

Project Report
on
Grocery Basket
(v. 3.0)

Contents

- *Introduction*
 - *Overview*
 - *Purpose*
- *Literature Survey*
 - *Existing problem*
 - *Proposed Solution*
- *Theoretical Analysis*
 - *Block Diagram*
 - *Hardware/ Software requirements*
- *Experimental Investigations*
- *Flow chart*
- *Material Components used*
- *Result*
- *Advantages & Disadvantages*
- *Application*
- *Conclusion*
- *Future Scope*
- *Bibliography*

Appendix

A. Source Code

Introduction

Hi....

This is Rammohan Locharla.

*I build an application called “**Grocery Basket**” in android using android studio and Kotlin Programming Language. Its version number is “**3.0**”.*

Overview

*Many times, we forget to purchase things that we want to buy, after all, we can't remember all the items. We are going to create a **Grocery Basket** Android App using MVVM and Room Database in Kotlin. In this project, we are using MVVM (Model View ViewModel) for architectural patterns, Room for database, Coroutines and RecyclerView to display the list of items.*

Purpose

So, with the help of this app, we can note down your grocery items that we are going to purchase, by doing this we can't forget any items that we want to purchase. We can make a list of the groceries we intend to buy. This is a Simple grocery shopping list app for pantry check and quick shopping! Make a grocery list in seconds

Yours faithfully,

Rammohan Locharla (Jungle Mystic)

Mail: locharla.rammohan@gmail.com

Mobile No: +91 8309583550

Google Developer Profile Link: <https://q.dev/rammohanlocharla>

SBID: SB20220248310

GitRepo ID: SPSGP-102704

Literature Survey

a. Existing problem

While we are going to market to purchase some groceries to our house. We probably put together a grocery list before heading to the store — whether it's in our head or on a piece of paper. But how often do we actually remember to bring that list with us so we can shop faster? When we went to market, in mind list -> we will forget to buy some groceries or we forget to take that list to market. Either way we end up in going again to market or else our mother or father or spouse will scold us.

b. Proposed Solution

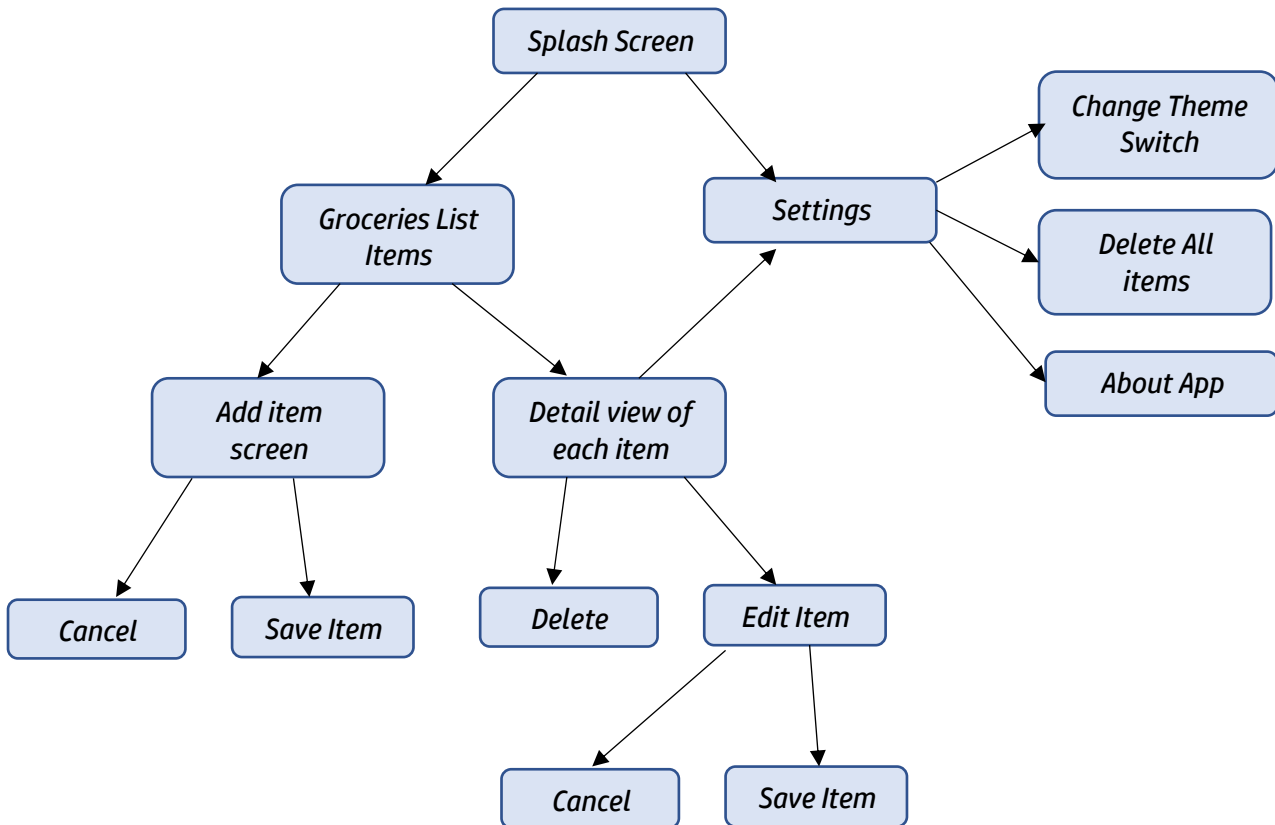
In this Digital world, we need digital solution for this problem. Anywhere we go, we won't forget to take our mobile with us. So, I created an app, that will take out all the work for us. Simplify the planning of our grocery shopping with this simple app.

So, with the help of this app, we can note down your grocery items that we are going to purchase, by doing this we can't forget any items that we want to purchase. We can make a list of the groceries we intend to buy.

Theoretical Analysis

a. Block Diagram

Diagrammatic overview of the project



b. Hardware/ Software requirements

- Windows PC/Laptop or Apple Mac
- Download and install Android Studio from the below link
<https://developer.android.com/studio>
- Physical (I used OnePlus 8 Pro mobile) or Virtual Android mobile.
- Internet Connection to access various resources.
- USB Cable (in case of using physical device).

Experimental Investigations

- **RecyclerView**

A RecyclerView is an advanced version of ListView with improved performance. When you have a long list of items to show you can use RecyclerView. It has the ability to reuse its views. In RecyclerView when the View goes out of the screen or not visible to the user it won't destroy it, it will reuse these views. This feature helps in reducing power consumption and providing more responsiveness to the application.

Steps for implementing your RecyclerView

If you're going to use RecyclerView, there are a few things you need to do. They'll be discussed in detail in the following sections.

First of all, decide what the list or grid is going to look like. Ordinarily you'll be able to use one of the RecyclerView library's standard layout managers.

Design how each element in the list is going to look and behave. Based on this design, extend the ViewHolder class. Your version of ViewHolder provides all the functionality for your list items. Your view holder is a wrapper around a View, and that view is managed by RecyclerView.

*Define the Adapter that associates your data with the ViewHolder views. When you define your adapter, you **need to override three key methods**:*

onCreateViewHolder (): *RecyclerView calls this method whenever it needs to create a new ViewHolder. The method creates and initializes the ViewHolder and its associated View, but does not fill in the view's contents—the ViewHolder has not yet been bound to specific data.*

onBindViewHolder (): *RecyclerView calls this method to associate a ViewHolder with data. The method fetches the appropriate data and uses the data to fill in the view holder's layout. For example, if the RecyclerView displays a list of names, the method might find the appropriate name in the list and fill in the view holder's TextView widget.*

getItemCount (): RecyclerView calls this method to get the size of the data set. For example, in an address book app, this might be the total number of addresses. RecyclerView uses this to determine when there are no more items that can be displayed.

- **Card View**

CardView is a new widget in Android that can be used to display any sort of data by providing a rounded corner layout along with a specific elevation. CardView is the view that can display views on top of each other.

The main usage of CardView is that it helps to give a rich feel and look to the UI design. This widget can be easily seen in many different Android Apps. CardView can be used for creating items in listview or inside RecyclerView.

The best part about CardView is that it extends Framelayout and it can be displayed on all platforms of Android. The cards are drawn to the screen with a default elevation, which causes the system to draw a shadow underneath them.

- **Fragments**

A Fragment represents a reusable portion of your app's UI. A fragment defines and manages its own layout, has its own lifecycle, and can handle its own input events. Fragments cannot live on their own--they must be hosted by an activity or another fragment. The fragment's view hierarchy becomes part of, or attaches to, the host's view hierarchy.

Fragments introduce modularity and reusability into your activity's UI by allowing you to divide the UI into discrete chunks. Activities are an ideal place to put global elements around your app's user interface, such as a navigation drawer. Conversely, fragments are better suited to define and manage the UI of a single screen or portion of a screen.

Dividing your UI into fragments makes it easier to modify your activity's appearance at runtime. While your activity is in the STARTED lifecycle state or higher, fragments can be added, replaced, or removed. You can keep a record of these changes in a back stack that is managed by the activity, allowing the changes to be reversed.

- **Navigation component**

The Navigation Architecture Component simplifies navigation implementation while also assisting you in visualizing your app's navigation flow. The library offers a variety of advantages, including:

- *Handling of fragment transactions automatically*
- *By default, up and back actions are handled correctly.*
- *Default animation and transition behaviors*
- *Deep linking is regarded as a first-rate operation.*
- *Implementing navigation UI patterns (such as navigation drawers and bottom navigation) with minimal additional effort.*
- *When navigating Android Studio tooling for visualizing and editing an app's navigation flow, use type safety when passing information.*

The Navigation Component is made up of three major parts:

Navigation Graph — *This is a resource that collects all navigation-related data in one place. This includes all of the locations in your app, referred to as destinations, as well as the possible paths a user could take through your app.*

NavHostFragment — *This is a unique widget that you can include in your layout. It shows various destinations from your Navigation Graph.*

NavController — *This is an object that keeps track of where you are in the navigation graph. As you move through a navigation graph, it orchestrates the swapping of destination content in the NavHostFragment.*

- **ViewModel**

The ViewModel class is designed to store and manage UI-related data in a lifecycle conscious way. The ViewModel class allows data to survive configuration changes such as screen rotations.

The Android framework manages the lifecycle of UI controllers, such as activities and fragments. The framework may decide to destroy or re-create a UI controller in response to certain user actions or device events that are completely out of your control.

Architecture Components provides ViewModel helper class for the UI controller that is responsible for preparing data for the UI. ViewModel objects are automatically retained during configuration changes so that data they hold is immediately available to the next activity or fragment instance.

ViewModel's only responsibility is to manage the data for the UI. It should never access your view hierarchy or hold a reference back to the Activity or the Fragment.

- **Coroutines**

On Android, coroutines help to manage long-running tasks that might otherwise block the main thread and cause your app to become unresponsive. Over 50% of professional developers who use coroutines have reported seeing increased productivity. This topic describes how you can use Kotlin coroutines to address these problems, enabling you to write cleaner and more concise app code.

Noteworthy features include the following:

Lightweight: *You can run many coroutines on a single thread due to support for suspension, which doesn't block the thread where the coroutine is running. Suspending saves memory over blocking while supporting many concurrent operations.*

Fewer memory leaks: *Use structured concurrency to run operations within a scope. Built-in cancellation support: Cancellation is propagated automatically through the running coroutine hierarchy.*

Jetpack integration: Many Jetpack libraries include extensions that provide full coroutines support. Some libraries also provide their own coroutine scope that you can use for structured concurrency.

- **Live Data**

LiveData is an observable data holder class. Unlike a regular observable, LiveData is lifecycle-aware, meaning it respects the lifecycle of other app components, such as activities, fragments, or services. This awareness ensures LiveData only updates app component observers that are in an active lifecycle state.

Using LiveData provides the following advantages:

- Ensures your UI matches your data state
- No memory leaks
- No crashes due to stopped activities
- No more manual lifecycle handling
- Always up to date data
- Proper configuration changes
- Sharing resources

- **Flow**

In coroutines, a flow is a type that can emit multiple values sequentially, as opposed to suspend functions that return only a single value. For example, you can use a flow to receive live updates from a database.

Flows are built on top of coroutines and can provide multiple values. A flow is conceptually a stream of data that can be computed asynchronously. The emitted values must be of the same type.

- **Room database**

The Room persistence library provides an abstraction layer over SQLite to allow fluent database access while harnessing the full power of SQLite. In particular, Room provides the following benefits:

- *Compile-time verification of SQL queries.*
- *Convenience annotations that minimize repetitive and error-prone boilerplate code.*
- *Streamlined database migration paths.*

There are three major components in Room:

- *The database class that holds the database and serves as the main access point for the underlying connection to your app's persisted data.*
- *Data entities that represent tables in your app's database.*
- *Data access objects (DAOs) that provide methods that your app can use to query, update, insert, and delete data in the database.*

- **SharedPreferences**

If you have a relatively small collection of key-values that you'd like to save, you should use the SharedPreferences APIs. A SharedPreferences object points to a file containing key-value pairs and provides simple methods to read and write them. Each SharedPreferences file is managed by the framework and can be private or shared.

You can create a new shared preference file or access an existing one by calling one of these methods:

getSharedPreferences () — *Use this if you need multiple shared preference files identified by name, which you specify with the first parameter. You can call this from any Context in your app.*

getPreferences () — *Use this from an Activity if you need to use only one shared preference file for the activity. Because this retrieves a default shared preference file that belongs to the activity, you don't need to supply a name.*

- **Preferences DataStore**

Preferences DataStore stores and accesses data using keys. This implementation does not require a predefined schema, and it does not provide type safety.

Create a Preferences DataStore

Use the property delegate created by preferencesDataStore to create an instance of DataStore<Preferences>. Call it once at the top level of your kotlin file, and access it through this property throughout the rest of your application. This makes it easier to keep your DataStore as a singleton.

Read from a Preferences DataStore

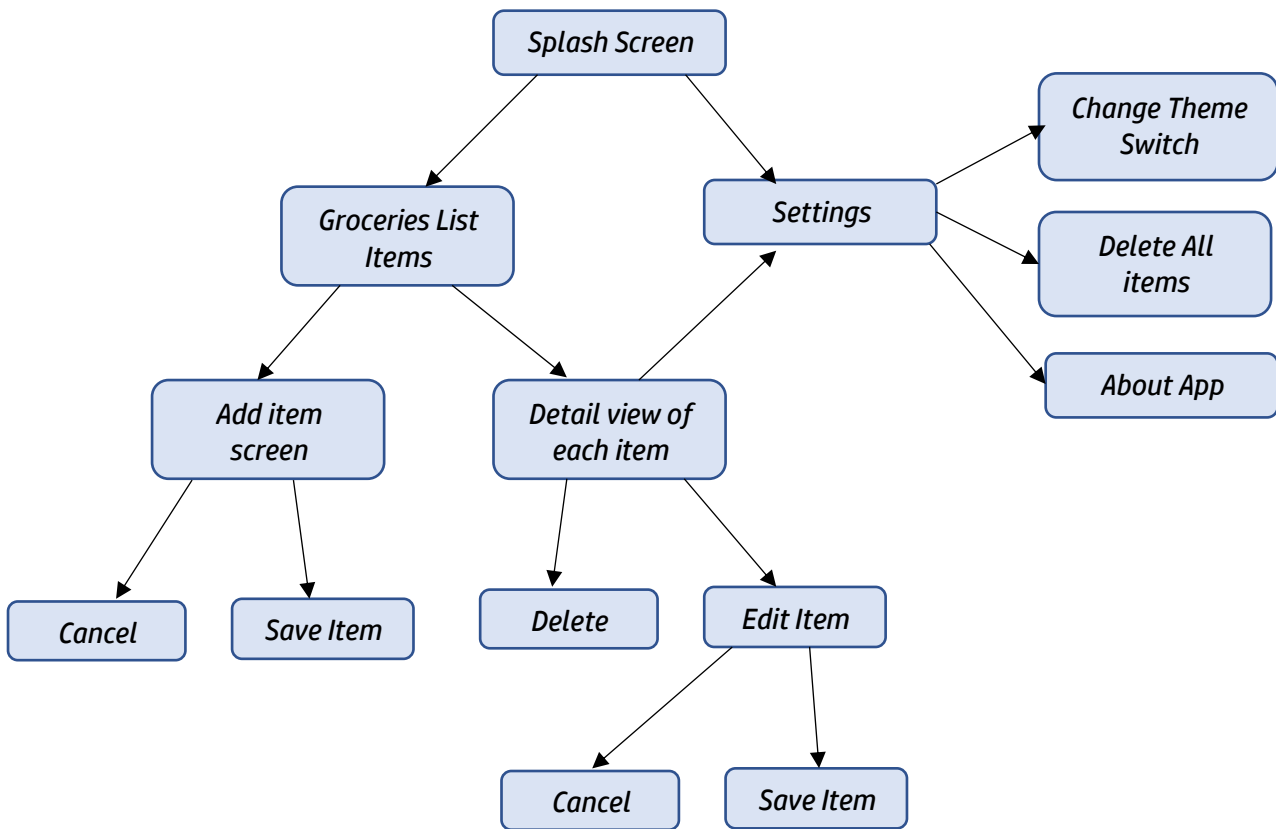
Because Preferences DataStore does not use a predefined schema, you must use the corresponding key type function to define a key for each value that you need to store in the DataStore<Preferences> instance. For example, to define a key for an int value, use intPreferencesKey (). Then, use the DataStore.data property to expose the appropriate stored value using a Flow.

Write to a Preferences DataStore

Preferences DataStore provides an edit () function that transactionally updates the data in a DataStore. The function's transform parameter accepts a block of code where you can update the values as needed. All of the code in the transform block is treated as a single transaction.

- *Also, added simple transition between fragments.*

Flow Chart

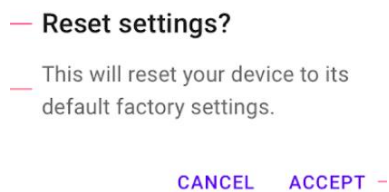


Material Components used

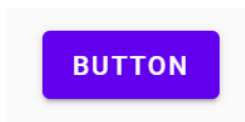
- **Text Fields**



- **Dialog**



- **Buttons**



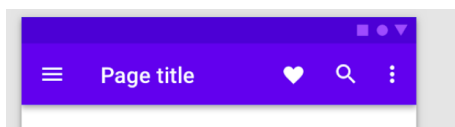
- **Card View**



- **Floating Action Button**

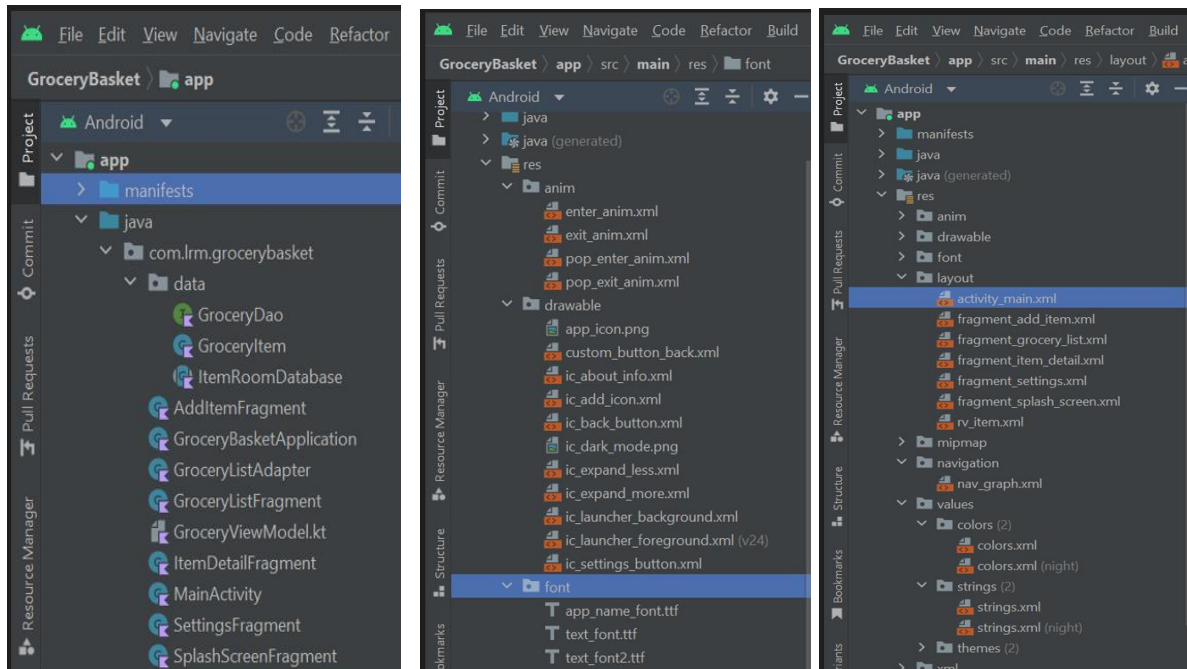


- **Top App bar**



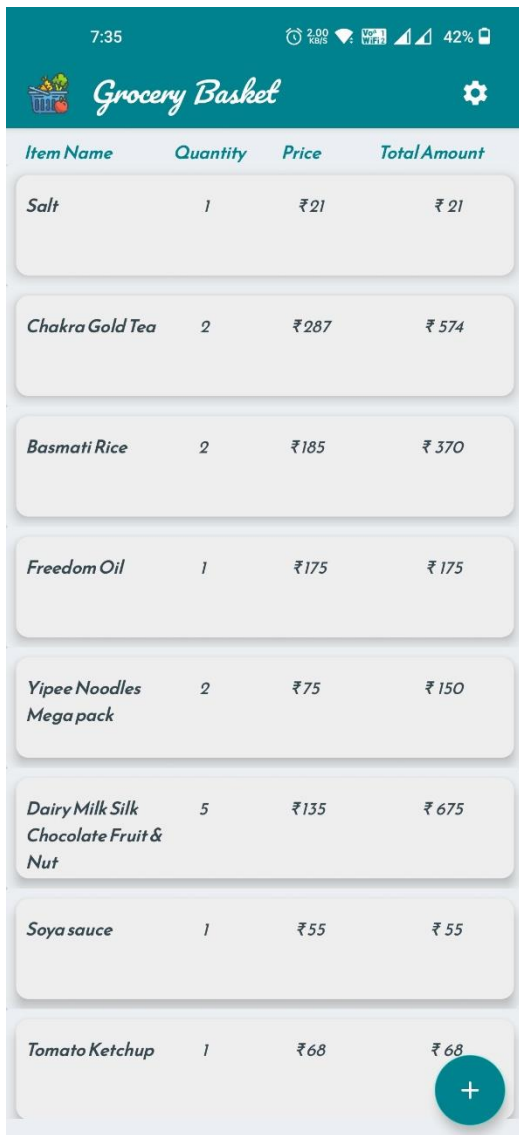
Result

1. Project tab in android studio screenshots

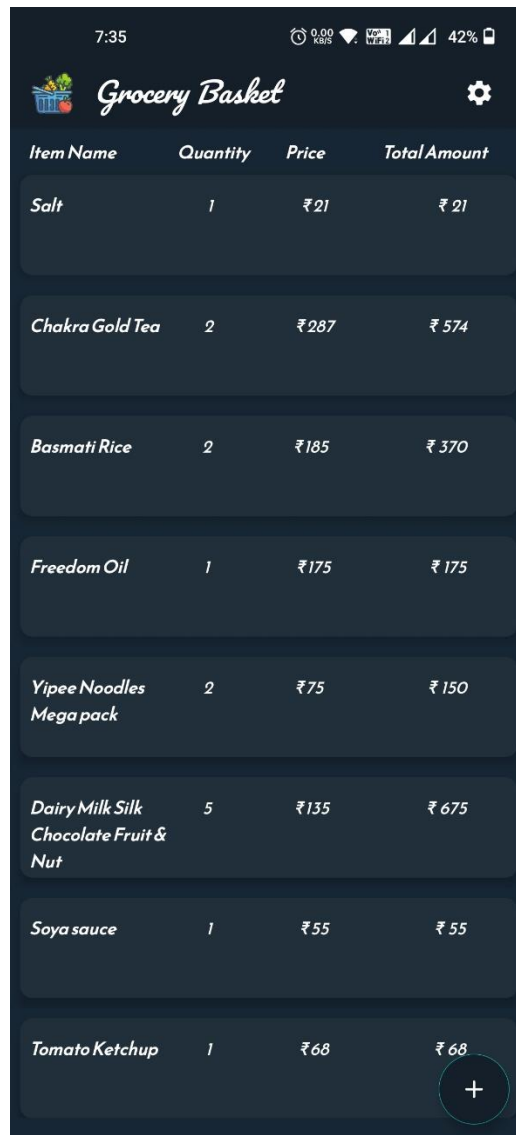


2. App Screenshots

- Final Output of the app will look like this in the Light and Dark Mode.

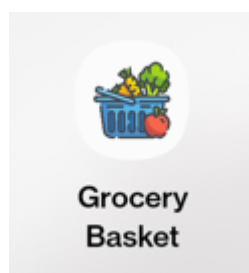


Item Name	Quantity	Price	Total Amount
Salt	1	₹ 21	₹ 21
Chakra Gold Tea	2	₹ 287	₹ 574
Basmati Rice	2	₹ 185	₹ 370
Freedom Oil	1	₹ 175	₹ 175
Yipee Noodles Mega pack	2	₹ 75	₹ 150
Dairy Milk Silk Chocolate Fruit & Nut	5	₹ 135	₹ 675
Soya sauce	1	₹ 55	₹ 55
Tomato Ketchup	1	₹ 68	₹ 68



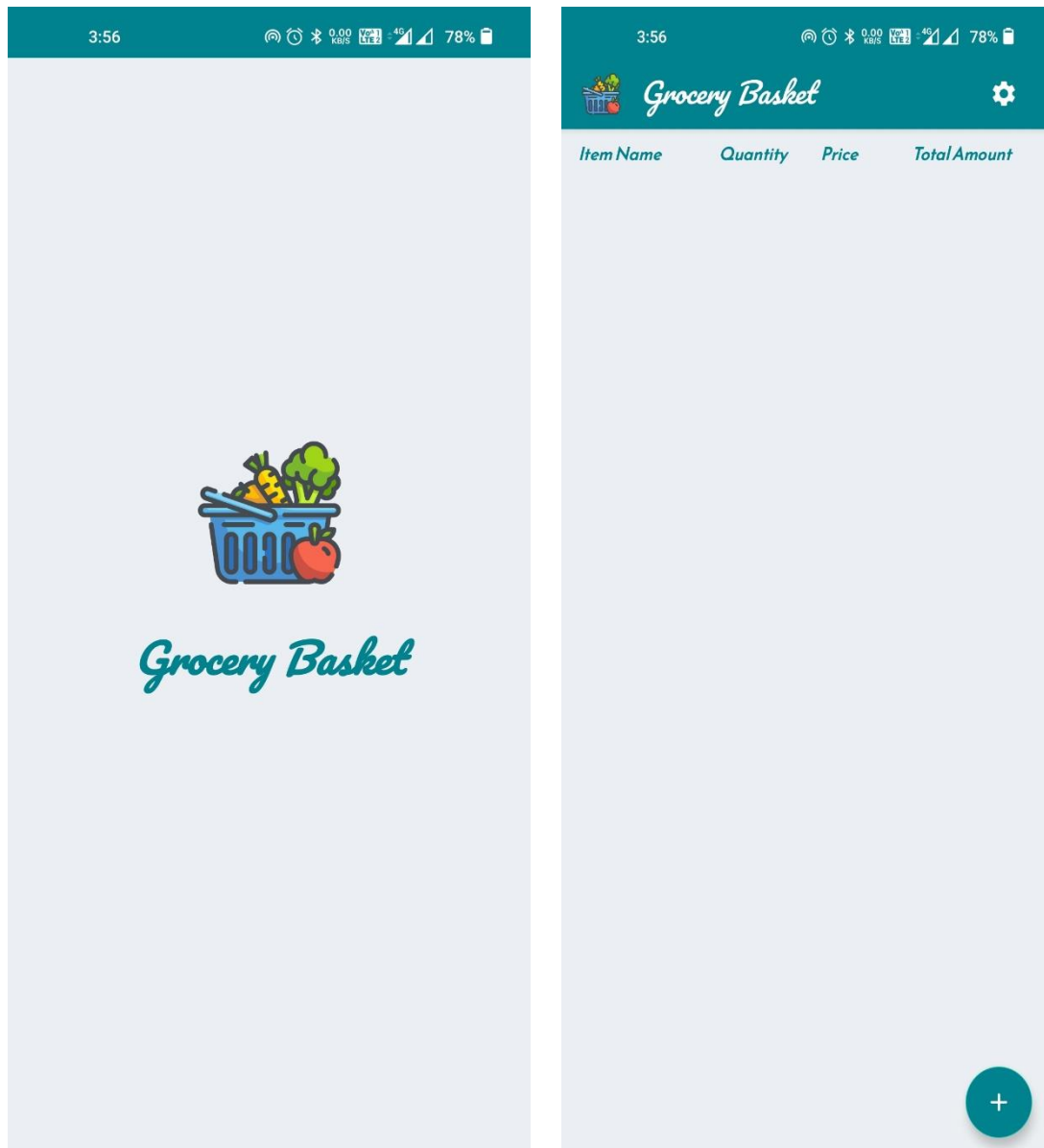
Item Name	Quantity	Price	Total Amount
Salt	1	₹ 21	₹ 21
Chakra Gold Tea	2	₹ 287	₹ 574
Basmati Rice	2	₹ 185	₹ 370
Freedom Oil	1	₹ 175	₹ 175
Yipee Noodles Mega pack	2	₹ 75	₹ 150
Dairy Milk Silk Chocolate Fruit & Nut	5	₹ 135	₹ 675
Soya sauce	1	₹ 55	₹ 55
Tomato Ketchup	1	₹ 68	₹ 68

- App Icon



- On opening app, Splash screen will be displayed

Then, List fragment with no items will be displayed



- *Add item screen and Item Details screen*

3:56
📶 🕒 🔌 0.00 KB/S 4G+ 78%

← Add Item to Basket

Enter Item Name *
Milk Packet

Enter Item Quantity *
2

Enter Item Price/Qty *
₹ 27

Cancel Save

3:58
📶 🕒 🔌 0.09 KB/S 4G+ 78%

← Item Detail
⚙️

Item Name Dairy Milk Silk Chocolate Fruit & Nut

Item Quantity 3

Item Price ₹135

Total Amount ₹405

Delete Edit

- *Item edit screen and Settings Screen*

3:58
📶 🕒 🔌 0.07 KB/S 4G+ 78%

← Edit item in Basket

Enter Item Name *
Dairy Milk Silk Chocolate Fruit & Nut

Enter Item Quantity *
3

Enter Item Price/Qty *
₹135

Cancel Save

3:59
📶 🕒 🔌 0.15 KB/S 4G+ 77%

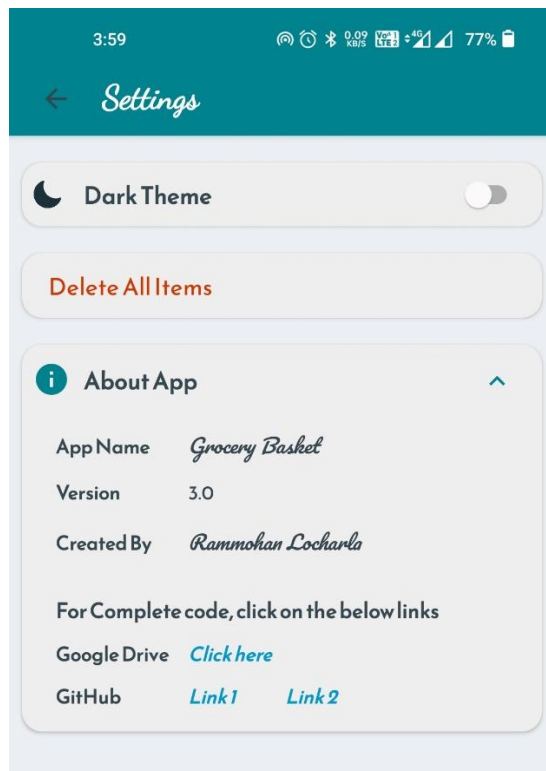
← Settings

🌙 Dark Theme

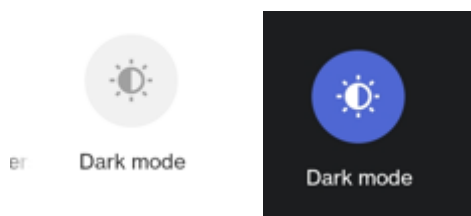
Delete All Items

📘 About App

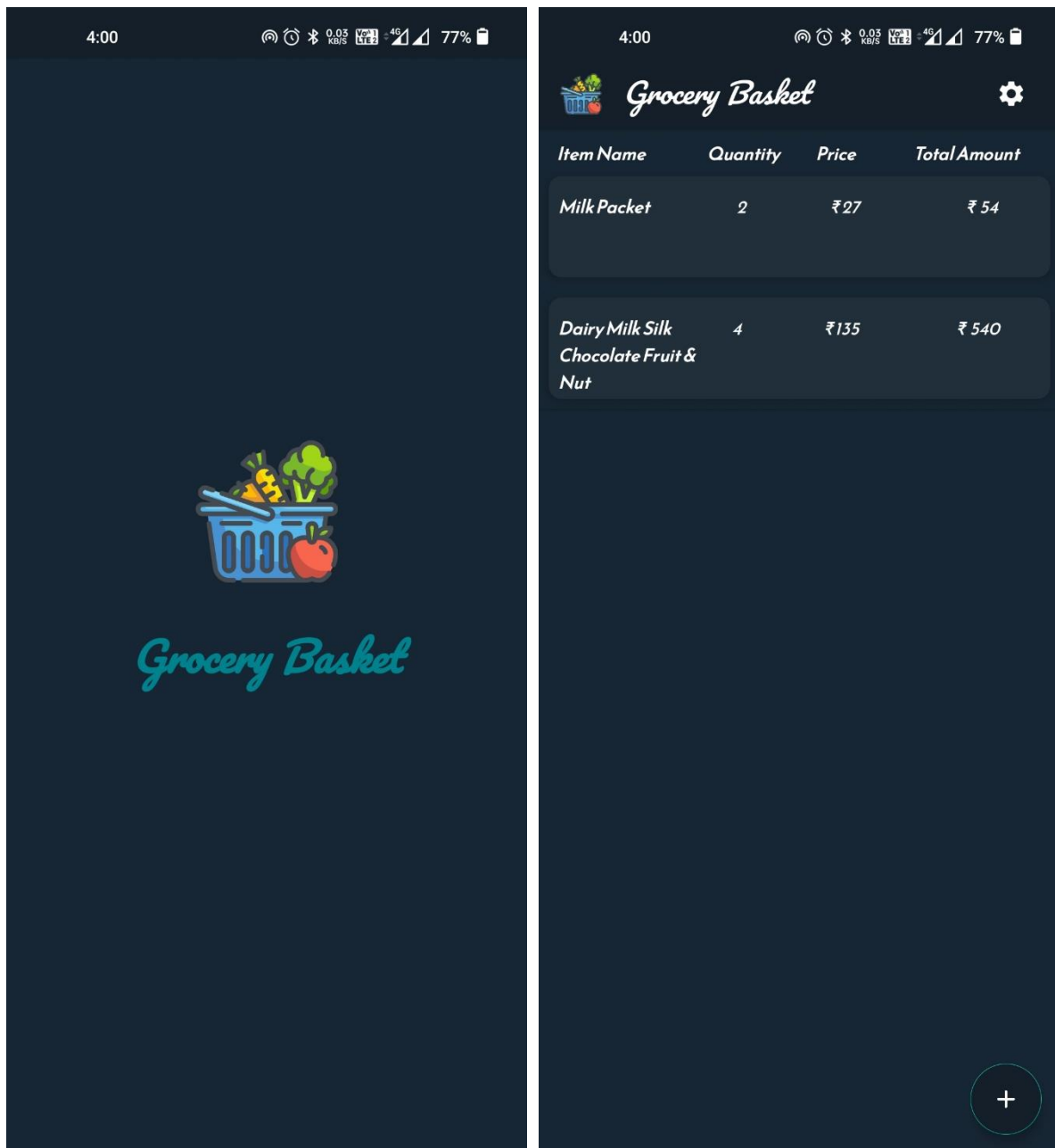
- *About App card expanded*



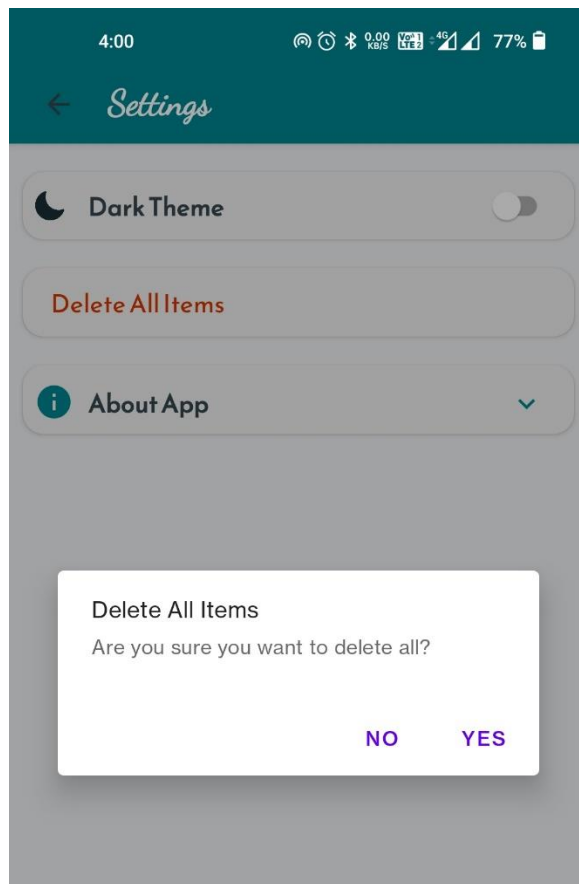
- *Dark mode on*



- After turn on dark mode in the device, Splash Screen and List Item Fragment will look like this...



- On clicking Delete All items in the settings screen



3. For Complete code, please refer

- a. Google Drive Link: [Project Grocery Basket - Google Drive](#)
- b. GitHub Link: [smartinternz02/SPSGP-102704-Virtual-Internship---Android-Application-Development-Using-Kotlin: Virtual Internship - Android Application Development Using Kotlin \(github.com\)](#)

Advantages and Disadvantages

Advantages

- *We can see a list of groceries to buy*
- *We can add a list item*
- *We can edit a list item*
- *We can delete an item*
- *We can delete all items*
- *We can change the theme to Light and Dark mode.*
- *We can see the About app*

Disadvantages

- *We are not saving theme selected, after restart of app the theme will be resetted.*
- *Unable to implement SharedPreferences and Preferences DataStore to store app settings data.*

Applications

- *We can use this app while we go to market to buy some groceries*
- *We can edit the quantity to buy.*
- *We use this app to delete and add new items, so we can buy new groceries*

Conclusion

With this app, we can note down your grocery items that we are going to purchase, by doing this we can't forget any items that we want to purchase. We can make a list of the groceries we intend to buy. This is a Simple grocery shopping list app for pantry check and quick shopping! Make a grocery list in seconds.

- *It saves us time and energy*
- *It saves us money*
- *It helps us to focus*
- *It improves our memory*

Future Scope

- *In the future, I would like to bring update to this app.*
- *Adding new features to app like Finger print access*
- *I want to implement SharedPreferences or Preferences DataStore to store relating to switches of Theme mode selection and Fingerprint access.*
- *I want to share the list to family members and also, they can add, edit and delete items.*

Bibliography

- For App Icons
 - ✓ <https://www.flaticon.com/search?word=Grocery%20Basket>
 - ✓ <https://icons8.com/icons/set/grocery>
- For colors used on layout, text, material components
 - ✓ <https://material.io/resources/color/#!/?view.left=0&view.right=0>
- For Material components
 - ✓ <https://m3.material.io/components>
- For Fonts styles
 - ✓ <https://fonts.google.com/>
- For Knowledge on Android development concepts

Books

- ✓ *Head First Android Development using Kotlin*
- ✓ *Android Apprentice*

Complete guide to Project code

- ✓ https://www.youtube.com/watch?v=vdCLb_Y71Ic
- ✓ <https://developer.android.com/courses/android-basics-kotlin/unit-5>

TRAINING > ANDROID BASICS IN KOTLIN > DATA PERSISTENCE > USE ROOM FOR DATA PERSISTENCE

Appendix

A. Source Code of the project

- *For Complete Code, Kindly, go to below mentioned links...*
 - *Google Drive Link: [Project Grocery Basket - Google Drive](#)*
 - *GitHub Link: [smartinternz02/SPSGP-102704-Virtual-Internship---Android-Application-Development-Using-Kotlin: Virtual Internship - Android Application Development Using Kotlin \(github.com\)](#)*