**App Development Proposal**

**Mobile2App: Inventory App**

The Mobile2App mobile inventory app is a native Android OS application designed to track items in a warehouse (or similar) environment. The app consists of a database that includes tables for inventory items and (encrypted) login credentials for users. The app starts with a login (or sign-up) screen and allows the user to progress to the main screen, displaying the inventory items (in a grid format). The user can add or remove items, increase or decrease the quantity of items, and receives a notification when an item is completely depleted.

**User Groups**

A “casual user” is expected to use this app for a very limited amount of time to keep track of valuable personal inventory, but not most items in their lives. This will require easy-to-learn and quick-to-use functionality, so as to not overwhelm the user with a cumbersome process that is not really worth learning for such limited use. A “power user” will use this app daily to keep track of many more items in their possession, compared to the more casual user. The app needs to be scalable, so that many users will be able to store a significant number of items in their inventory, without storage or network limitations. A “professional user” will use this app multiple times a day on an ongoing, long-term basis. All these user groups demand reliable and stable user experiences (UXs), as well as robust security to protect their customers’ (and the company’s) sensitive data. The needs and wants of each of these user groups overlap with one another, as all these factors are critical for anybody using the app, no matter the user group they belong to.

**Development Overview**

This app will be developed using Android Studio and Jetpack Compose with Oracle’s Java programming language and XML (eXtensible Markup Language), although updates may include Kotlin integration as well. Following industry standard guidelines, the app should be designed to be intuitive and delightful to use, provide significant value to the user, make the most of premium device features, and be designed for safety (Google, n.d.). The app should provide a wide range of accessibility options and support a dynamic array of devices. Privacy will be supported by minimizing permission requests, minimizing location access, and minimizing data visibility across apps. Security practices will be followed, such as industry standard encryption technology, data integrity, and proper user authentication.

**User Interface (UI)**

The user will initially be presented with a login (or sign-up) screen using TextViews in the top row to display the screen title “Login” in large, stylized font. “Username: “ and “Password: “ labels (TextViews) should appear in the 1st column, in the 2nd and 3rd rows, respectively. Then, EditText boxes can appear to the right of each of these two labels, so that the user may enter their login (or sign up) credentials, with the password characters being obscured using asterisks (‘\*’). A “Submit” Button should be placed near the bottom row to allow the user to trigger the authentication event and query the database. If the user account does not exist, an account will be created, and the user will be logged in. A TextView at the bottom of the screen will inform the user of this possible account creation. A toast or Snackbar notification should be created to indicate to the user of authentication success or failure.

The user will then be presented with the “Permissions” screen that alerts the user that the following pop-up dialog will ask for (dangerous runtime) permission to send SMS text messages to their device when low inventory for any item is detected. Upon clicking a “Next” button, the Android permission request dialog will display with the title “SMS Request” and message “Receive text messages when inventory items are low?” with the buttons (left to right) “MAYBE LATER”, “NO”, and “YES”. The “Login” and “Permissions” screens will be bypassed during subsequential app restarts, assuming the user has successfully been authenticated and the permission request has been completed.

The main user interface (UI) screen will be implemented using the GridLayout or FrameLayout ViewGroup (with RecyclerView for dynamic item addition) to arrange the inventory items in a grid pattern and provide a fast interface with low memory requirements. An Item class will be created that consists of name, id, and quantity attributes to be displayed in TextViews to the user. Member methods should be created to implement the required functionality for inventory management and notifications. Each item in the grid shall display a logical label and header. An “Add” floating action button (FAB) centered at the bottom of the main screen will be featured, in order to allow the user to add items to the inventory grid. Upon clicking this button, an EditText with a label of “Item Name: “ will be shown, followed by an EditText with a label of “Quantity: “ and default value ‘1’. Upon entering text into the “Item Name: “ EditText, an “add” button will be dynamically enabled to finalize the addition of the item to the inventory. Each row of data in the grid shall contain a “Delete” button to delete that row of data, with a pop-up confirmation dialog with title “Delete Item” and message “Completely delete this item from your inventory?” and “NO” and “YES” buttons. A Snackbar or toast will confirm the addition and deletion of any items. The user will be able to change the number of items for each item, using either ‘+’ or ‘-‘ buttons or optionally entering a value directly into an EditText. Changes are automatically changed.

**Architecture**

The mobile app may be based on the Model-View-Controller (MVC) architecture. For example, activity\_main.xml is a View that will display the app UI of View objects and handle user and Android operating system interactions. MainActivity.java can contain the app logic code that responds to the View and controls flow between the Model and View. The Models include repository classes, database helper classes, service classes, and other data model classes that define the properties and behaviors of each and provide a layer of abstraction between code and data sources. These data models will ideally be persistent and will drive the UI to avoid user data loss caused from Android OS destroying the apps due to limited resources or when network connections are poor.

A Model-View-ViewModel (MVVM) architecture that implements a Repository design pattern is more complicated but provides better modularity and easier testing and maintenance and will be considered (McCown, F., 2023). A single source of truth (SSOT) can act as the owner of any new data type defined in the app and will enforce immutable (unchangeable) data types. SSOT can be used in conjunction with a unidirectional data flow (UDF) pattern to ensure state only flows in a single direction, and events modify the data flow in the opposite direction. In a UDF pattern, the Data Layer is composed of this SSOT and obtains and exposes data to a reactive UI. Although the specific implementation may vary a little from the described architectures, the key is to separate the code across different layers, including functionality, states, services, APIs (application programming interfaces), data stores, data sources, data repositories, and other components of the application. By following these guidelines, the architecture will help provide a robust, modular, testable, and maintainable application that provides data consistency and is less prone to errors (Google, n.d.).

**Implementation Details**

For the login/sign-up screen, the user enters an email by interacting with the View (EditText) objects with the code found in activity\_login.xml. When the user clicks the “Submit” button, the data is sent to the Controller, LoginActivity.java, which will handle input sanitization/validation and error handling. The input for the email address will need to follow a standard email format, along with a password of at least 8 characters that start with a letter and used in conjunction with at least one numerical digit and special character. The Controller will send the database requests to the Model, which will implement 256-bit AES encryption for data sources and include DAOs (database access objects), repositories, and other classes, such as User.java, Inventory.java, and Item.java. Accessing a database server will require network permissions from the Android device using the app. Once the authentication of the user is complete, the Model sends the requested data back to the Controller to enforce the appropriate authorization (permissions and role). The Controller then updates the View with the up-to-date inventory information received from the Model. The user then sees the update on their screen, as a toast notification that either confirms or denies successful login. The same basic process will be executed when the user interacts with the activity\_inventory.xml View, which will send data to the Controller (InventoryActivity.java) A user may manually input an integer value for the quantity of an item; otherwise, a toast notification will notify the user of this rule upon attempted submission. The Controller will then interact with the Model to update and retrieve inventory data. The Controller will receive a response from the Model and update the View accordingly for the user to see. This is the general framework that will be used. I look forward to any questions, concerns, and thoughts you may have about this App Development Proposal for the Mobile2App Android inventory app!

Thank you,

Matthew Pool

**References**:

Google. (n.d.). Android Developer. *Design*. <https://developer.android.com/>design.

McCown, Frank. (2023). *CS 360: Mobile Architecture & Programming*. ZyBook. <https://learn.zybooks.com/zybook/CS-360-R3349-OL-TRAD-UG.24EW3>.