "alias"
                keyPassword "password"
            }
        }
    }
    buildTypes {
        release {
            ...
            signingConfig signingConfigs.release // Apply the config
        }
        debug {
            // Debug uses default debug signing
        }
    }
    ...
}
```

    - **Importance:** Release builds *cannot* be generated or distributed without proper signing.

4. **Mock Data Configuration:**
    - **Purpose:** Allows running the app using mock API responses for development and testing without needing a live backend.
    - **Location:** Open app/src/main/java/com/example/therapyai/TherapyAIApp.java.
    - **Action:** Find the call to initRepositories(boolean useMockData) within the onCreate method.
        - Set useMockData to true to use mock data (defined in TherapyApiImpl.java).
        - Set useMockData to false to use the real API defined by BASE_URL.
    - **Importance:** Ensure this is set to false for builds intended to connect to the actual backend.

5. Building the Application

You can build the application using Android Studio or the command line (Gradle wrapper).

1. **Build Variants:** Select the desired build variant in Android Studio (Build -> Select Build Variant...) or specify it on the command line. Common variants:
   - debug: For development and testing. Uses the debug signing key. Typically not obfuscated or optimized.
   - release: For distribution. Requires the release signing configuration set up in Section 4.3. Usually includes code shrinking (ProGuard/R8) and optimization.
2. **Using Android Studio - Recommended:**
   - Go to Build -> Build Bundle(s) \ APK(s) -> Build APK(s) or Build Bundle(s). ("Build" might be written as "Generate")
   - Choose the desired variant (debug or release).
   - The output file(s) will be located in app/build/outputs/apk/[variant]/ or app/build/outputs/bundle/[variant]/.
3. **Using Command Line:**
   - Open a terminal or command prompt in the project's root directory.
   - **Debug APK:** ./gradlew assembleDebug (Linux/macOS) or gradlew.bat assembleDebug (Windows)
   - **Release APK:** ./gradlew assembleRelease or gradlew.bat assembleRelease
   - **Release App Bundle (AAB):** ./gradlew bundleRelease or gradlew.bat bundleRelease
   - Output files will be in the same locations mentioned above.

6. Deployment

- The output of the build process is either an **APK** (.apk) file or an **Android App Bundle** (.aab) file.
- **APK:** Can be directly installed on devices (requires enabling installation from unknown sources if not distributed via Play Store). Move the file from the computer to the device, and open the file on the device using “File Explorer” or your preferred directory explorer on android.
- **AAB:** The preferred format for uploading to the Google Play Store. Google Play processes the AAB to generate optimized APKs for different device configurations.
- Deployment methods (Google Play Console, Mobile Device Management (MDM), direct installation) are outside the scope of this guide. Ensure the correct build type (Release) and signing configuration were used for the deployed artifact.

7. Maintenance

Regular maintenance is required to keep the application up-to-date and secure.

1. **Updating Dependencies:**
   - Periodically check for updates to libraries listed in app/build.gradle and the project-level build.gradle.
   - Android Studio often highlights available updates.
   - Use stable versions unless there's a specific reason to use alpha/beta releases.
   - After updating dependencies, perform a clean build (Build -> Clean Project) and thorough testing to ensure compatibility.
2. **Updating Android Gradle Plugin & Build Tools:**
   - Update the Android Gradle Plugin version in the project-level build.gradle and the Gradle version in gradle/wrapper/gradle-wrapper.properties as recommended by Android Studio or official releases.
   - Update compileSdk, targetSdk, and buildToolsVersion in app/build.gradle according to current Android best practices and requirements. Test thoroughly after updates.
3. **Configuration Changes:** If backend URLs, API keys, or other configurations change, update them in the relevant files (ApiServiceProvider.java, local.properties, potentially AndroidManifest.xml or resource files) and rebuild.
4. **Code Cleanup & Refactoring:** Periodically review code for potential improvements, removal of unused code/resources, and adherence to best practices. Use Android Studio's lint checks and analysis tools.
5. **Source Code Backup:** Regularly back up the source code, especially the signing keystore file (.jks) and local.properties (which contains sensitive passwords), using a secure version control system (like Git with a private repository) or other secure backup methods.

8. Troubleshooting

- **Gradle Sync Errors:** Check network connection, proxy settings. Ensure JDK path is correct in Android Studio. Try "File -> Invalidate Caches / Restart". Check the "Build" window for specific error messages (e.g., dependency resolution failures).
- **Build Errors:**
  - *Missing SDK/Build Tools:* Install required versions via SDK Manager.
  - *Signing Errors (Release):* Ensure local.properties exists, contains the correct keys (KEYSTORE_FILE, KEYSTORE_PASSWORD, etc.), paths are correct, and passwords match the keystore. Verify the signingConfigs.release block in app/build.gradle is correctly configured.
  - *Compilation Errors:* Address syntax errors or unresolved references reported in the "Build" window.
  - *Resource Errors:* Check XML layouts, drawables, etc., for errors.
- **Runtime Crashes on Startup:**
  - *API URL Incorrect:* Verify BASE_URL in ApiServiceProvider.java.
  - *Firebase Initialization Failed:* Ensure google-services.json is present in the app/ directory and is valid for the configured Firebase project. Check Logcat for Firebase initialization errors.
  - *Initialization Order Issues:* Ensure Repositories and Managers are initialized correctly in TherapyAIApp.java.
- **Authentication Issues:** Check network, API URL. Verify token refresh logic in TokenAuthenticator and API endpoint. Ensure EphemeralPrefs is storing/retrieving tokens correctly. Check for UserNotAuthenticatedException logs if using Keystore features.

9. Security Considerations

- **API Keys & Secrets:** NEVER commit API keys, signing key passwords, or other secrets directly into version control (build.gradle, source files). Use local.properties (added to .gitignore) or environment variables.
- **Keystore Security:** Protect your release keystore file and its passwords diligently. Back it up securely. Loss means you cannot update the app on the Play Store.
- **google-services.json:** Treat this file as sensitive, as it links your app to your Firebase project. Add it to .gitignore if using public repositories.
- **FLAG_SECURE:** The use of this flag in BaseSignedActivity and BaseSessionActivity is important to prevent screen capture of potentially sensitive session/user data.
- **Data Encryption:** EphemeralPrefs relies on Android Keystore. Be aware of its limitations (e.g., key invalidation on screen lock changes if not configured carefully, UserNotAuthenticatedException). File encryption in RecordingRepository also depends on this key.
- **Network Security:** Retrofit/OkHttp configuration in ApiServiceProvider enforced TLS 1.2/1.3. Ensure the backend server also uses strong TLS configuration.

10. Running the Application:

After configuring the environment and loading the code, all that is left to do is deploy the application, use the provided file from the environment and install it on your smartphone.

If using mock, the application will run without any need for a cloud server.
Otherwise, make sure the cloud server is up and running and the phone application is communicating with the correct server.

On first login after booting the app and cloud it might throw a 403 exception, try logging in again as azure sometimes does a lazy boot !!!