Project Documentation

Members:
Manisan, Jodessey Swet C.
Ledesma, Reyver
Hinayon, Alexander

TRAVEL BUCKET LIST
(Mobile App)

Application Design and Architecture

This application is designed as a personal travel log to help users track their planned trips and commemorate their completed journeys. The architecture follows modern Android development practices, emphasizing a clean separation of concerns and a user-friendly, responsive interface.

1.1 Core Design Philosophy:

Single-Activity Architecture: The app uses a *HomepageActivity* to host multiple fragments (*HomeFragment, HistoryFragment, ProfileFragment*). This creates a seamless user experience, allowing for quick navigation without reloading the entire application.

Fragment-Based Navigation: A *BottomNavigationView* serves as the primary navigation tool, enabling users to switch between the main sections of the app: Pending Travels, Travel History, and User Profile.

ViewModel-Driven UI: The app leverages the *ViewModel* architecture component. This ensures that UI data (like the list of travels) survives configuration changes (e.g., screen rotation) and is separated from the UI logic in the fragments. *LiveData* is used to create a reactive UI that updates automatically when the underlying data changes.

User-Centric Interface: Dialogs are used for user input (e.g., adding a new trip, marking a trip as complete) to keep the user within the current context. The UI includes clear visual cues, such as icons and Material Design components, for an intuitive experience.

1.2 Key Screens & Flow:

Authentication (*MainActivity, RegisterActivity*): The user journey begins with a login or registration screen. User credentials are saved locally for session management.

Homepage (*HomepageActivity*): This is the main container for the user experience.

- **Pending Travels** (*HomeFragment*): The default screen, showing a list of upcoming or planned trips. Users can add new trips and mark existing ones as complete.

- **Travel History** (*HistoryFragment*): Displays a list of all completed trips, including details like the destination, date of completion, and a captured photo.

- **Profile** (*ProfileFragment*): A personalized space where users can set a profile picture, view travel statistics (total places visited, pending trips), and manage their account (e.g., logout, change theme).

2. Data Model

The data model is straightforward and designed to capture the essential details of a user's travels. The primary data object is the Travel class.

2.1. Travel Class:

This Plain Old Java Object (POJO) represents a single travel destination.

- *String city*: The primary location (e.g., "Paris, France").
- *String place*: The specific landmark or place of interest (e.g., "Eiffel Tower").
- *String status:* The current state of the trip, typically "Awaiting" or "Completed".
- *String photoUri:* (For completed travels) A Uri stored as a String, pointing to the image the user captured for this memory.
- *String dateCompleted:* (For completed travels) The date the trip was marked as done.

2.2. Data Management:

PendingTravelsViewModel: Holds the *LiveData<List<Travel>>* for all travels with the "Awaiting" status. It is responsible for adding new trips to this list.

CompletedTravelsViewModel: Holds the *LiveData<List<Travel>>* for all travels with the "Completed" status. It receives completed trips from the HomeFragment.

SharedPreferences: Used for lightweight, persistent data storage, including:

- User login credentials and session status.
- The URI of the user's chosen profile picture.

Key Challenges Faced

The development process involved several technical hurdles that were successfully overcome.

1.**UI State Preservation**: A significant challenge was ensuring the app's state was not lost during screen rotations or when the OS terminated the process. Initially, fragment state was lost on recreation. This was solved by:

- Implementing *ViewModel* to hold UI data, making it lifecycle-aware.

- Using *onSaveInstanceState* in the *HomepageActivity* to remember which fragment was active, ensuring the user returns to the same screen after a configuration change.

2.Build and Environment Errors:

- **JDK/jlink Incompatibility**: The project initially failed to build due to a jlink.exe error. This was resolved by explicitly configuring the Gradle JVM in Android Studio settings to a compatible version (JDK 17) instead of relying on the bundled runtime.

- **"Constant Expression Required"**: A classic Android issue arose when using resource IDs *(R.id.history*) in a *switch* statement. The solution was to refactor the switch blocks into *if-else if* chains, which is the modern, recommended approach.

- **SDK Download Corruption:** The emulator system image failed to download with a "Not in GZIP format" error. This was traced to a corrupted download cache and was fixed by clearing the SDK's *.temp* folder and restarting the download.

3.Fragment Communication: A core architectural challenge was passing data between different fragments. For instance, when a travel item is marked as "complete" in *HomeFragment*, it needs to be removed from the pending list and added to the history list (managed by *HistoryFragment*). This was solved by using a Shared ViewModel (*PendingTravelsViewModel* and *CompletedTravelsViewModel*), scoped to the *Activity*. This allows both fragments to access and observe the same data source, ensuring consistency.