Cross-Platform Café Mobile App with Cloud Storage

Shawn Anuptra Martin

Undergraduate dissertation submitted in
partial fulfilment of the regulations governing
the award of the degree of
BSc (Hons) Web and Mobile Development
at the University of Sunderland
2023

Abstract

This project was inspired by the swift advancement of technology, specifically in the mobile industry. In a world where mobile devices are now considered a necessity, it will be a missed opportunity to not engage in the industry. For independent café owners, there are opportunities to grow using technology; a cross-platform mobile application, for example, which is the main point of discussion of this paper. In order to develop this project, research on the evaluation between native mobile applications and cross-platform mobile applications was initially carried out. The evaluation took into account the 'what' of each type of mobile application, and also the 'why'. There were critical findings found within this research relating to the difference in the user interfaces of mobile operating systems and the best practices for developing a usable user interface.

After the research was the analysis stage of the project. This was required to set appropriate goals in tandem with the deadlines. Technical analysis were done in order to pick the best tool for the job. It was decided that the app was to be developed using Flutter, integrated with Google Cloud to provide online functionality. While Flutter is a comparatively newer technology for cross-platform development, the research done had shown that it is a promising framework with longevity. However, due to its age, a large learning curve needed to be overcome to develop with Flutter. Designs for both the front-end and the back-end of the app was also carefully considered. Competitor analysis was done to see how more experienced teams designed their app. Figma was used to draw the UI mockups, draw.io was used to draw diagrams such as User Flow and ERDs.

During the development phase of the project, testing was also continuously done. While this slowed down the rate of development, it allowed the app to be continuously iterated and improved upon. Most of the testing done were functionality testing; promoting a low number of bugs with the previously planned features correctly implemented.

The project was finally evaluated and reanalysed. First, it was evaluated to the requirements and goals set from the start of the project. The evaluation discusses how well the project was done on all aspects: the analysis and design, the technology chosen for this project, the time management, and more. Most importantly, it also discusses the improvements that could be done for the future and how the project could be expanded, including the use of data analytics to gain insight on how users are using the app, hence allowing further improvements to the app overall.

Table of Contents

Cross-Platform Café Mobile App with Cloud Storage

1. Introduction

This document is a definitive brief for the CET300 project, titled “Cross-Platform Café Mobile App with Cloud Storage”. This project does not have a real-world client, however, it does aim to solve a real-world problem. This development of this project will be supervised by Paul Graham, a senior lecturer in the School of Computer Science at the University of Sunderland.

This document will be going into details of the CET300 project, starting with this Introduction chapter. The context of the problem will be explored in Chapter 2 of this document. This is where the origin of the problem will be explained while giving the necessary context on why this problem arises and how this problem can be solved. The problem statement of this project, “How independent cafés can use technology to compete in the market”, will also be formed in this chapter.

Before solving the problem, prior research will be needed to make the most informed decision on how to solve the problem; this will be done in Chapter 3 of this document. The research context in this chapter will investigate two different topics: which cross-platform technology is the best fit for the problem, and how to build a satisfactory user interface.

Chapter 4 will go through the solution composed for the problem. It is broken down into further subchapters, namely functional and non-functional requirements, hardware and software requirements, and the constraints of the project.

The procedure of this project will be discussed in Chapter 5. This is where all the potential social, ethical, professional, and legal issues for this project will be addressed. There will also be a subchapter discussing the chosen methodology of this project, Rapid Application Development, and how this methodology was chosen.

Chapter 6 will be where all progress on this project that had been done will be reported. This is all the progress that was made before the deadline of this definite brief on 18 November 2022.

The final part of this document is the references and the appendices. The reference will be a collection of all the sources cited in this document. The appendices will include the author’s Programme Route Diagram, the author’s current Project Schedule, evidence of Supervisory Meetings, any Analysis and Design Diagrams that have been created, and the slides that are going to be used for the presentation of this definite brief.

1.1. Problem Context

Due to the COVID-19 pandemic, small and medium size businesses are struggling with business uncertainty (Rakshit, Islam, Mondal, & Paul, 2021), including independent cafés. In order to survive, independent cafés will need to compete with chain cafés and other independent cafés. They will need to be able to attract new customers, retain current customers, and increasing the number of returning customers so that the cafés will not run at a loss.

Independent cafés will need to be more discoverable and relevant in their market (e.g. in their local area). It would also be advantageous for them to be able to provide the same, if not higher quality service compared to their peers or chain cafés. For example, providing a collection service and a loyalty reward scheme. While advertising in social media has proven to be effective in increasing awareness of a brand leading to an increase in sales (Dolega, Rowe, & Branagan, 2021), it lacks a more direct “call-to-action” path for potential customers to engage with the business.

Using a traditional method of calling the café to order a coffee can be tedious and time consuming. Not only does the variance in cell reception increases the difficulty of communication between the café and the customer, it also adds another layer of potential problem in the transaction. For example, the customer might have a stutter, or the café employee might not speak the same language as fluently.

A café can use printed cards, signed or stamped by employees to mark the number of points the customer have accumulated. However, this can pose several problems. Customers might have access to the printed cards, and fake the signature of the employees to get free rewards. They also might also forgot to bring their card to the café, and they will have to be given new cards to keep accumulating the points. The café will need to keep stock of cards to give out to customers. This will also have an impact on the environment, as a significant amount of papers are used and wasted when in a relatively short period of time.

Independent cafés can also make use of transaction data in order to improve their products. By analysing the data, they are able to see which of their products are most popular, the trends of their products, which products can be improved, etc.

1.2. Proposed Solution

The overall solution of the project is a cross-platform mobile app, available on both Android and iOS that features the functionalities mentioned in the Problem Context chapter, mainly the collection service and the loyaly reward scheme.

Developing a cross-platform app essentially cuts both the cost and the development time in half. Instead of developing two native apps, which costs twice as much due to requiring the services of 2 separate teams of developers, a single team of developers to develop the cross-platform app would be sufficient.

The app will include the loyalty programme functionality, collection service functionality, and other functionalities such as accounts and Shopping Cart to provide the highest quality of experience to customers using the app. All of the functionalities will be explored deeper in the next sections.

Cloud storage will also be used with this app. This will be used to securely store user data, and can be used to store transactional data to analyse and process.

1.3. Dissertation structure

This is going to be the dissertation structure. Will update later on

2. An Evaluation of Native vs Cross-Platform Mobile Development for Independent Café/Business Owners

2.1. Introduction

The Food & Beverage (F&B) industry faced a decline in sales during the pandemic. Based on a report, the UK cafés and coffee shops market size has significantly decreased in 2021 to only £4.1bn, compared to £7.1bn market size in 2020 (IBISWorld, 2022). Though the market has not fully recovered from the pandemic, the 2022 estimated market size of the cafés and coffee shops industry is £5.6bn – around a 36.1% increase in 2022 compared to 2021 (IBISWorld, 2022).

This means that the café industry is growing again, coming out of the pandemic. This presents an opportunity for independent cafés to grow and expand; an opportunity for new independent cafés to enter the market and compete with existing coffee shops.

However, this raises the question – how can an independent café compete in the growing market? Following the trend will certainly help; since 2019, 521% more coffee sellers now offer a collection service (Specialty Coffee Association, 2020), where customers can order their drinks for collection at a later time. In spring 2021, National Coffee Association released a report of data trends for the US, which shows an increase in the number of people who ordered coffee through an app by 30% since January 2020.

The use of loyalty programmes, proven by branded coffee shops, will also help sales. Loyalty programmes give customers an incentive to become return customers. For example, Costa's loyalty programme, Costa Coffee Club, which is integrated into the Costa Coffee mobile app, generates one-third of Costa's sales. (Grochowska, 2022). A research done by ___ also found that implementing a loyalty programme, customer retention increases. ___ also proved that customer satisfaction does not play a mediating role between the loyalty programme and customer retention. In other words, even if customers are not very satisfied with the loyalty program, it does not necessarily mean they will stop being loyal customers. The loyalty programme itself is what keeps the customer coming back to the business, regardless of how satisfied they are with the programme. However, it is to note that the study is conducted on Malaysian national car sector, not in the F&B industry.

A custom mobile app is able to package the two above factors into an accessible medium where customers can access. Stocchi *et al.* (2021), concluded in a market research on mobile apps that "apps can enhance consumer perceptions of value from the early stages of the customer journey". An enhanced perceived value of a business increases customer retention (Dube & Renaghan, 2000; Liu, Petruzzi, & Sudharshan, 2007, cited in Hanaysha, 2018).

2.2. Native Mobile Development

Based on the statistics compiled by Statcounter GlobalStats, the market is dominated by the Android mobile operating system with over 70% of the marketshare, then iOS with 27.6% of the marketshare (Statcounter GlobalStats, 2023). With the rest of the mobile operating systems marketshare totalling to only 0.6%, discussing native mobile development in this paper will only cover the Android OS by Google and iOS by Apple.

Native mobile development is the development of apps that are specifically developed for a particular mobile operating system (Singh, 2023). In other words, native mobile apps are apps that are only designed for either Android using Java or Kotlin, or iOS using Swift or Objective-C.

The main advantage of a native mobile app is that it is built in accordance with guidelines and specifications provided by each mobile operating system (White, 2013). For the user interface design aspect of the app, Android apps will adhere to Material Design Guidelines (Google, 2023), and iOS apps will adhere to the Human Interface Guidelines (Apple, 2023). This will provide a more consistent experience between apps in each operating system. The components of the user interface such as the buttons and how the app handles navigation will be consistent and similar between apps, providing a better user experience for users.

This is due to the lesser need to 'learn' the app everytime a user uses a new app; stated by Jakob's Law (Nielsen, 2000).

Another advantage of a native mobile app is its performance compared to cross-platform apps or hybrid apps. Based on a test-based research study, an Android app developed with React Native generally has a worse CPU performance than a natively-coded Kotlin Android app (Svensson & Käld, 2021). Accessing a device's native hardware will also benefit the native app, as the native code will have access to the device's native hardware, while the non-native code have to access the device hardware one degree outside its shell. This is proven in the same study by the shown higher percentage of CPU load in the React Native app when loading the GPS compared to the native app. However, there are some conflict in the literature with a study conducted by (Biørn-Hansen, Rieger, Grønli, Majchrzak, & Ghinea, 2020) that shows that native code sometimes are outperformed by non-native code. The result is based on TTC (Time To Completion) in accessing Geolocation of the hardware. This study however covers each task (accessing accelerometer, contacts, file systems, and geolocation) individually, and does not put the task in context (e.g. in an actual app with background tasks). This does not reflect a realistic real-life scenario, and results may vary when a more realistic and practical approach is taken.

Native mobile apps are also perceived to be less buggy (Weichelt, Heimonen, Pilz, Yoder, & Bendixsen, 2019). This can be attributed to only using a single codebase to develop and compile an app for a single operating system (Singh, 2023) – there will be no reliance on other cross-platform tools, thus eliminating a source of error.

It is important to note that there are some situations where native app development do not shine. For instance, if a business wants to develop native apps for both Android and iOS, it will be costly. This is due to the business having to pay twice as many developers; 2 teams for each operating system, versus one for a cross-platform app (Shevtsiv & Striuk, 2020).

Having 2 different teams for each operating system working on the same app also surfaces a communication challenge (Shevtsiv & Striuk, 2020). The app needs to feature the same functionalities in both platforms, and synchronizing this and update dates will be more difficult. The two developer teams may also disagree on how to implement changes and/or functionalities, leading to a longer overall TTM (Time to Market).

2.3. Mobile Cross-Platform Development

In contrast to native mobile development, mobile cross-platform development (MCPD) uses a single codebase to develop and compile apps for multiple operating systems (Biørn-Hansen, Rieger, Grønli, Majchrzak, & Ghinea, 2020). In this case, for both Android and iOS.

According to Biørn-Hansen et al., MCPD can be categorised into two main categories: runtime environment and generative approaches. Runtime environment can be further broken down into 3 more categories: web, hybrid, and self-contained runtime. Generative approaches can be broken down into MDSD (model-driven software development) and transpiling.

However, this classification is not universal. MCPD can also only be categorised into 4 approaches: web, hybrid, interpreted, and cross-compiled (Huber, Demetz, & Felderer, 2022).

Since MDSD is not prevalent outside of academia (Biørn-Hansen, Rieger, Grønli, Majchrzak, & Ghinea, 2020), this paper will not discuss this category of MCPD.

After further reading of both of these classifications, it is clear that they are both one and the same (outside of MDSD), just with difference in the name. Self-contained runtime apps are the same as interpreted apps; same with transpiled apps and cross-compiled apps.

Moving forward, this paper will use the categorisation of MCPD based on Huber et al.

2.3.1. Hybrid approach

Hybrid approaches use both native and web technologies to build the app, hence the term. The app has a web-based core, developed in web development language, namely HyperText Markup Language (HTML), Cascading Style Sheets (CSS), and JavaScript (JS). The core will then be wrapped in a *native app wrap*per to allow the app to have access to the device's features such as the camera. Frameworks such as Ionic Capacitor or Apache Cordova develop apps with this MCPD approach (Kotlin, 2022).

The disadvantage of using this hybrid approach is the potential for a decrease in app performance (Gonsalves, 2018). The app will also lack a native navigation feel (Gonsalves, 2018), and developing an intuitive user experience (UX) and navigation pattern would be challenging (Kidecha, 2022).

2.3.2. Interpreted approach

Interpreted approaches use web technology such as CSS and JavaScript, with native user interface (UI) interpreted during the app runtime. The developed app has an integrated interpreter to generate the native UI (Huber, Demetz, & Felderer, 2022). Apps developed with React Native are examples of this approach (Biørn-Hansen, Rieger, Grønli, Majchrzak, & Ghinea, 2020).

The apps developed with this approach can feel and perform like a native app, however, due to the integrated interpreter, would need a larger memory bandwidth (Gonsalves, 2018).

2.3.3. Cross-compiled approach

Cross-compiled approaches make use of a source code that is developed in a single programming language that will be compiled into a native app with its native programming language. It is an app that is designed to run on different operating systems (Kotlin, 2022), such as Android and iOS. Apps developed with the Flutter framework are developed with this approach (Biørn-Hansen, Rieger, Grønli, Majchrzak, & Ghinea, 2020). However, there is a discrepancy in the literature as Flutter has been categorised in the interpreted approach before (Gonsalves, 2018).

Developers would still need to implement some native code to get full access to the device's hardware such as fingerprint sensors (Kidecha, 2022). It will also be challenging to make the app perfect in every operating system, as developers will need to be aware of minor differences between the operating systems when implementing complex functionalities (Singh, 2021).

3.1.4. Web Approach

The web approach makes use of web technologies to develop the app. They are built to be used with mobile web browsers. A Progressive Web App (PWA) is an example of this approach (Huber, Demetz, & Felderer, 2022).

A PWA is a web app that uses web platform technologies and features (MDN Web Docs, 2022). A regular website or web app can be turned into a PWA just by including the three foundations of a PWA: a secure connection (HTTPS), a service worker, and a manifest file (Toonen, 2020). PWAs aim to provide a native app experience, while simultaneously still being able to be accessed through a web browser; they will still have a URL that users can navigate to and share. PWAs are available for offline use and can use push notifications to encourage users to return.

The main drawback of PWAs is their compatibility with different browsers. For example, in iOS, only Safari allows the installation of PWAs (Firtman, Andrew, Jara, LePage, & Medley, n.d.). At the time of writing, Safari also does not support push notifications – Apple announced in June 2022 that this feature will only be supported in 2023 (Khandelwal, 2022).

Access to native features of the device is also limited for PWAs by the web browser it is run in (Biørn-Hansen, Rieger, Grønli, Majchrzak, & Ghinea, 2020). If the web browser does not

allow the access to contact details or file management system of the device, the PWA could not access them.

2.4. Performance

Overall, the performance of a natively developed mobile application is faster than an application developed for cross-platform use. This is proven by a comparative test study of performance between a native Android and iOS apps and a Xamarin app (Grzmil, Skublewska-Paszkowska, Łukasik, & Smołka, 2017). In this study, the native app performed better than the Xamarin app in three tests, including calculations, file saves, and time to location fix. The two apps have insignificant performance difference in using the internet connection to download files. However, in the file-reading test, the Android app performs the worst, followed by the Xamarin app and the iOS app. This outlier might be caused by an unoptimized code developed in the native Android app.

This fact is also supported by other studies, such as in a study done by Barros, Medeiros, Moares, & Junior (2020), where it is stated that native applications are faster than cross-platform apps. A study comparing native Android app with Flutter and React Native app also further supports this statement (Mahendra & Anggorojati, 2020).

Comparing native and cross-platform apps speed in performing logical operations, such as calculations and CRUD operations, is a valid way to compare the performance between the two apps. However, it is also important to compare the apps from the perspective of the user: the user experience.

To be able to determine user experience quantitavely, comparisons can be done on several metrics, including: framerate, CPU usage, RAM usage, and GPU usage (Biørn-Hansen, Grønli, & Ghinea, 2019). Other metrics, such as response time, can also be included (Mahendra & Anggorojati, 2020). It is important to note that framerate is most likely to be the most important metric. This is due to the fact that most mobile devices have a screen refresh rate of 60Hz, and if the application refreshes in less than 60 Hz, the app will be noticeably “janky” (Lewis, 2018), a term used to describe a poor quality interface.

On average, native applications are able to produce more framerates with less resources (less CPU, RAM, and GPU usage) (Mahendra & Anggorojati, 2020). Although, there are instances where this is not the case. An example would be a Flutter app being able to render 66 frames per second (FPS), while a native Android app only rendering 58 FPS (Hussain, Khan, Farooqui, Arain, & Siddiqui, 2021). Another example is in a study done by Biørn-Hansen, Grønli, and Ghinea (2019), where the native implementation has higher CPU

and RAM usage than their cross-platform counterparts, albeit with lower number of *jank* frames and GPU usage.

Mahendra and Anggorojati (2020) test results in their study shows that response time of native applications is faster than cross-platform applications when opening the camera. This supports the statement that native apps overall has a better user experience as native applications run smoother than cross-platform apps.

2.4. Size

Cross-platform apps have a bigger bundle size in comparison to native applications. Mahendra & Anggorojati (2020) found that when installed, Flutter apps can be around 10x larger than an identical native Android app, while React Native apps can be around 7.5x larger. This is due to the cross-platform apps requiring additional components to run the apps, i.e. the JavaScript bridge needed in React Native. However, this comparison study is done on emulators instead of real devices, which might lead to different results when done on real Android devices. This study also did not mention whether the cross-platform apps are compiled in debug or release mode. Apps in debug mode contain debugging overload to aid in debugging the app during development, leading to an inaccurate, larger than normal app bundle size (Flutter, n.d.).

This statement is also supported by a study done by Jia, Ebone, and Tan (2018), showing that native Android and iOS apps are significantly smaller in size than their cross-platform counterparts, developed with Xamarin, Cordova, and Titanium.

2.5. Which one for independent business owners?

To answer this question, it is important to define what constitutes as an independent business. Based on Women's Economic Council of Canada,

"Independent businesses are defined by their ownership structure, for the most part. They are owned, not by a larger corporation or by shareholders, but by private owners."

In other words, it is a business with no outside influence; they do not have public shares or shareholders that have an influence in how the business is run.

The important part is independent businesses are typically small in size – usually fewer than 100 employees, and usually with less than $10M in annual revenue (Shopify, 2022). Due to the small size and revenue, most independent businesses do not have a large capital to pour its resources into developing an app in order to grow.

Based on the previous subchapters on performance differences between native mobile applications and cross-platform mobile applications, independent business owners might choose to build a native app over a cross-platform app. However, it is important to factor in more business-oriented reasons in order to make an informed business decision.

While native mobile apps on average is superior to cross-platform apps in terms of performance and user experience, it might be more costly in both monetary and time cost. As mentioned briefly in 2.2, developing native apps for both Android and iOS concurrently is very costly to the business. Two separate teams, one for each operating system, needs to be hired in order to develop the apps. This will impact the TTM for these apps, especially if both apps are required to be released simultaneously. Updating and maintaining two different codebases will require a significant amount of resources (Schmitt, 2022). Independent business owners might need to reconsider putting a large amount of capital to developing two different apps; valuable amount of capital that can be used to other resources, such as hiring staff, venue cost, cost of raw materials, etc.

Independent businesses might also want a faster time-to-market (TTM), in order for the app to quickly get quality reviews from customers and to be improved upon. A faster TTM allows a business to gain competitive edge over other businesses by beating the competition to the market (in this case, other independent café owners) (Carter, 2023). For this purpose, cross-platform apps has an advantage over native apps. This is due to the relative speed of building a cross-platform app compared to two independent Android and iOS apps (Schmitt, 2022).

While the business-oriented factors lean towards adopting a cross-platform mobile app, is the performance trade off worth the faster TTM and less capital needed? As mentioned previously, native apps are more performant than cross-platform apps. However, in a real-world scenario, the difference is minute.

In conclusion, if the goal of an independent business owner is to release a mobile app that will have the most reach, they will have to release both an Android and an iOS app. The most efficient way to do this is by developing a cross-platform app. Not only does cross-platform application allow faster TTM, but it also has a lower cost-to-reach ratio, compared to developing two different mobile apps for the two major mobile operating systems.

3. Requirements Analysis

This chapter will discuss the requirements analysis done for this project. This ranges from functional and non-functional requirements of the project itself, the software and hardware requirements, and the constraints put on this project.

3.1 Functional Requirements

For the app to work, the functional requirements of the app are as follows:

1. Users will be able to log in and log out of the account in the app
2. Users will be able to make and edit their account, including:
    a. Login details
        i. Email/password
        ii. Mobile phone number
    b. Displayed name
3. Users will be able to delete their account
    a. This is a requirement based on GDPR rules
4. The app will display a menu for the customer to choose from
5. Shopping Cart functionality:
    a. Users can add products to the Cart
    b. Users can remove products from the Cart
    c. Shopping cart contents will be saved when the app is closed, and can still be accessed when the app is reopened
6. Users will be able to check out their order
    a. Users will be able to choose what time they are going to pick up their order
    b. Every order will have a unique order ID for verification in store
7. Users will be able to cancel their order within a time limit
8. Users will be able to see their order history
9. Users will have access to a loyalty programme
    a. Users will get a ‘stamp’ for every drink purchased
    b. Users will be able to get a free drink after 8 ‘stamps’
    c. Users will have access to a free drink voucher on their birthday week

There are some limitations in this app that disallows the app to be production-ready. For example, a fully functional payment functionality. This will be discussed further in a later subchapter, 3.5 Constraints.

3.2 Non-Functional Requirements

The non-functional requirements of the app are as follows:

1. Cold launches of the app should take less than 5 seconds
2. The app should smoothly run on 60fps, with no frame losses
3. The UI of the app should be intuitive
4. The UI of the app should be colour-blind accessible
5. The use of skeleton loading in the UI to inform the user that the app is still loading, instead of not working completely when there are any long load times
6. Signing in and out should take less than 3 seconds on a 4G LTE connection

The purpose of these requirements is for a satisfactory and enjoyable experience for the users. According to Google (Android Developer Tools, 2019), if an app cold launches longer than 5 seconds, the performance of the app is considered concerning.

3.3 Software Requirements

The development of this app can be split into two parts: the programming of the app itself, and the designing of the app.

For the programming side of the development process, these software applications will be needed:

1. Android Studio as the IDE (Integrated Development Environment)
2. Google Cloud as the backend provider of this project. Google Cloud offers Firebase, which aids in user authentication and security, and Firestore, a NoSQL based database.

For the designing side of the development process, these software applications were needed:

1. Figma
    a. Used for designing and rapid prototyping of the UI
    b. Moodboarding of the UI
2. Draw.io
    a. Used for drawing diagrams and flowcharts

3.4 Hardware Requirements

The hardware requirements to develop, test, and run the app will include:

1. A Windows PC complete with a keyboard, mouse, and monitor. Based on Android Studio, the minimum requirements for the PC are:
    a. Operating System: 64-bit Windows 8/10/11
    b. CPU: x86_64 CPU architecture; 2nd generation Intel Core or newer, or AMD CPU with support for a Windows Hypervisor
    c. RAM: 8 GB or more
    d. Disk Space: 8 GB minimum (IDE + Android SDK + Android Emulator)
    e. Screen resolution: 1280 x 800
2. An Android phone to test the app on a real Android mobile phone

3.5 Constraints

In order to complete this project and produce an app with the highest quality, it was important to put constraints on the project scope to prevent bloating.

The app does not include actual payment processes. This is due to the complications of setting up an online payment service, the potential cost, and the legal issues that may come with it. This project was not done with a real client, nor does it contain any real monetary transactions.

Since no real payments are being processed, there will also not be any real verification of the orders. There will not be an external app that a store employee would normally have to accept and verify orders, as developing an external app would not be feasible in the given time frame.

Another constraint is that no testing was done for iOS. This was due to the limitation of the author not having any direct access to any macOS computers, which were essential requirements in order to develop for the iOS platform.

4. Design

This chapter will discuss the design phase of the development of this project. It starts with a competitor analysis for a more informed and educated design decisions in the UI design aspect.

4.1. Competitor Analysis

Using and analysing the competition products gives a better understanding on what the competitor's strengths and weaknesses are (Lloyd, 2021). This allows business owners the

potential to exploit the competition's weaknesses and provide a service that fills the gap in the market. This also allows business owners to ensure the service they provide are up to market standard, by making sure their service has similar strengths to their competition.

In this section, analysis will be conducted on 3 different café apps available in the Google PlayStore: Starbucks UK, Costa Coffee Club, and Trading Post Coffee. The analysis will be done on the Android platform and discusses navigation, features, and overall aesthetic choices of the app.

4.1.1. Starbucks UK

The Starbucks UK app's security policy disallows screenshots to be taken while the app is open. Hence, the analysis done will not be supported by any screenshots of the app itself.

The first screen after the splash screen, which will be referred as the Welcome Screen, contains a brief explanation on how the loyalty reward scheme works, a Log In Button, and a Register Account Button. The Log In Button navigates to a Log In Screen, and the Register Account Button to a Sign Up Screen. When a user successfully logs in or registers an account, both screens will redirect to a Home Screen.

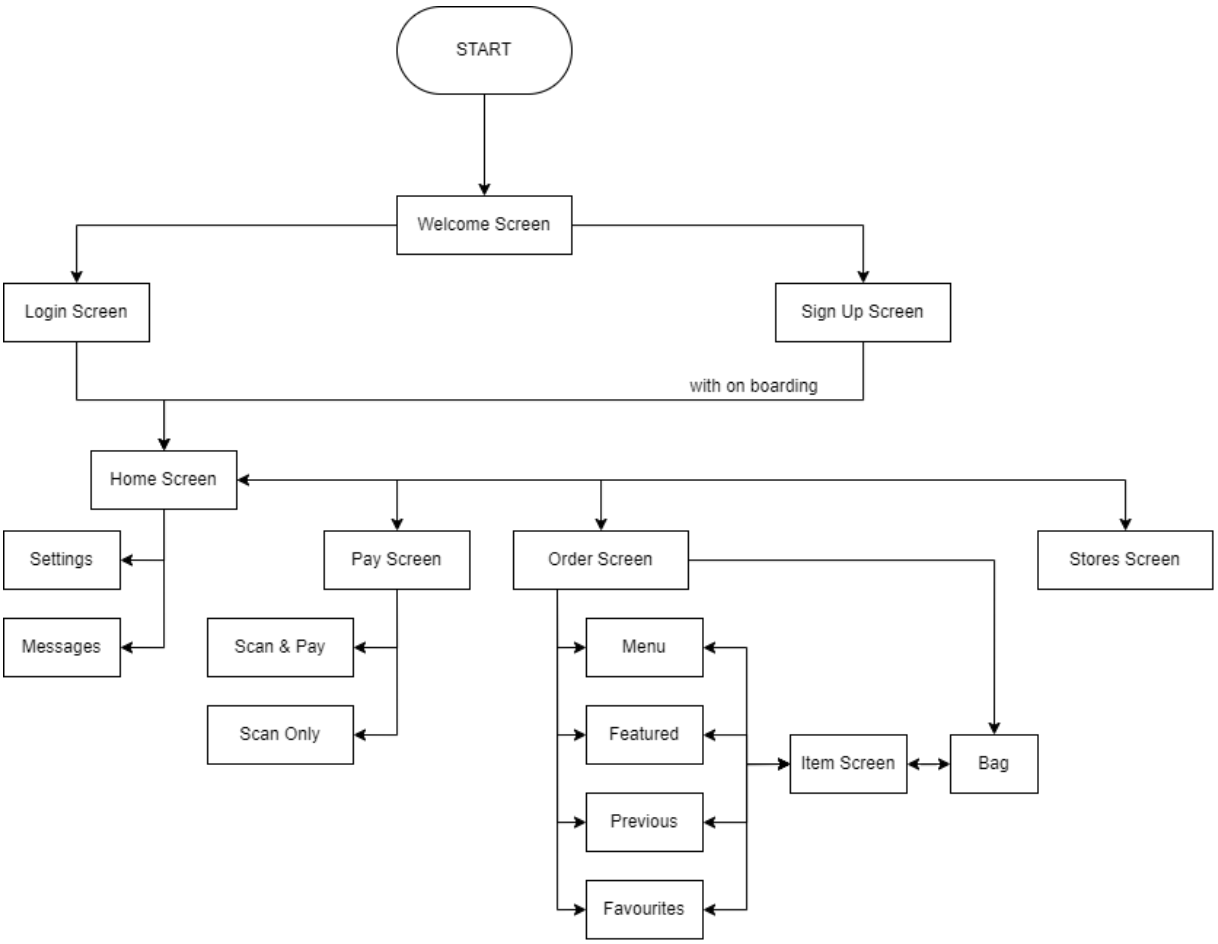


Figure 4.1. Starbucks screen flow diagram.

Once the user is authenticated and redirected to the Home Screen, there is a Bottom Navigation Bar (BotNavBar) with 4 options: Home, Pay, Order, and Stores. Home Screen contains the loyalty scheme explanation present previously in the Welcome Screen, the Reward Stars count of the user, the currently Featured Drinks, and the available stores nearby.

The Pay Screen contains 2 tabs: Scan & Pay and Scan Only. Scan & Pay contains a starbucks card bardcode that users can pay with, while the Scan Only contains only QR Code which will be scanned to add Stars to the account. Scan & Pay is used when users have or want to register a Starbucks Card to pay with, while Scan Only is used when users want to pay via other payment methods, such as contactless or cash.

When first opened, if there is no selected store, the Order Screen automatically redirects to the Stores tab to prompt user to select a store to order from. Once user selects a store, it will redirect back to the Order Screen. The Order Screen contains 4 tabs: Menu, Featured, Previous, and Favourites. The Order Screen also has a Snackbar, showing the selected store and number of items in the Bag. The Menu tab features all of the items categories, and

will open all of the items in the categories when the user clicks. When an item is selected, it will redirect to an Item Screen where the user can customise their selected drink, add to their Favourites, or Add item to the Bag.

The Featured tab contains Featured Drinks, Food, Coffee and Merchandise. The Previous tab contains the previously placed orders of the user, and the Favourites tab contains the items users have marked as Favourite.

The Stores Screen uses the location of the user, if the user allows, to find and show the nearest store available. If access to location is not allowed, the user will have to find their nearest store in the map themself.

Settings can be accessed through the Home Screen. Users will need to access the Settings Screen if they want to change account details, or sign out of the app.

Notable features of the app include: Onboarding process when users first register an account, searching the menu, and marking items as Favourite.

The app is developed with the Starbucks brand colours: green and gold as highlights, and using the neutral colours, black and white for the majority of the app. Important information, such as Call-To-Action (CTA) Buttons and active tabs are in green.

The app makes use of the same font throughout, which gives the app consistency. However, it also uses different font types and sizes, allowing a clear visual hierarchy to the UI. The app is filled with animations such as fading and sliding in and out of different screens, giving the app a smooth feeling when navigated around.

There could be improvements in the app, such as the inconsistent padding and spacing around elements in certain screens such as the Item Screen. There are also inconsistent alignment of icons and text. The “No favourites yet” message in Order>Favourites Screen is centre aligned, while “No previous orders yet” message in Order>Previous Screen is left aligned.

4.1.2. Costa Coffee Club

The Welcome Screen contains a very simple loyalty reward scheme description, “BUY 8 GET 1 FREE”, a Register Button and Log In Button. The Register Button leads to a Sign Up form, and the Log In Button leads to a Log In form. Once successfully authenticated, users are directed to the Home Screen.

After successful user authentication, the BotNavBar contains 5 tabs: Home, Order, Stores, Rewards, and Settings.

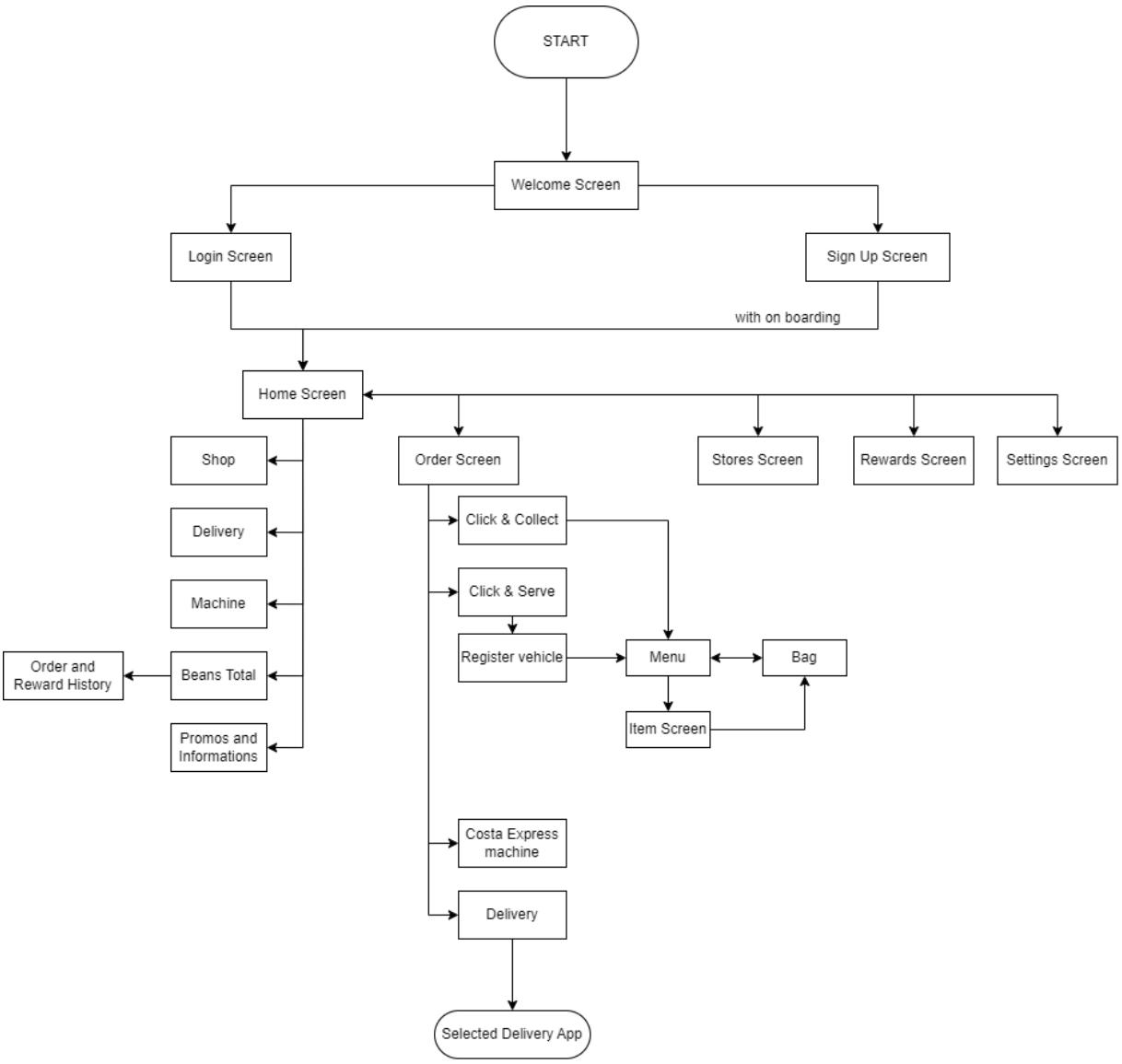


Figure 4.2. Costa screen flow diagram.

At the top of Home Screen are 3 buttons, namely Shop, Delivery, and Machine. This is used to gain Beans (points) for the loyalty rewards scheme for in shop purchase, delivery purchase, and in Costa Express machines respectively. Below that are cards showing deals, total Beans for user, promotions, and other information such as referral code, recycling information, and information about the free-range eggs used in Costa.

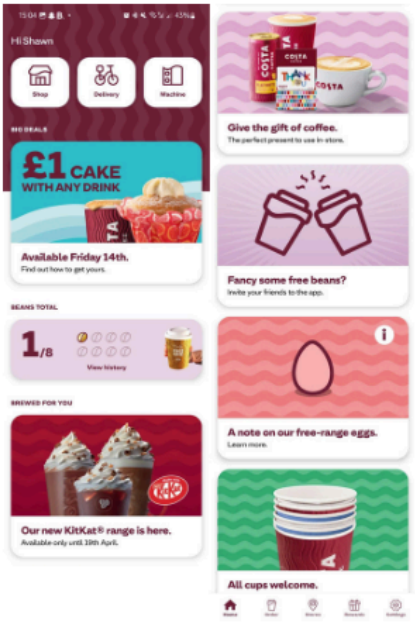


Figure 4.3. Costa Home tab, displayed in two separate images.

In the Order Screen, there are 4 different ways for a user to order. Click & Collect, Click & Serve, Costa Express machine, and Delivery. Each of this button will redirect user to the respective Screen.

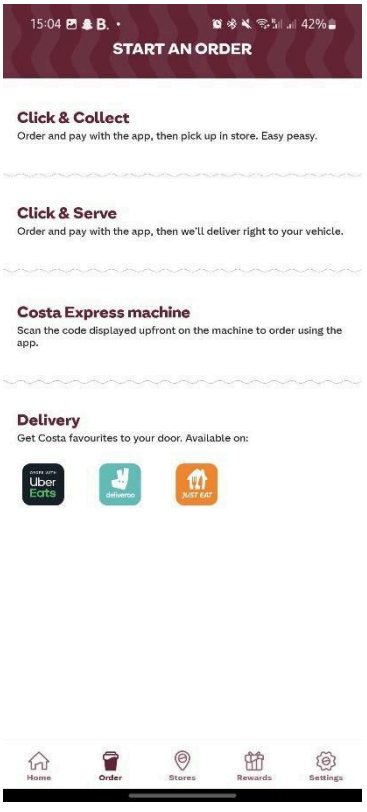


Figure 4.4. Costa Order tab.

Selecting Click & Collect will prompt the user to select a nearby store, and show the menu of available items. Clicking on the item will redirect to Item Screen where user can customise their order, and add to order. The Menu Screen contains the currently selected store and number of items in Bag.

Figure 4.5. Costa Menu screen.

Stores Screen will show the current location of user, and the stores available sorted by the nearest to user. Rewards Screen prompts user to input a Reward Code, such as a referral code from another user, to submit and claim rewards. Settings Screen is used for users to change account information, accessing support and legal information for Costa, and to sign out.

Notable features of the Costa app includes: onboarding process when user first registers an account and referral codes.

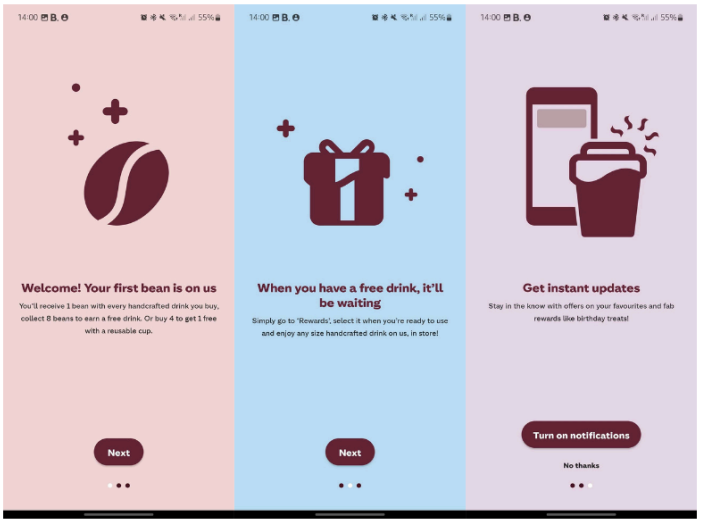


Figure 4.6. Onboarding process when user is first successfully authenticated.

The Costa app is developed with the Costa colours; Red is used for highlights, accent colours, and title colours, with white background and black text. The app mostly uses the same font throughout, with difference in thickness, size, and colour to provide visual hierarchy. It also contains a large number of images to provide more context to the text.

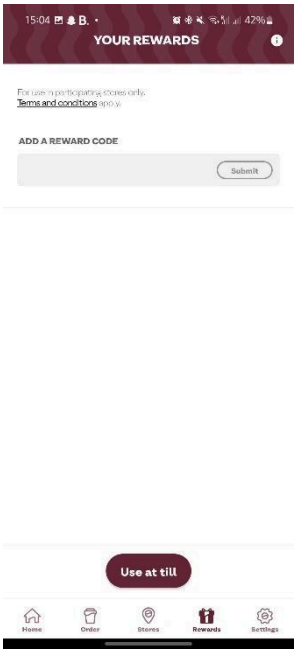


Figure 4.7. Costa Rewards tab.

The main improvement of the app is to reduce the size of cards to allow more content to be visible in less screen space. This reduces the amount of scrolling the users need. While the app has more features than the Starbucks app, such as 3 different ways to order, this might be too much for first time users. Having to navigate to more than 2 layers of screen to order items is a layer of friction that can be reduced.

4.1.3. Trading Post Coffee

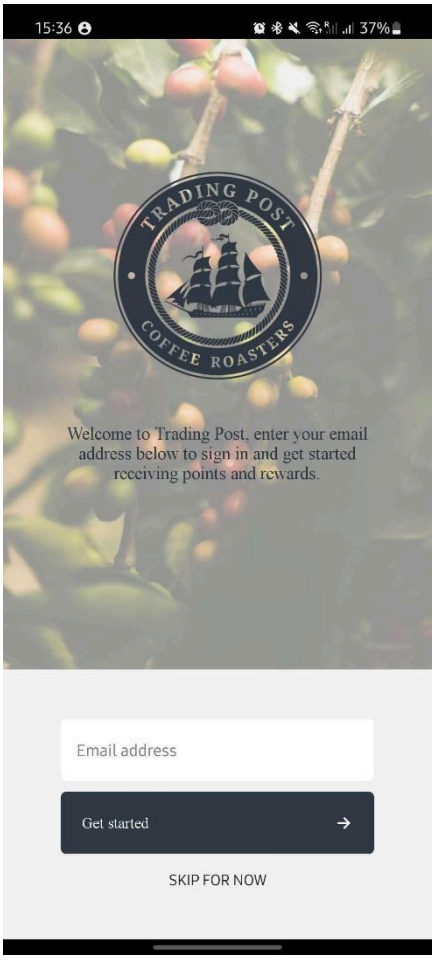


Figure 4.8. Trading Post Welcome screen.

The Welcome Screen prompts a user's email to sign in/sign up, with a "SKIP FOR NOW" button if a user does not want to sign in to use the app. When that button is clicked, a WebView modal will appear with the list of stores. Users can see what stores are serving which drinks. However, when trying to access the Bag, a login form is prompted.

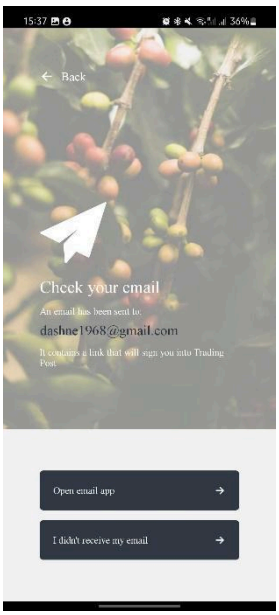


Figure 4.9. Check Email screen.

Once a user has input their email, the screen will prompt the user to check their email, and find a link to be used to sign in. This CheckEmail Screen has 2 buttons, one to open the email app, and the other to resend the email.

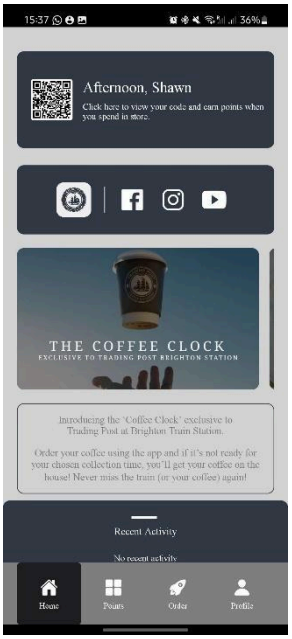


Figure 4.10. Trading Post Home screen.

Once the login link is accessed through the email, it will automatically redirect to the Home Screen with the user state signed in. The app has a BotNavBar with 4 tabs: Home, Points, Order, and Profile. The Home Screen contains the QR Code used to earn points for the

loyalty reward scheme, links to all their social media, and a brief explanation on “Coffee Clock”, which is a collection service with specified time.

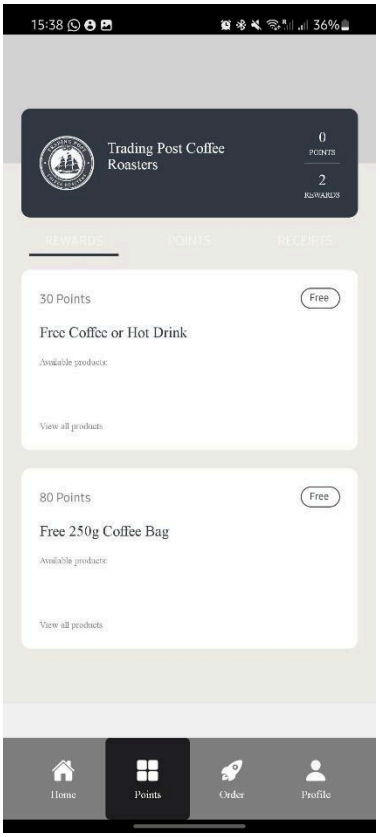


Figure 4.11. Trading Post Points screen.

The Points Screen contains the user’s current points and rewards and 3 tabs: Rewards, Points, and Receipts. The Rewards Tab shows available products to claim with sufficient rewards. The Points Tab shows available products to claim with sufficient Points, and Receipts shows previous orders a user has completed.

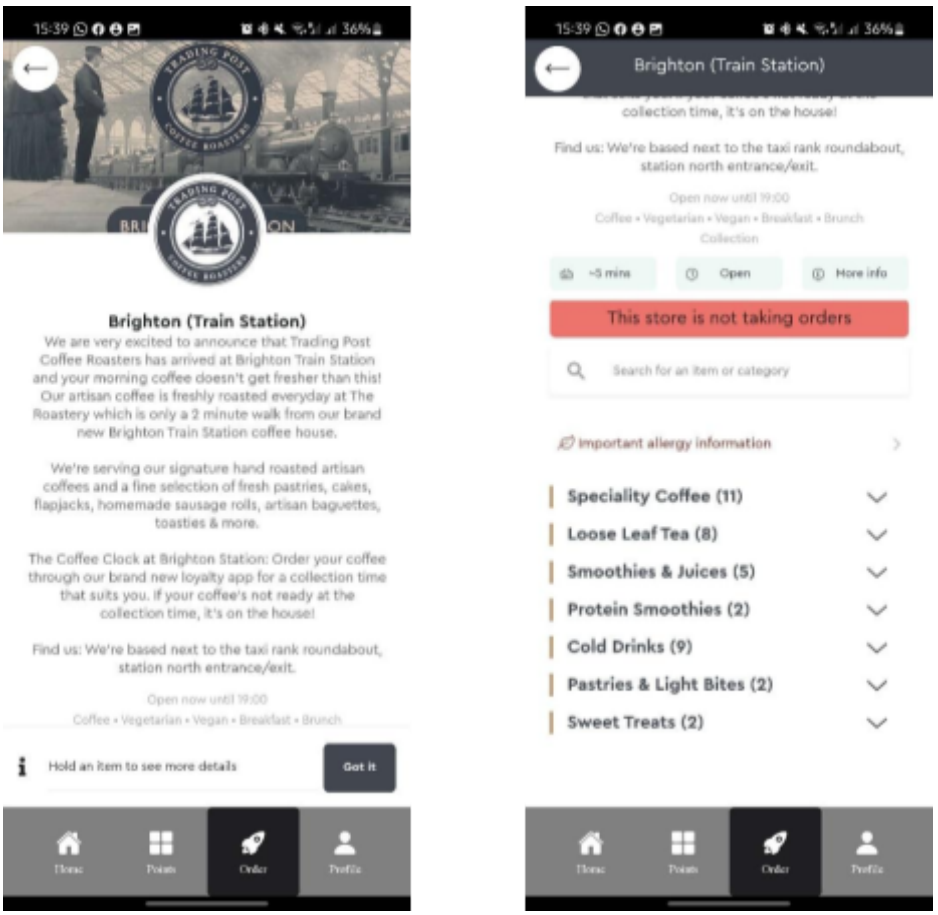


Figure 4.12. Trading Post Order tab when a store is selected.

The Order Tab shows different stores of Trading Post Coffee Roasters, and their status: Open or Closed, and if available, waiting time. When clicked on one store, the Store Screen shows the full description of the store and the available items in the store. When first accessed the Order Tab, or if user does not have a name or phone number, the app will prompt the user to go to Edit Profile and fill in the information.

The Profile Screen allows user to edit account information, provide feedback, and log out.

Unfortunately, during analysis, the Trading Post Coffee Roasters have momentarily stopped this service unannounced, hence it is not possible to conduct analysis on the flow of ordering items through the app.

Notable features of the app include: Email link sign in, timed collection service at chosen stores.

The app is developed with the brand's colour scheme, navy blue and white. The use of serif font gives the app a feeling of sophistication, of being formal. The lack of animation makes the app feel snappy.

There are numerous significant improvements this app can make. While the colour scheme is consistent throughout the app, there are instances where text are illegible. A more understandable language will also improve the UX if applied to the app (e.g. there are no explanations on what the difference between Rewards and Points are). The use of email sign in link instead of the standard account registration and sign in is also distracting; it takes the user out of the app to open their email.

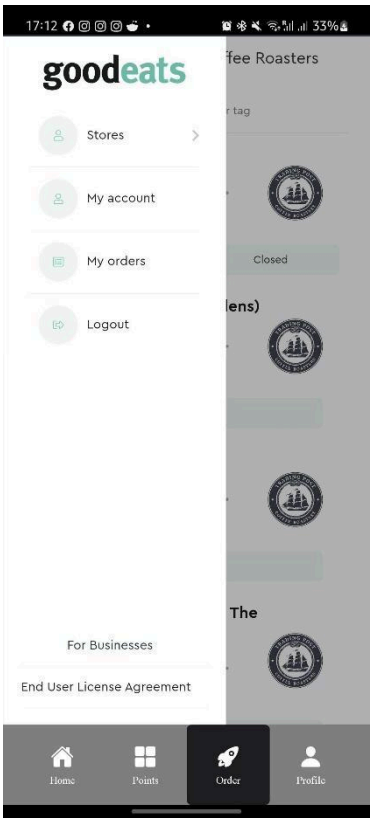


Figure 4.13. GoodEats WebView powering the Order tab in the Trading Post app.

The distracting UX is amplified further with the use of WebViews inside the app. The Order Screen uses a WebView powered by GoodEats. The colour scheme of this screen is different than the rest of the app, as GoodEats uses lime green as accents, compared to the navy blue of the rest of the app. Another factor why using WebView is discouraged is an issue with navigation. While navigating in the Order Screen, when pressing the back button in Android, it does not go back from the previous page in the Order Screen, but it goes back to the previous navigation inside the app (e.g. Home -> Order -> clicking through items in the WebView -> Back button is clicked -> Home).

4.1.4. Conclusion

Costa Coffee and Starbucks are in the top 3 leading coffee shop chains in the UK (Statista, 2022), hence their mobile apps are chosen for analysis to see what the top competitors in the

café industry offers. Trading Post Coffee Roasters app was chosen because it is an independent café with a custom mobile app available in Google Playstore. Analysing an independent café app gives insight to what independent cafés are competing with at a smaller scale. By analysing all three apps, an educated conclusion can be made on what an independent café should include in their custom app to provide a better user experience and service compared to their competition.

Based on the analysis above, it is essential for the app to not include any WebView for a better user experience. The app should also not be bloated like Costa, but instead focusing on fewer features executed well. It will have better layout consistency than both Starbucks and Trading Post app, with legible text and clear visual hierarchy. The app will have consistent colour scheme, appropriate with the branding of the café. A more comprehensive description of this project's app design will be discussed in the next part.

4.2. Design Choices of the App

The design choices made will be based on the author's creative decisions and competitor analysis from the previous part. This subchapter will discuss all the front-end design decisions of the app.

4.2.1. Branding

In order to apply the café's branding, a faux café is constructed. The café will be called CameoCoffee; a recently opened artisan café with only a single location. The café is based in the UK and is looking to further improve their service to customers by making a custom mobile app.

The logo of the café is a single line in the shape of a coffee bean. This symbolises the continuous demand of superior coffee that is supplied by CameoCoffee. The gradient applied to the logo is a parallel to the mission of providing great coffee to everyone from any age, any background, with no discrimination – no segmented lines in the logo and no distinct colour separation.

Figure 4.14. CameoCoffee logo.

The fontface chosen for the logo is a serif font, "DM Serif Display", which evokes a feeling of professionalism and seriousness, the approach taken from CameoCoffee to their products.

Serif fonts also suggests the weight of experience and history. In combination with the minimalistic logo, this showcases that CameoCoffee is a modern café that aims to provide the best quality coffee, building on top of the experience of other cafés in the history of coffee. The font colours are brown and gold, relating to their products of coffee and excellence. The distinct colourations of the two words, Cameo and Coffee, is for the purpose of legibility.

In order to reflect the brand, the app colour scheme will revolve around the brown and gold colouration.

4.2.2. Navigation

The app's navigation is going to be build on the analysis done on the previous part. It will mostly be based on the navigation pattern of the StarbucksUK app, with features and characteristics from Costa and Trading Post apps incorporated.

The first major change is the lack of a Stores Screen. This is due to CameoCoffee only having a single location. If in the future CameoCoffee wants to expand to different locations, this feature can be added into the app. This reduces the number of screens users will need to go through if they are navigating the app; reduces clutter.

The navigation overall on this app was simple. Unlike both Starbucks and Costa with multiple 'layers' of screen users have to navigate through, CameoCoffee only has 2 layers at most: the Item Screen whenever a user taps on an item on the Menu. The app was purposefully designed much *flatter*, with five base screens all accessible through a BotNavBar.

4.3. Storage

Storage for this project will use a cloud storage solution. There are two main categories of databases that can be used for cloud storage: SQL based and NoSQL based. SQL databases are the more traditional type, including relational databases, hierarchical database, network database, and object-oriented database (Oamii Tech, 2022). NoSQL database models include the graph database, document database, key-value stores, and column-oriented database (MongoDB, 2023).

The decision was made to use a NoSQL database, as NoSQL on average is faster on reading and writing data to a database (Wang & Yang, 2017). More specifically, Google Firestore. Firestore was chosen due to the decision that the app's backend is going to be handled by Firebase. This allows a better developer experience, as both authentication and database can be accessed through the Firebase Console.

4.3.1 Entity-Relationship Diagram

In order to construct an ERD, a non-exhaustive list of read and write processes the app will undergo was written down. Entities for the database was also identified. The information are shown below.

No.	Read	Write
1	Read menu items	Add, update, or remove items from Shopping Cart
2	Read item descriptions	Checkout Shopping Cart
3	Read content of Shopping Cart items of a current logged in user account	Update order status (confirming, mark as finished, or cancelling)
4	Read account information for profile page and/or ‘stamps’	Add ‘stamps’, reset ‘stamps’ number if applicable
5	Read order history of a user	Add order to order history of each user
6	Read account information for eligibility of free drink on birthday week	

Table 4.1. A non-exhaustive list of read and write operations of the app.

List of entities:

1. User
2. Order
3. Product

Using these information, the entity-relationship diagram can be constructed below.

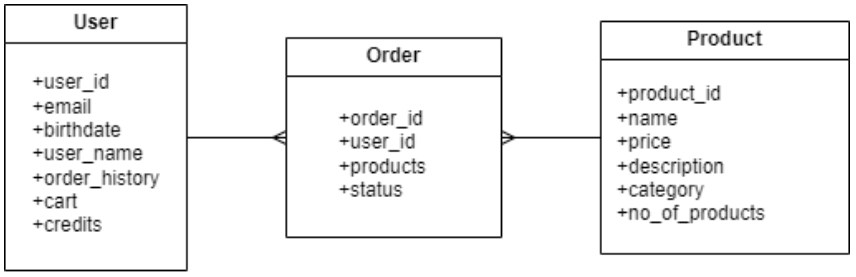


Figure 4.15. ERD diagram using the aforementioned entities: User, Order, and Product.

The relationship between the User collection and Order collection is one-to-many, and the Product collection to the Order collection is one-to-many. This is because one user is able to have multiple orders, whether completed or not, and the list of orders will be saved in the *order_history* document. A product is available in multiple orders, hence its one-to-many relationship, and the list of products will be saved in the *products* document.

5. Development and Testing

This chapter discusses about the development and testing stage of the project. This includes the discussion of which technology to use, any social, ethical, professional, or legal issues, and the development methodology used throughout the project.

5.1. Flutter vs React Native

In order for the app to be built to the highest quality, it needs to be built with the most suitable technologies. This section is going to talk about what technologies are going to be used throughout the development of this project.

Both Flutter and React Native's aim is to build an app that is available cross-platform (Flutter, n.d.) (React Native, n.d.) accessed at 2022. Research was done to determine which technology is better suited for this project.

The technology used to build the app was chosen by considering two main factors: the performance of an identical app built with the differing technologies, and the support and documentation available for the technologies.

According to a study A. Banwarie in 2022, comparing Flutter, React Native, Xamarin and Native app performances, Flutter achieved a rating of 4.5 out of 5, while React Native achieved a rating of 4.0 out of 5.0. This is due to React Native using a JavaScript bridge in the app. Flutter does not use any bridges, hence allowing higher efficiency in loading UI (Banwarie, 2022).

Another study also proved this point, with Flutter having a faster start up time compared to React Native, 1.69 seconds against 4.26 seconds (Kho, Aliyazis, & Galinium, 2022). In the same study, it is also shown that a Flutter app consumes less memory bandwith, compared to a React Native app. An average of 6.52MB for the Flutter app, and 27.6MB for the React Native app (Kho, Aliyazis, & Galinium, 2022).

In the study conducted by A. Banwarie, which collected developers opinions on documentation of cross-platform technologies, it is concluded that Flutter has a 5.0 out of 5.0 rating, while React Native has a 4.0 out of 5.0 rating. The availability of standardised documents and lab exercises are the reasons Flutter is scored higher than React Native in documentation and resources (Banwarie, 2022). Another reason might also be the lack of structure and maintainence of React Native libraries, as they heavily rely on third-party libraries (Joshi, 2022).

Based on the discussion above, the author has chosen Flutter to be used in this project. However, for developers with a strong web background, React Native might be more

suitable. This is due to React Native was built by the same team that developed React, a popular library used to make websites and webapps. The syntax between the two products are highly similar, thus the flatter learning curves.

5.2. Social, Ethical, Professional, and Legal Issues (SEPLI)

This project aims to build a fully functional mobile app that will interact with users. For the app to work properly, there will be user data that needs to be stored. The data needs to be stored securely, according to the EU General Data Protection Regulation (GDPR).

In practical terms, users' details need to be stored in a secure database, and their passwords need to be encrypted. Users will also need to be able to delete their accounts whenever they think it necessary.

This project will also need other people to be interacted with. This includes the author's supervisor, internal and external markers, and people aged 18+ for gaining feedback and testing purposes.

Interacting with the supervisor and markers, while formal language is not required, is still important to use academic and professional language while interacting with them.

Since the project aims to build a mobile app for the public, a user-centric design approach is going to be used. This means during the development, there will be times when users' feedback is needed. This can be done using interviews or questionnaires, both of which require a formal and professional manner while interacting. The end users will also need to consent before being included in this project, and their anonymity protected when they provide information and feedback that will be collected.

The use of media in this project will also be subject to legislation, namely under the Copyright Act. It is important to not use designs and media protected under Copyright, unless with permission of the original artist.

5.3. Development Methodology

A development methodology dictates how a project will go; it will determine how a project is managed, scheduled, and progressed. Hence, it is quite important to adopt the most suitable methodology to produce the deliverables most efficiently and productively as possible while maintaining a high standard of quality.

This project will likely be using a hybrid of the Waterfall Model and Agile. More specifically, Rapid Application Development (RAD).

Due to the nature of this project with a specific deadline, and the fact that this project will not be done by a team, but rather by a single person, a purely Agile methodology will not work. This is due to the very flexible nature of Agile, and the fact that there is a lack of formal documentation that is required under the Agile methodology (Risener, 2022). This project requires frequent documentation of the deliverables to achieve a high grade. Since Agile is flexible, it means that it is rather uncertain – projects are very prone to changes based on external factors (Risener, 2022). While this may be an advantage in other situations, with a set deadline and an unchanging set of expected deliverables, using purely Agile methodology will not be wise.

The Waterfall Model, though many describe it as outdated (Appelbaum, 2019), would be effective as there is quite a sizable amount of planning allocated to the initial parts of the project development. However, the Waterfall model is not without flaws. Its main drawback is the lack of a feedback loop (Risener, 2022). Projects that use a strict Waterfall Model will be slow to implement any changes, and any mistakes found at the end stages of the Waterfall Model will generally require the project to start over from its Planning Phase again (Appelbaum, 2019).

Combining the pros of both Agile and the Waterfall Model and minimizing both their cons, this project will be run using RAD. RAD in general has 4 steps: defining project requirements, prototyping, construction and feedback gathering, and finalizing (Chien, 2020). This fits perfectly for this project – a set project scope at the start, constantly improving prototype over iterations based on feedback, and the final implementation of the project.

RAD has more structure compared to Agile, like the Waterfall method, while still having the flexibility of Agile during the development stage.

6. Evaluation

This chapter will be on the evaluation of the project. The chapter covers the initial planning and the development process of the project.

Upon reflection, the initial planning went pretty well. Schedules were made, solid goals were set on what to do when, and the main objective of this project was well thought out. However, I have only realised later on that I did not take into account the holidays and breaks that we do, resulting in a mess of a schedule later on. It also occurs to me that I have spent too much

time planning, and not enough time actually executing the plan. Moving forward, this is a large opportunity for growth that needs to be cultivated.

During the developmental phase of the project, there was a lot of learning occurred. There were a lot of hours spent on just learning the tools that I was using, and even more debugging careless mistakes due to the inexperience of using Flutter. Doing the literature review also helped in the learning of mobile application development and design. The differences between Human Design Guidelines and Material Design and how they affect user interaction with the screen, the pros and cons of different MCPD technologies, and how business owners factor in different things in order to grow using technology. On a personal level, the literature review has taught me to be patient and critical. Reading different papers that do not agree with each other provokes me to think and be more diligent in finding the correct answer.

7. Conclusions and Recommendations for Further Work

In conclusion, I am not very happy with how the work turned out. The artefact does not meet the initial requirements due to very bad time management. Lots of things to improve.

For future work, the app can also be further developed. Including but not limited to analytics, actual card payment services, the use of profile pictures, different themes, and more. And better code too.

References

Flutter. (n.d.). *Flutter - Build apps for any screen*. Retrieved from Flutter: https://flutter.dev/

Android Developer Tools. (2019). *App Startup Time*. Retrieved from Android Mobile App Developer Tools – Android Developers: https://developer.android.com/topic/performance/vitals/launch-time#:~:text=Cold%20startup%20takes%205%20seconds,takes%202%20seconds%20or%20longer.

Appelbaum, B. (2019). *Top 6 Software Development Methodologies*. Retrieved from https://blog.planview.com/top-6-software-development-methodologies/

Apple. (2023). *Human Interface Guidelines*. Retrieved from Apple Developer: https://developer.apple.com/design/human-interface-guidelines/guidelines/overview/

Babich, N. (2019). *The 4 Golden Rules of UI Design*. Retrieved from https://xd.adobe.com/ideas/process/ui-design/4-golden-rules-ui-design/

Banwarie, A. (2022). Choosing the right framework for Android development: which mobile development frameworks are chosen and why?

Barros, L. P., Medeiros, F., Moares, E., & Junior, A. F. (2020). Analyzing the Performance of Apps Developed by using Cross-Platform and Native Technologies. *SEKE*, 186-191.

Biørn-Hansen, A., Grønli, T.-M., & Ghinea, G. (2019). Animations in Cross-Platform Mobile Applications: An Evaluation of Tools, Metrics and Performance. *Sensors*, p. 19.

Biørn-Hansen, A., Rieger, C., Grønli, T., Majchrzak, T. A., & Ghinea, G. (2020). An empirical investigation of performance overhead in cross-platform mobile development frameworks. *Empirical Software Engineering 25*, 2997-3040. Retrieved from https://doi.org/10.1007/s10664-020-09827-6

Carter, J. (2023). *Time To Market (TTM) – 5 Ways to Reduce it and Market Quickly*. Retrieved from TCGen Product Management Consulting Experts: https://www.tcgen.com/time-to-market/

Chappal, M. S. (2021). *The 6 key principles of UI design*. Retrieved from https://maze.co/collections/ux-ui-design/ui-design-principles/

Chien, C. (2020). *What is Rapid Application Development (RAD)?* Retrieved from https://codebots.com/app-development/what-is-rapid-application-development-rad

Corrigan, S. (2021). *Introduction to User Interface Design: 6 Important Principles*. Retrieved from https://www.flux-academy.com/blog/introduction-to-user-interface-design-6-important-principles

Dolega, L., Rowe, F., & Branagan, E. (2021). Going digital? The impact of social media marketing on retail website traffic, orders and sales. *Journal of Retailing and Customer Services*.

Dube, L., & Renaghan, L. (2000). Creating visible customer value. *Cornell Hotel and Restaurant Administration Quarterly, Vol. 41*, 62-72.

Firtman, M., Andrew, R., Jara, A., LePage, P., & Medley, J. (n.d.). *Progressive Web Apps*. Retrieved from https://web.dev/learn/pwa/progressive-web-apps/

Flutter. (n.d.). *Measuring your app's size*. Retrieved 2023, from Flutter documentation: https://docs.flutter.dev/perf/app-size

Footprint. (2021, February 18). *Coffee shop sales fall 40% but big chains expand*. Retrieved from https://www.foodservicefootprint.com/coffee-shop-sales-fall-40-but-big-chains-expand/

Gonsalves, M. (2018). *EVALUATING THE MOBILE DEVELOPMENT FRAMEWORKS.*

Google. (2023). *Material Design*. Retrieved from Material Design: https://m3.material.io/

Grochowska, I. (2022). *Coffee loyalty programs: 10 successful examples (2022)*. Retrieved from https://www.openloyalty.io/insider/coffee-loyalty-programs-10-successful-examples-2022

Grzmil, P., Skublewska-Paszkowska, M., Łukasik, E., & Smołka, J. (2017). *PERFORMANCE ANALYSIS OF NATIVE AND CROSS-PLATFORM MOBILE.* Lublin University of Technology, Institute of Computer Science.

Hanaysha, J. R. (2018). Customer retention and the mediating role of perceived value in retail industry. *World Journal of Entrepreneurship, Management and Sustainable Development, Vol. 14 No. 1*, 2-24.

Huber, S., Demetz, L., & Felderer, M. (2022). *A comparative study on the energy consumption of Progressive Web.*

Hussain, H., Khan, K., Farooqui, F., Arain, Q. A., & Siddiqui, I. F. (2021). Comparative Study of Android Native and Flutter App Development. *Memory*, 36-37.

IBISWorld. (2022). *Cafes & Coffee Shops in the UK - Market Size 2011-2029*. Retrieved from https://www.ibisworld.com/united-kingdom/market-size/cafes-coffee-shops/

Ifunanya, U. (2020). *12 key UI/UX design principles to boost your designs*. Retrieved from https://bootcamp.uxdesign.cc/fundamental-principles-of-ui-ux-design-3b1434e90a99

Inkbot Design. (2020). *What Makes A Good User Interface? 13 UI Design Principles*. Retrieved from https://inkbotdesign.medium.com/what-makes-a-good-user-interface-13-ui-design-principles-b156b1fb4c13

Jia, X., Ebone, A., & Tan, Y. (2018). A Performance Evaluation of Cross-Platform Mobile Application Development Approaches . *MOBILESoft '18: Proceedings of the 5th International Conference on Mobile Software Engineering and Systems*, 92-93.

Joshi, M. (2022). *Flutter vs React Native: A Comparison*. Retrieved from BrowserStack.

Khandelwal, V. (2022). *iOS Web Push Notifications: What Does This Mean For Publishers?* Retrieved from https://www.izooto.com/blog/ios-safari-push-notifications

Kho, I. E., Aliyazis, A. W., & Galinium, M. (2022). Front-End Application For Multiple Storefronts Ecommerce Using Cross-Platform Technology.

Kidecha, S. (2022). *Native vs Hybrid vs Cross-Platform: How and What to Choose?* Retrieved from https://dzone.com/articles/native-vs-hybrid-vs-cross-platform-how-and-what-to

Kotlin. (2022). *What is cross-platform mobile development?* Retrieved from https://kotlinlang.org/docs/cross-platform-mobile-development.html

Kuniavsky, M. (2010). *Smart Things: Ubiquitous Computing User Experience Design.*

Lewis, P. (2018). *Rendering Performance*. Retrieved from web.dev: https://web.dev/rendering-performance/

Liu, B. S.-C., Petruzzi, N. C., & Sudharshan, D. (2007). A service effort allocation model for assessing customer lifetime value in service marketing. *Journal of Services Marketing, Vol. 21, No. 1*, 24-35.

Lloyd, A. (2021). *The Importance of a Competitor Analysis and How to Conduct One*. Retrieved from KAYO Digital: https://kayo.digital/news/the-importance-of-a-competitor-analysis

Mahendra, M., & Anggorojati, B. (2020). Evaluating the performance of Android based Cross-Platform App Development Frameworks. *ICCIP '20: Proceedings of the 6th International Conference on Communication and Information Processing*, 32-37.

MDN Web Docs. (2022). *Progressive web apps (PWAs)*. Retrieved from MDN Web Docs: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps

MongoDB. (2023). *Types of NoSQL Databases*. Retrieved from MongoDB: https://www.mongodb.com/scale/types-of-nosql-databases

National Coffee Association. (2021). *National Coffee Data Trends, Media Highlights, Spring 2021.* Retrieved from https://www.ncausa.org/Portals/56/Images/MarketResearch/20210325%20NCA%202021%20media%20highlights.pdf?ver=DRpZe_V7CfE3O3GkEmSybw%3D%3D

Nielsen, J. (2000). *End of Web Design.* Nielsen Norman Group.

Oamii Tech. (2022). *What Are The 4 Types Of Database Management Systems?* Retrieved from Oamii Tech: https://www.oamiitech.com/what-are-the-4-types-of-database-management-systems

Pratama, M. A., & Cahyadi, A. T. (2020). Effect of User Interface and User Experience on Application Sales. *IOP Conference Series: Materials Science and Engineering 879.* IOP Publishing.

Rakshit, S., Islam, N., Mondal, S., & Paul, T. (2021). Mobile apps for SME business sustainability during COVID-19 and onwards. *J Bus Res*, 28-39.

React Native. (n.d.). *React Native - Learn once, write anywhere*. Retrieved from React Native: https://reactnative.dev/

Risener, K. (2022). A Study of Software Development Methodologies. Retrieved from https://scholarworks.uark.edu/csceuht/103/

Rose, A. (2021). *Accessibility is too often forgotten about in website design*. Retrieved from https://www.moneyandmentalhealth.org/accessibility-in-website-design/

Schmitt, J. (2022). *Native vs cross-platform mobile app development*. Retrieved from CircleCI: https://circleci.com/blog/native-vs-cross-platform-mobile-dev/

Shevtsiv, N. A., & Striuk, A. M. (2020). Cross platform development vs native development.

Shopify. (2022). *Definition of a Small Business: How Big Is Still Considered Small?* Retrieved from Shopify: https://www.shopify.com/blog/what-is-considered-a-small-business

Singh, S. (2021). *Native vs Hybrid vs Cross Platform – What to Choose in 2022?* Retrieved from https://www.netsolutions.com/insights/native-vs-hybrid-vs-cross-platform/#what-is-a-native-app

Singh, S. (2023). *Native vs Hybrid vs Cross Platform – What to Choose in 2023?* Retrieved from Net Solutions: https://www.netsolutions.com/insights/native-vs-hybrid-vs-cross-platform/#what-is-a-native-app

Specialty Coffee Association. (2020). *Specialty Coffee Consumption and COVID-19: The 2020 Square x SCA Coffee Report*. Retrieved from https://sca.coffee/sca-news/news/2020-square-report

Statcounter GlobalStats. (2023). *Mobile Operating System Market Share Worldwide - January 2023*. Retrieved from statcounter GlobalStats: https://gs.statcounter.com/os-market-share/mobile/worldwide

Statista. (2022). *Selected leading coffee shop chains in the United Kingdom (UK) as of January 2022, by number of units*. Retrieved from Statista: https://www.statista.com/statistics/297863/leading-coffee-shop-chains-in-the-united-kingdom-uk-store-number/

Stocchi, L., Pourazad, N., Michaelidou, N., Tanusondjaja, A., & Harrigan, P. (2021). Marketing research on Mobile apps: past, present and future. *Journal of the Academy of Marketing Science 50*, 195-225.

Supriadi, O. A. (2019). User Interface Design of Mobile-based Commerce. *IOP Conference Series: Materials Science and Engineering 662.* IOP Publishing.

Svensson, O., & Käld, M. P. (2021). *React Native and native application development.* Jönköping: Jönköping University School of Engineering.

Toonen, E. (2020). *What is a progressive web app (PWA)? Why would you want one?* Retrieved from https://yoast.com/what-is-a-progressive-web-app-pwa/

UXPin. (n.d.). *The Basic Principles of User Interface Design*. Retrieved from https://www.uxpin.com/studio/blog/ui-design-principles/

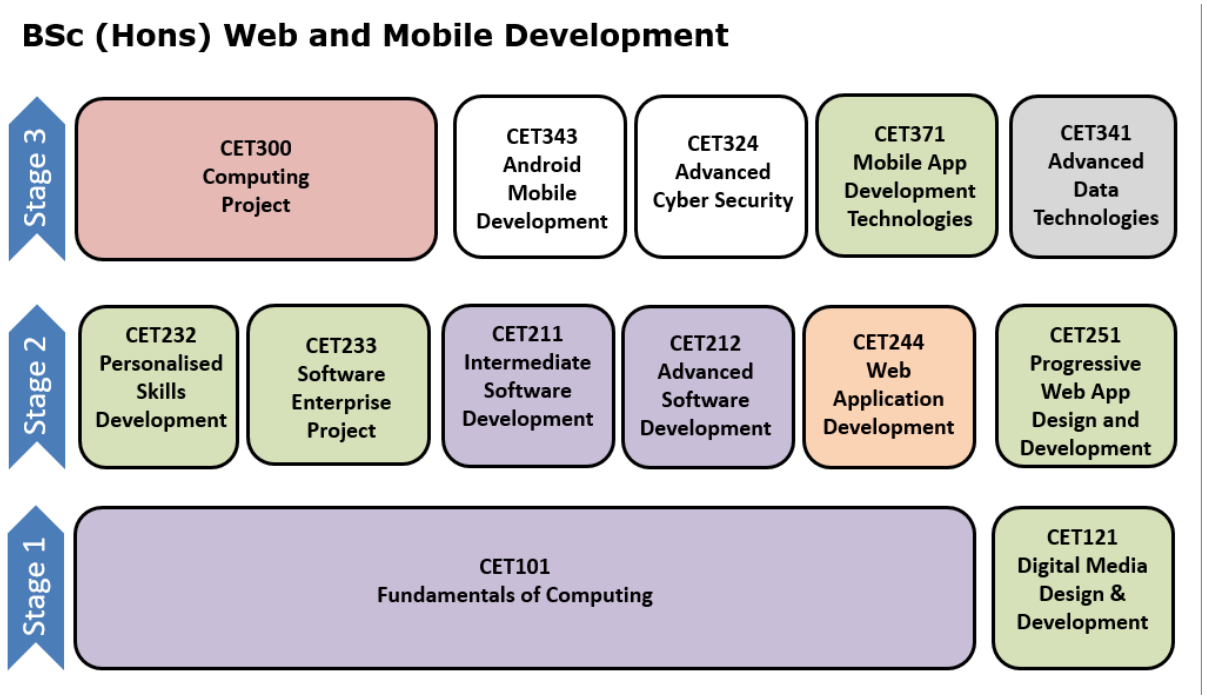
Wang, R., & Yang, Z. (2017). SQL vs NoSQL: A Performance Comparison.

Weichelt, B., Heimonen, T., Pilz, M., Yoder, A., & Bendixsen, C. (2019). An Argument Against Cross-Platform Development: Lessons From an Augmented Reality App Prototype for Rural Emergency Responders. *JMIR mHealth and uHealth.*

White, J. (2013). Going native (or not): Five questions to ask mobile application developers. *Australas Med J.*

Appendices

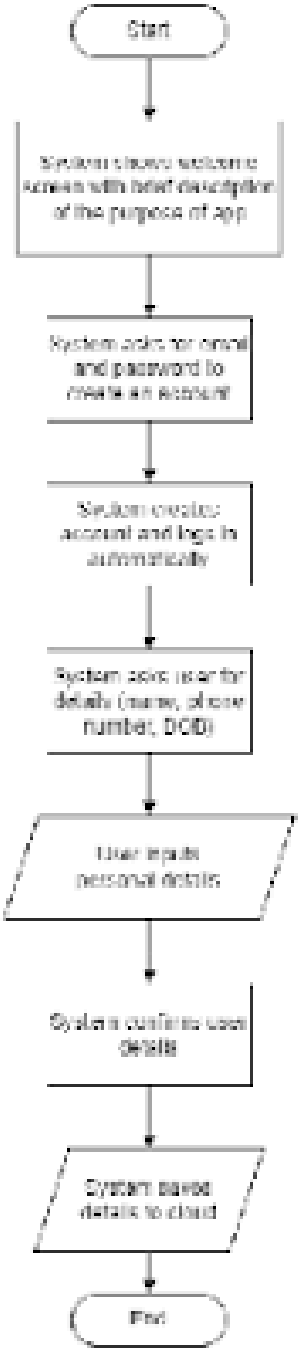
Appendix A: Programme Route Diagram



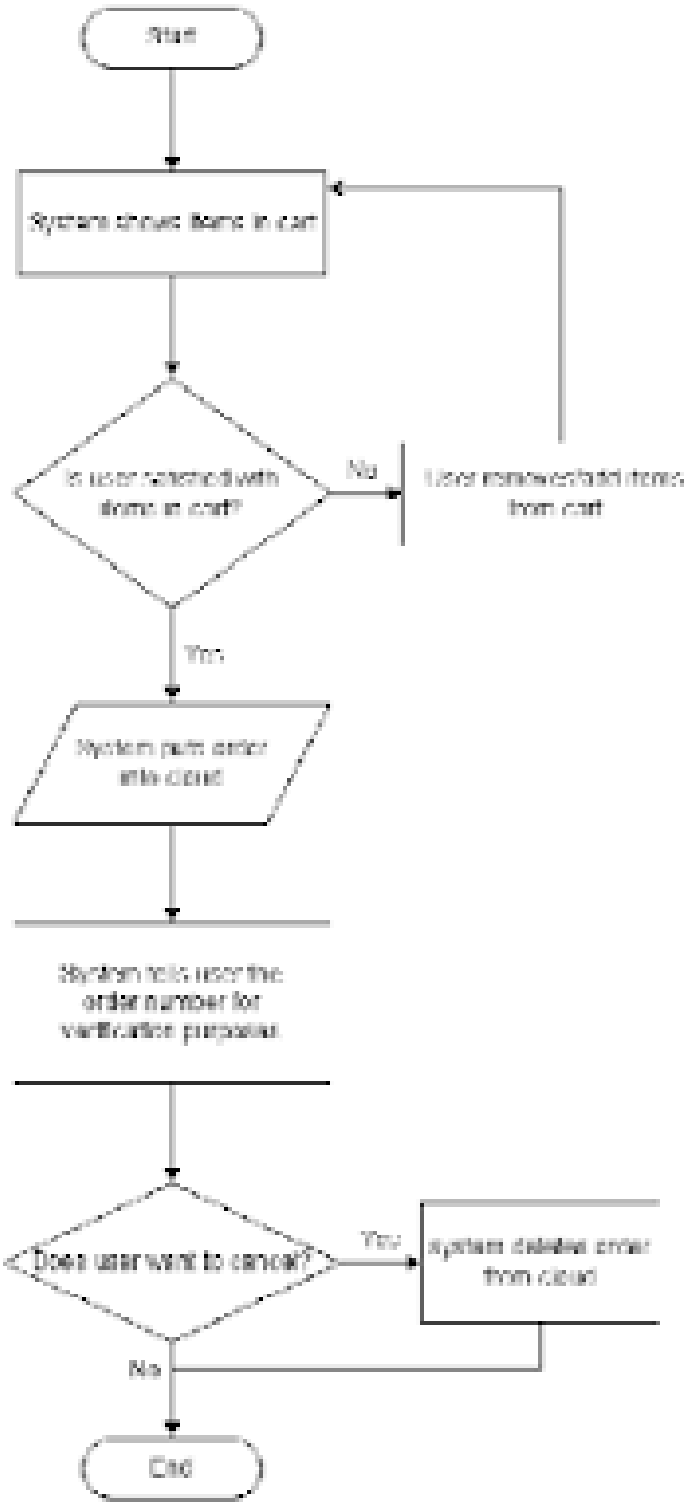
MAIN FLOW


Start
User opens the app
Does user have an account?
Yes
User sign in to app
No
Account creation/Onboarding
App home screen is shown
User selects menu
User selects cart
User selects Account
User selects item to add to cart
System saves user addition to cart to cloud
User satisfied with items selected?
No
Yes
user checkout process
System loads users current cart from cloud
System shows order history
User selects order history
User edits account
User selects logout
System shows confirmation message
User confirms
User is logged out
User closing the app?
No
Yes
User exits the app
End

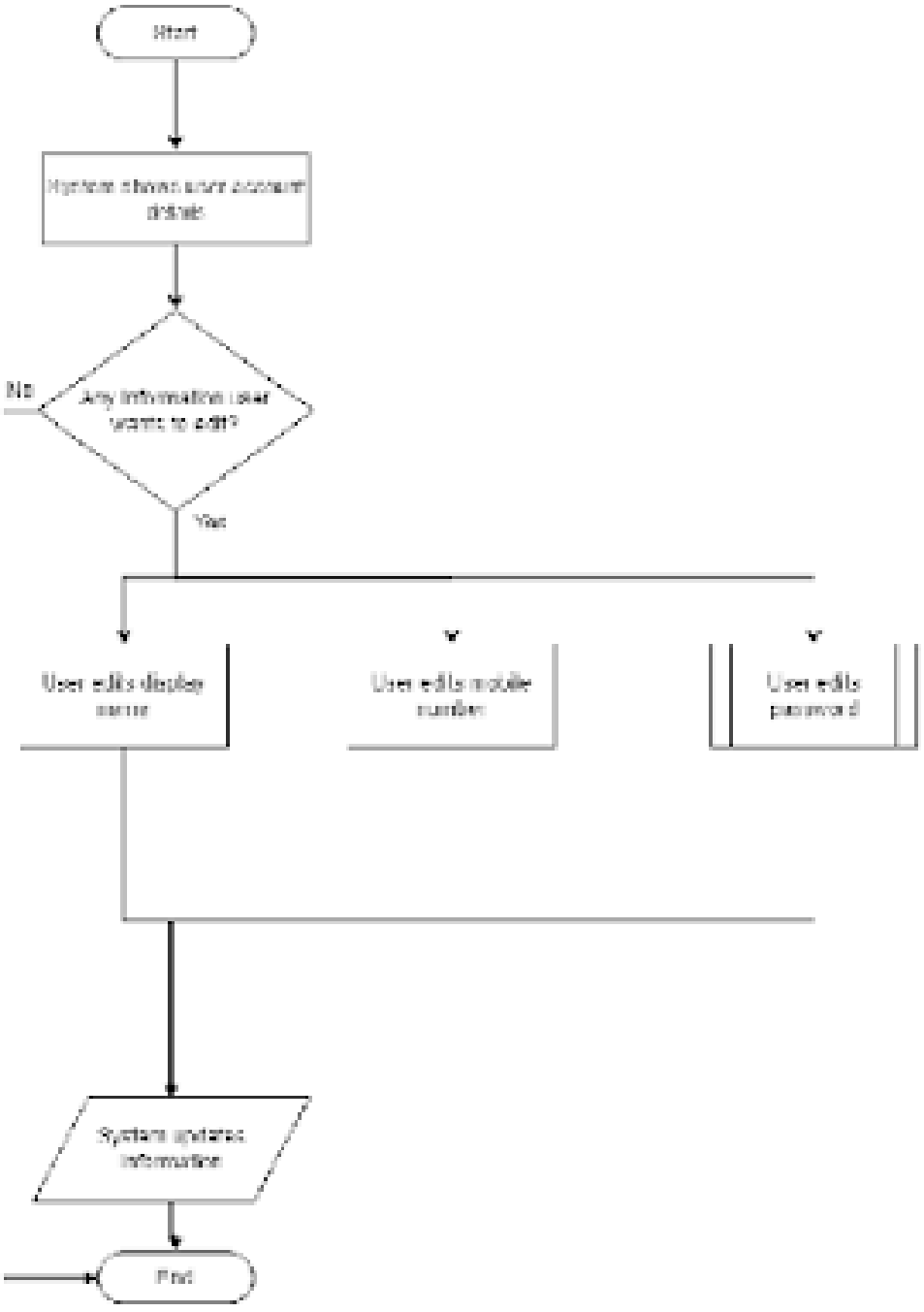
ACCOUNT CREATION /ONBOARDING PROCESS



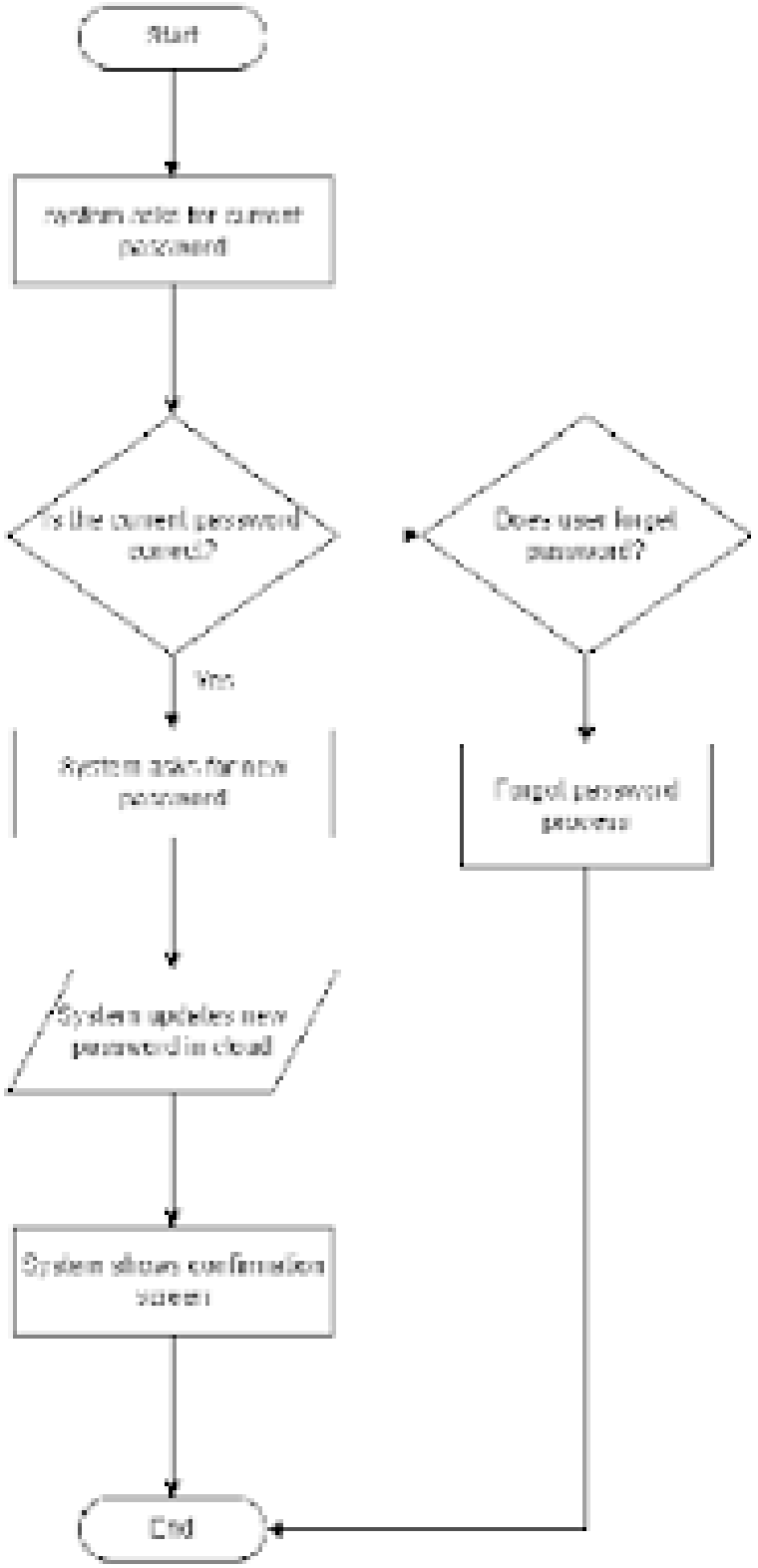
USER CHECKOUT PROCESS



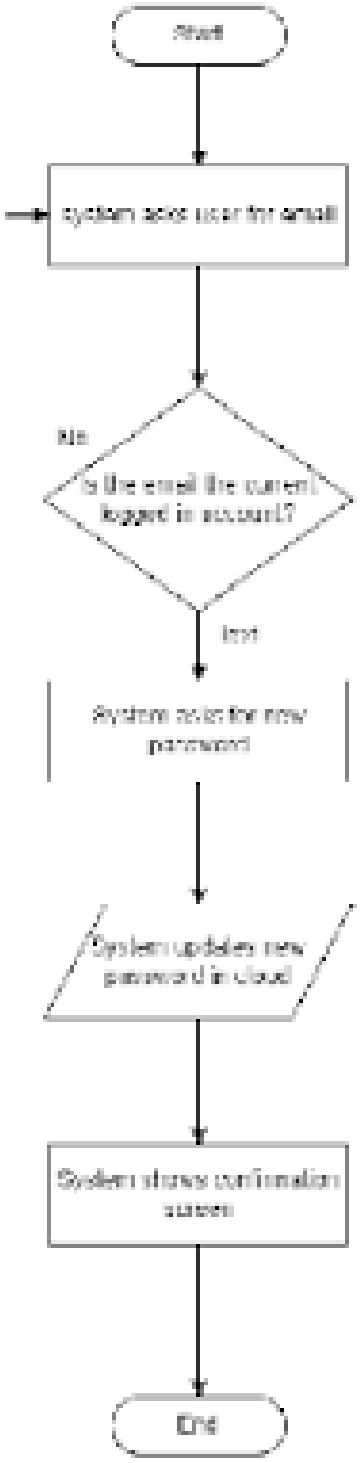
USER EDIT ACCOUNT PROCESS



EDIT PASSWORD PROCESS



FORGOT PASSWORD PROCESS



Appendix E.2: ERD

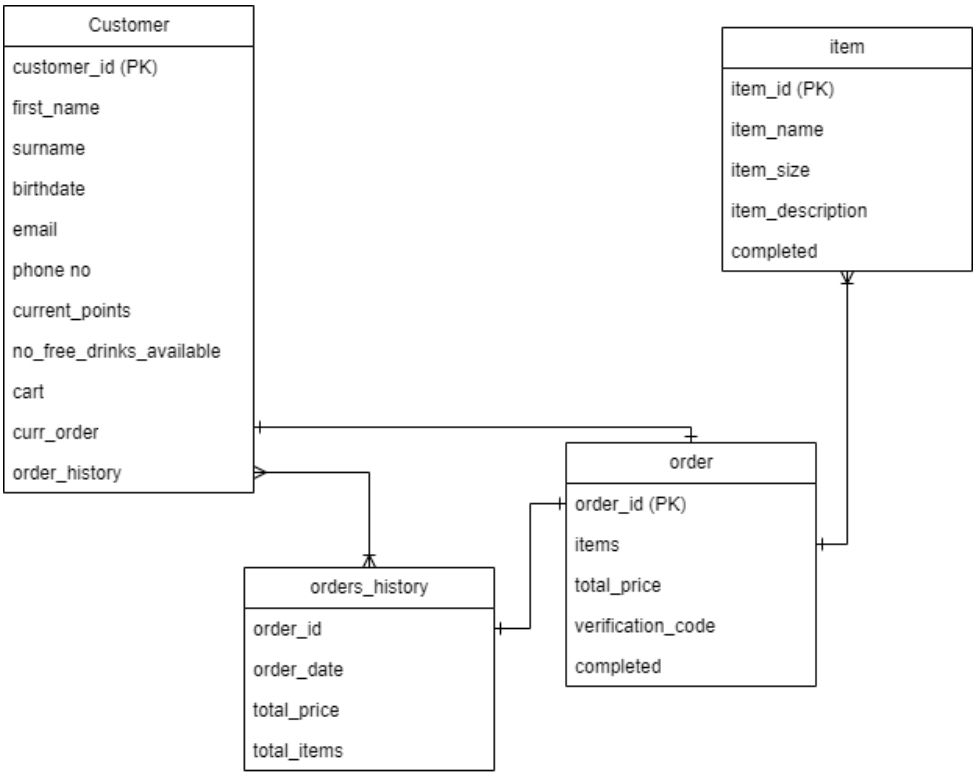


Figure 16: First iteration of the Entity Relationship Diagram of the app

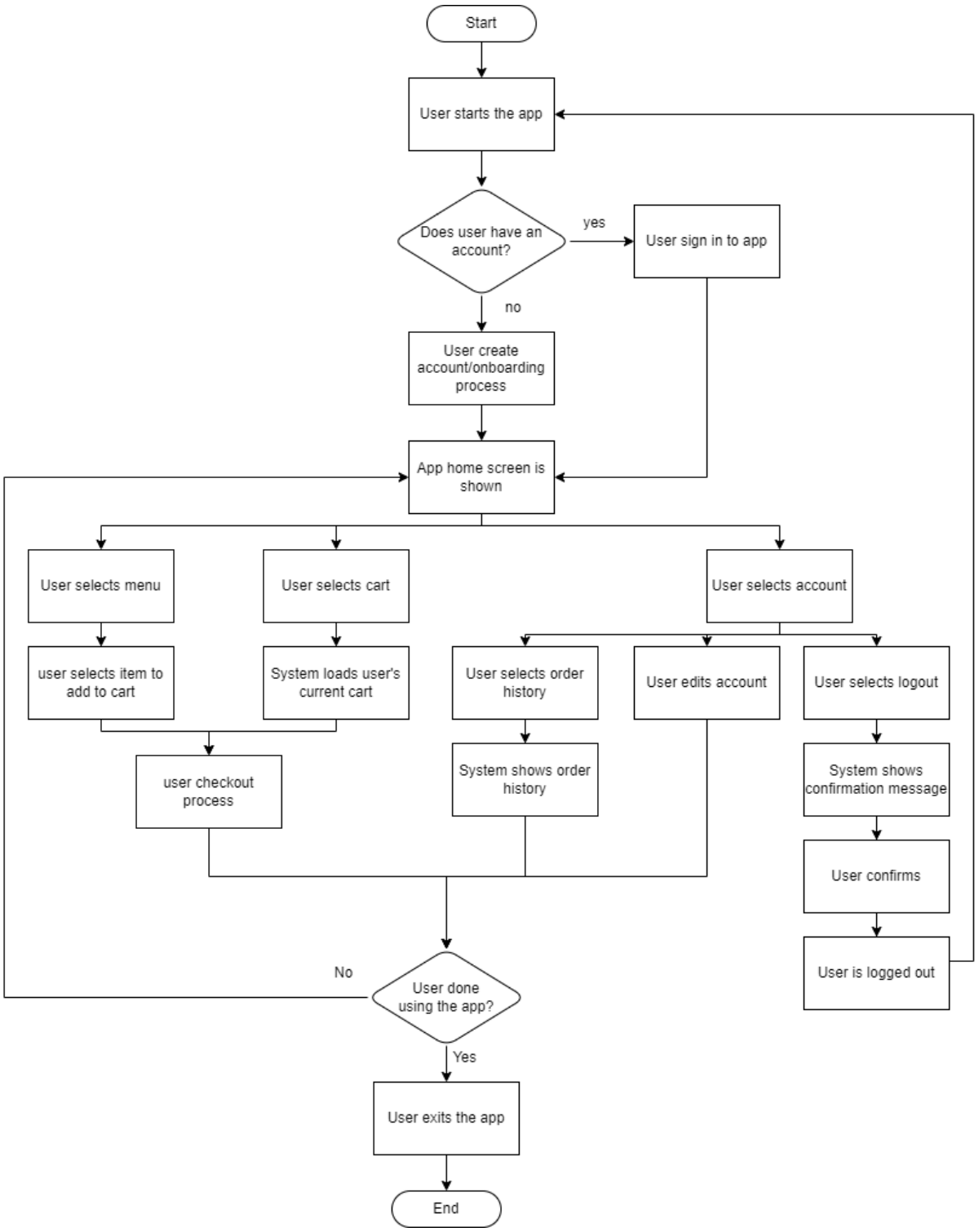


Figure 17: Initial simplified flowchart of how the app runs