E-Commerce Mobile Application

Final Year Project
B.Sc.(Hons) in Software Development

BY
Darren Regan

May 10, 2020

Advised by Martin Hynes
Department of Computer Science and Applied Physics
Galway-Mayo Institute of Technology (GMIT)

Contents

List of Figures

Chapter 1

Introduction

My project is a Native Android E-Commerce Application built in Android Studio using Kotlin programming languages. The general concept of my application is to create a e-commerce app that mirrors Amazon or Alibaba.
A specific goal of this project was to learn how native development differs from hybrid development(Xamarin, Ionic, React Native etc). Hybrid development was a topic that had multiple course projects, but i never got to experience building a project based on native development in Android or iOS.

A focus i made on my application is integrating different technologies such as Firebase, Picasso and Android Jetpack. Using these technologies made certain aspects of my application easier to implement. For example Picasso which is a image downloading and caching library can manage many aspects of image processing in an android environment such as handling ImageView recycling, complex image transformations with minimal memory use and automatic memory and disk caching which all have a heavy amount of coding involved to be implemented manually.

Firebase is another technology i use which has a great number of features and products such as Cloud Firestore which stores and syncs data between users and devices - at global scale - using a cloud-hosted, NoSQL database.
Authentication to manage users in a secure way by offering authentication through email, password and third-party providers like GitHub, Google, Facebook and Twitter.Realtime Database which is an efficient, low-latency solution for mobile apps that require synced states across clients in realtime.

Jetpack is a suite of libraries, tools, and guidance to help developers write high-quality apps more easily. These components help you follow best practices, free you from writing boilerplate code, and simplify complex tasks, so you can focus on the code you care about.

1.0.1 Objectives for project

Learn Android Development, Kotlin and Android Studio

The main objective of this project is to learn Android Development, Kotlin and Android Studio as i am interested in Android Development as a career and throughout my course i have only developed hybrid mobile applications.

Some Objectives and Requirements for features

- Learn Android Development, Kotlin and Android Studio (requirement)
- Create Fast responsive application (requirement)
- Create a Scalable and Re-Usable application (requirement)
- Admin Panel for adding Products (objective)
- User can change profile details (objective)
- Login & Logout (objective)
- Register (objective)
- Display Products with Description, Price etc. (objective)
- Add Social Integration (Employee Features, Chat/Calendar etc.) (objective)
- Purchasing Cart etc. (objective)
- Add Statistics for Purchasing and Consumer Habits (objective)

1.0.2 Sections

Methodology After setting objectives for my project, i set out to make a plan for development using Kanban boards. I used an agile approach for my development setting specific goals for each week. Validation and Testing was done using Junit.

Technology Review In this section i explain the Technologies used in my project, technologies such as Firebase, Kotlin, Android Studio are explained in great detail. I will also review programming in android with kotlin compared to my experience through out my four years in college. Comparisons such as Native development vs Hybrid development, use cases for different programming languages, language features such as more use of lambda expressions and more Kotlin features that make it a great alternative to java for android development.

System Design In this section i provided an explanation of the overall system architecture using XML class, sequence and interaction diagrams as well as screenshots of UI components such as how each view is formed with ImageView, TextBox etc.

System Evaluation In this section i evaluate my project against the objectives i set out.

Conclusion
In my conclusion i evaluate my overall goals for my project and list outcomes of the project. I go over discoveries made, what I've learned and what i can improve on in the future

Github Repository `https://github.com/DarrenRegan/Final-Year-Project`

My GitHub Repository contains my Dissertation, APK file which are both available for download with a click of a button at the top of the README.

- The README contains a quick explanation of the project as well as an installation guide, Devices used in testing as well as resources used along with links to research material.
- The code for my project is located at Final-Year-Project/app/src/main/ Java folder contains the code for activities and models
- Res folder contains the code for UI elements of the project, main/res/layout contains XML for all activities

Chapter 2

Methodology

This chapter covers the various methodologies that were implemented in this project, this includes Research methodologies, Software development methodologies, project management, supervisor meetings, developments tools, testing and source control.

An overview of methodologies used in this project; **Software development methodologies** which includes Agile Development, Continuous Delivery, Test Driven Development, Feature Driven Development, Extreme Programming etc. **Project management** which includes GitHub Kanban board and supervisor meetings. **Development tools** Android studio, Firebase. **Source Control** GitHub, Overleaf, Badge/Shield

2.1 Project management

Project Management is a key element of any task to ensure that the project is laid out correctly and each component party is complete at a given date. Because this project is a solo project i used the GitHub Kanban board to first pick features that i may want to implement and then broke them down into separate cards that i could work on individually. This helped brake down tasks which helped in development.

2.1.1 Agile

This project used Agile project management methodologies. An iterative approach was taken. This meant that each week i had certain tasks laid out to be completed be that either a feature for the app or documentation. I also had short weekly/bi-weekly meetings with my supervisor where i would update him, and asks for advice on my dissertation, app, etc.

Agile Road-map

An agile road-map in agile development helps state the goals of the project from the outset. It helps break down the tasks which are then further broken down for sprints. This is usually a high level view of the project. It is used often in industry and it helps to understand a project as a whole and learn what other teams are contributing to the overall project. The roadmap also shows how the project can grow in the future.

Planning and Development

During the meeting and planning phase, the milestones & objectives for the project were identified and broken down into simpler, and more manageable

tasks. The tasks are were then grouped into sprints or iterations lasting one to three weeks depending on the task. These are the tasks which mostly completed before each meeting with my supervisor. With each meeting i had a task somewhat completed and i could then know the following task which i could talk with my supervisor about before hand. So each weekly meeting with the supervisor started a sprint and the aim was to complete most of that sprint before the next meeting. The plans for the project and the sprints are taken from the User stories i created on the Kanban project and Design document. Through this iteration i convert the plans into working code.

Design Phase

Throughout the agile development life cycle design is a step in every iteration/sprint. The design is gradually built on. The design is never defined at the beginning of the project. The gradual evolution of the design enables taking advantages of new technologies that come on stream as well as meeting any new requirements brought forward by clients. The user experience is key when design a project/application. Every design must be user friendly. The type of end user of the item must be consider when designing. This comes to play in android development often as there is many different tools in development to create different features. You need to test them to know whats best for the end product.

Testing

Testing was carried out continuously with Android Studio features. Not only those Android Studio have a huge amount of features the Emulator can test a massive array of devices. So testing for this application was a continuous process throughout every build.
In a real-world environment The user tests would be carried out when meeting with the client. The client using the application would then comment on any changes that should be or could be made.

2.2 System Integration

Producing this project as a solo person meant i used different Software Engineering techniques to development my app. The main approaches used are test driven development(TDD), feature driven development(FDD), and continuous delivery

2.2.1 Test Driven Development

Test Driven Development is based predominately on unit testing. However they are very different because test driven development is when the unit tests are written before the code has been created. This means the developers are using the unit tests as a gild and the tests alone break down the overall task needing

to be completed. The tests will initially fail and the developers write minimal code in order for the tests to pass. This process is continuously repeated in order for the application to pass each test. These tests are commonly automated, in Android Studio a huge amount of testing is automatically completed for you without any manual interference.

2.2.2 Feature Driven Development

Feature Driven Development (FDD) is an agile framework that, as its name suggests, organizes software development around making progress on features. Features in the FDD context, though, are not necessarily product features in the commonly understood sense. They are, rather, more akin to user stories in Scrum. In other words, "complete the login process" might be considered a feature in the Feature Driven Development (FDD) methodology.[15]

FDD planning

CP prepares iteration planning meeting by grouping the right amount of relevant features into a work package. Normally the team has a number of CPs, each of them conducts iterations independently. Each iteration does not have to include the whole 10 development team. Instead, the new iteration team is formed for each iteration. CP selects iteration team members based on availability. Each developer receives a subset of features; strong class ownership is encouraged. The iteration team goes through the list of features, which should be familiar from Process One. If needed, the team contacts the customer to clarify any obscure features. In the complex cases the team may create sequence diagrams. The planning meeting also sets timeline for iteration milestones.[16]

Lifecycle of FDD

1. **Develop an overall model** - FDD project starts with high-level walk through of the scope of the system and its context. Next, detailed domain models are created for each modelling area by small groups and presented for peer review.

2. **Build a features list** - Knowledge gathered during the initial modeling is used to identify a list of features by functionally decomposing the domain into subject areas. Subject areas each contain business activities, and the steps within each business activity form the basis for a categorized feature list.

3. **Plan by feature** - After the feature list is completed, the next step is to produce the development plan and assign ownership of features (or feature sets) as classes to programmers.

4. **Design by feature** - A design package is produced for each feature. A chief programmer selects a small group of features that are to be developed within two weeks. Together with the corresponding class owners, the chief programmer works out detailed sequence diagrams for each feature and refines the overall model. Next, the class and method prologues are written and finally a design inspection is held.

5. **Build by feature** - After a successful design inspection for each activity to produce a feature is planned, the class owners develop code for their classes. After unit testing and successful code inspection, the completed feature is promoted to the main build.

Coding in FDD

Coding process in FDD is not as exciting and challenging as it is in XP. This happens because by the coding time the features have been extensively discussed during Process One, iteration kick-off meeting, design review meeting. Classes and methods are 13 defined by now, their purpose is described in code documentation. Coding often becomes a mechanical process. Unlike XP FDD strongly discourages refactoring. The main argument against refactoring here is that it takes time and does not bring any value to the customer. The quality of code is addressed during code review meetings. FDD encourages strong code ownership. The main idea is that every developer knows the owned code and better realizes the consequence of changes.[16] FDD fights the problem of leaving team members from the different angle:

- Sufficient code documentation simplifies understanding somebody else's code
- Developers know what other people's code does, since they reviewed the design
- Developers will look at each other's code during code review

2.2.3 Continuous Delivery

Continuous delivery is an extension of continuous integration to make sure that you can release new changes to your customers quickly in a sustainable way. This means that on top of having automated your testing, you also have automated your release process and you can deploy your application at any point of time by clicking on a button. In theory, with continuous delivery, you can decide to release daily, weekly, fortnightly, or whatever suits your business requirements. However, if you truly want to get the benefits of continuous delivery, you should deploy to production as early as possible to make sure that you release small batches that are easy to troubleshoot in case of a problem. [13]

Continuous Delivery in an Android Environment

In the context of mobile development, Continuous Delivery is the ability to produce valid builds that are ready to be published in an app store. Continuous Deployment is the final status of the process, which ensures that any of the changes applied to the code is automatically deployed to the production environment.

Continuous Delivery for mobile development can be tricky. For Android app development requires a key to sign off the production build before it can move to the Google Play Store. Keeping keys and passwords on a Version Control Software (VCS) is a bad practise, because VCS is available to any member with access to the project, which includes any temporary 3rd parties. It's common practise therefore to create release build on a local computer, but this erodes one of the great advantages to Continuous delivery: having a scalable online platform and the ability to easily deploy any stages of the Continuous Integration at any time.[2]

Because of this you need to secure the signing keys of the app, building a Continuous Delivery around a mobile development environment can be tricky. For Android app development, you can improve this by implementing Google Play App Signing, which applies a second key when your app is uploaded to the Google Play Store. But even after employing this method, there will still be at least a key and a couple passwords required to build a release (signing key, key and upload password). To get past this you can use a Encrypted Cloud Storage service for this problem. They can keep the keys safe and are only limited by the integration capabilities of your chosen Continuous Delivery system.

Implementing Continuous Delivery

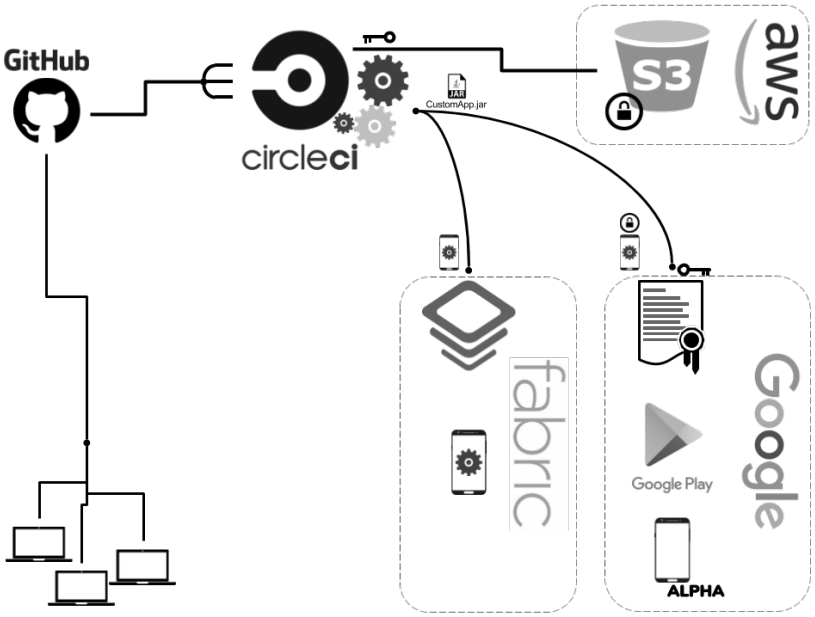
Implementing Continuous Delivery, it requires you pick the right tools for the job. So choosing your Continuous Integration system is important. For this project because i was solo i could just use my local computer for deployment and tool integration inside Android Studio is so good, i didn't need to implement these tools, but in a real-world environment working with a team this implementation is mandatory

Some examples of Continuous integration systems:

- GitLab CI
- CircleCI
- CodeShip
- Github Actions
- AWS Pipeline

Using these systems with a Version Control System such as GitHub is an easy feat to accomplish and results in a final system as shown below

Figure 2.1: Continuous Delivery Setup for an Android App [2]



2.3 Development tools

In order to manage this project i used different development tools and Source Control tools. This was very useful as it enables working on different branches for features that i want to implement and leaves the master branch as a working state of the project. This could be used for industry standard as continuous delivery. For this project it's meant that after each sprint there was a working version to present to the customer. Although the working version does not mean it doesn't need changes in next sprints. I was solo this project so i did not take advantage of branches as much as a group would, as each member would have there own branch to work on.

2.3.1 Android Studio

Android Studio is the official integrated development environment (IDE) for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development. It is available for download on Windows, macOS and Linux based operating systems.It is a replacement for the Eclipse Android Development Tools (ADT) as the primary IDE for native Android application development.
The integration between Android Studio and Firebase made Android Studio the most superior development platform to use.

2.3.2 Firebase

Firebase is Google's mobile and web application development platform that helps you build, improve, and grow your application. Firebase frees developers to focus on crafting excellent user experiences. You don't need to manage servers. You don't need to write APIs. Firebase is your server, your API and your database, everything is written generically so that you can modify every-thing to suit most needs.

2.4 Source Control

2.4.1 GitHub

Github provides hosting for software development version control using Git. It offers the distributed version control and source code management (SCM) functionality of Git, plus its own features. It provides access control and several collaboration features such as bug tracking, feature requests, task management, and wikis for every project. GitHub offers plans free of charge, and professional and enterprise accounts. The Github kanban board was a feature i used in this project to track sprints.

2.4.2 Overleaf

Overleaf is an open-source online real-time collaborate LaTeX editor. It is hosted at http://www.overleaf.com, but can also run on your own local version, and contribute to the development of Overleaf.

2.4.3 Badge/Shield

Badge/Shield.io is a service for concise, consistent, and legible badges in SVG and raster format, which can easily be included in GitHub READMEs or any other web page. The service supports dozens of continuous integration services, package registries, distributions, app stores, social networks, code coverage services, and code analysis services. Every month it serves over 470 million images. [10] This is used for easy access to a copy of the APK and Dissertation at the top of my README, you can further customize these badges to show Beta versions, stable versions, downloads, ratings, code coverage etc.

Chapter 3

Technology Review

This chapter discusses the different technologies used in throughout the project. It discusses the the advantages and disadvantages of each technology and why certain technologies were used over others. It also discusses hybrid applications compared to native applications, advantages, disadvantages, uses at different business structures and other topics.

3.1 Overview

This project is a Native android app built with Kotlin in Android Studio, Fire-base is used for the database, statistics, verification etc.
Topics:

- Kotlin/Java comparison
- Fire-base
- Picasso
- Paper
- Android Studio/IntelliJ
- Native Applications
- Hybrid Applications
- Hybrid vs Native comparison

3.2 Main Technologies

This section will discuss the main technologies currently in use in the android application.

3.2.1 Kotlin

Kotlin is a cross-platform, statically typed, general-purpose programming language with type inference. Kotlin is designed to inter-operate fully with Java, and the JVM version of its standard library depends on the Java Class Library, but type inference allows its syntax to be more concise. Kotlin mainly targets the JVM, but also compiles to JavaScript or native code (via LLVM). Language development costs are borne by JetBrains, while the Kotlin Foundation protects the Kotlin trademark.
Kotlin is the preferred language for Android app developers as of May 2019, since the release of Android Studio 3.0 in October 2017, Kotlin has been included as an alternative to the standard Java compiler. The Android Kotlin compiler targets Java 6 by default, and lets programmers choose between Java 8 to 14 for optimization purposes.
Kotlin originated at JetBrains, which is the company behind IntelliJ IDEA. Kotlin has been open source since 2012 and has a large team of full-time developers working on it, there is also the Kotlin project of GitHub which has more than 370 contributors.

Advantages

Kotlin has many advantages, many are quite serious improvements in readability and workflow which was noticeable when creating my project

- **Less code combined with greater readability** - Spend less time writing code and working to understand the code of others.
- **Mature language and environment** - Kotlin has developed continuously over the years not only as a language but as a whole ecosystem with very robust tooling. Its seamless integration with Android Studio, makes it actively used by companies to develop Android applications.
- **Kotlin support of Android Jetpack and other libraries** - KTX extensions adds kotlin language features, such as coroutines, extension functions, lambdas, and named parameters, to existing Android libraries.
- **Interoperability with Java** - You can use Kotlin along with the Java programming language in your applications without needing to migrate all your code to Kotlin.
- **Support for multi-platform development** - You can use Kotlin for developing not only Android but also iOS, back-end, and web applications by sharing the common code among the platforms.
- **Code safety** - Less code and better readability lead to fewer errors. The Kotlin compiler detects the remaining errors, making the code safe.
- **Easy to Learn** - Kotlin is very easy to learn, especially for any Java experienced developers.
- **Large community** - Kotlin a great support and many contributions from the community, which is growing all over the world. According to Google, over 60% of the top 100 apps on the Google Play Store use Kotlin. Many startups and Fortune 500 companies have already developed Android applications using Kotlin and more and more companies are prioritizing Kotlin Native application development over other options due to the robust toolkit and optimizations that make your applications the best that they can be.

Disadvantages

- **Shift from Java to Kotlin** - Kotlin is an amazing programming language and there is a reason why leading lead companies have started using kotlin, but at their core their two different languages. Developers won't be able to quickly shift from one to another without taking time to learn Kotlin. Therefore company's have to consider different approaches to Android app development as additional expenses are required on training a team of developers.

- **Hard to find experienced developers** - There is a high demand for specialists in Kotlin as Google made it the preferred language for Android development in 2019, but there is still a very large amount of Java programmers on the market compared to Kotlin developers. This means on average the Kotlin developers may be younger meaning less senior developers available for hire. This is quite a large disadvantage, but will quickly fade away as many leading tech companies have switched which creates a ripple effect down the chain of companies.

- **Limited learning resources** - Although the number of Android app developers who use Kotlin instead of Java increase everyday, there is still a limited number of resources in the market compared to Java. Many College courses will teach Java over Kotlin as both are so similar, meaning most Kotlin developers come from a background in Java and learn to code in Kotlin themselves.

Kotlin Syntax

Kotlin syntax is familiar to any programmer that is from a OOP domain and be be more or less understood from the get-go. There are differences from Java such as primary and secondary constructors, val & var variable declarations and more.

Below you can see the basic structure of a class in kotlin

```
class Foo {

    val b: String = "b"      // val means unmodifiable
    var i: Int = 0           // var means modifiable

    fun hello() {
        val str = "Hello"
        print("$str World")
    }

    fun sum(x: Int, y: Int): Int {
        return x + y
    }

    fun maxOf(a: Float, b: Float) = if (a > b) a else b

}
```

String Interpolation - A smarter and more readable version of Java's String.format() that is built into the language

```
val x = 5
val y = 10
print("sum of $x and $y is ${x + y}")  // sum of 5 and 10 is 15
```

Type Inference - Kotlin will infer your types wherever you feel it will improve readability

```
val a = "abc"                          // type inferred to String
val b = 4                              // type inferred to Int

val c: Double = 0.7                    // type declared explicitly
val d: List<String> = ArrayList()      // type declared explicitly
```

Smart Casts - The Kotlin compiler tracks logic and auto-casts types if possible, which means you do not need to use instanceof checks followed by explicit casts

```
if (obj is String) {
    print(obj.toUpperCase())     // obj is now known to be a String
}
```

When Expression - The switch case is replaced with the more readable and flexible when() expression

```
when (x) {
    1 -> print("x is 1")
    2 -> print("x is 2")
    3, 4 -> print("x is 3 or 4")
    in 5..10 -> print("x is 5, 6, 7, 8, 9, or 10")
    else -> print("x is out of range")
}

// It also works as an expression or a statement
// with or without an argument
val res: Boolean = when {
    obj == null -> false
    obj is String -> true
    else -> throw IllegalStateException()
}
```

Setter & Getter behavior - You can make custom set & get behaviors that are added to public fields, which means getter & setters won't bloat your code

```
class Frame {
    var width: Int = 800
    var height: Int = 600

    val pixels: Int
        get() = width * height
}
```

Data Classes - We frequently create classes whose main purpose is to hold data. In such a class some standard functionality and utility functions are often mechanically derivable from the data. In kotlin, this is called a data class and is marked as data. Its a Plain Old Java Object so its complete with toString(), equals(), hashCode() and copy().

```
data class Person(val name: String,
                  var email: String,
                  var age: Int)

val john = Person("John", "john@gmail.com", 112)
}
```

3.2.2 Kotlin vs. Java in Context of Android Development

3.2.3 Why would a Java Developer switch to Kotlin?

Android development has always been tagged with Java Development. But Kotlin has outpaced it in performance, simplicity, maintainability, robustness, and lightweight code. Developers and users lure for feature-rich, effective and cost-efficient mobile apps that help increase the efficiency of processes, increasing competencies, meet customers' expectations, help maintain business' reputation and remove uncertainty. This is why Google has made Kotlin as the preferred language for Android app developers.

Although Kotlin seems to work back to back with Java for front-end development, it is already being utilized in backend projects like Spring 5. It can compile to almost every platform including Android, JVM, Native and JavaScript and can extract one common code-base that will target all of them at the same time. Its scripting capabilities make it easy to handle all Gradle build scripts. Its built-in null safety support makes it win over Android that is full of old Java-style API's. Being compatible with all Java libraries and frameworks, the JVM, Gradle integration and Maven build systems, it allows Android developers to write new modules and write them alongside existing Java code. Plus it supports modern programming concepts like higher-order functions, extension functions, delegates which help build clean API's.

3.2.4 Why Java isn't preferred for Android Development

- **Support a subset of Java 8 Features** - Android Developers are not able to reap the full benefits of Java. To make full use of Java you must use Java 7. [6]
- **Issues addressed in Kotlin** - Java comes with some well-documented language issues like endless try-catch blocks, null-unsafety, NullPointerExceptions and lack of extendability that are addressed with kotlin.
- **Syntax of Java** - Syntax of Java is very complex and writing longer code leans to more errors, bugs and is much harder to read.

3.2.5 Switch from Java to Kotlin

- **Kotlin and Java can co-exist** - Kotlin is simultaneously operable with Java. You can have both Java and Kotlin code co-exist within a single project. Both codes can compile perfectly and users are not even able to make out which part of the code is Kotlin and which part is Java.
- **Kotlin and Java classes can be written simultaneously** - There is no requirement to convert the entire project to Kotlin or starting a new project. Alternatively, small chunks of codes can be written in Kotlin and can be inserted as and when required, without affecting the entire code.
- **Kotlin is an enhancement of Java** - a java developer or an existing java application user will be easily able to comprehend what Kotlin code is doing. It looks familiar, the code is easy to read and understand.
- **Android Studio support for Kotlin** - Android studio has an excellent support for Kotlin. After you install Kotlin plugin, Android Studio makes it easy to configure Kotlin plugin in your project. Once this is done, the IDE will automatically understand the underlying functionality like Kotlin code compilation, and execution. Debugging, code navigation, unit testing, auto-completion and full refactoring support. Once all of this is set, converting an entire Java code into Kotlin can be accomplished in a single click.

3.2.6 Firebase

Firebase is Google's mobile and web application development platform that helps you build, improve, and grow your application. Firebase frees developers to focus on crafting excellent user experiences. You don't need to manage servers. You don't need to write APIs. Firebase is your server, your API and your database, everything is written generically so that you can modify everything to suit most needs. Firebase has a huge amount of features, real time databases, cloud storage, hosting, machine learning, authentication, statistics, analytics and more.

Firebase products are setup in three different area's.

1. **Build better apps**
2. **Improve app quality**
3. **Grow your business**

Build better apps

Firebase lets you build more powerful, secure and scalable apps, using world-class infrastructure. There are seven different products focused on building a better app. My project takes advantage of Authenication, Realtime Database and Cloud storage. Almost all of Firebase products are extremely useful and are worth mentioning as they could be implemented into the project at some point. I will first summarize the products, and then go into further detail on the specific products i used in my project. How the code is implemented, why i used it etc.

Products for building better apps

- **Cloud Firestore** - Store and sync data between users and devices - at global scale - using a cloud-hosted, NOSQL database. Cloud Firestore gives you live synchronization and offline support along with efficient data queries. Its integration with other Firebase products enables you to build truly serverless apps.

- **ML Kit** - Bring powerful machine learning features to your mobile app whether you're new or experienced in ML. Get started easily by using our ready-to-use APIs for common mobile use cases, or import your own custom models which can be hosted and served to your apps by Firebase. ML Kit APIs can run on-device or in the cloud, depending on the functionality, and some give you both choices.

- **Cloud Functions** - Extend your app with custom backend code without needing to manage and scale your own servers. Functions can be triggered by events, which are emitted by Firebase products, Google Cloud services, or third parties, using webhook

- **Authentication** - Manage your users in a simple and secure way. Firebase Auth offers multiple methods to authenticate, including email and password, third-party providers like Google or Facebook, and using your existing account system directly. Build your own interface, or take advantage of our open source, fully customizable UI.

- **Hosting** - Simplify your web hosting with tools made specifically for modern web apps. When you upload your web assets, we automatically push them out to our global CDN and give them a free SSL certificate so your users get a secure, reliable, low-latency experience, no matter where they are.

- **Cloud Storage** - Store and share user-generated content like images, audio, and video with powerful, simple, and cost-effective object storage built for Google scale. The Firebase SDKs for Cloud Storage add Google security to file uploads and downloads for your Firebase apps, regardless of network quality.

- **Realtime Database** - Realtime Database is Firebase's original database. It's an efficient, low-latency solution for mobile apps that require synced states across clients in realtime. We recommend Cloud Firestore instead of Realtime Database for most developers starting a new project.

Improve app quality

Firebase gives you insights into app performance and stability, so you can channel your resources effectively. These products weren't used in the project, but are worth mentioning as they are quite valuable in a commercial development environment where app performance and crashing have a huge impact on user engagement and app performance on the Google Play Store.

Products for improving app quality

- **Crashlytics** - Reduce your troubleshooting time by turning an avalanche of crashes into a manageable list of issues. Get clear, actionable insight into which issues to tackle first by seeing the user impact right in the Crashlytics dashboard. Realtime alerts will help you stay on top of stability even on the go. Crashlytics is the primary crash reporter for Firebase.
- **Performance Monitoring** - Diagnose app performance issues occurring on your users' devices. Use traces to monitor the performance of specific parts of your app and see a summarized view in the Firebase console. Stay on top of your app's start-up time and monitor HTTP requests without writing any code.
- **Test Lab** - Run automatic and customized tests for your app on virtual and physical devices hosted by Google. Use Firebase Test Lab throughout your development lifecycle to discover bugs and inconsistencies so that you can offer up a great experience on a wide variety of devices.
- **App Distribution** - Firebase App Distribution allows developers to send pre-release versions of their app to trusted testers from the console or using command line tools, as well as manage testers in one place.

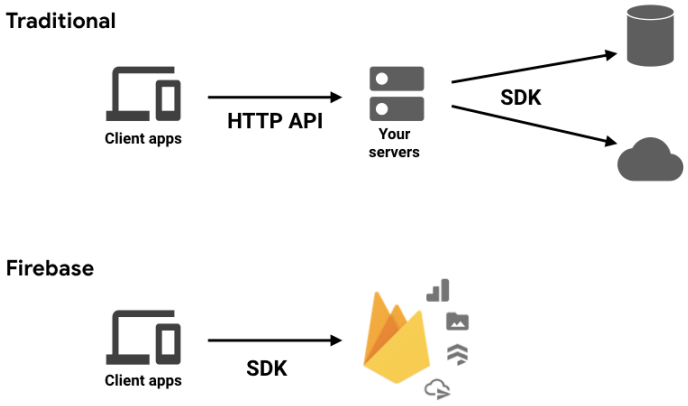
Grow your business

Firebase helps you grow to millions of users, simplifying user engagement and retention.

Products for Growing your business

- **In-App Messaging** - Engage and nurture your active users with targeted and contextual messages that encourage them to complete meaningful actions within your app. You have the power to trigger messages based on user behavior and interests. You can also customize the design of in-app messages to fit your brand. In-App Messaging supports a variety of use cases and formats.
- **Google Analytics** - Analyze user attributions and behavior in a single dashboard to make informed decisions on your product roadmap. Gain realtime insights from reports, or export your raw event data to Google BigQuery for custom analysis.
- **Predictions** - Harness the power of Google's machine learning to get insight into which segments of users are likely to churn or spend (or complete another conversion event). Use these smart predictive segments for targeting in other products like Remote Config, Cloud Messaging, and In-App Messaging.
- **A/B Testing** - Improve your app by running product and marketing experiments, without worrying about setting up the infrastructure to run A/B tests. Customize experiments to suit your goals. Test a variety of updates to your app, like message copy or new features. Then, only roll-out changes proven to move the needle on your key metrics.
- **Cloud Messaging** - Send messages and notifications to users across platforms—Android, iOS, and the web—for free. Messages can be sent to single devices, groups of devices, or specific topics or user segments. Firebase Cloud Messaging (FCM) scales to even the largest apps, delivering hundreds of billions of messages per day.
- **Remote Config** - Customize how your app renders for each user. Change the look and feel, roll out features gradually, run A/B tests, deliver customized content to certain users, or make other updates without deploying a new version—all from the Firebase console. Monitor the impact of your changes and make adjustments in a matter of minutes.
- **Dynamic Links** - Use Dynamic Links to deliver a customized user experience for iOS, Android, and the web. You can use them to power mobile web to drive native app conversions, user to user sharing, social and marketing campaigns, and more. Dynamic Links provides you with the attributions you need to better understand your mobile growth.

Traditional Architecture vs. Serverless Architecture

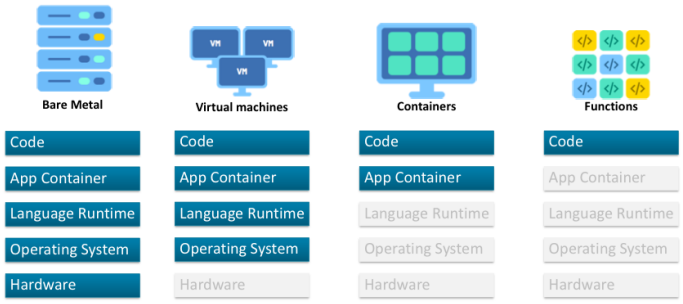


Firebase and other serverless architectures have rapidly emerged as a new technology concept in recent years. Using this Architecture, developers can create a variety of applications for various industries. Many enterprises have already started adopting serverless products, serverless architecture provides computation-light, highly-flexible, stateless applications and more. Developers are constantly on the lookout for more effective ways to maintain the software development lifecycle, doing so introduces new technologies that are accompanied with an increase in productivity.
Serverless architecture was introduced to help businesses focus on application development. With serverless, businesses no longer have to worry about server infrastructure, reducing development costs and shortening the development cycle.

A traditional architecture approach is frequently a large computer or cluster of computers that is accessed over the internet to provide access to information or servers. The machines are commonly located at specific area's around the world depending the size of the company. A traditional approach requires a team to manage these servers that which starts to become a problem what different scales of companies. As you get towards Enterprise sized servers they can become increasingly expensive. Not only do they cost Hundreds of Thousands to millions, you need to have space to house them, a staff of engineers to maintain them. In addition, the average life cycle of a given server is about 5 years. Which means not only do you have to potentially replace your machines, but in the interim you are constrained by the limitations of that machine.
An example of these would be if a websites server does not have the capacity to handle it's increased load, the only traditional solution is to purchase more servers to handle extra volume, or replacing existing servers with a better model. Both have exceptionally high cost and come with large downsides.

Serverless Architectures aims to solve these problems. Rather than having one machine to host a job, you can now utilize servers, such as Firebase, Amazon AWS and other cloud hosted databases. All have a huge array of additional features that you opt into to spec out a server that fits your needs and removes the additional costs and headaches of a traditional server.

Figure 3.1: Serverless Evolution



Advantages of Serverless

Advantages Serverless architectures has are the following:

1. Providers scale and manages the required resources
2. Rapid provision of resources in real-time, even for unforeseen peak loads and disproportionate growth
3. Highly scalable and flexible architecture
4. Users only need to pay for the resources they use
5. High error tolerance thanks to flexible hardware infrastructure in the provider's computer centers

Disadvantages of Serverless

Disadvantages Serverless architectures has are the following:

1. No access to virtual machines, operating system or runtime environments
2. Implementing serverless structures is very labor-intensive
3. Lock-in effect – for example, when changing provider, you generally have to recode all event-based functions
4. Relatively complex monitoring and debugging process, as in-depth performance, and error analyses are generally not possible

As shown Firebase is an excellent option for any developer interested in creating a mobile application, finally i will go over the Pros and Cons of choosing Firebase

The Pros of Firebase

- **Databases** - Depending on your budget, Google offers robust databases to use with your apps. Both realtime and Firestore databases can be scaled in terms of size, suggesting a fully secure managed solution, that still provides you easy access to your data via firebase console. Data updates and offline access makes databases usable for real time application, as well as keeping multiple databases in sync

- **Wide selection of products** - Firebase suggest a lot of products to make your application work. You can choose between realtime databases and firestore, store data in the cloud and build serverless applications with the integrated Cloud functions.

- **Free to use for small developers** - Firebase pricing[22] is free as you start with it. This will allow you to understand whether it fits your application and understand all the peculiarities. Once you reach certain amount of database memory or need a specific product, you can choose a different plan around that product or memory you need. You can do this by using the Blaze plan calculator provided on firebase pricing website[22]

- **Excellent Documentation** - The whole firebase platform is extremely well documented. Good technical documentation, API documentation, SDK references, which makes all the products easier to use and accessible for the user. The firebase products page contains all the required information concerning the integration's, available platforms, guidance's, docs, guides and lists supported technologies.

- **Accessible UI and ease of Integration** - Firebase requires minimal programming language knowledge, and suggests integration's via its user interface, it also is integrated into Android Studio which further increases ease of use. While eliminating the need for complex configurations, anyone can set up the application.

- **Google products** - Firebase comes with a Content Delivery Network (CDN) in-built with Google Cloud platform, while also having integration with all google products. A product owned by google also has the advantage of unlikely hood of the company collapsing or not supporting the product frequently.

The Cons of Firebase

The cons of firebase are limited and depends on the scope of you project and the size of your company. Some of the cons are currently being fixed through new features that are currently in Beta. For example Firebase Realtime Database, which i used in this project has limitations of not being able to do complex query's and Data Modeling. Google aims to fix this with its new product Firestore.

- **Firebase Realtime Database limitations** - Realtime Database is used as the main storage for my project, which has cons. One of the mains problems with it, is the limited querying capabilities. Realtime database provides no way to filter capabilities, because the whole Database is a huge JSON file, which makes it pretty difficult to make complex queries.
- **Data Modeling** - Firebase Realtime Database and it data modeling as a problem where because of "Database as a single JSON file" structure,m you can't implement relations between data items.
- **Locked into a vendor** - This is a wide problem with BaaS solutions in general, not having the ability to migrate data to another platform can be considered a con.
- **Support for iOS** - While firebase is a cross-platform product, it concentrates more on Android mobile platform being that its a google product. Android Studio easily integrates all Firebase products such as Test Lab, Auth etc. While iOS devices are lacking behind on such integration's.

3.2.7 Picasso

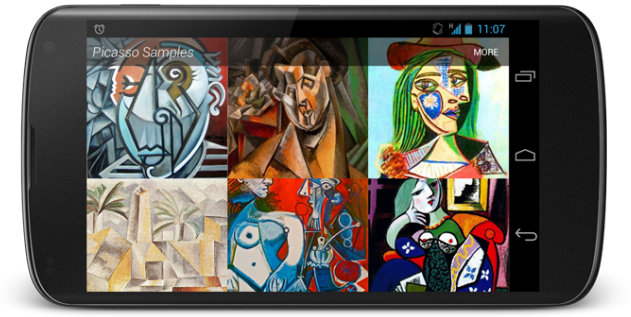
Picasso is a powerful image downloading and caching library, it is widely used and requires very little code to implement. This is used because working with images in android is difficult, you need to work with network requests, caching, background threads and decoding & encoding of image which uses a lot of memory. Picasso hides all this and adds additional features like resizing images while being memory efficient.

Images add much-needed context and visual flair to Android applications. Picasso allows for hassle-free image loading in your application—often in one line of code!

```
Picasso.get().load("http://i.imgur.com/DvpvklR.png").into(imageView);
```

Many common pitfalls of image loading on Android are handled automatically by Picasso:

- Handling ImageView recycling and download cancellation in an adapter.
- Complex image transformations with minimal memory use.
- Automatic memory and disk caching.



Features

ADAPTER DOWNLOADS

Adapter re-use is automatically detected and the previous download canceled.

```
@Override public void getView(int position, View convertView, ViewGroup parent) {
  SquaredImageView view = (SquaredImageView) convertView;
  if (view == null) {
    view = new SquaredImageView(context);
  }
  String url = getItem(position);

  Picasso.get().load(url).into(view);
}
```

IMAGE TRANSFORMATIONS

Transform images to better fit into layouts and to reduce memory size.

```
Picasso.get()
  .load(url)
  .resize(50, 50)
  .centerCrop()
  .into(imageView)
```

Can also specify custom transformations for more advanced effects.

```
public class CropSquareTransformation implements Transformation {
  @Override public Bitmap transform(Bitmap source) {
    int size = Math.min(source.getWidth(), source.getHeight());
    int x = (source.getWidth() - size) / 2;
    int y = (source.getHeight() - size) / 2;
    Bitmap result = Bitmap.createBitmap(source, x, y, size, size);
    if (result != source) {
      source.recycle();
    }
    return result;
  }

  @Override public String key() { return "square()"; }
}
```

Pass an instance of this class to the transform method.

PLACE HOLDERS

Picasso supports both download and error placeholders as optional features.

```
Picasso.get()
    .load(url)
    .placeholder(R.drawable.user_placeholder)
    .error(R.drawable.user_placeholder_error)
    .into(imageView);
```

A request will be retried three times before the error placeholder is shown.

RESOURCE LOADING

Resources, assets, files, content providers are all supported as image sources.

```
Picasso.get().load(R.drawable.landing_screen).into(imageView1);
Picasso.get().load("file:///android_asset/DvpvklR.png").into(imageView2);
Picasso.get().load(new File(...)).into(imageView3);
```

DEBUG INDICATORS

For development you can enable the display of a colored ribbon which indicates the image source. Call setIndicatorsEnabled(true) on the Picasso instance.

3.2.8 Paper

Paper is a fast NoSQL-like storage for Java/Kotlin objects on Android with automatic schema migration support. Paper's aim is to provide a simple yet fast object storage option for Android. It allows to use Java/Kotlin classes as is: without annotations, factory methods, mandatory class extensions etc. Moreover adding or removing fields to data classes is no longer a pain – all data structure changes are handled automatically.

Using Paper is very easy and extremely useful, for this project its used to save the User's login information in memory so when the user opens up the application after exiting they will automatically be logged in.

Initialize Paper [32] Initializing of Paper is needed in the onCreate or OnCreateView in the Activities were it is used

```
Paper.init(context);
```

Threading [32]

- Paper.init() should be called in UI thread;
- All other APIs (write, read etc.) are thread-safe and obviously must be called outside of UI thread. Reading/writing for different keys can be done in parallel.

Save [32] Save any object, Map, List, HashMap etc. including all internal objects. Use your existing data classes as is. Note that key is used as file name to store the data and so cannot contain symbols like.

```
List<Person> contacts = ...
Paper.book().write("contacts", contacts);
```

Read [32] Read data objects is as easy as

```
List<Person> = Paper.book().read("contacts");
```

the instantiated class is exactly the one used to save data. Limited changes to the class structure are handled automatically. See Handle data class changes. Use default values if object doesn't exist in the storage.

```
List<Person> = Paper.book().read("contacts", new ArrayList<>());
```

Destroy [32] To Remove all keys for the given Book. Paper.init() must be called prior calling destroy(). This is remove all user login data from memory which is called on logout.

```
Paper.book().destroy();
```

3.2.9 Native Applications

What is a Native Mobile App?

Native Mobile applications are applications that have been written using the native development language and tools specific to that platform. For example: A native iOS application would be written in either Swift or Objective-C and compiled using Xcode, while a native android application would have been developed using Kotlin or Java and compiled using Android Studio.

Since these applications are developed using the platform's default solutions, developers have full and easier access to the device's capabilities; like all the device's sensors, the user's address book, and whatever the latest and greatest new bit of technology the phone offers. Native applications tend to also be more performant since their code is closer to the 'metal'. In addition to being faster, you will also have access to all of the native user interface (UI) controls and layouts. While you will probably want to style them to fit your applications' theme, you will also want them to behave and interact like any other UI element on that platform.

However, any application written in a native development language cannot be run on the opposite platform. Meaning, you have to develop separately for each platform, which leads to larger budget and team size, assuming you want to release your application on both iOS and Android. In addition, your application is only available through each platform's app stores, each having their own set of rule and restrictions. This also means that do make a new release for a new feature/fix bugs etc. you same end up having to wait a minimum of one day for it to be released.

Advantages

- **Performance** - Native apps deliver the best performance of all development approaches. This is a major reason why platforms like Instagram and Airbnb [1] [34] moved from react native to native android development. Performance is one of the most important components of keeping a user on your application, the actions of your application need to be snappy and can't leave the user feeling like they need to wait.
- **Better Support** - Native apps receive complete support from app stores and the overall app marketplace. Distribution in app stores helps with discover-ability. Overtime Google has made Native development a more favoured development standard, along with the pressure of consumers waiting a extremely high standard from mobile applications. Having better support a company such as Apple or Google is a strong advantage over hybrid development.
- **Better Overall** - Native apps are interactive, intuitive, and run more smoothly in terms of user input and output. As mentioned native apps are just smoother for large applications.
- **Native Integration Features** - Native development allows developers to access the full feature set of the selected operating system. Not only do you have the full feature set from the selected operating system, you also have a the full feature set of tools like Android Studio.
- **Flow of app** - The user experience of native apps is far superior to web apps or hybrid apps. To the user, the flow is more natural because of each mobile operating system's specific UI guidelines and standards.
- **Goggle App Store Quality** - A native app must be approved by its respective operating system which assures quality, security, and device compatibility. While all Apps on the Google App Store aren't all at the highest standard, Google has an aggressive stance on the expected quality of applications and will quickly remove a developers access to the store if they are seen to be having a negative on the store as a whole.

Disadvantages

- **Higher Difficulty** - Native apps use difficult programming languages which require experienced developers.
- **Higher Cost** - Expenses are more costly upfront for native apps compared to web or hybrid apps.
- **Not the best option for Simple apps** - Native apps are not the best option for simple applications.
- **Hiring Experienced Developers** - Hiring experienced native developers can be more difficult and costly compared to a hybrid developer.

3.2.10 Hybrid Applications

What is a Hybrid App?

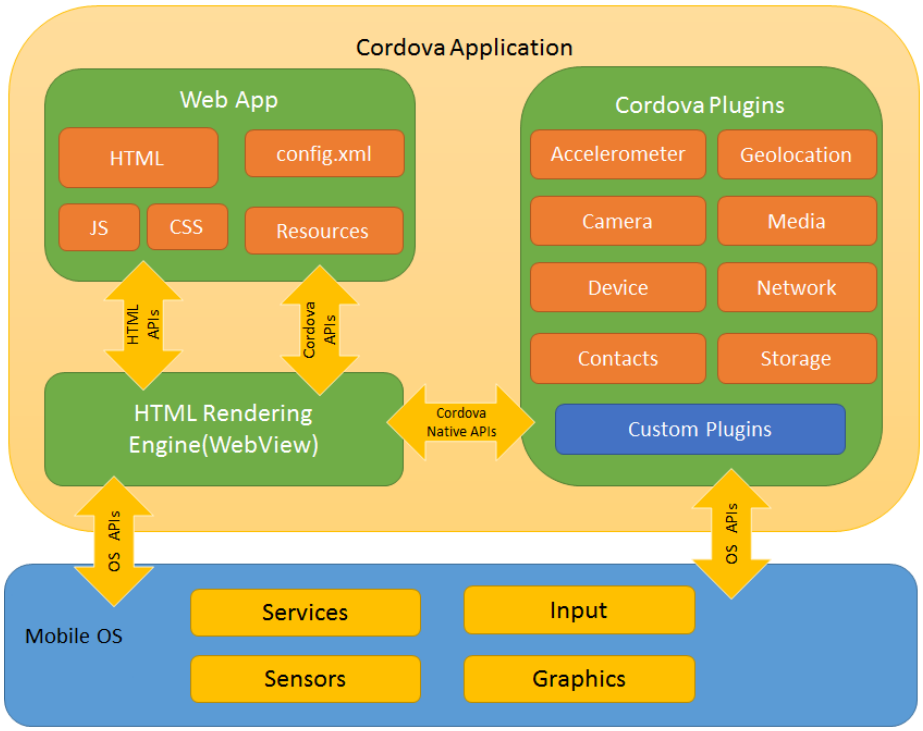
Hybrid apps are a blend, hence the name hybrid, of both native and web solutions. Where the core of the application is written using web technologies.

The core of an hybrid application is written using web technologies (HTML, CSS and JavaScript), which are then encapsulated within a native application. Through the use of plugins, these applications can have full access to the mobile device's features.[28]

The heart of a hybrid-mobile application is still just an application that is written with HTML, CSS, and JavaScript. However, instead of the app being shown within the user's browser, it is run from within a native application and its own embedded browser, which is essentially invisible to the user. For example, an iOS application would use the WKWebView to display our application, while on Android it would use the WebView element to do the same function.[28]

This code is then embedded into a native application wrapper using a solution like Apache Cordova and creates a native shell application that is just the platform's webview component in which it will load your web application. This gives you the ability to create and publish true native applications that can be submitted to each of the platform's app stores for sale.

Figure 3.2: Architecture of a Hybrid App (Cordova) [14]



When to use Hybrid Applications

Hybrid mobile applications have their place in situations where fast development is the main priority or when the high cost of targeting each separate platform with an individual native application would prohibit the creation of the application. Large companies who are in an extremely competitive app market are not going to sacrifice performance and control with hybrid development. However smaller developers have the option of taking advantage of the gap between native and hybrid development in terms of the most cost effective options and when it comes to cost hybrid development is very cost effective for a lot of companies.

Advantages

- **Unified Development** - By far the single biggest benefit that hybrid mobile apps can offer is the unified development. Companies can save a substantial amount of money that would otherwise have to be spent on developing and maintaining separate code bases for different mobile platforms. They can develop just a single version and let their hybrid framework of choice do the heavy lifting and ensure that everything will

work flawlessly. This, of course, directly leads to lower cost of development and, potentially, greater revenue. Many small businesses wouldn't be able to afford to target all major mobile platforms, if there wasn't the option to do so with a hybrid framework.

- **Fast Deployment** - The Minimum Viable Product (MVP) approach necessitates the fast deployment of functional solutions in order to be the first to penetrate the market and gain a substantial competitive advantage. Those who need to have their app in the App Store as fast as possible should seriously consider using hybrid applications.[30]

- **Scaling** - Hybrid applications are limited only by the underlying framework. Companies who partner with a good provider can instantly target all major platforms without any additional effort at all. It the platform is popular enough, it can be expected that it will quickly add support for any new mobile operating systems and their respective incremental updates.

- **Return on Investment** - Hybrid applications built with software such as Ionic have, 1. 60-80% time savings on average building with hybrid versus native. 2. Less hiring costs. 3. Project is more feasible with lower costs. [27]

Disadvantages

- **App Performance** - Hybrid apps add an extra layer between the source code and the target mobile platform: the particular hybrid mobile framework, such as Ionic, Cordova, Xamarin, React Native, and many others. The unsurprising result is a possible loss of performance. It really varies from application to application just how noticeable the difference can be, but the fact that Facebook migrated their mobile application from HTML5 to native shows that there really can be a significant difference, at least for large-scale applications. Mark Zuckerberg even went on to say that "The biggest mistake we've made as a company is betting on HTML5 over native."

- **Debugging** - That extra layer also makes debugging a potential nightmare. Developers have to rely on the framework itself to play nicely with the targeted operating system and not introduce any new bugs. Since developers are not likely to have a deep knowledge of the targeted platform, figuring out the exact cause of an issue can be a lengthy affair.

- **Lack native features** - Hybrid apps lack the integration with native operating systems that limit to app at certain points, once your app reaches a certain point of complexity Hybrid apps start to struggle.

Chapter 4

System Design

4.1 Chapter Introduction

In this section I will discuss the architecture of the application. I will discuss the components of an android application mainly Android Activities and the tops that come with Activities; Fragments, Activity lifecycle, persistence, process lifecycle etc. I will also discuss how the Front-End communicates with the Back-End after explaining of activities work.

4.2 Introduction to System

This application was made using Kotlin with Android Studio. My application uses Single Activity App Architecture, which is recommended by Google as the best way to make a Native Android Application[36]

4.3 Android Activities

An activity is a single, focused thing that the user can do. Almost all activities interact with the user, so the Activity class takes care of creating a window for you in which you can place your UI with setContentView. While activities are often presented to the user as full-screen windows, they can also be used in other ways: as floating windows (via a theme with android.R.attr#windowIsFloating set), Multi-Window mode or embedded into other windows.[37] There are two methods almost all subclasses of Activity will implement:

- onCreate is where you initialize your activity. Most importantly, here you will usually call setContentView(int) with a layout resource defining your UI, and using findViewById to retrieve the widgets in that UI that you need to interact with programmatically.

- onPause is where you deal with the user pausing active interaction with the activity. Any changes made by the user should at this point be committed (usually to the android.content.ContentProvider holding the data). In this state the activity is still visible on screen.

To be of use with Context.startActivity(), all activity classes must have a corresponding ¡activity¿ declaration in their package's **AndroidManifest.xml.**

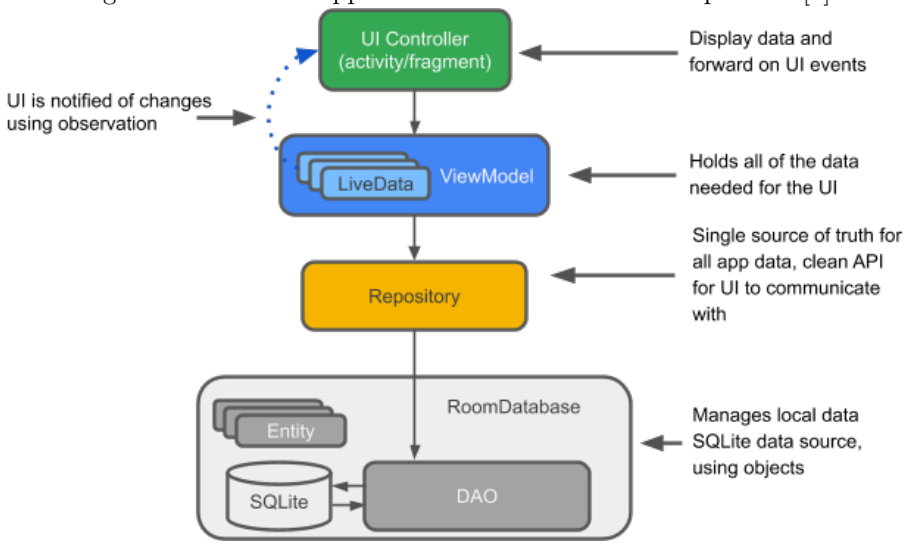
4.3.1 Fragments

A Fragment represents a behavior or a portion of user interface in a FragmentActivity. You can combine multiple fragments in a single activity to build a multi-pane UI and reuse a fragment in multiple activities. You can think of a fragment as a modular section of an activity, which has its own lifecycle, receives its own input events, and which you can add or remove while the activity is running (sort of like a "sub activity" that you can reuse in different activities).[3]

A fragment must always be hosted in an activity and the fragment's lifecycle is directly affected by the host activity's lifecycle. For example, when the activity is paused, so are all fragments in it, and when the activity is destroyed, so are all fragments. However, while an activity is running, you can manipulate each fragment independently, such as add or remove them. When you perform such a fragment transaction, you can also add it to a back stack that's managed by the activity—each back stack entry in the activity is a record of the fragment transaction that occurred. The back stack allows the user to reverse a fragment transaction (navigate backwards), by pressing the Back button.[3]

When you add a fragment as a part of your activity layout, it lives in a ViewGroup inside the activity's view hierarchy and the fragment defines its own view layout. You can insert a fragment into your activity layout by declaring the fragment in the activity's layout file, as a ¡fragment¿ element, or from your application code by adding it to an existing ViewGroup.[3]

Figure 4.1: Android app which uses Architecture components [8]



4.3.2 Creating a Fragment

Example of Fragment in Project to create the Navigation Bars:

HomeViewModel

```
class HomeViewModel : ViewModel() {

    private val _text = MutableLiveData<String>().apply {
        value = "This is home Fragment"
    }
    val text: LiveData<String> = _text
}
```

HomeFragment

```
class HomeFragment : Fragment() {

    private lateinit var homeViewModel: HomeViewModel

    override fun onCreateView(
        inflater: LayoutInflater,
        container: ViewGroup?,
        savedInstanceState: Bundle?
    ): View? {
        homeViewModel =
```

```
            ViewModelProviders.of(this).get(HomeViewModel::class.java)
        val root = inflater.inflate(R.layout.fragment_home, container, false)
        val textView: TextView = root.findViewById(R.id.text_home)
        homeViewModel.text.observe(this, Observer {
            textView.text = it
        })
        return root
    }
}
```

After creating a fragment you then need to declare it inside an activity layout file

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="http://schemas.android.com
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <TextView
        android:id="@+id/text_home"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:layout_marginEnd="8dp"
        android:textAlignment="center"
        android:textSize="20sp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

4.3.3 Communicating with a Activity

Although a **Fragment** is implemented as an object that's independent from a **FragmentActivity** and can be used inside multiple activities, a given instance of a fragment is directly tied to the activity that hosts it.
Specifically, the fragment can access the FragmentActivity instance with getActivity() and easily perform tasks such as find a view in the activity layout:

```
val listView: View? = activity?.findViewById(R.id.list)
```

Likewise, your activity can call methods in the fragment by acquiring a reference to the Fragment from FragmentManager, using findFragmentById() or findFragmentByTag(). For example:

```
val fragment = supportFragmentManager.findFragmentById(R.id.example_fragment) as ExampleFrag
```

4.4 Implementation of Activities

Here i will describe a list of Activities and their functionality with code examples.

4.4.1 List of Activities

Each of the following subsections describe the anatomy of each Activity and will discuss their purpose and the code used to implement each component.

4.4.2 Main Activity

The MainActivity contains all the code that will appear when first launching the application. It contains Login and Register button as well as the OnClick-Listener for those buttons. The MainActivity also contains the code for the navigation controller between fragments.

```
val drawerLayout: DrawerLayout = findViewById(R.id.drawer_layout)
val navView: NavigationView = findViewById(R.id.nav_view)
val navController = findNavController(R.id.nav_host_fragment)

// Passing each menu ID as a set of Ids because each
    // menu should be considered as top level destinations.
    appBarConfiguration = AppBarConfiguration(
        setOf(
            R.id.nav_home, R.id.nav_gallery, R.id.nav_slideshow,
            R.id.nav_tools, R.id.nav_share, R.id.nav_send
        ), drawerLayout
    )
    setupActionBarWithNavController(navController, appBarConfiguration)
    navView.setupWithNavController(navController)
} //onCreate end

override fun onCreateOptionsMenu(menu: Menu): Boolean {
    // Inflate the menu; this adds items to the action bar if it is present.
    menuInflater.inflate(R.menu.main, menu)
    return true
}

override fun onSupportNavigateUp(): Boolean {
    val navController = findNavController(R.id.nav_host_fragment)
    return navController.navigateUp(appBarConfiguration) || super.onSupportNavigateUp()
}
```

4.4.3 Register

The Register Activity is the activity that is triggered when the user clicks on Join Now Button, the RegisterAcivity is used to create an account on the Firebase Database and to also verify that the Phone Number given is unique to the database.

Similar to the LoginAcivity there are four steps that i will explain in four different subsubsections.

The Trigger Event

1. The user clicks on the Join Now Button, this will activate a OnClickListener in the MainAcivity

```
// This code is triggered by a OnClickListener,
// once triggered the view context of the app will move from MainAcivity to RegisterActivity
// RegisterActivity::class.java
joinNowButton.setOnClickListener{view ->
    val intent = Intent(view.context, RegisterActivity::class.java)
    view.context.startActivity(intent)
}
```

onCreate and Variables

2. Once the user triggers the event the UI will be shown

```
class RegisterActivity : AppCompatActivity() {

    private lateinit var CreateAccountButton: Button
    private lateinit var InputName: EditText
    private lateinit var InputPhoneNumber: EditText
    private lateinit var InputPassword: EditText
    private lateinit var loadingBar: ProgressDialog

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_register)

        CreateAccountButton = findViewById(R.id.register_btn)
        InputName = findViewById(R.id.register_user_name_input)
        InputPhoneNumber = findViewById(R.id.register_phone_number_input)
        InputPassword = findViewById(R.id.register_password_input)
        loadingBar = ProgressDialog(this)

        CreateAccountButton.setOnClickListener{
            CreateAccount()
        } }
```

Creating a Account

3. Once the user clicks the Join Now Button another OnClickListener will trigger which calls this method to Create an Account, CreateAccount() will also call validatePhoneNumber to valid the user's phone number input. This is used to remove any duplicate phone number as its the Primary Key in the Database.

```
  private fun CreateAccount() {
    var name = InputName.text.toString()
    var phone = InputPhoneNumber.text.toString()
    var password = InputPassword.text.toString()

    if (TextUtils.isEmpty(name)){
        Toast.makeText(this, "Write your name... ", Toast.LENGTH_SHORT).show()
    }
    else if (TextUtils.isEmpty(phone)){
        Toast.makeText(this, "Write your phone number... ", Toast.LENGTH_SHORT).show()
    }
    else if (TextUtils.isEmpty(password)){
        Toast.makeText(this, "Write your password... ", Toast.LENGTH_SHORT).show()
    }
    else{
        loadingBar.setTitle("Create Account")
        loadingBar.setMessage("Please wait, while we validate your information.")
        loadingBar.setCanceledOnTouchOutside(false)
        loadingBar.show()

        validatePhoneNumber(name, phone, password)

    }
}
```

Account Verification

4. This is the last step for the user to register a account. This method will enter the users input into the Firebase Database and will also check if any users contain the same Phone Number. If a user has the same phone number the account will not be created and the user will be prompted to enter a new number. The reason for this is the Phone Number is the Primary Key in the Database and the Phone Number is also setup for phone authentication with google products, so having multiple accounts with the same number would serve no purpose so eliminating this is the best option.

```
//WORKS AFTER TESTING - SCHEMA USERS - KEY PHONENUMBER - DETAILS NAME:< PASSWORD:< PHONE:
private fun validatePhoneNumber(name: String, phone: String, password: String) {

val RootRef:DatabaseReference
RootRef = FirebaseDatabase.getInstance().getReference()

RootRef.addListenerForSingleValueEvent(object: ValueEventListener{
override fun onDataChange(dataSnapshot: DataSnapshot) {
    //val post = dataSnapshot.getValue(String::class.java)
    //Update the UI with received data
    if (!(dataSnapshot.child("Users").child(phone).exists())){
        val childUpdates = HashMap<String, Any>()
        childUpdates.put("phone", phone)
        childUpdates.put("password", password)
        childUpdates.put("name", name)

        RootRef.child("Users").child(phone).updateChildren(childUpdates)
            .addOnCompleteListener(object: OnCompleteListener<Void>{
                override fun onComplete(@NonNull task: Task<Void>){
                    if (task.isSuccessful()){
                        Toast.makeText(this@RegisterActivity,
                        "Your Account has been created.", Toast.LENGTH_SHORT).show()
                        loadingBar.dismiss()

                        val intent = Intent(this@RegisterActivity,
                        LoginActivity::class.java)
                        startActivity(intent)
                    }//if
                    else{
                        loadingBar.dismiss()
                        Toast.makeText(this@RegisterActivity,
                        "Network Error: Please Try Again...", Toast.LENGTH_SHORT).show()
                    }
                }//onComplete
            })//addOnCompleteListener
    }//Data
    else{
        Toast.makeText(this@RegisterActivity,
        "This " + phone + "already exists.", Toast.LENGTH_SHORT).show()
        loadingBar.dismiss()
        Toast.makeText(this@RegisterActivity,
        "Please try another number", Toast.LENGTH_SHORT).show()

        val intent = Intent(this@RegisterActivity, LoginActivity::class.java)
        startActivity(intent)
    } }//onDataChange
```

4.4.4 Login

The login Activity is the activity that the user will see when clicking the Login button at the home page. The activity features a welcome message and prompts user to either Login or create a new account. From here depending on which button is pressed two things can happen either a Login Button is click or a Join Now button is clicked here i will explain of the LoginAcivity works.

There are four steps to this that i will explain in four different subsubsections.

The Trigger Event

1. The user clicks on the Login Button, this will activate a OnClickListener in the MainAcivity.

```
// This code is triggered by a OnClickListener,
// once triggered the view context of the app will move from MainAcivity to LoginAcivity
// LoginActivity::class.java
loginButton.setOnClickListener{view ->
    val intent = Intent(view.context, LoginActivity::class.java)
    view.context.startActivity(intent)
}
```

onCreate and Variables

2. Once the user triggers the event the UI will be shown

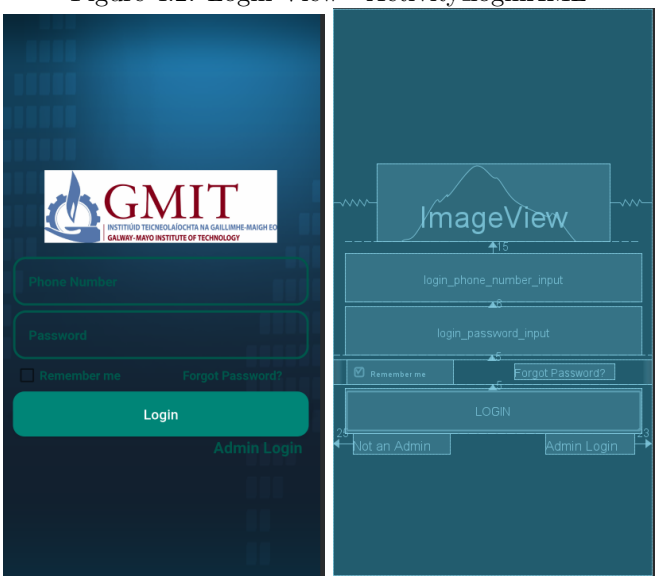
```
class LoginActivity : AppCompatActivity() {

    private lateinit var InputPassword: EditText
    private lateinit var InputPhoneNumber: EditText
    private lateinit var LoginButton: Button
    private lateinit var loadingBar: ProgressDialog
    private val parentDbName = "Users"

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_login)

        LoginButton = findViewById(R.id.login_btn)
        InputPhoneNumber = findViewById(R.id.login_phone_number_input)
        InputPassword = findViewById(R.id.login_password_input)
        loadingBar = ProgressDialog(this)

        LoginButton.setOnClickListener{
            LoginUser()
        }
    }//onCreate
```

Login the User

3. Once user now clicks the Login Button another OnClickListener will trigger which calls this method to login, LoginUser will also call AllowAccessToAccount to verify the user in the Firebase DB.

```
private fun LoginUser() {
    var phone = InputPhoneNumber.text.toString()
    var password = InputPassword.text.toString()

    if (TextUtils.isEmpty(phone)){
        Toast.makeText(this, "Write your phone number... ", Toast.LENGTH_SHORT).show()
    }
    else if (TextUtils.isEmpty(password)){
        Toast.makeText(this, "Write your password... ", Toast.LENGTH_SHORT).show()
    }
    else{
        loadingBar.setTitle("Login Account")
        loadingBar.setMessage("Please wait, while we validate your information.")
        loadingBar.setCanceledOnTouchOutside(false)
        loadingBar.show()

        AllowAccessToAccount(phone, password)
    }
}//LoginUser
```

Figure 4.2: Login View - Activity_loginẊML



Login Verification and Remember me

4. This is the last step for the user to Login. this method will called in the LoginUser method to verify that the details the user inputs are valid. If they're valid the user will be sent to the HomeAcivity page where they will see Products.

```
private fun allowAccessToAccount(phone: String, password: String) {

if (checkBoxRememberMe.isChecked){
    Paper.book().write(Prevalent.UserPhoneKey, phone)
    Paper.book().write(Prevalent.UserPasswordKey, password)
}
val RootRef: DatabaseReference
RootRef = FirebaseDatabase.getInstance().getReference()
RootRef.addListenerForSingleValueEvent(object: ValueEventListener {

    override fun onDataChange(dataSnapshot: DataSnapshot) {
        //var userData:Users
        if(dataSnapshot.child(parentDbName).child(phone).exists()){
            var usersData = dataSnapshot.child(parentDbName).child(phone).getValue(Users::cl

            if (usersData?.getPhone().equals(phone)){
                if (usersData?.getPassword().equals(password)){
                    if(parentDbName.equals("Admins")){
                        Toast.makeText(this@LoginActivity, "Admin Login Successful", Toast.L
                        loadingBar.dismiss()
                        //Sends user to AdminActivity
                        val intent = Intent(this@LoginActivity, AdminPanelActivity::class.ja
                        startActivity(intent)
                    }
                    else if(parentDbName.equals("Users")){
                        Toast.makeText(this@LoginActivity, "Login Successful", Toast.LENGTH_
                        loadingBar.dismiss()
                        //Sends user to HomeActivity
                        val intent = Intent(this@LoginActivity, HomeActivity::class.java)
                        Prevalent.currentOnlineUser = usersData!!
                        startActivity(intent)
                    }
                }
                else{
                    Toast.makeText(this@LoginActivity, "Password is Incorrect...", Toast.LEN
                    loadingBar.dismiss()
                }
            }//getPhone if
        }
        else{
            Toast.makeText(this@LoginActivity, "Account with this " + phone + " Number does
```

4.4.5 Navigation

The Navigation for my project is created using the Navigation Drawer Activity in combination with Fragments

Setting up Drawer Resources

First you must create an drawer resource. Here i will show aciviity_home_drawer.xml

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    tools:showIn="navigation_view">

    <group android:checkableBehavior="single">
        <item
            android:id="@+id/nav_home"
            android:icon="@drawable/ic_menu_camera"
            android:title="Home" />
        <item
            android:id="@+id/nav_gallery"
            android:icon="@drawable/ic_menu_gallery"
            android:title="Cart" />
        <item
            android:id="@+id/nav_slideshow"
            android:icon="@drawable/ic_menu_slideshow"
            android:title="Orders" />
        <item
            android:id="@+id/nav_tools"
            android:icon="@drawable/ic_menu_manage"
            android:title="Categories" />
    </group>

    <item android:title="Settings">
        <menu>
            <item
                android:id="@+id/nav_share"
                android:icon="@drawable/ic_menu_share"
                android:title="Settings" />
            <item
                android:id="@+id/nav_send"
                android:icon="@drawable/ic_menu_send"
                android:title="Logout" />
        </menu>
    </item>

</menu>
```

Define Fragments

After this next is to define fragments that will be displayed within the Navigation Activity;. These fragments are created in the ui folder each with a ViewModel and a host Fragment. Once created we then need to create Mobile_navigation.xml which is used to navigation between each fragment.

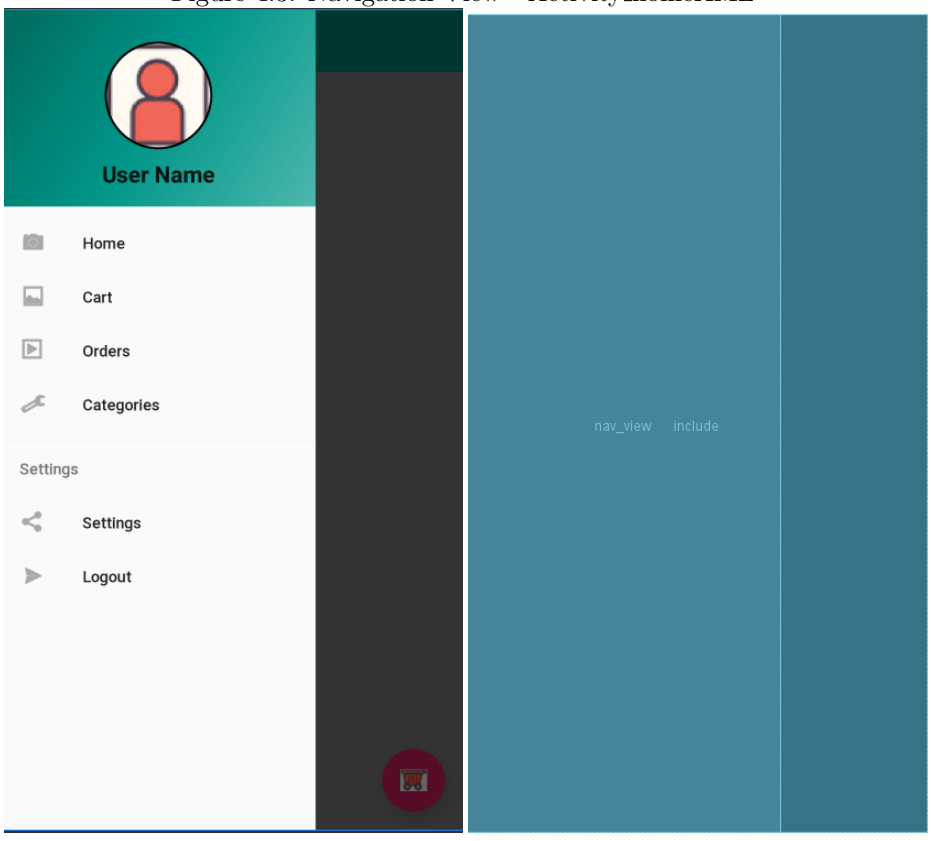
```
    <fragment
        android:id="@+id/nav_home"
        android:name="com.example.ecommerce.ui.home.HomeFragment"
        android:label="@string/menu_home"
        tools:layout="@layout/fragment_home" />

    <fragment
        android:id="@+id/nav_gallery"
        android:name="com.example.ecommerce.ui.cart.CartFragment"
        android:label="@string/menu_gallery"
        tools:layout="@layout/fragment_cart" />

    <fragment
        android:id="@+id/nav_slideshow"
        android:name="com.example.ecommerce.ui.orders.OrdersFragment"
        android:label="@string/menu_slideshow"
        tools:layout="@layout/fragment_orders" />

    <fragment
        android:id="@+id/nav_tools"
        android:name="com.example.ecommerce.ui.categories.CategoriesFragment"
        android:label="@string/menu_tools"
        tools:layout="@layout/fragment_tools" />

    <fragment
        android:id="@+id/nav_share"
        android:name="com.example.ecommerce.ui.settings.SettingsFragment"
        android:label="@string/menu_share"
        tools:layout="@layout/fragment_settings" >
    </fragment>

    <fragment
        android:id="@+id/nav_send"
        android:name="com.example.ecommerce.ui.logout.LogoutFragment"
        android:label="@string/menu_send"
        tools:layout="@layout/fragment_logout" />
</navigation>
```

Navigation Host Fragment

Once we've create all the Fragments and connected them with the Navigation View we must choose a Nav Host Fragment. This is basically the highest level activity of the navigation view, anything created on this Activity will appear on other views when you change to other Fragments in the navigation.

```
<fragment
    android:id="@+id/nav_host_fragment"
    android:name="androidx.navigation.fragment.NavHostFragment"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:defaultNavHost="true"
    app:navGraph="@navigation/mobile_navigation" />
```

Figure 4.3: Navigation View - Activity_homeẊML



Setup Navigation in Activity

Setting up the Navigation within the Activity does not take up much code in kotlin compared to Java.

First i initialize all the needed Views and setup the navController

```
val toolbar: Toolbar = findViewById(R.id.toolbar)
setSupportActionBar(toolbar)
val drawerLayout: DrawerLayout = findViewById(R.id.drawer_layout)
val navView: NavigationView = findViewById(R.id.nav_view)
val navController = findNavController(R.id.nav_host_fragment)
setupActionBarWithNavController(navController, appBarConfiguration)
navView.setupWithNavController(navController)
```

I will also attach a addOnDestinationChangedListener to the navController.

```
navController.addOnDestinationChangedListener { _, destination, _ ->
            when {
                destination.id == R.id.nav_share -> {
                    toolbar.visibility = View.VISIBLE
                    recyclerView.visibility = View.GONE
                }
                destination.id == R.id.nav_tools -> {
                    toolbar.visibility = View.VISIBLE
                    recyclerView.visibility = View.GONE
                }
                destination.id == R.id.nav_slideshow -> {
                    toolbar.visibility = View.VISIBLE
                    recyclerView.visibility = View.GONE
                }
                destination.id == R.id.nav_gallery -> {
                    toolbar.visibility = View.VISIBLE
                    recyclerView.visibility = View.GONE
                }
                destination.id == R.id.nav_send -> {
                    toolbar.visibility = View.VISIBLE
                    recyclerView.visibility = View.GONE

                    Paper.book().destroy()
                    val intent = Intent(this, LoginActivity::class.java)
                    intent.flags = Intent.FLAG_ACTIVITY_NEW_TASK or Intent.FLAG_ACTIVITY_CLE
                    startActivity(intent)
                }
                else -> {
                    toolbar.visibility = View.VISIBLE
                    recyclerView.visibility = View.VISIBLE
                }
```

After implementing all of these code snippets the Navigation Drawer is now fully connected and ready to have code implemented. The Navigation will display as shown in Figure 4.5

4.4.6 Admin

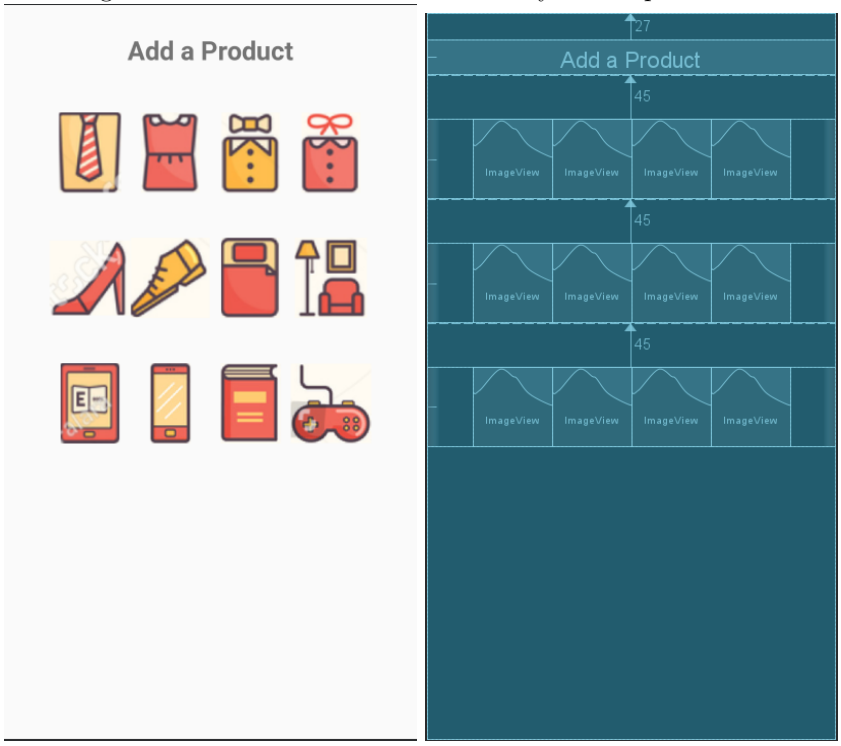
The admin portion of my project is entered through the LoginActivity where the user must choose the admin login to reach the AdminPanelActivity.

The Admin feature consists of two different Activities, AdminActivity & AdminPanelActivity

AdminPanelActivity

The AdminPanelActivity is the first View the user will see after logging in with an Admin Account. This Panel is used for the user to add products to the application store. The panel consists of Twelve different categories, each being a different category for the store (Men's clothes, Women's Clothes, furniture, phones etc.)

Figure 4.4: Admin Panel View - Activity_admin_panelẊML



Each ImageView shown in Figure 4.5 as its on setOnClickListener which will pass on the Category and the value of the Categoryt to the AdminActivity when an imageview is chosen.

```
//*********************************************************************
//Layout 3 OnClickListeners - tablets, phones, books, gaming
//*********************************************************************

tablets.setOnClickListener{
    val intent = Intent(this, AdminActivity::class.java)
    intent.putExtra("category", "Tablets")
    startActivity(intent)
}
phones.setOnClickListener{
    val intent = Intent(this, AdminActivity::class.java)
    intent.putExtra("category", "Phones")
    startActivity(intent)
}
books.setOnClickListener{
    val intent = Intent(this, AdminActivity::class.java)
    intent.putExtra("category", "Books")
    startActivity(intent)
}
gaming.setOnClickListener{
    val intent = Intent(this, AdminActivity::class.java)
    intent.putExtra("category", "Gaming")
    startActivity(intent)
}
```

AdminActivity

After Choosing a Category in the AdminPanelActivity the setOnClickListener will send the user to the AdminActivity here the Products are added with a Name, Desc and Price. The user will also choose the image that will be displayed in the RecyclerView after a user logs in.

Figure 4.5: Admin Panel View - Activity_admin_panelẊML

Saving Products to Database

1. Initialize all the needed Views

```
CategoryName = getIntent().getExtras()?.get("category").toString()
ProductImageRef = FirebaseStorage.getInstance().reference.child("Product Images")
ProductRef = FirebaseDatabase.getInstance().reference.child("Products")
InputProductImage = findViewById(R.id.select_product)
InputProductName = findViewById(R.id.product_name)
InputProductDesc =  findViewById(R.id.product_desc)
InputProductPrice = findViewById(R.id.product_price)
AddProductButton =  findViewById(R.id.add_new_product)
loadingBar = ProgressDialog(this)
```

2. Set up OnClickListerners

```
    //OnClickListeners
    InputProductImage.setOnClickListener{
        openProducts()
    }

    AddProductButton.setOnClickListener{
        validateProducts()
    }
```

3. Validation Before saving any to the database, validation to be sure all fields are appropriately filled is needed. If all fields are not empty the storeProduct-Info() function will be called to store the details

```
    private fun validateProducts() {
    Desc = InputProductDesc.getText().toString()
    Price = InputProductPrice.getText().toString()
    Prod_Name = InputProductName.getText().toString()

    if(TextUtils.isEmpty(Desc)){
        Toast.makeText(this, "Please write a Product Desc ", Toast.LENGTH_SHORT).show()
    }
    else if(TextUtils.isEmpty(Price)){
        Toast.makeText(this, "Please write a Product Price ", Toast.LENGTH_SHORT).show()
    }
    else if(TextUtils.isEmpty(Prod_Name)){
        Toast.makeText(this, "Please write a Product Name ", Toast.LENGTH_SHORT).show()
    }
    else{
        storeProductInfo()
    }
}
```

4. Storing the Product Details
First we must declare all the necessary variables

```
private fun storeProductInfo() {
    loadingBar.setTitle("Adding a Product.. ")
    loadingBar.setMessage("Please wait, while we add the product.. ")
    loadingBar.setCanceledOnTouchOutside(false)
    loadingBar.show()

    var c = Calendar.getInstance()

    var currentDate = SimpleDateFormat("MMM dd, yyyy")
    saveCurrentDate = currentDate.format(c.time)
```

```
    var currentTime = SimpleDateFormat("HH:mm:ss a")
    saveCurrentTime = currentTime.format(c.time)

    //Create a Random key
    productKey = saveCurrentDate + saveCurrentTime

    //Store Img in Firebase - https://firebase.google.com/docs/storage/android/upload-files#
    var filePath = ProductImageRef.child(ImageUri.lastPathSegment + productKey + ".jpg")
    val ut = filePath.putFile(ImageUri)
```

We now have all the necessary variables to create a product in the database.

5. Validation and Storing Image Before calling the finally function that will store all the products details we must validate and store the Image so it can be put in the HashMap which will then put the product details into the database.

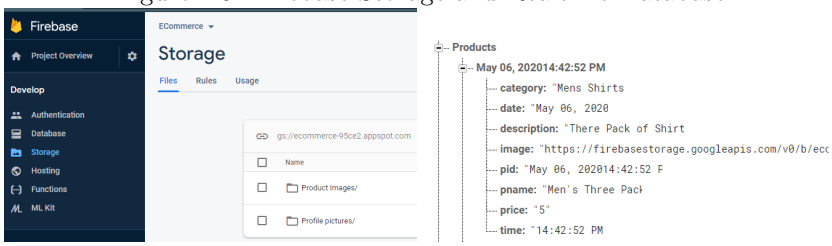
```
val urlTask = ut.continueWithTask { task ->
        if (!task.isSuccessful) {
            task.exception?.let {
                throw it
            }
        }
        downloadImageUrl = filePath.downloadUrl.toString()
        return@continueWithTask filePath.downloadUrl
    }.addOnCompleteListener { task ->
        if (task.isSuccessful) {
            downloadImageUrl = task.result.toString()
            Toast.makeText(this@AdminActivity, "Product Image Uploaded ", Toast.LENGTH_SHORT
            saveToDatabase()
        } else {
            loadingBar.dismiss()
            var ex = task.exception.toString()
            Toast.makeText(this@AdminActivity, "Error: $ex", Toast.LENGTH_SHORT).show()

        }
    }.addOnFailureListener {task ->
        var ex = task.cause.toString()
        Toast.makeText(this@AdminActivity, "Error: $ex", Toast.LENGTH_SHORT).show()
        loadingBar.dismiss()
    }
}
```

6. Saving to Database

```
private fun saveToDatabase() {
    var pMap:HashMap<String, Any> = HashMap()
    pMap.put("pid", productKey)
    pMap.put("date", saveCurrentDate)
    pMap.put("time", saveCurrentTime)
    pMap.put("description", Desc)
    pMap.put("image", downloadImageUrl)
    pMap.put("category", CategoryName)
    pMap.put("price", Price)
    pMap.put("pname", Prod_Name)

    ProductRef.child(productKey).updateChildren(pMap).addOnCompleteListener{ task ->
        if (task.isSuccessful) {
            //Sends user to Admin Panel
            val intent = Intent(this@AdminActivity, AdminPanelActivity::class.java)
            startActivity(intent)

            loadingBar.dismiss()
            Toast.makeText(this@AdminActivity, "Product Added Successfully ", Toast.LENGTH_S
        }
        else{
            loadingBar.dismiss()
            var ex = task.exception.toString()
            Toast.makeText(this@AdminActivity, "Error: $ex", Toast.LENGTH_SHORT).show()

        }
    }
}
```

After this the images are stored in Firebase Storage and product details in the Realtime database shown in Figure 4.6

Figure 4.6: Firebase Storage and Realtime Database

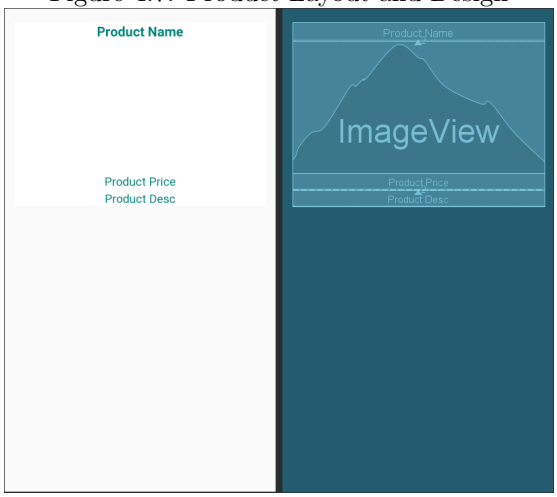


4.4.7 RecyclerView

The RecyclerView is used to display all the products stored on the Database. Implementation of this goes as follows:

1. Create the Product layout in a CardView, products_layoutẋml

Figure 4.7: Product Layout and Design



2. Create a Model Kotlin Class

```
companion object{
    public lateinit var category:String
    public lateinit var date:String
    public lateinit var description:String
    public lateinit var image:String
    public lateinit var pid:String
    public lateinit var pname:String
    public lateinit var price:String
    public lateinit var time:String }

init {
        lateinit var category:String
        lateinit var date:String
        lateinit var description:String
        lateinit var image:String
        lateinit var pid:String
        lateinit var pname:String
        lateinit var price:String
        lateinit var time:String }
```

3. Create a ProductViewHolder which implements ItemClickListener inferface

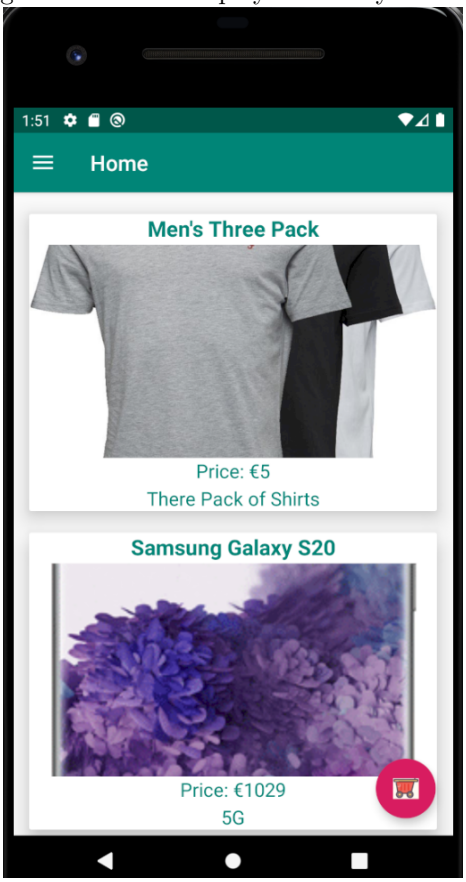
```
class ProductView:RecyclerView.ViewHolder, View.OnClickListener{
    var txtProdName:TextView
    var txtProdDesc:TextView
    var txtProdPrice:TextView
    var txtImageView:ImageView
    lateinit var listener: ItemClickListener
    constructor(itemView: View) : super(itemView)

    fun ProductView(itemView:View){
        super.itemView
        txtImageView = itemView.findViewById(R.id.product_card_img)
        txtProdName = itemView.findViewById(R.id.product_card_name)
        txtProdDesc = itemView.findViewById(R.id.product_card_desc)
        txtProdPrice = itemView.findViewById(R.id.product_card_price)
    }

    init {
        txtImageView = itemView.findViewById(R.id.product_card_img)
        txtProdName = itemView.findViewById(R.id.product_card_name)
        txtProdDesc = itemView.findViewById(R.id.product_card_desc)
        txtProdPrice = itemView.findViewById(R.id.product_card_price)
    }

    fun setItemClickListener(listener: ItemClickListener){
        this.listener = listener
    }


    override fun onClick(v: View?) {
        listener.onClick(v!!, adapterPosition, false)
    }
}
```

4. Finally display the recyclerView in the HomeActivity

```
override fun onStart() {
    super.onStart()

    val options = FirebaseRecyclerOptions.Builder<Products>().setQuery(productsRef, Products

    val adapter:FirebaseRecyclerAdapter<Products, ProductView> = object:FirebaseRecyclerAdap
        override fun onBindViewHolder(holder: ProductView, position: Int, model: Products) {
            holder.txtProdName.setText(model.getPname())
            holder.txtProdDesc.setText(model.getDescription())
```

```
            holder.txtProdPrice.setText("Price: " + "€" + model.getPrice())        //"Price:
            Picasso.get().load(model.getImage()).into(holder.txtImageView)
        }

        override fun onCreateViewHolder(parent: ViewGroup, viewType: Int): ProductView {
            val view = LayoutInflater.from(parent.context).inflate(R.layout.products_layout,
            return ProductView(view)
        }
    }//adapter
    recyclerView.adapter = adapter
    adapter.startListening()
}//onStart
```

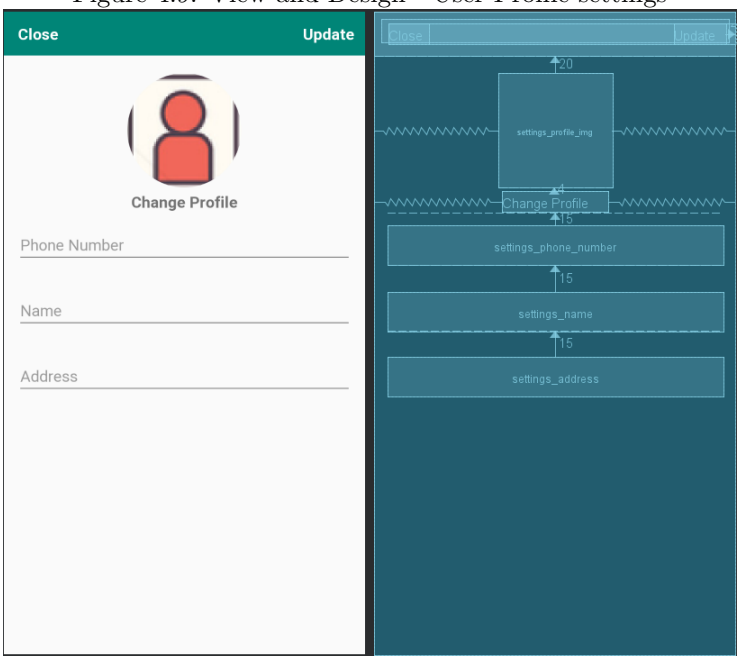
Figure 4.8: Items displayed in RecyclerView



4.4.8 Settings

The settings are used for the user to edit there User Profile details. The user can choose a Profile image, as well as change there Phone Number, Name and Address

Figure 4.9: View and Design - User Profile settings



The code for settings is located in the SettingsFragment, this fragment contains three different aspects: Allowing User to choose a Profile Image, Saving the Users Details and Displaying Users New Details

1. Allowing User to choose a Profile Image

Once the user chooses to change there profile image a onClickListener will trigger to create the process

```
profileChangeTxtBtn.setOnClickListener{ it: View? ->
    check = "clicked"

    // for fragment (DO NOT use `getActivity()`)
    this.context?.let { it1 ->
        CropImage.activity()
            .setAspectRatio(1, 1)
            .start(it1, this)
```

Once triggered this will prompt the User to choose a folder in there phone and contains the images they want to display as there profile picture. Once an image is chosen the user will be prompted with a screen to Crop that image

Figure 4.10: Crop image for User Profile Picture

Saving Cropped Image

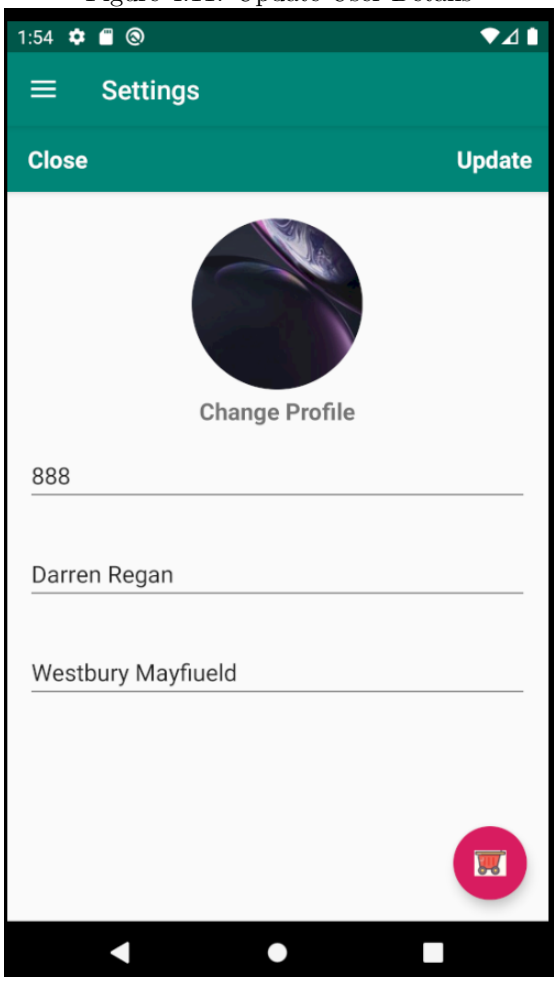
Once the User has chosen the cropped image that they like, a function onActivityResult will be called. This function will validate the cropped image as well as saving the image if it is suitable.

```
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)

    if (requestCode == CropImage.CROP_IMAGE_ACTIVITY_REQUEST_CODE){
        val result = CropImage.getActivityResult(data)
        if (resultCode == RESULT_OK){
            imageUri = result.uri
            profileImageView.setImageURI(imageUri)
        }else if (resultCode == CropImage.CROP_IMAGE_ACTIVITY_RESULT_ERROR_CODE){
            var error = result.error
            Toast.makeText(this.context, "Error, please try again ", Toast.LENGTH_SHORT).sho
        }
    }
}
```

2. Saving the Users Details

There is two separate functions for saving the User Details. One function uploadImage() contains code for uploading the Cropped Image that the user has chosen to the Database and updateUserInfo() will update users data if they wish to not choose to upload a new profile picture.

Figure 4.11: Update User Details



Updating all user details

Once the user clicks the update button shown in Figure 4.11 it will trigger the onClickListener

```
saveTextBtn.setOnClickListener{ it: View? ->
    if (check == "clicked"){
        saveUserInfo()
    }
    else{
        updateUserInfo()
    }
}
```

SaveUserInfo() is called when the user decides to upload a image, this will first validate that the user has entered details before calling the uploadImage() function

```
private fun saveUserInfo() {
    when {
        TextUtils.isEmpty(fullNameEditText.text.toString())
        -> Toast.makeText(this.context, "Must have a name ", Toast.LENGTH_SHORT).show()
        TextUtils.isEmpty(addressEditText.text.toString())
        -> Toast.makeText(this.context, "Must have a address ", Toast.LENGTH_SHORT).show()
        TextUtils.isEmpty(userPhoneEditText.text.toString())
        -> Toast.makeText(this.context, "Must have a phone number ", Toast.LENGTH_SHORT).sho
        check == "clicked" -> uploadImage()
    }
}
```

Upload Image and Save Data

Once triggered the process of saving the users detail will now start

```
run {
    val fileRef =
        storageProfilePicRef.child(Prevalent.currentOnlineUser.getPhone() + ".jpg")
    uploadTask = fileRef.putFile(imageUri)

    val ut = uploadTask.continueWithTask { task ->
        if (!task.isSuccessful) {
            task.exception?.let {
                throw it
            }
        }
        myUrl = fileRef.downloadUrl.toString()
        return@continueWithTask fileRef.downloadUrl
    }.addOnCompleteListener { task ->
        if (task.isSuccessful) {
            val downloadUrl = task.result
            myUrl = downloadUrl.toString()

            var ref = FirebaseDatabase.getInstance().reference.child("Users")
            var userMap:HashMap<String, Any> = HashMap()
            userMap["name"] = fullNameEditText.text.toString()
            userMap["address"] = addressEditText.text.toString()
            userMap["phoneOrder"] = userPhoneEditText.text.toString()
            //userMap["phone"] = userPhoneEditText.text.toString()
            userMap["image"] = myUrl
            ref.child(Prevalent.currentOnlineUser.getPhone()).updateChildren(userMap)

            progressDialog.dismiss()

            val intent = Intent(activity, HomeActivity::class.java)
            startActivity(intent)
            Toast.makeText(this.context, "Profile Updated! ", Toast.LENGTH_SHORT).show()
        }
```

3. Display new User Details

After saving all the details the User is sent to the HomeActivity, if user returns to the settings page there new details will be displayed shown in Figure 4.11, the function displayUserInfo() is used to achieve this

```
private fun displayUserInfo(profileImageView: CircleImageView?, fullNameEditText: EditText?,
userPhoneEditText: EditText?, addressEditText: EditText?) {
    val usersRef = FirebaseDatabase.getInstance().reference.child("Users").child(Prevalent.c

    usersRef.addValueEventListener(object: ValueEventListener {
        override fun onCancelled(p0: DatabaseError) {
        }

        override fun onDataChange(dataSnapshot: DataSnapshot) {
            if (dataSnapshot.exists()){
                if (dataSnapshot.child("image").exists()){
                    val image:String = dataSnapshot.child("image").value.toString()
                    val name:String = dataSnapshot.child("name").value.toString()
                    val phone:String = dataSnapshot.child("phone").value.toString()
                    val address:String = dataSnapshot.child("address").value.toString()

                    Picasso.get().load(image).into(profileImageView)
                    fullNameEditText?.setText(name)
                    userPhoneEditText?.setText(phone)
                    addressEditText?.setText(address)
                }
            }
        }
    })//EventListener
}//displayUserInfo
```

Chapter 5

System Evaluation

5.1 Robust

It was very important from me to have this application be robust as a basic requirement. I did this by following Android Jetpack[4] guidelines that Accelerates Development, Eliminates boilerplate code and helps build higher quality, robust apps. Using Android Jetpack, i will list some of the features used.

5.1.1 Benchmark

The Jetpack Benchmark library allows you to quickly benchmark your Kotlin-based or Java-based code from within Android Studio. The library handles warmup, measures your code performance, and outputs benchmarking results to the Android Studio console.

I used this with its integration into Android Studio for testing the RecyclerView performance, this helped with receiving accurate logs that helped with debugging.

5.1.2 Multidex

In my application i quickly ran into a problem where my app and the libraries references exceed 65,536 methods. This caused many issues for me but was fixed with the implemention of Mutlidex

5.1.3 Navigation

The use of Jetpack guidelines for implementing Navigation made the navigation extremely fast compared to my first implementation. The Navigation component consists of three key parts, Navigation Graph, NavHost and NavController which was detailed in the Navigation section of System Design.
Implementing the compontent features provides a number of benefits

- Handling fragment transactions
- Handling Up and Back actions by default
- Provides standardized resources for animations and transitions.
- Implementing and handling deep linking.
- Including Navigation UI patterns, such as navigation drawers and bottom navigation, with minimal additional work.
- Safe Args - a Gradle plugin that provides type safety when navigating and passing data between destinations.
- ViewModel support - you can scope a ViewModel to a navigation graph to share UI-related data between the graph's destinations.

5.1.4 Lifecycles

Implementing Lifecycle for my navigation fragments contributed heavily to no errors or performance issues. Lifecycle-aware components perform actions in response to a change in the lifecycle status of another component, such as activities and fragments. These components help you produce better-organized, and often lighter-weight code, that is easier to maintain.

By following the Best practices for lifecycle-aware components [5] i was able to keep my UI controllers (activities and fragments) as lean as possible.

5.1.5 Speed of App and Saving to Database

The application components load very fast thanks to following the Jetpack guidelines. Using the Firebase Realtime Database once a product or user details buttons are pressed the Database is updated extremely fast within one to three seconds.

5.2 Accessibility

As a E-Commerce focused application Accessibility is the highest priority, as people of all ages may use your application. Because of this your application must emphasize a Minimalistic UI. The user should perform the least amount of tasks as possible to navigate your application. To reflect this ideal i will reflect on how i think i accomplished this task.

5.2.1 Minimalistic

With the Goal of a Minimalistic UI i chose to only display very clear buttons, images etc. The buttons are generally very large so that the user cannot be overwhelmed by an influx of information. Each section of the app contains at most two buttons and a few text fields to accomplish this task.

5.2.2 Intuitive

Along with the Minimalistic UI the application is Intuitive as there is no complicated or technical information displayed. Instead there is very clear text on each action so that the vast majority of the users will at a glance know exactly what each page does.

5.3 Evaluation of Objectives

5.3.1 Learning Android Development

My main goal for this project was to learn Android Development, as i am interested in choosing Android Development as my Career option. Through creating this project i have very clearing accomplished this goal. Along with using Google Recommend design principles and through sheer hours of working with Android Studio i have become accustomed to working with Android Development and gained the understanding of building a high-quality application.

5.3.2 Features

My goals for features where to high, while i feel like i gained exactly what i wanted for this project, i set my objectives to high without realising of long it won't take to learn Kotlin, Android Development and Android Studio features at the same time. I would like to revisit the project in the summer to add more features as i now have a good grasp of development and have learned from my mistakes with Scope-Creep and planning

5.3.3 Scalable and Re-Usable

The application scales very well and functions are set with Googles recommend standards. The application is re-usable for any type of business and implementing a new feature is extremely easy with how the application is set up.

5.3.4 Responsiveness

My goal for the project to be responsive worked out well thanks to the development principles that i followed. Not only is the Navigation very fast, the read/write from the database is almost instantaneous which i am very happy with/

Chapter 6

Conclusion

This chapter serves as a conclusion to the project. In it, the objectives of the project are analysed again briefly to evaluate whether or not they were met. It also analyses any improvements and/or changes that could be added/made to the project if it was to be developed again.

6.1 Objectives Review

The main Goal and Objective of my project was to Learn Android Development, Kotlin and Android Studio. I set my requirements to implement the current recommend development methods created by Google. Everything else was an objective or extra feature if i had time. I feel confident that i achieved my goals for this Project. I spent well over 200 Hours inside Android Studio while learning how to implement and get accustomed to developing with Kotlin. Kotlin removes are large amount of boilerplate code, which for a large amount of the project slowed me down because i am so accustomed to Java development. A java method that is 20 to 30 lines of code can be reduced to a simple Lambda expression in Kotlin which infers all the logic that java would have to specify.

Overall i am happy with what I've achieved with my Project, while the COVID-19 pandemic did affect how much i implemented i achieved the most important Goals of my project which were to Learn Android Development, Kotlin and Android Studio.

6.1.1 What i learned

The main objectives for this project was to learn a new programming language and to learn and create a native android application. These two objectives are the most important as the main goal was to get insight on Android Development as Android Development was something i was debating on choosing as a main career option. Through completing this Project i feel like i gained exactly what i wanted out of this project and i am now highly considering Android Development as my career going forward.

6.2 Improvements

There are many improvements i would want to make to my application, integrating the many products from Firebase shown in the tech review, creating a product analysis, creating consumer statistics for buying habits, integrating Google Analytics for advertisements etc. All of these we're considered and even attempted but were ultimately out of scope of a non-team project.

6.3 Downfalls

These are some downfalls of the project, i give my thoughts on what i learned and how i would fix these issues going forward. These two i feel like are actually a huge benefit for me to have a downfall on before i go into industry. I know of these problems from our study but have never actually experienced them properly, this experience should give me insight before going into industry on how i should plan out my work and how valuable Software Methodologies such as Agile are to developing software.

6.3.1 Scope Creep

Scope creep was hard to manage in this project as i kept wanting to add more features before i had a solid foundation. I originally wanted an employee chat and calendar to manage work days, additionally i wanted to add statistics for purchasing habits etc. But all these features made no sense in the scope of this project as i am solo. If i had two addition persons then these features would be feasible, but i wasted a good amount of time researching how i would develop these features instead of just coding the more basic features of the project.
Scope Creep had a very negative impact on this project for this reason, and is something i have learned a great deal amount now that i have actually experienced it.

Ways i found to combat scope-creep:

- **Planning**- Carefully planning out the features/components and sticking to that plan and only branching out when the plan is complete or is needed. This would make it less likely to fall prey to scope-creep
- **Focus on Main Features** - Focusing on the main features that i had described in my deign document is the easiest way to combat scope-creep. I had laid out features but over time i focused less and less on those features and branched off to area's and didn't add anything to the actual project.

  After this project i now know a lot more on how to focus on the important features, making sure they're implemented and tested adequately before implementing new features.
- **Utilizing Development Cycles** - I feel like i should have utilized Test driven development more, i believe i should of just implemented the feature and gone back to refactor it to save a lot of time. But i mostly ended up not implementing a lot of stuff due to time constraints.

6.3.2 Utilize Docs

What i noticed with Kotlin and Android Development, is that there is a huge amount of documentation that i didn't end up reading, so i would implement for example Firebase authentication and Login/Sign-up features through trial and error. But there was a resource available for it for free which i previously thought cost a fixed amount. So what i learned is that reading the full documentation is a huge benefit before trying to implement a feature. Documentation for Android, not only helps with implementing features but also have a large amount of tips on UI Design, Marketing Strategies, Performance Tips and many more topics.

6.4 Final Conclusion

This project was a great learning experience. It has given me real insight on how important actual Software Methodologies are to creating Software in a timely manner. I gained experience with a new language and development platform as well. I am actually interested in Android Development as a career after working with Android Studio and Kotlin. The project as a whole just highlights the benefit of Software Development Methodologies and their usage within Industry. Time management is also something highlighted as it played a large role and shows just how important deadlines are within companies. This project provided great experience and knowledge that i know will help me within industry in the future.

As a Final Word, i would like to extend my thanks and gratitude to all of the lecturers at GMIT who have helped me in various ways throughout the last four years. It is thanks to the staff at GMIT that i can go forward into industry with a positive mindset and the ability to learn and adapt to what the industry requires of me. Finally i would like to thank my personal supervisor Martin Hynes and the external examiners who will be grading this project, i hope you find the documentation and code to be of rigorous academic standards.

Chapter 7

Appendices

Github Repository

https://github.com/DarrenRegan/Final-Year-Project

Github Kanban Board

https://github.com/DarrenRegan/Final-Year-Project/projects/1

Github Kotlin Code

https://github.com/DarrenRegan/Final-Year-Project/tree/master/app/src/main/java/com/example/ecommerce

Github XML Files

https://github.com/DarrenRegan/Final-Year-Project/tree/master/app/src/main/res/layout

Screencast

https://youtu.be/lY1Y_oAYKdc

References

[1] *Airbnb moving away from react native.* URL: https://medium.com/airbnb-engineering/react-native-at-airbnb-f95aa460be1c.

[2] *Anatomy of Android Continuous Delivery.* URL: https://medium.com/the-telegraph-engineering/android-continuous-delivery-fb41da63176.

[3] *Android Acivity - Fragments.* URL: https://developer.android.com/guide/components/fragments.html.

[4] *Android Jetpack.* URL: https://developer.android.com/jetpack.

[5] *Android Jetpack LifeCycles.* URL: https://developer.android.com/topic/libraries/architecture/lifecycle.

[6] *Android onyl supports Java 7 and a Subset of Java 8.* URL: https://developer.android.com/studio/write/java8-support.

[7] *Android Performance tips.* URL: https://developer.android.com/training/articles/perf-tips.html.

[8] *Android, Guide to app architecture.* URL: https://developer.android.com/jetpack/docs/guide.

[9] *AWS Pipeline CI/CD tool.* URL: https://aws.amazon.com/getting-started/projects/set-up-ci-cd-pipeline/.

[10] *Badges/Shield.io for Readme's.* URL: https://github.com/badges/shields.

[11] *CircleCI Continuous Delivery Software tool.* URL: https://circleci.com/docs/2.0/about-circleci.

[12] *CodeShip Continuous Delivery Software tool.* URL: https://documentation.codeship.com/basic/quickstart/getting-started.

[13] *Continuous Delivery Article.* URL: https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment.

[14] *Cordova, What is Hybrid App Development?* URL: https://cordova.apache.org/docs/en/latest/guide/overview/.

[15] *Feature-Driven Development.* URL: https://en.wikipedia.org/wiki/Feature-driven_development.

[16] *Feature-Driven Development compared to XP*. URL: http://www.featuredrivendevelopment.com/files/FDD_vs_XP.pdf.

[17] *Firebase*. URL: https://firebase.google.com/.

[18] *Firebase Android Documentation*. URL: https://firebase.google.com/docs/android/setup.

[19] *Firebase Android Google tutorial*. URL: https://codelabs.developers.google.com/codelabs/firebase-android/#0.

[20] *Firebase Authenication Docs*. URL: https://firebase.google.com/docs/auth.

[21] *Firebase Cloud Storage*. URL: https://firebase.google.com/docs/storage.

[22] *Firebase pricing*. URL: https://firebase.google.com/pricing.

[23] *Firebase Products*. URL: https://firebase.google.com/products.

[24] *Firebase Realtime Database Docs*. URL: https://firebase.google.com/docs/database.

[25] *Github Actions CI/CD tool*. URL: https://github.com/features/actions.

[26] *GitLab Continuous Delivery Software tool*. URL: https://docs.gitlab.com/ee/ci/.

[27] *Ionic, Return on Investment vs Native applications*. URL: https://ionicframework.com/resources/articles/roi-hybrid-vs-native.

[28] *Ionic, What is Hybrid App Development?* URL: https://ionicframework.com/resources/articles/what-is-hybrid-app-development.

[29] *Key Signing for Google App Store*. URL: https://developer.android.com/studio/publish/app-signing.

[30] *Minimum Viable Product Approach*. URL: https://link.springer.com/chapter/10.1007/978-3-319-33515-5_10.

[31] *Native, HTML5, or Hybrid: Understanding Your Mobile Application Development Options*. URL: https://developer.salesforce.com/index.php?title=Native,_HTML5,_or_Hybrid:_Understanding_Your_Mobile_Application_Development_Options&oldid=51601.

[32] *Paper Document ion and GitHub*. URL: https://github.com/pilgr/Paper.

[33] *Picasso Source Code/GitHub*. URL: https://github.com/square/picasso.

[34] *React Native article by a Airbnb, Google Android Developer*. URL: https://medium.com/airbnb-engineering/sunsetting-react-native-1868ba28e30a.

[35] *Real Challenges in Mobile App Development*. URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.724.2463&rep=rep1&type=pdf.

[36] *Single Activity: Use Android Jetpack to Accelerate Your App Development.* URL: https://android-developers.googleblog.com/2018/05/use-android-jetpack-to-accelerate-your.html?m=1.

[37] *Single Activity: Use Android Jetpack to Accelerate Your App Development.* URL: https://developer.android.com/reference/kotlin/android/app/Activity.

[38] *Single Activity: Why, When, and How (Android Dev Summit '18).* URL: https://www.youtube.com/watch?v=2k8x8V77CrU.

[39] *Smartphone Market Share.* URL: https://www.idc.com/promo/smartphone-market-share/os.

[40] *Technical Documentation in Software Development.* URL: https://www.altexsoft.com/blog/business/technical-documentation-in-software-development-types-best-practices-and-tools/.

[23] [17] [31] [1] [34] [7] [18] [20] [24] [21] [39] [19] [22] [33] [10] [15] [16] [40] [2] [13] [29] [26] [11] [12] [25] [9] [35] [30] [28] [14] [27] [6] [38] [36] [37] [3] [8] [32] [4] [5]