Victor Feight

11/13/2021

CS-360

Option 1: Inventory App Development Proposal

Contents

[User Requirements 1](#_Toc87830180)

[Project Goals 2](#_Toc87830181)

[Potential Users and User Goals 2](#_Toc87830182)

[Setting Up Material UI Components 3](#_Toc87830183)

[Necessary Screens and Feature Requirements 4](#_Toc87830184)

[Functional Requirements (Data flow) 7](#_Toc87830185)

# User Requirements

We have several user requirements which we need to develop and test, including:

1. A database with at least two tables, one to store the inventory items and one to store user logins and passwords
2. A screen for logging into the app. Note that this should also be used to create a login if the user has never logged in before.
3. A screen, with a grid, that displays all items in the inventory
4. A mechanism by which the user can add and remove items from inventory
5. A mechanism by which the user can increase or decrease the number of a specific item in the inventory
6. A mechanism by which the application will notify the user when the amount of any item in the inventory has been reduced to 0 (zero)”

# Project Goals

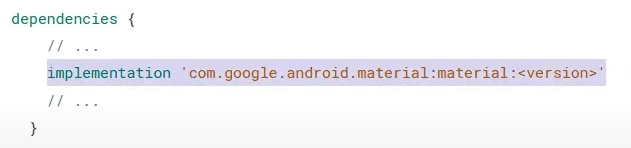
The goal is to create a minimalistic, KISS inventory tracking app using material UI guidelines and interface components that can be later tailored to company-specific guidelines. At the moment, the primary focus is implementing all of the features in a seamless UI using Material Design components. The app’s design purpose is specifically for inventory item counts in a warehouse, thus, in default column mode there should be a list-based interface, where each item is a product to track. Each line item should should be connected to an increment stepper Material UI component to increase or decrease stock count, and each should track when a count has reached 0. Each button should have a delete (x) button, which will decrement the total count of that current type. Thus each item should be clickable, and there should be an easy to click and appropriately sized ( + ) button on the bottom app bar. At any time the last inventory item in the database is reduced to 0 count, a toast notification should notify the user that the inventory is empty.

# Potential Users and User Goals

Potential users include warehouse managers or owners who are looking for a streamlined app geared toward their specific needs or products. Another type of user might be a general purpose entreupeneur who needs a simplistic product tracker with no frills. Last might be a small business owner who is looking for a fast and easy app for tracking, as they may not have the time to fiddle with advanced features and is looking for simple functionality and a clean easy interface. All three seek a responsive and aesthetic, clean and easy to use interface. In the end, speed, ease of use may be the 2 most defining aspects for an app of this type followed by UI design.

# Setting Up Material UI Components

App-wide Material Design-based UI requires importing the dependency:



Without our gradle build file, add to dependencies section (ensuring to add the correct version via Google’s MVN repository and searching for ‘material’, 1.5.0-beta01 at the time of this writing).

Extending AppCompatActivity is necessary to ensure all Material Design components work correctly.

The Material App theme can also be set allowing an app-wide theme, in this case in will be Theme.MaterialComponents.Light.

This can be set in app 🡪 res 🡪 values 🡪 styles.xml

<style name="Theme.MyApp" parent="Theme.MaterialComponents.Light">

<!-- ... -->

</style>

# Necessary Screens and Feature Requirements

1. A screen for logging into the app. Note that this should also be used to create a login if the user has never logged in before.

One of our databases will hold a list of usernames and their associated password credentials. The upper portion of the GridLayout ViewGroup could contain a logo for the app, followed by Username and Password prompts and a login button on the following grid cells. As per the Material Design guidelines, visual hints should be used to distinguish selected input, and prefilled text can save space when labelling fields. The following is an example of Material Design login UI fields:



A “New User?” link will bring the user to a new screen with text input fields prompting for a username and password. Upon entering and pressing “Submit” button, the new user information will be processed into the user credential Sqlite database. The screen will then revert to the login screen, prompting again for credentials. Visual hints should be used again to distinguish selected input in this case.

1. A screen, with a grid, that displays all items in the inventory

Contents can be placed into a single column, with appropriate gutter space between items, and margins at either end of the layout. All three elements can be achieved with Material UI **responsive grid layout**. The responsive grid shall adjust to appropriate screen size dpis as necessary from extra small phone display up to Large desktop displays (“Responsive column grid”, Material Design guidelines).

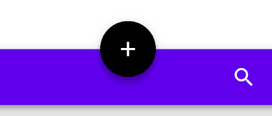
We create a visual grouping of the items pulled from an SQLite database, and scale them to the recommended 40-60 character per list item line. This way we can have images for each item scaled alongside a description at a reasonable height. Type should be vertically center-aligned within its component on the 4dp grid as in the following image:



1. A mechanism by which the user can add and remove items from inventory.

The main view which pulls entries from the database will display a scrollable list of all inventory items.

**Adding items**: The bottom App bar should be actionable, as per Material Guidelines. Future requirements might require a search functionality, which would be placed along the buttom bar. Adding an item can be done by pressing a ( + ) add button on the bottom app bar, which would transition the user to dialog asking for a new item name and quantity. After entering appropriate item name and quantity, the item will be added to the database and updated in the list items. The following shows an add item button for the app:



**Removing items**: The list title should be a primary action, which would trigger an edit action. Pressing and holding the list title of an item will trigger the primary action, opening a dialog to delete an item or cancel. This can be implemented using a **dialog** Material Design component. Pressing cancel will revert the user to the initial screen of list items in a grid column. As per the Google Core App quality guidelines, the app should preserve user state when leaving the leaving the foreground, the app should correctly preserve user state when reverting to the list items, preventing any data loss due to back-navigation or state change.

1. A mechanism by which the user can increase or decrease the number of a specific item in the inventory.

Each list item in the list view will have a right-justified input-stepper as a secondary action. A ( - ) icon for decrement, an integer to display current number of items, and a ( +) button for increment on each list item would allow for easily increasing and decreasing stock count. Negative quantities should be rejected with a visual hint or popup.

The following shows an example:



Note the ( - ) decrement button is greyed out when there are no remaining items. Upon reloading the screen, the item will be removed from the view.

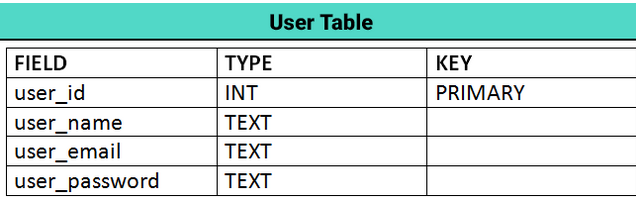
1. A mechanism by which the application will notify the user when the amount of any item in the inventory has been reduced to 0 (zero)”

This feature requires use of a Material Designs “Snackbar” component. This will allow a toast popup notification that notifies the user of an empty database without detracting from the overall UI experience, thus adhering to the Google Android core app guideline “Set timeouts for notifications where appropriate.”

# Functional Requirements (Data flow)

1. **SQLite tables**: A database with two tables is required for the app, one to store the inventory items and one to store user logins and passwords. An onCreate() method can be used to create two initial SQLite databases, one for “Products”, and one for “User Credentials”.
2. **Database schema:** The products database will contain two fields, one for product name and one for initial product quantity. Negative quantities should be rejected with a visual hint. The user credentials database will contain four fields, for user\_id, user\_name, user\_email, and user\_password.

The following image shows a user database schema:



1. **Login form**: Activity\_login.xml will contain the login form, with the logo on top, two input fields for email and password, a login button, and a link to registrate. These are wrapped in LinearLayoutCompat which is useful for items displayed in a single column.
2. **Input validation class**: Validation methods for input such as validating empty input and validating email and password.

The grid will be assigned within ConstraintLayout in activity\_main.xml, with an id such as @+id/grid\_item.

Update strings.xml with appropriate values such as app\_name, hint\_name, hint\_email, hint\_password, text\_login, text\_register, and also error messages for bad input.

Create a User model class with getter and setter methods used to retrieve user’s name, email, and password, and set the three.

Create a database helper class, and extend the class with SQLiteOpenHelper. Add some methods to manipulate data in the database, such as addUser, updateUser, and checkuser.

RecyclerView library can be used to display a large list of scrollable data that may change frequently. In our case, the inventory item list may be edited frequently, so we use a Recycler view.

To implement the Snackbar, we reference our grid items in our MainActivity.java, and create an onClickListener, allowing snackbars in response to “zero items in database” event, passing a Snackbar.LENGTH\_LONG for the timeout length.

**Data flow summary:**

Initial Screen: Login (user name input, password input), button 🡪

* Press button with incorrect or incomplete credentials 🡪 a snackbar warns you of incomplete/invalid credentials
* Press button with correct credentials 🡪 Successful login snackbar 🡪 Switch to Inventory items screen
* Press Registration button 🡪 Switch to Registration screen

Registration screen: Text field inputs accepts user\_name, user\_email, user\_password and stores them to the database upon successful entry and pressing “Register” button.

* Upon successful registration, add user account to database and revert to Initial login screen to re-input credentials.

Inventory data screen: When a user increments, increase count for that particular line item. When a user decrements, decrease count for that particular line item unless zero, then delete the item from the list using RecyclerView implementation.

* When a user presses ( + ) add button, display add new item dialog.

Add new item dialog: Text input fields accept item name and initial item quantity. Upon pressing “Enter item”, item is inserted into SQLite product database. Press and hold the item to bring up an edit screen, prompting user to either cancel or delete the item outright. Upon deleting an item, reduce item’s count to 0 and remove from the view.

* Upon pressing cancel, return user to the inventory data screen.

Discuss how the functional app requirements will be represented in the code and connected to the UI. You should explain the calls that show the flow of data between code and screens. When explaining what data calls you may need to make, you should list the major UI components on each screen, then determine what data each component will either display or accept as input and where the data might come from.

Works Cited

*Material Components*. Material design. (n.d.). Retrieved November 14, 2021, from https://material.io/components.

*Understanding Layout*. Material design. (n.d.). Retrieved November 14, 2021, from https://material.io/design/layout/understanding-layout.html#principles.