



الجامعة الألمانية الأردنية
German Jordanian University

German Jordanian University

School of Electrical Engineering and Information Technology

Department of Computer Science

CS4922 - Senior Project 2

Creating an Android Application to help users view any item and its information from a store, and admins modify, update, or add information via a user-friendly application.

Sief Addeen Freitekh – 20189501049

Supervised by: Dr. Ismail Hababeh

Table of Contents

Chapter I: Introduction.....	3
Chapter II: Project Explanation.....	4
i. Software Usage.....	4
ii. Database Design.....	6
Chapter III: Project APIs and External Sources.....	7
i. Back4App.....	7
ii. Parse.....	10
Chapter IV: App Design and Structure.....	14
i. Design and Flow.....	14
ii. UML Diagram.....	24
Chapter V: Conclusion.....	25
i. Future Plans.....	25
ii. References.....	27

Chapter I: Introduction

Databases are used all around the world for many different purposes. One thing I noticed is there is usually not a detailed and well-designed application I could use with a detailed list for checking items in stock for purchase at a specific location.

This project aims to create an adaptable mobile application system for admins that can view and modify/add items to the database, and for users to view and decide what to do with this knowledge.

In this day and age, smartphones are in abundance and a fair majority of people own them. They can be used for social media, entertainment of all types, and offer a various list of applications available at the user's leisure and preference.

My application will be a general purpose mobile android application that can be modified and used to help users know what they will expect before going somewhere. For example if you are to enter a store or location and ask for a specific item, if it isn't available you will feel you wasted a lot of your time going there when it could be solved by digitally checking the offered application and seeing what is in stock and what isn't in detail via an application.

In regards to real life data, it is difficult to access data for a local business or shop, so I will be using a test dataset and database for this application, but it will be a simple process for applying it to a real world situation or objective. For the users, the application will offer a very detailed list of each item, such as the price, item type, color, category and most importantly if it is available or not.

There will need to be a list of prerequisite knowledge and software for this project. These include but are not limited to:

- Programming language knowledge
- Mobile development software
- Database knowledge

- Online database software

Chapter II: Project Explanation

This is a project that requires two essential components to function. These are:

- Development software
- Database

i. Development Software:

The most important component for this project is the programming language to be used, and the software. Mobile applications for Android are almost always programmed in Kotlin or Java. For this application it is coded in Java, for I have essential background knowledge of this language the most. For GUI design coding I am using Android XML, for it is the most accessible and detailed way to design the application. All of this is coded on Android Studio, for it has support for these languages and it is a popular free application for design and coding of mobile Android applications.

In Android Studio, a project consists of two different main files, a source code file that consists of Java or Kotlin code, and an XML file, which is used for design and static user interfaces. XML stands for Extensible Markup Language, and the difference between XML and a programming language is as follows:

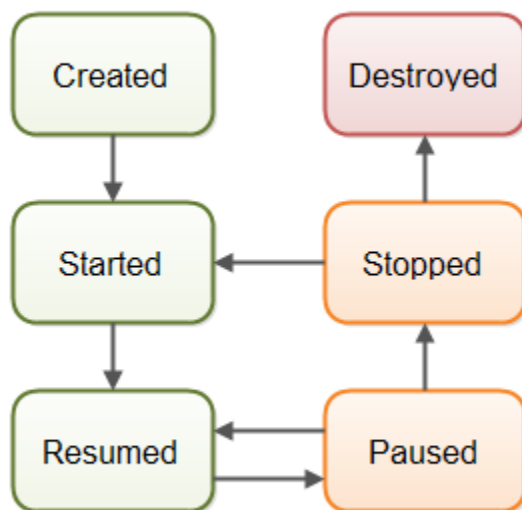
A markup language is slightly different from a programming language. Whereas a programming language (C#, C++, Java, Kotlin, Python, BASIC) will allow you to define behaviors, interactions, and conditions; a markup language is used more to describe data, and in this case, layouts. Programming languages create dynamic interactions, whereas markup languages generally handle things like static user interfaces. {1}

Both of these combine to form an Android Activity, which is essentially a single, focused screen of your application.

An Android activity is one screen of the Android app's user interface. In that way an Android activity is very similar to windows in a desktop application. An Android app may contain one or more activities, meaning one or more screens. The Android app starts by showing the main activity, and from there the app may make it possible to open additional activities. {2}

Refer to figure 2.1 of an Activity Life Cycle:

Figure 2.1



As you can see, There are 6 essential states of an Activity, which are:

- onCreate()
- onStart()
- onResume()
- onPause()
- onStop()
- onDestroy()

These allow many different interactions between activities, for example, Activity A opens Activity B, then Activity A enters a pause state, until the user presses the back button and goes back to Activity A. Activities can be paused in the background while the user goes to back to the home screen, but due to how Android functions, these Activities can be destroyed to free up memory for the device. Note that all Activities follow the path of the arrows in figure 1.1, and cannot jump from state to state not designated by the diagram, for example an Activity cannot jump from Created to Resumed state, or vice versa, but it can go from the Stopped to Started state.

Android Studio is an IDE developed by JetBrains and Google, and it is used worldwide with support for many libraries and APIs, and it can emulate almost any popular android device by downloading its respective SDK.

ii. Database:

This project also includes an online database, which will be used to hold data and information, and interact with the application on the back-end. In respect to the difficulty required to host and create a personal database, I will be using an online hosted one, which includes an API (which is essentially an interface or method for two pieces of software to communicate and interact) that provides many features and comprehensible functions to interact with the application on the front-end.

An online database is a database accessible from a local network or the Internet, as opposed to one that is stored locally on an individual computer or its attached storage (such as a CD). Online databases are hosted on websites, made available as software as a service products accessible via a web browser. {3}

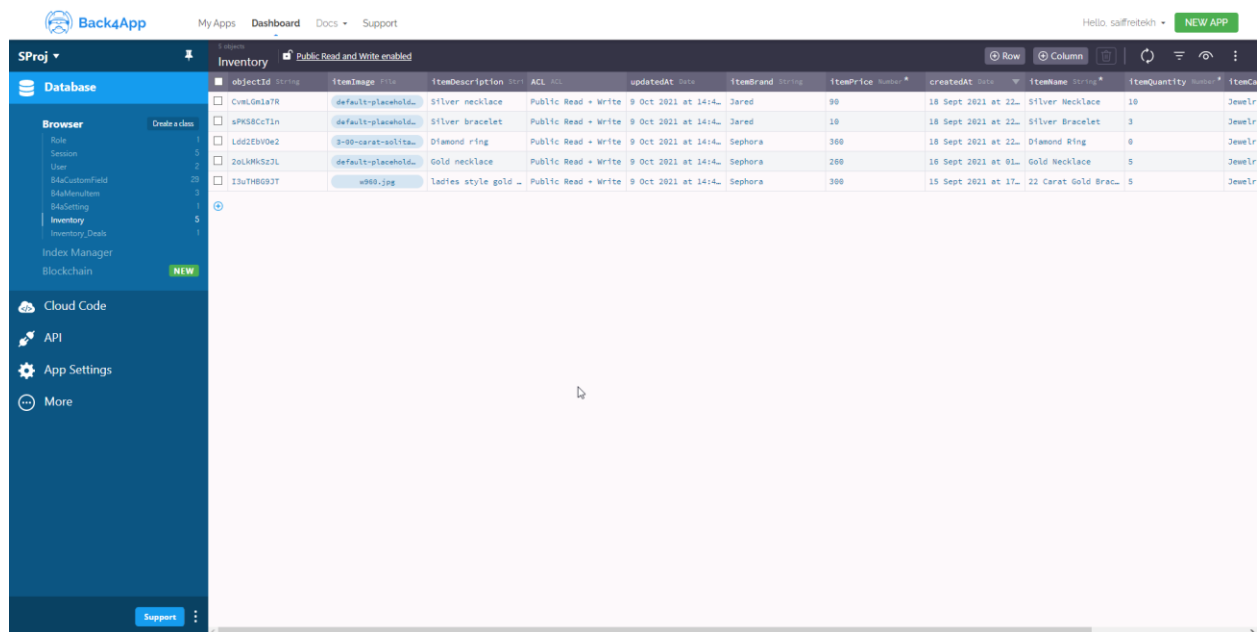
Chapter III: Project APIs and External Sources

i. Back4App:

My project will be using Back4App, which is: *an open source (based on Parse Platform), low code platform that helps you to develop modern apps.*{4}

Back4App is a hosting platform for Parse databases, and is the main back-end source of my project. In the below figure 3.1 shown, is the UI of the Back4App website which hosts our Parse database.

Figure 3.1



The screenshot shows the Back4App dashboard for a project named 'SProj'. The 'Database' section is active, displaying a table of inventory items. The table has columns for objectId, itemImage, itemDescription, ACL, updatedAt, itemBrand, itemPrice, createdAt, itemName, itemQuantity, and itemCategory. There are four rows of data, each representing a different jewelry item.

objectId	itemImage	itemDescription	ACL	updatedAt	itemBrand	itemPrice	createdAt	itemName	itemQuantity	itemCategory
CvM6G1a7R	default-placehold...	Silver necklace	Public Read + Write	9 Oct 2021 at 14:4...	Jared	90	18 Sept 2021 at 22...	Silver Necklace	10	Jewelr
4PK38Ct7In	default-placehold...	Silver bracelet	Public Read + Write	9 Oct 2021 at 14:4...	Jared	10	18 Sept 2021 at 22...	Silver Bracelet	3	Jewelr
L69ZEDV0w2	3-90-carat-solita...	Diamond ring	Public Read + Write	9 Oct 2021 at 14:4...	Sephora	360	18 Sept 2021 at 22...	Diamond Ring	0	Jewelr
2dLKH6523L	default-placehold...	Gold necklace	Public Read + Write	9 Oct 2021 at 14:4...	Sephora	260	18 Sept 2021 at 22...	Gold Necklace	5	Jewelr
13uTH8693T	w860.jpg	ladies style gold ...	Public Read + Write	9 Oct 2021 at 14:4...	Sephora	360	19 Sept 2021 at 17...	22 Carat Gold Brac...	5	Jewelr

In figure 3.2 below, you can see the linking class between the application and the Parse database created on the Back4App website (more information on: <https://docs.parseplatform.org/android/guide/#getting-started>)

Figure 3.2



```
import com.parse.Parse;
import android.app.Application;

public class App extends Application {
    @Override
    public void onCreate() {
        super.onCreate();
        Parse.initialize(new Parse.Configuration.Builder(this)
            .applicationId("YOUR_APP_ID")
            // if defined
            .clientKey("YOUR_CLIENT_KEY")
            .server("http://localhost:1337/parse/")
            .build()
        );
    }
}
```

Parse has a default `_User` table, which has its own function calls within the application, such as `getEmail()`, `getUsername()`, etc. It also interacts with the `Session` table in the sense that it will automatically create a session entry, which includes a pointer to the `_User` table to the user that logs in to the application, and built-in password encryption, so even database administrators cannot view or modify said password.

It includes sign up and log in functions, as you can see in figures 3.3 and 3.4:

In the below figure, the function gives default fields such as setUsername and setPassword, but also allows for other fields such as phone numbers, etc.

Figure 3.3

```
ParseUser user = new ParseUser();
user.setUsername("my name");
user.setPassword("my pass");
user.setEmail("email@example.com");

// other fields can be set just like with ParseObject
user.put("phone", "650-253-0000");

user.signUpInBackground(new SignUpCallback() {
    public void done(ParseException e) {
        if (e == null) {
            // Hooray! Let them use the app now.
        } else {
            // Sign up didn't succeed. Look at the ParseException
            // to figure out what went wrong
        }
    }
});
```

And below, it will take two fields, username/email or password, and check if the matching set exists in the _User table and logs them in.

Figure 3.4

```
ParseUser.logInInBackground("Jerry", "showmethemoney", new
LoginCallback() {
    public void done(ParseUser user, ParseException e) {
        if (user != null) {
            // Hooray! The user is logged in.
        } else {
            // Signup failed. Look at the ParseException to see what happened.
        }
    }
});
```

Back4App also has useful features such as email verification and the ability for the user to change his/her password via method calls that are added by the API and it's corresponding libraries.

ii. Parse:

Parse tables are referred to as Classes. To further understand how to develop an application with Back4App and Parse, we must understand how to correlate our project with the database back-end. To do so, we know our project is an Inventory based application, so the Inventory table will be the main object of our task.

Each ParseObject has a class name that you can use to distinguish different sorts of data. For example, we could call the high score object a GameScore. We recommend that you NameYourClassesLikeThis and nameYourKeysLikeThis, just to keep your code looking pretty.{4}

A Parse database is relatively different from SQL databases, the main difference being the data types that Parse offers. They are as follows:

- Boolean
- String
- Number
- Date
- Object
- Array
- GeoPoint
- Polygon
- File
- Pointer
- Relation

Explanations for each:

- Boolean: Is universally the same idea in which it holds true or false (0 or 1)
- String: Also known as a character array, it holds a set of alphanumeric characters, that must begin with a character and continue with any mix of numbers or characters. Example: dogCat1, itemName2, newInt
- Number: Made up of any type of number such as Integer or Float values, and can be casted within the code. Example: 1, 2.4, 144.56, 65
- Date: Time and date
- Object: As the name suggests, it is an object that can hold a value of any type

- Array: Holds an array of numbers, strings, objects, etc
- GeoPoint: It associates real-world latitude and longitude coordinates with an object
- Polygon: It associates polygon coordinates with an object
- File: Holds data of a file such as an image, text document, etc.
- Pointer: this functions similar to a foreign key, it holds an objectid of an object(such as another table) to reference it.
- Relation: consists of a table of many objects, similar to a many to many relation.

Our projects main focus consists of the _User Table, and the Inventory Table.

We will begin with the _User Table shown as figure 3.5:

Figure 3.5

objectId	emailVerified	isAdmin	ACL	updatedAt	authData	inventory_favorites	username	createdAt	password	email
Dj6V0j8UC	True	False	Public Read, Dj6V0j8UC	10 Oct 2021 at 11:12	(undefined)	View relation	saif	9 Oct 2021 at 17:08	(hidden)	saiffreitakh@gmail.com
Xk5Vx0pKh	True	False	Public Read, Xk5Vx0pKh	10 Oct 2021 at 12:12	(undefined)	View relation	saif1234	9 Oct 2021 at 16:42	(hidden)	saiffreitakh@yahoo.com

As you can see, the User table consists of 11 fields:

- objectId (String)
- emailVerified (Boolean)
- ACL (ACL)
- createdAt (Date)
- updatedAt (Date)
- authData (Object)
- username (String)
- password (String)
- email (String)
- isAdmin (Boolean)
- inventory favorites (Relation)

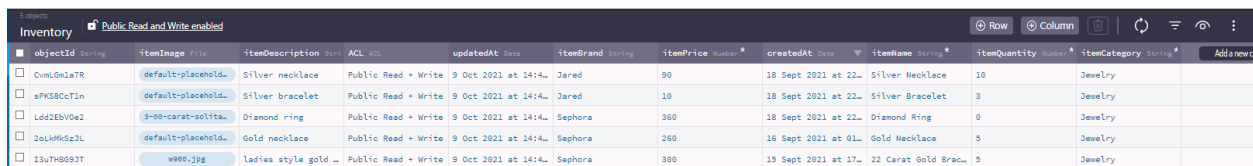
ObjectId is a key generated automatically for each entry in the User table. This is the main key used to reference objects and entries in the source code.

emailVerified is a boolean check to see if the user is registered by email or not to be able to login effectively. ACL stands for Access Control List, its for managing permissions such as read/write,private/public. The columns createdAt and updatedAt are used to know when an entry was created, and when it was last updated. The column authData is used for third party authentication. Of course, there are username, email, and password columns for each user to have their own unique set of data to be able to log in to the app.

I have created two extra columns: isAdmin, and inventory_favorites. isAdmin is a check to see if a user has admin privileges to be able to access the admin menu for the application, for modifying and updating users, inventory items, etc. Inventory_favorites is a relation filled with inventory entries. Each user has their own subtable of favorited items that are available in the Inventory table.

Next is the Inventory Table, shown as figure 3.6:

Figure 3.6



objectId	itemImage	itemDescription	ACL	updatedAt	itemBrand	itemPrice	createdAt	itemName	itemQuantity	itemCategory
CvnlOn1a7R	default-placeholder	Silver necklace	Public Read + Write	9 Oct 2021 at 14:4...	Jared	90	18 Sept 2021 at 22...	Silver Necklace	10	Jewelry
sPK58Ct7In	default-placeholder	Silver bracelet	Public Read + Write	9 Oct 2021 at 14:4...	Jared	10	18 Sept 2021 at 22...	Silver Bracelet	3	Jewelry
Ldd2EBv0e2	3-99-carat-solita...	Diamond ring	Public Read + Write	9 Oct 2021 at 14:4...	Sephora	360	18 Sept 2021 at 22...	Diamond Ring	0	Jewelry
2oLkHk5z3L	default-placeholder	Gold necklace	Public Read + Write	9 Oct 2021 at 14:4...	Sephora	260	16 Sept 2021 at 01...	Gold Necklace	5	Jewelry
I3u7H8693T	w999.jpg	ladies style gold ...	Public Read + Write	9 Oct 2021 at 14:4...	Sephora	300	15 Sept 2021 at 17...	22 Carat Gold Brac...	5	Jewelry

This table consists of 11 fields. We discussed the first four columns in the previous table explanation, so I will be discussing the ones I have created.

Each entry represents one item in the stock. Each item has its own unique distinction between every other one in the table. To achieve this, there are seven columns that consist of information for an item.

The first field is itemImage. This can be a custom uploaded image of an item available in stock, or a default placeholder image if the image does not exist currently or simply is not available.

The second one is itemName. This field is used for giving a name basis to the item. According to the client's needs, this can range from a simple name such as:

- necklace
- cup
- mug
- scissors

or something such as:

- silver studded necklace
- white World Cup coffee mug
- green safety scissors

The 3rd field is itemDescription. This can be a brief or long description of the item.

The 4th field is itemBrand. This simply gives a clear idea of the brand of the item.

The next field itemPrice is a Number field that can be in Float or Integer format.

itemQuantity gives a clear idea to the amount of that item there is in stock. This allows for checking if something is out of stock or other specifics. itemCategory allows sorting for every item in its respective category.

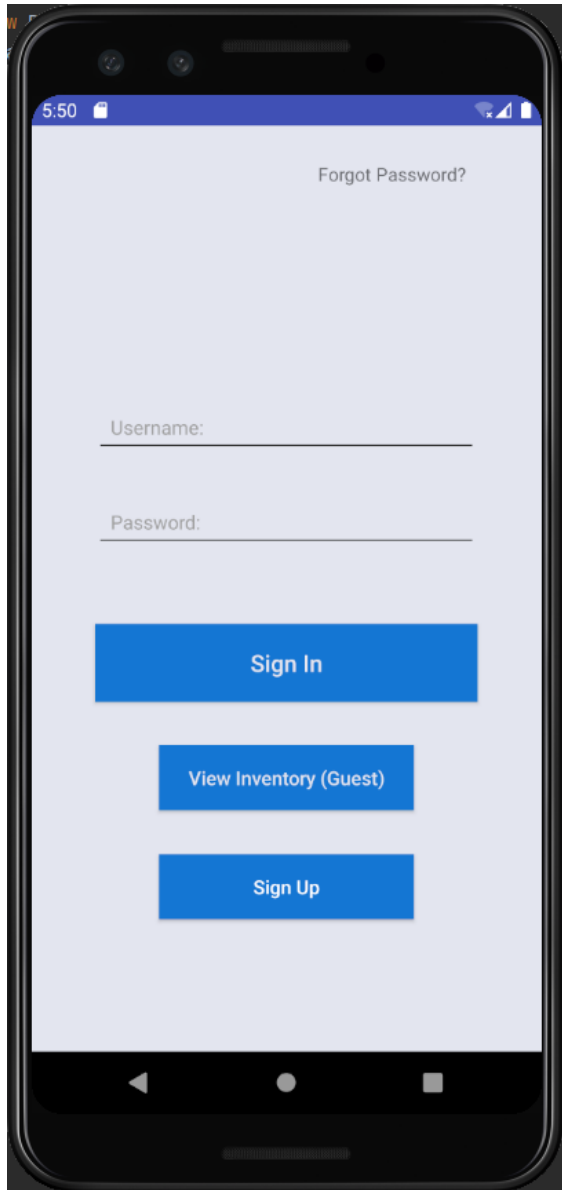
Chapter IV: App Design and Structure

i. Design and Flow:

In the previous sections we explained the back-end part of the project, and the objects associated with it. In this chapter we will explain the application itself. This application is supposed to be minimalistic and user friendly, so I have implemented the guest and user approach. By this, I mean the application will have a registered user UI, and an alternative guest UI (more will be explained later).

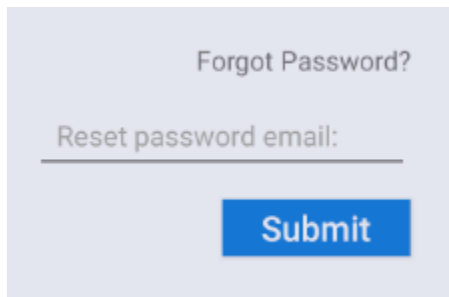
In figure 4.1, you can see the initial Activity of the application, called MainActivity. This is a generic Android format where the default activity that opens is the main activity. The android phone SDK used for these examples is the Pixel 3, API version 30.

Figure 4.1



As shown above, this is the login (Main) activity that appears when the app is launched from a new device. You have 3 buttons, sign in, view inventory as a guest, and sign up. There is also a hidden prompt when the “Forgot Password?” TextView has been pressed, as shown in figure 4.2:

Figure 4.2

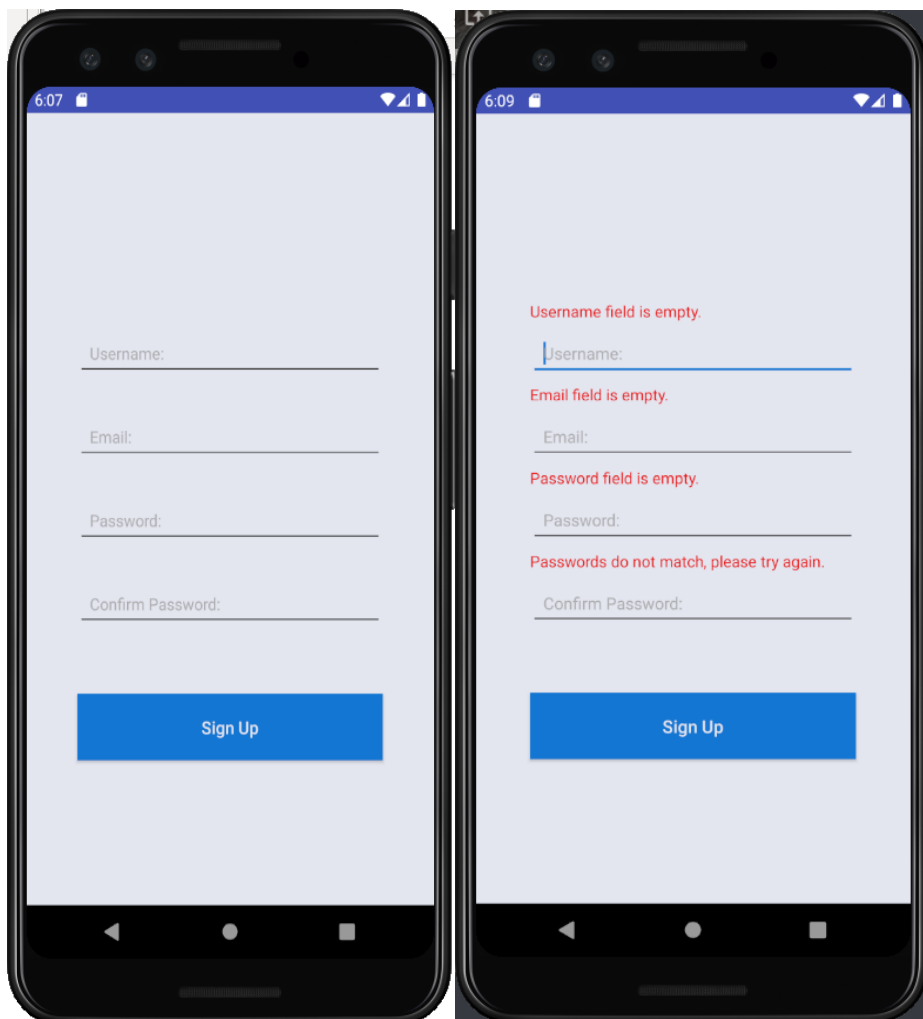


A screenshot of a mobile application interface for a 'Forgot Password?' screen. It features a light blue background. At the top, the text 'Forgot Password?' is centered. Below it is a text input field with the placeholder 'Reset password email:'. At the bottom, there is a blue rectangular button with the white text 'Submit'.

To be able to login, the user must create an account that will be saved in the User Table. This is important to be able to access useful features.

Pressing on “Sign Up” will open the RegisterActivity and its respective layout_register.xml, as shown in figure 4.3.

Figure 4.3



Two side-by-side screenshots of a mobile application's RegisterActivity. Both screens show a registration form with four input fields: 'Username:', 'Email:', 'Password:', and 'Confirm Password:'. A blue 'Sign Up' button is at the bottom. The left screenshot shows the form in its initial state. The right screenshot shows the form after validation, with red error messages above each field: 'Username field is empty.', 'Email field is empty.', 'Password field is empty.', and 'Passwords do not match, please try again.'.

Using what we learned in the previous chapters, this means that MainActivity will call onPause() and enter a pause state, while RegisterActivity will go through the registration process at the user's own pace. When the user is done and registered successfully, a loading prompt will appear while the data is being saved to the database, then RegisterActivity will call onDestroy(), as we no longer need it in the memory. MainActivity() will go to onResume(), and from there a user can login (after they verify their email) and access the next Activity of the application.

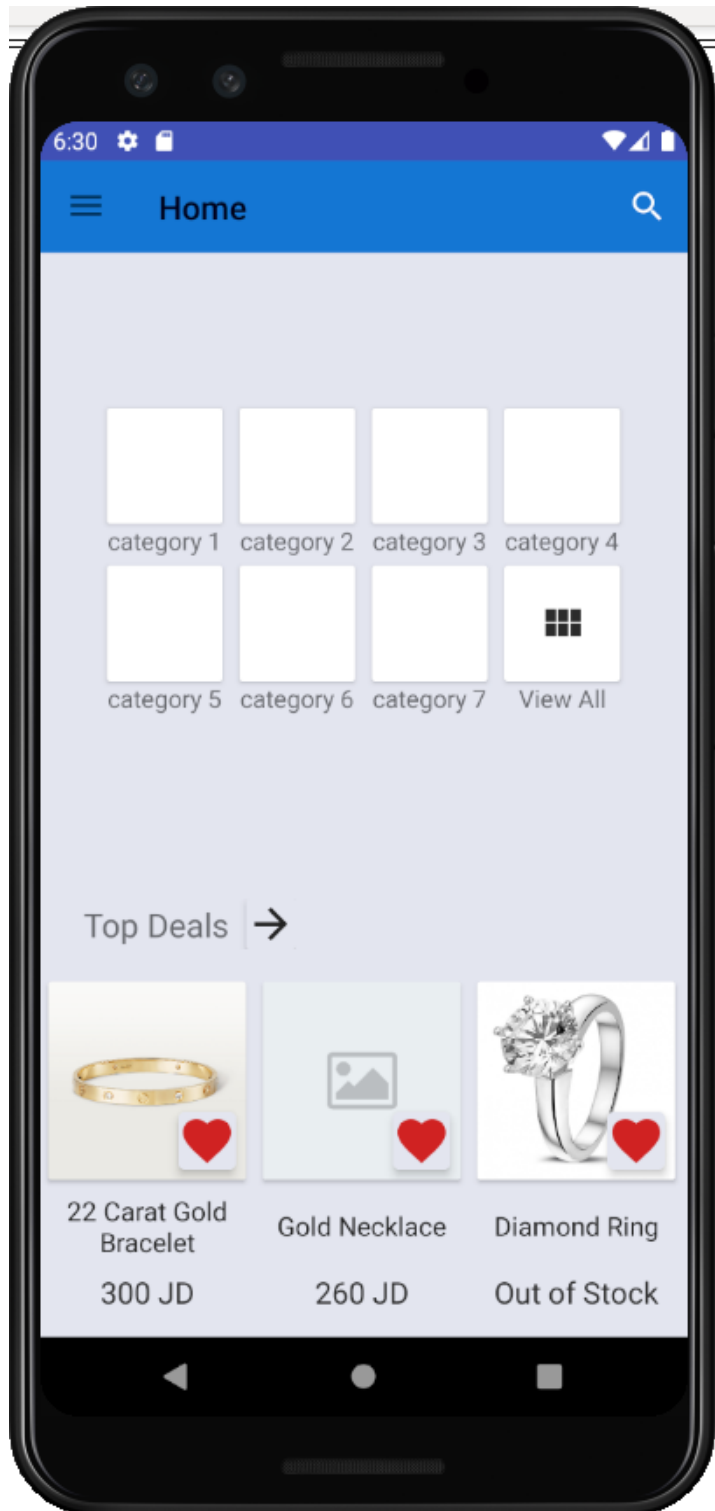
When either "View Inventory (Guest)", or "Sign In" are initiated successfully, the application will launch InventoryActivity and MainActivity will be destroyed and released from memory.

In InventoryActivity, there is a scrollable View called nestedScrollView that allows you to scroll indefinitely according to the amount and size of the views and data shown on the screen. At the top are the 7 category icons and names, and an additional view all button that can be changed and updated at the client's request. For the sake of the presentation, they will be named category 1-7, and will not have an icon.

Below the categories, there is a top deals button which contains a RecyclerView that can be changed and updated at the client's request to include current deals or offers specific to the store or market.

Figure 4.4 shows the screen you see when it is initiated.

Figure 4.4



Necessary to this project are fragments. Fragments are essentially a part of the interface that is connected to an Activity. A fragment will always be assigned to an Activity, and it cannot exist by itself without an accompanying Activity reference.

Before going any further, it's essential to understand what fragments are or mean. As we stated above, a fragment is a part of application's user interface that is bound to an activity. Fragments also have their logic and can thus, accept and handle different events. Fragments are beneficial since they allow code to be divided into smaller and more manageable chunks. {5}

On the toolbar at the top of the screen, the name of the current fragment will be shown. This is a segway into the NavigationUI component that I utilize in my project. In the InventoryActivity, we have 7 different fragments that all serve their own purpose. These are: homeFragment, profileFragment, settingsFragment, favoritesFragment, adminFragment, searchFragment, and itemFragment. In figure 4.5, shown is the code to initialize the navigationController and navigationUI components with the toolbar, drawerLayout, and navigationView component.

Figure 4.5

```
Toolbar mToolbar = findViewById(R.id.toolbar);
setSupportActionBar(mToolbar);
Objects.requireNonNull(getSupportActionBar()).setDisplayShowTitleEnabled(false);

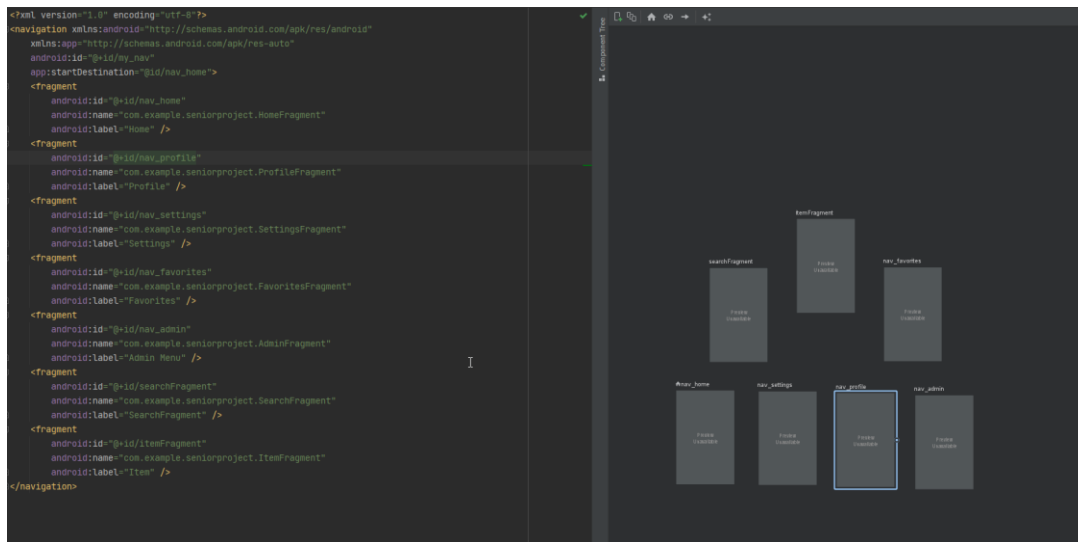
NavHostFragment navHostFragment = (NavHostFragment) getSupportFragmentManager().findFragmentById(R.id.navHostFragment);
assert navHostFragment != null;
navigationView = findViewById(R.id.nav_view);
navigationView.setItemIconTintList(null);
navController = navHostFragment.getNavController();
NavigationUI.setupWithNavController(navigationView, navController);
NavigationUI.setupActionBarWithNavController(activity: this, navController, drawerLayout);
NavigationUI.setupWithNavController(mToolbar, navController, drawerLayout);
navigationView.setNavigationItemSelectedListener(this);
```

As you can see, there is a main navHostFragment, which represents a container for all the fragments.

All of these are bound to the InventoryActivity, and they all can be viewed by replacing the navHostFragment container at any time by using their respective triggers and events.

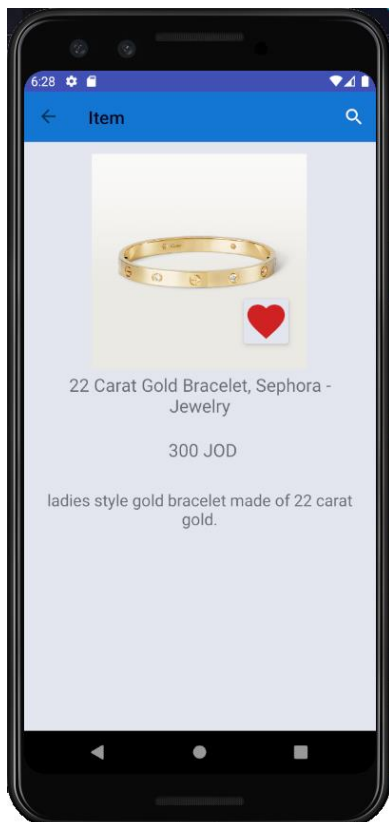
In figure 4.6, shown is the my_nav.xml that keeps track of the different navigation states we have. These are coded dynamically, so we can navigate to them by using the NavigationUI component.

Figure 4.6



SearchFragment, HomeFragment, and FavoritesFragment contain a `recyclerView` component. This represents Pressing on an individual item in the `recyclerView` List will open up an `itemFragment`, which will show the full information of that item, as shown in figure 4.7.

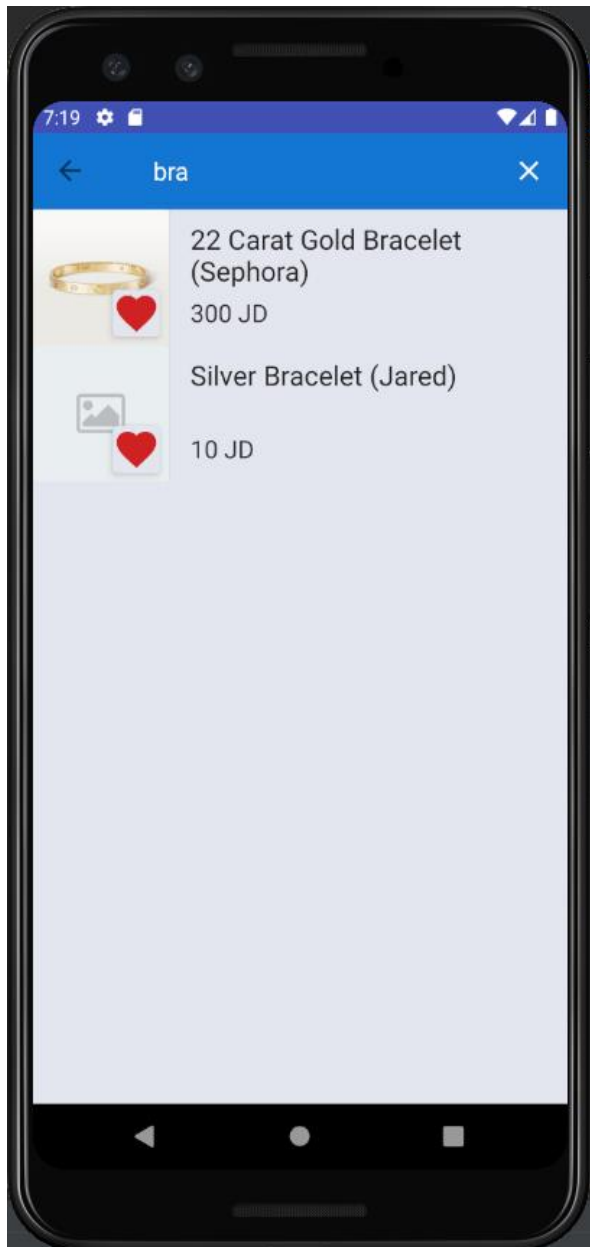
Figure 4.7



Also shown on the toolbar, is a search icon. This will call an event to open the SearchView associated with the icon, which will give the option to input text to send to the SearchFragment via an argument bundle.

In figure 4.8 is the search fragment interface:

Figure 4.8



Next is the navigation drawer, otherwise known as the side menu as shown in figure 4.9 (for users) and 4.10 (for guests). This will always be available by pressing the hamburger icon on the top left, and will open up a side menu consisting of home, profile, settings, admin menu (if you have the admin role), and register/login (if you are a guest or a user). All icons in the side menu are vector style graphics, available as XML file drawables.

Figure 4.9

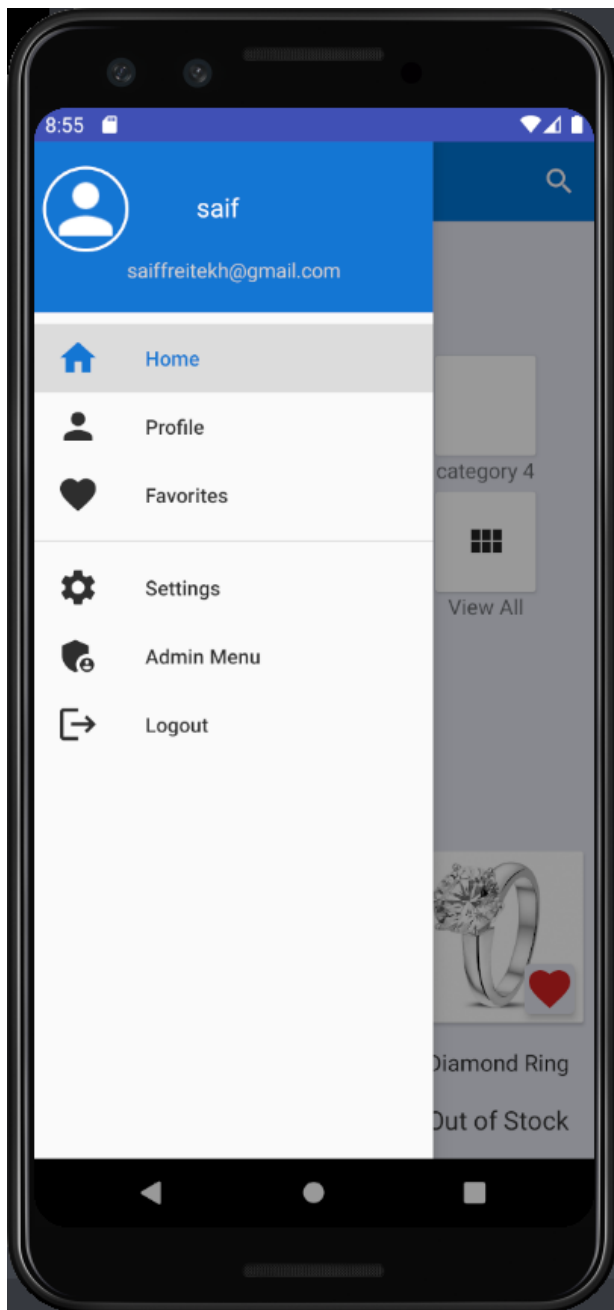
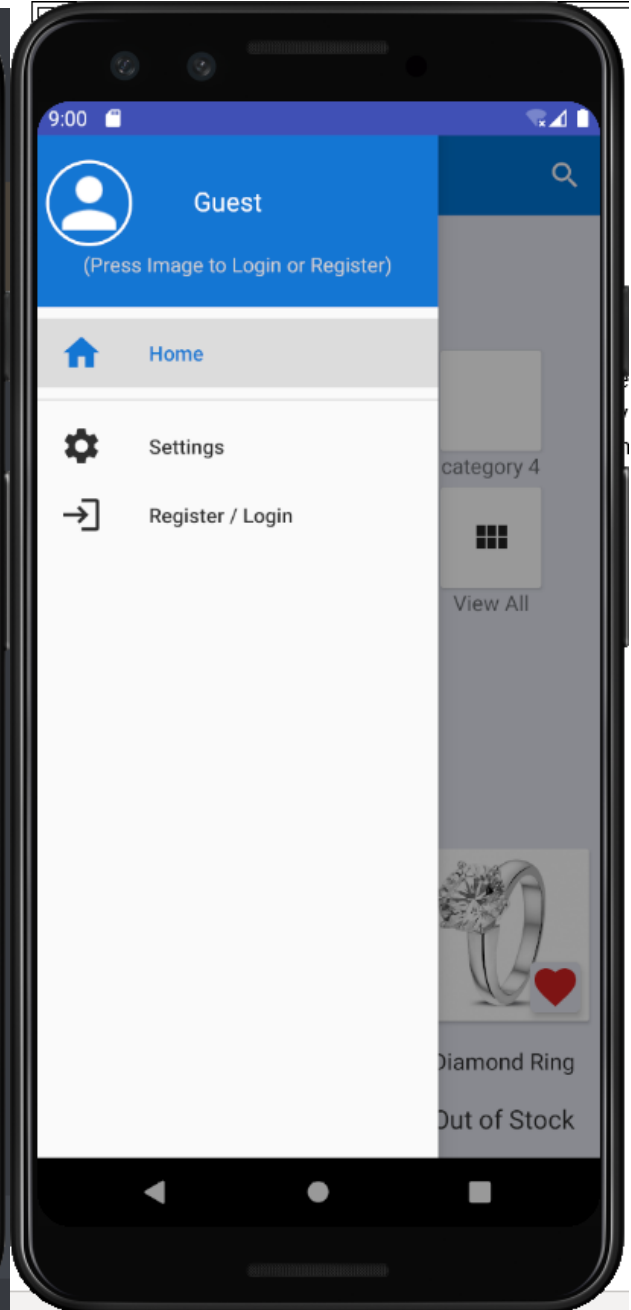


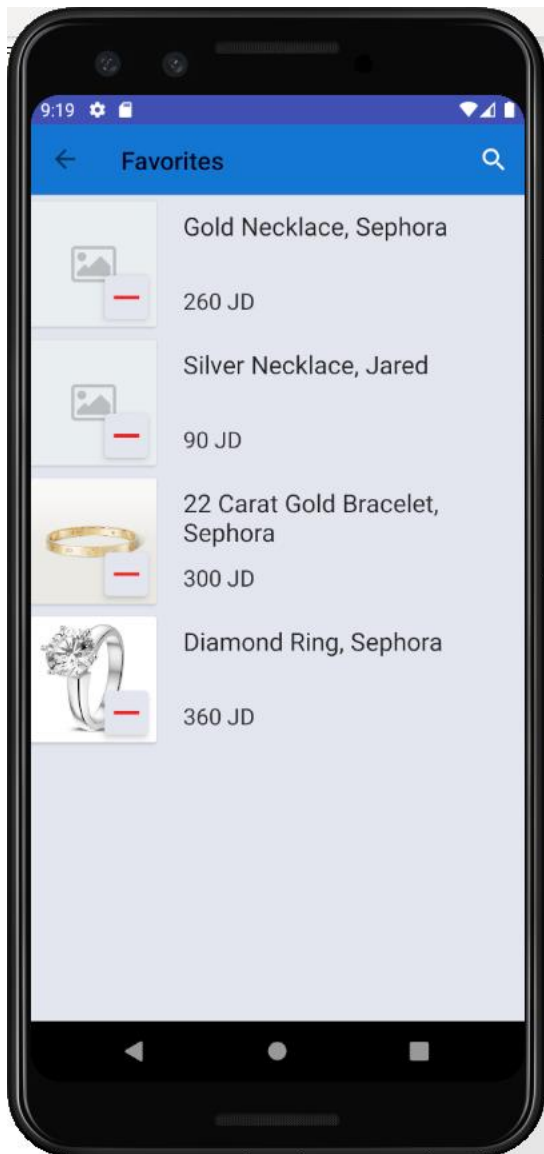
Figure 4.10



By pressing on the Home button, it will open the main fragment which is homeFragment as shown in figure 4.4 above. By pressing Profile, it will open the profileFragment which will allow you to either change your password or logout. If you are a guest, profile and favorites will not be shown (refer to figure 4.10).

By pressing on favorites, it will open up the user's personalized favorites menu, which has items added by pressing the heart icon shown on every item in the recyclerView. Each user has their own favorites list, saved in inventory_favorites which is a relation inside the _User table. Example of favorites is as shown on figure 4.11:

Figure 4.11



By pressing on Admin Menu (for admins with the admin role inside the Role Table) The app will open a link to Back4App's web admin interface. This will allow you to modify, update, and add items and content to the database, and has an easy to use interface, as shown in figures 4.12, and 4.13:

Figure 4.12

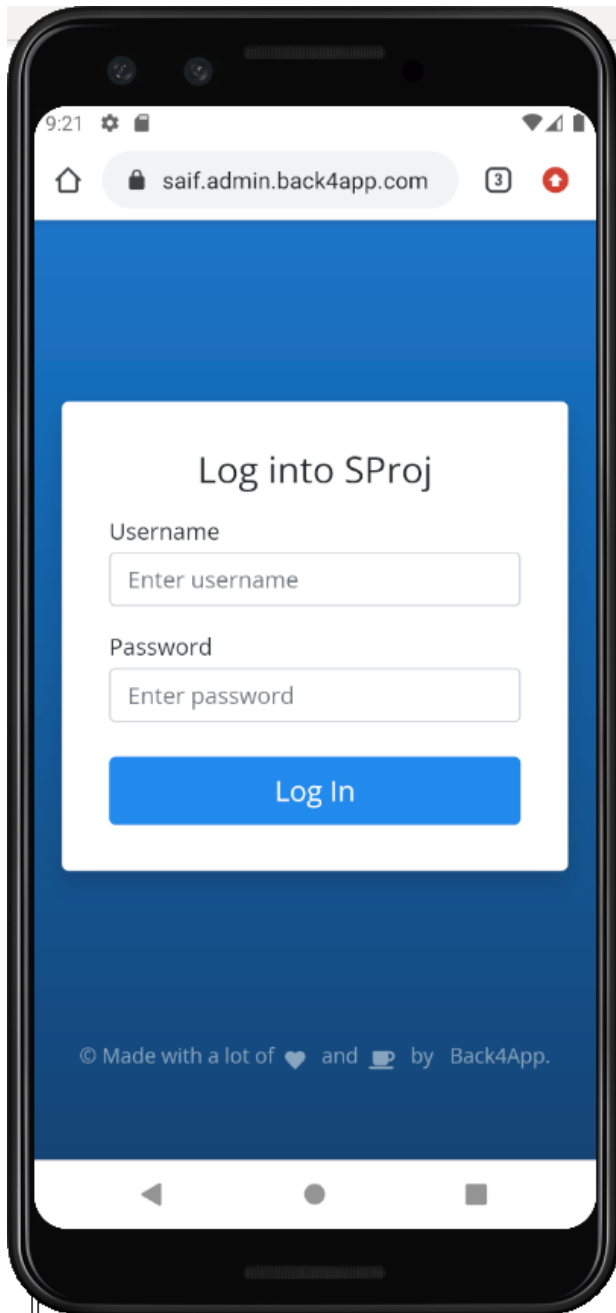
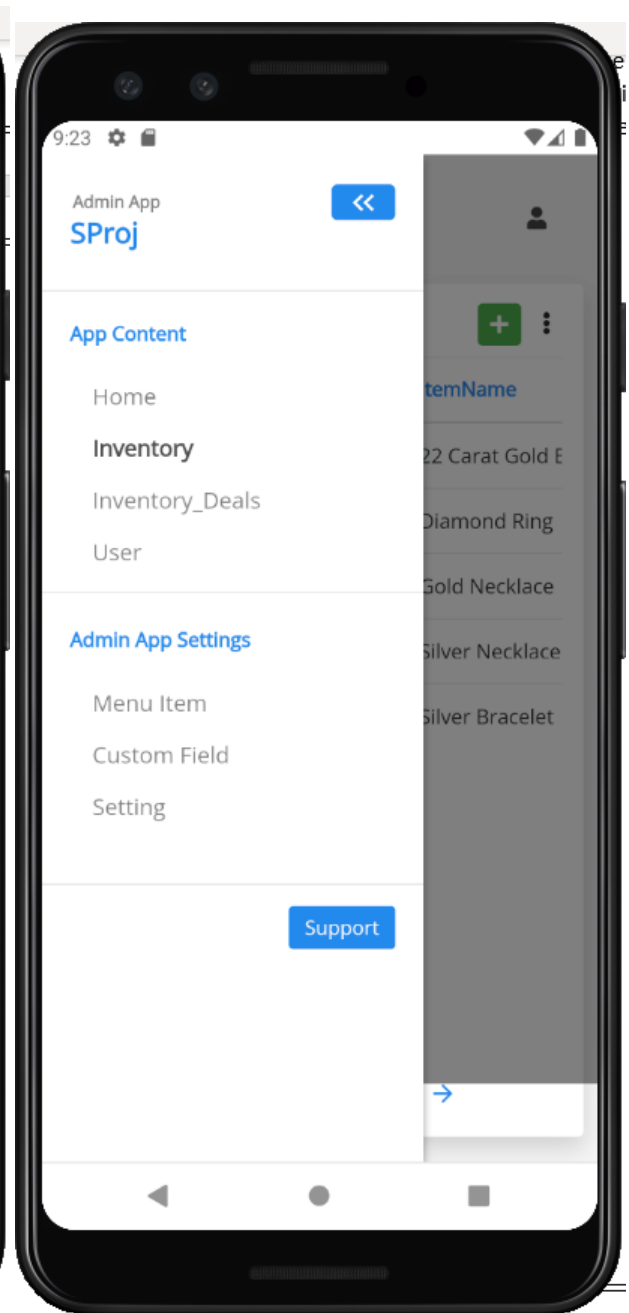


Figure 4.13



That concludes the design and flow of the application, source code is available on GitHub.

ii. UML Diagram:

Below in figure 4.14 is the UML diagram for the user interface flow, and in figure 4.15 is for the guest UI.

Figure 4.14 (users)

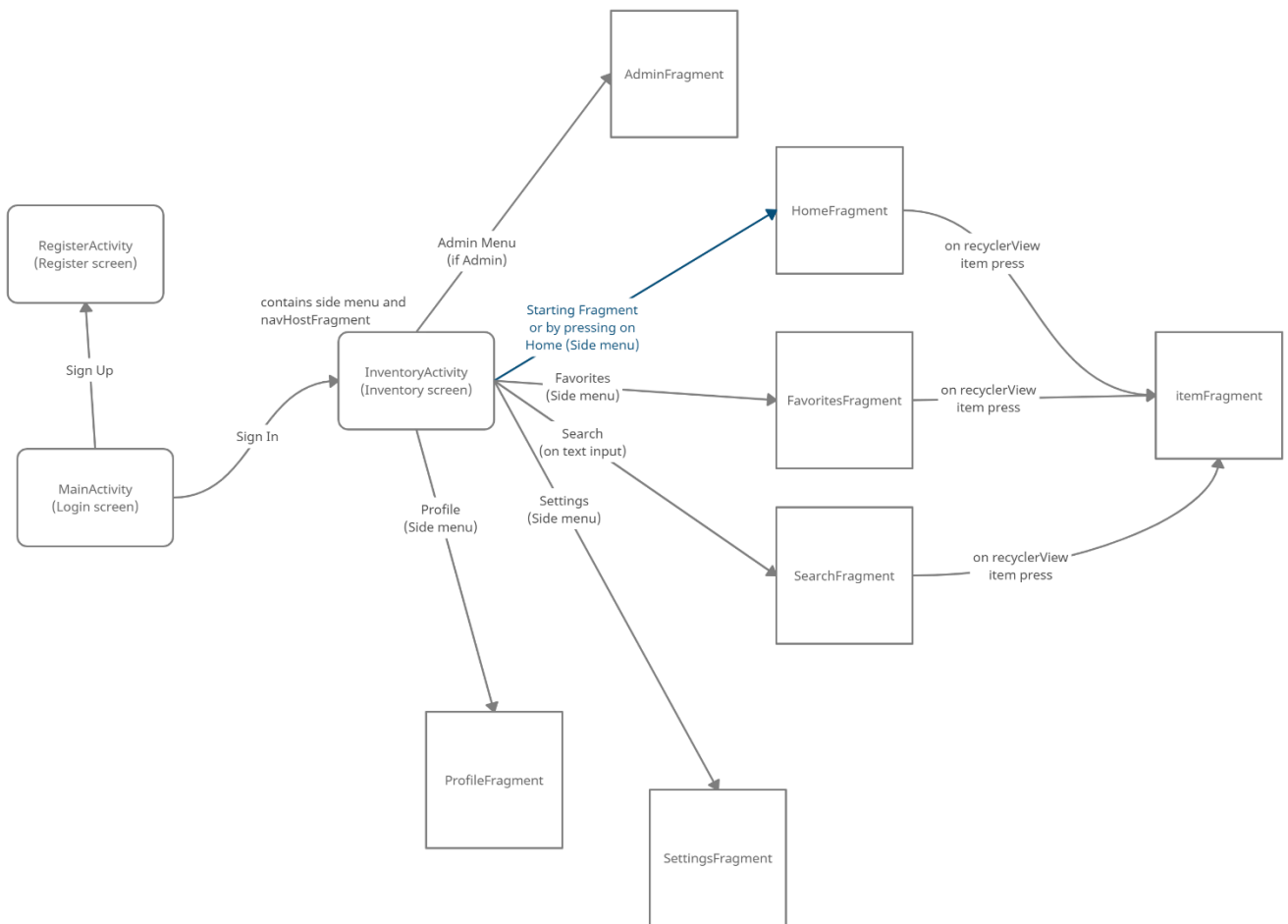
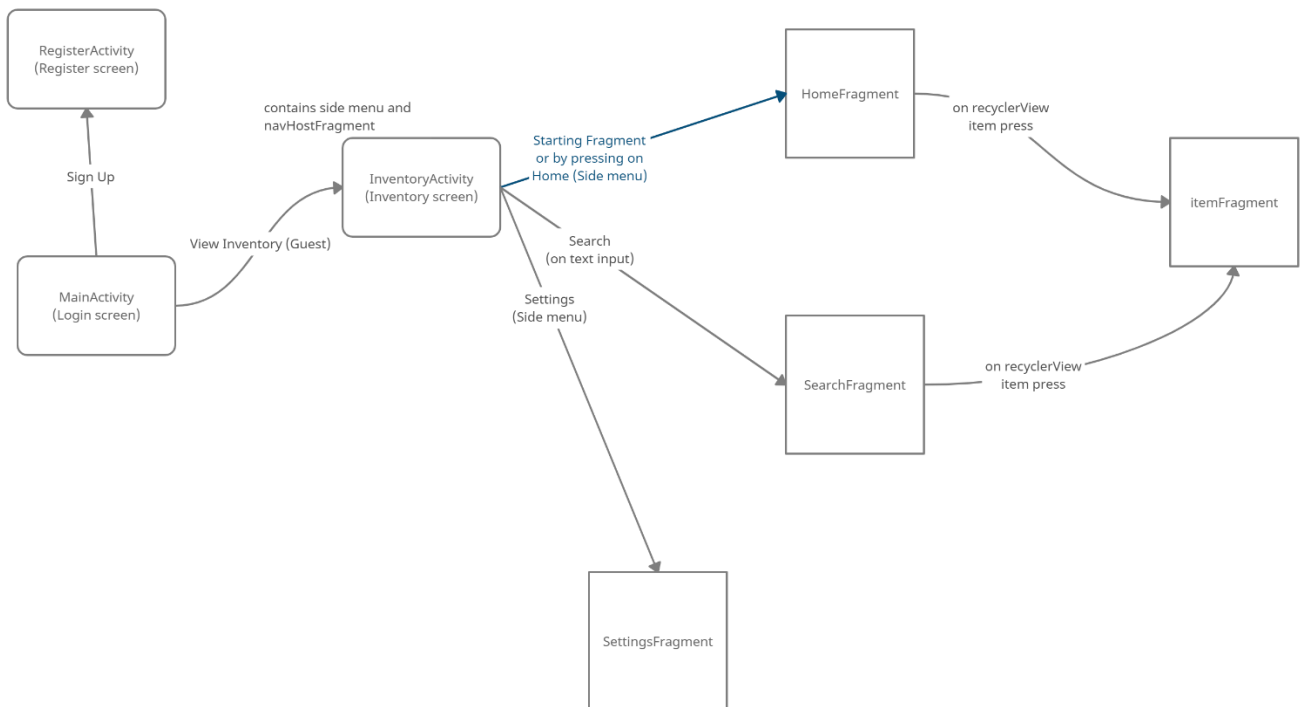


Figure 4.15 (guests)



Chapter V: Conclusion

i. Future Plans:

This concludes the project. All in all, I believe it was a satisfactory effort and my goals were mostly achieved (in respect to the one man challenge I faced and lack of experience in Android development). A challenge that I assumed would be faced is using real world information from a local business or store, so for the sake of the project and the limited Parse API network calls per month, I used a small amount of data I created to elaborate on some of the functionality. Further testing with large amounts of data should be conducted to fully confirm functionality. Of course, there are more features I wish to include, such as:

Data analysis and AI inclusion to the project using python and it's respective libraries. With this, I can create recommendation elements and a recommendation algorithm of sorts. Expanding on this, could lead to content personalized to each

user depending on their favorited items, and what category they prefer, etc. and can be viewed by a recyclerView list to be implemented in the the Home screen.

Another planned is using a barcode/QR code scanning system and fully implement such a system to be able to view, remove, and add items using a printed QR code on the item, and make it accessible to the database system and application. The application would have a QRCode scanner (in which I believe this can be done using APIs or external libraries for Java/Android Studio, but would require further testing and work) and would view data and information of an item using it's custom generated QR code that is only available to the local store. No progress has been made for this idea, even though it was mentioned in the first report that I wanted to include this, but unfortunately the goal was not reached, and I will have to work on this in the future.

I believe in regard to the efficiency of the source code, design, and the notion of keeping a simple and clean format, using a large amount of data would not impose an issue, but of course there will need to be further testing to confirm.

This application would help a local business or store in keeping track of their inventory, and allow users that wish to know more about the stock and items available in this store.

Source Code available on GitHub (references)

ii. References:

<https://www.androidauthority.com/introduction-to-xml-968598/> {1}

<http://tutorials.jenkov.com/android/activity.html> {2}

https://en.wikipedia.org/wiki/Online_database {3}

<https://www.back4app.com/docs/android/data-objects/android-data-types> {4}

<https://www.section.io/engineering-education/all-about-fragments-in-android-applications/> {5}

Additional links:

<https://docs.parseplatform.org/android/guide/#getting-started>

<https://github.com/zanger1234/Senior-Project.git>