# MSc Project

~

# Mobile Barber Application

Joshua Robertson
20 September 2021

*"A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Science by advanced study in Computer Science in the Faculty of Engineering."*

School of Computer Science, Electrical and Electronic Engineering, and Engineering Maths (SCEEM)

# Abstract

This thesis discusses the development of a minimum viable product (MVP) of an application aimed to facilitate the delivery and execution of home haircuts and other beauty treatments. Whilst there exists a range of applications that serve a similar purpose, none allow for immediate booking and delivery, similar to that seen in the ”Uber” model, which we therefore aim to provide in this application. In describing the process, we first begin by examining the project management and software methodologies that will be used for the project, which pertains to agile, or more specifically lean and user centric design (UCD). Next, we discuss the software and framework chosen for both the front and back-end, along with the ideas influencing these decisions before moving on to examining the target user.

In the software requirements section, we discuss the scope, user needs, requirements and use cases, similar to what would be found within a software requirements specifications (SRS) document. The system design section then builds upon those requirements and discusses how they were used to drive decisions around the architecture and state management used for the application. Within the prototyping section, we discuss the sketching and wire-framing that was carried out to represent the user interface of the app, along with some basic functionality, before this is implemented within the implementation section. Finally, we discuss testing and any future work.
The main contributions of the project are as follows:

- Market research was carried out to highlight any market gaps within existing applications
- User surveys and testing was used throughout to adhere to both UCD and lean development methodologies
- A cross-platform MVP using Flutter and Dart was created that exhibited the following features
  - Allowed a user and barber to create an account and login through authentication
  - Allowed for the booking and checkout of a variety of barber products
  - Allowed the user and barber to manage their account within the application

# Acknowledgements

I would like to thank my supervisor, Dr Alex Kavvos; I am extremely grateful for our weekly meetings, the guidance and support of which were highly influential in the outcome of the project.

I would like to acknowledge any participants who have given their valuable time to the project in any capacity.

I would like to thank my friends and family whose unconditional support has been unwavering throughout a challenging year.

# Authors Declaration

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Taught Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, this work is my own work. Work done in collaboration with, or with the assistance of others, is indicated as such. I have identified all material in this dissertation which is not my own work through appropriate referencing and acknowledgement. Where I have quoted or otherwise incorporated material which is the work of others, I have included the source in the references. Any views expressed in the dissertation, other than referenced material, are those of the author.

This project fits within the scope of ethics application 0028, as reviewed by my supervisor Dr Alex Kavvos.

SIGNED: Joshua Robertson DATE: 19.09.2021

# Contents

# Acronyms

**API** Application Programming Interface. 17, 23, 26, 27, 43–45

**GIMP** GNU Image Manipulation Program. 46

**IDE** Integrated Developer Environment. 33

**MVC** Model-view Controller. 25, 26, 41

**MVP** Minimum Viable Product. 1, 5, 10, 11, 50–53

**NPD** New Product Development. 10, 11, 15

**SDK** Software Development Kit. 17

**SLA** Service-level Agreement. 34

**SRS** Software Requirements Specification. 1, 22

**UCD** User Centric Design. 1, 10, 13, 17, 30, 48, 50

**UI** User Interface. 4, 26–30, 33, 34, 41, 42, 52

# List of Figures

# List of Tables

# 1 Introduction

The COVID-19 pandemic has had a ubiquitous impact on our lives, from disrupting how we work and travel, to influencing how we socialise and interact with one another. This has brought a shift in consumer and business behaviours; some of which are temporary but others, such as the adoption of more digital services and flexible home-working seem likely to be permanent [4]. Furthermore, the restriction of movement has caused consumers to migrate to online services at an unprecedented rate. With the global home services market expected to grow by almost 20% per year until 2026 [31], there exists a wealth of opportunity for companies to capitalise through digitalising previously physical services.

Here, we propose an application directed towards targetting the self-care market, which, unsurprisingly saw a sizeable uptake in demand during the pandemic [11]. Despite many hairdressers and salons re-opening their stores, consumers are still hesitant to return with levels yet to reach those seen before the pandemic [5].

There already exists a range of applications suited towards providing home haircuts. For example, Shortcut [21], TRIM-IT [38] and TrimCheck [39] all provide bookable home haircuts. Despite this, no application offers a comprehensive set of features, such as allowing for immediate home haircuts, rather than requiring an advanced booking and allowing for booking of multiple beauty products.

Here, we therefore propose an application aimed to facilitate this and allow for immediate booking and delivery of home haircuts and other beauty treatments. We will aim to explore the novelty of the application through market research to elucidate any further gaps that may be apparent. Together, this will be achieved through the following objectives:

- Conduct initial user research to elucidate any market gaps
- Through a user centric design (UCD) methodology plan and prototype the user interface of the application
- Create an MVP using New Product Development (NPD) methodologies

For the implementation of the application, a heavy focus on the end user was taken through using a UCD methodology, along with NPD, which refers to the entirety of processes leading to bringing a product to market and encompasses several steps as seen in Figure 1.1 below and discussed throughout the proceeding sections.


Figure 1.1: The 7 Steps of New Product Development
[33]

## 1.1 Ideation and Concept

The first stage of NPD starts with conceptualisation of a product idea. For the application the initial concept came from an external partner, who proposed a mobile hairdresser application to capitalise on the increased need for home delivery services. This was further defined during several meetings carried out in the initial stages of the project. This amounted to several features required for the product to be considered a viable MVP, including:

- The user should be able to create an account and login using authentication
- The user should be able to view a list of barbers in their area (using geolocation)
- The user should be able to view the barbers products and add them to their cart
- The user should be able to checkout their cart
- The user should be able to schedule in a time when they will receive their home hair cut
- The parent barber should be able to create an account and login using authentication
- The parent barber should be able to add a new barber and their products

## 1.2 Project Management

The project management for the application focused on two key aspects; a clear vision and scope, including a detailed project plan; and an execution phase, which utilized an agile methodology.

### 1.2.1 Choosing a Project Management Framework

Historically, the waterfall framework was the most dominant software methodology [6]. The Waterfall method involves an iterative model, whereby a step-by-step approach is taken, with each stage being extensively planned out and execution being rigid. It was decided that Waterfall was not to be used due to several considerations, most relevantly:

- A lack of programming experience in the given language and therefore it is likely that development would be hindered by a structured approach
- It being a time-limited project, not enough resources would be available to be assigned to the time needed to meticulously plan the project

Alternatively, it was decided to adopt an agile methodology, arguably the industry standard in software development. Agile was introduced as a means to make the execution of projects more flexible and timely and it is estimated that around 86% of software projects are carried out using such methods [35]. Within agile, there exists a variety of methodogloies, of which lean was beleived to be the most suitable [30], which involves seven basic principles:

- eliminate waste
- amplify learning and create knowledge
- decide as late as possible
- deliver as fast as possible
- empower the team
- build integrity/quality in
- see the whole

Using lean development allowed for short, iterative cycles of production, providing value in the form of quick creation and constant revision. Taking on an agile methodology also served the project well in the sense that there was a strong focus on prioritising quality over comprehensive documentation and lengthy processes. This methodology also involves constant variation and improvement of previously implemented code, allowing for learning and development as the project grows. Finally, giving a focus of the project as a whole added tremendous value due to it being solo developed and therefore allowed for bigger picture thinking at every stage of production.

Although this approach is more commonly relevant to a team of developers, approaching the project management in this way allowed for a stringent and well defined timeline to be used, aiding in project delivery and outcome. This involved several key stages. Firstly, individual epics were defined, which included:

- Define the Scope and Market Research
- Design and Architect the Application
- Setup and Create The Back End
- Write the Dissertation

These were then used to create stories which were further split into individual tasks, placed into a timeline and carried out in sprints. For this, the project management tool monday.com [27] was used, which can be seen in Figure 1.2. Using Monday.com allowed for a timeline to be easily created, along with updating the status of each story when relevant to follow the completion of the project. In this way, a change management approach [22] could be carried out, in which the project could easily be updated and reflected in the tasks. This was especially important as both agile methodology and UCD rely on constant feedback from the end user which informs the project.

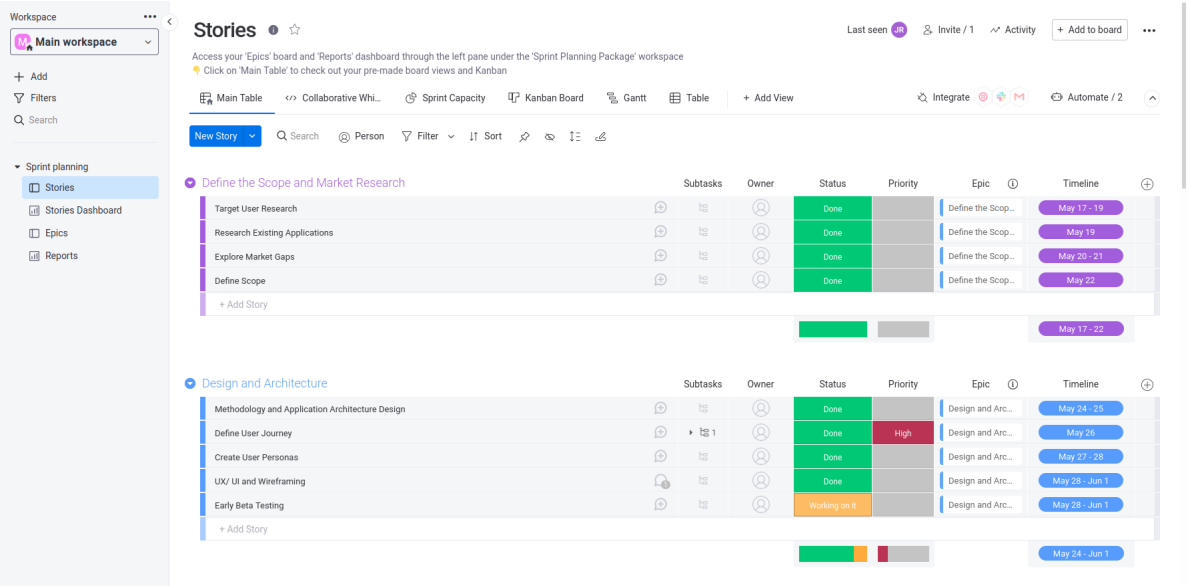



Figure 1.2: Screenshot of Monday.com Displaying the Project Stories [27]

Finally, using the created timeline, a gantt chart was implemented, which gave an overarching view of the project, with tasks performed represented along the vertical axis and the timeline represented along the horizontal axis.

## 1.3 Research and Market Analysis

In order to gauge whether there is a market for the proposed analysis, a survey was carried out in which users were asked about whether they could see themselves using the application features, among other things. The full survey can be found at section 7.3. The results of the survey indicated that there was indeed a market for the proposed application. For example, 77% of users indicated that it was a service that they would potentially use and many users commented that they would also use the service for other beauty treatments, such as nail and eyebrow treatment.

### 1.3.1 Existing Applications

As previously discusses there exists a variety of similar applications, for which the most prominent will be discussed below, along with the salient and limiting features of each.

TRIM-IT

TRIM-IT is a UK based mobile hairdresser application that currently operates in London, UK. Their business model is franchise based model, whereby barbers would invest through a monthly fee and be provided with the tools necessary to provide a haircut, such as a mobile barber unit and salon items. The positives of the service are that barbers are able to provide immediate haircuts and are not limited in having to book for a future date. The current negative features are that potential barbers must buy in to the service and there are therefore some initial start up costs which may dissuade barbers from using the service.

TrimCheck

TrimCheck is another UK based application operating in London. They work by allowing users to book a home hair cut through the application to be delivered at a future date and time. The positives of the application are that barbers must be registered at a barber shop, therefore there is some level of vetting that occurs and they also are widely known and used. The negatives of the application is that they only allow for booking of home haircuts and also do not provide a way to book a haircut immediately and only for a future date and time.

Shortcut

Shortcut is a US based application that was founded in 2014 and allows the user to book either a haircut or a hair cut and beard trim. One of the defining features of the shortcut app is the availability functionality, with the app providing the ability to request a haircut from 8am to as late as midnight.

The application is limited on its features that it provides, with it only having the option to request a Hair Cut only or a Hair Cut and Beard Trim. Another limiting feature of the application is the price. For a single haircut the cost starts at 75$ (around £54), which is most likely a reflection of high start up costs and is a problem seen in other similar applications, such as uber and lyft that can only be mitigated through losses [40]. As the application is currently only operating in the US, it does not present any current competition.

### 1.3.2 Novelty of the Proposed Application

As discussed in the previous sections, there exists a range of applications that are suited towards providing a mobile barber service, however, they are each limited by several factors. The novelty of the proposed application therefore aims to build upon these applications and act to unify the features found previously, such as:

- Allowing the user to book a home hair cut for immediate delivery

- Allowing for the barber to enter a range of services, rather than just a hair cut
- Keeping costs low by reducing initial start up costs to the barber (i.e. allowing them to use their own salon equipment and providing the products at the users address rather than a mobile unit)
- Providing the application throughout the U.K. by standardising the sign up process and focusing on ease of use

## 1.4 Deciding on a Platform

### 1.4.1 Mobile vs Desktop

An important consideration in NPD is determining which platform best suits the project, mobile or desktop. Here, we will discuss the merits and pitfalls of each, before concluding which is most relevant for the project.

Market Share

Consumers are now for the first time viewing web pages on mobile devices at a higher rate than on desktop, at 54.8%, compared to just 31.16% in early 2015 [26]. Further to this, over the last year desktop usage has dropped from 46.39% to 41.36%, whilst mobile phone usage has increased from 50.88% to 55.89%, following on a several year long trend [13] that reflects a saturated mobile market driving down the cost of phones. However, this analysis is slightly premature due to only being indicative of the world market, whereas the proposed application will only operate within the United Kingdom. When we analyse just the U.K. data (Figure 1.3) we see that the results are not so conclusive, with only a 0.97% difference between the two in favour of Desktop. Therefore a decision based entirely on market share is unfavourable and other metrics must be explored.

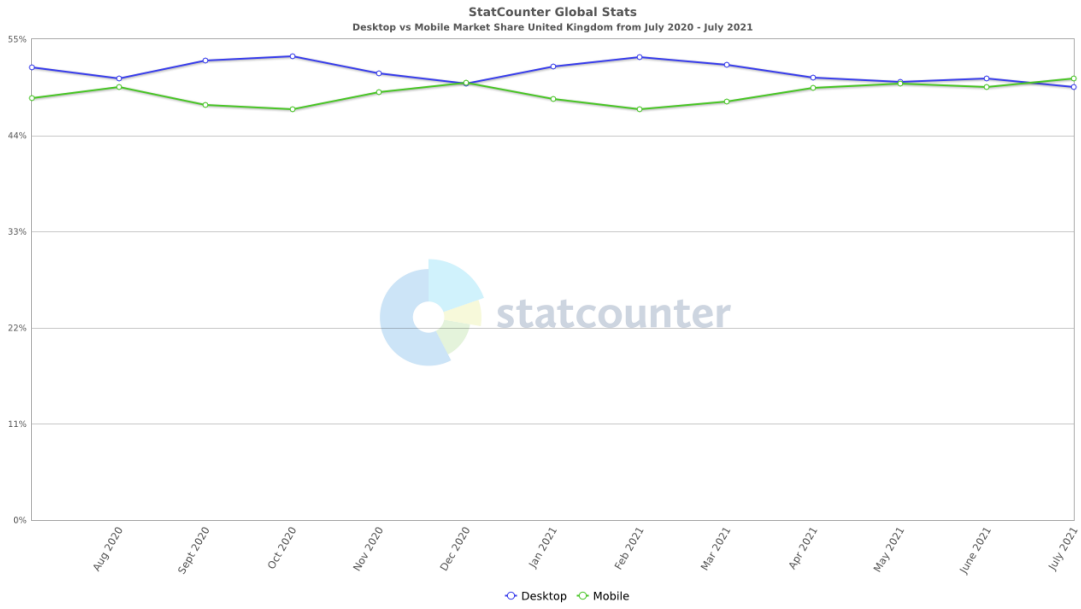



Figure 1.3: Desktop vs Mobile Market Share in the United Kingdom [12]

Features and Performance

Another measure to take into consideration is which features are required for the application and how this is reflected in mobile vs desktop applications. One the most pertinent features required is geolocation, which is much more suited to a mobile application due to inbuilt GPS. Another important consideration for the project is speed, for which mobiles perform actions much faster than a website along with ease of use, for which mobile again comes out on top. Finally, as continuation of the project to market is expected, speed of creation is essential, for which an application is better suited, therefore for this project we have decided to create a mobile application.

### 1.4.2 Mobile Platform: Android vs iOS

An important consideration when creating a mobile application is deciding on which platform to choose. The two largest mobile platform providers currently are android and apple (iOS). Historically, iOS has dominated the market share, with a 42.02% market share in January 2011 compared to Androids 12.42% (Figure 1.4). Despite this, in recent years android OS has become more popular, even holding a greater share several times over the last few years and currently trails by only around 2%.

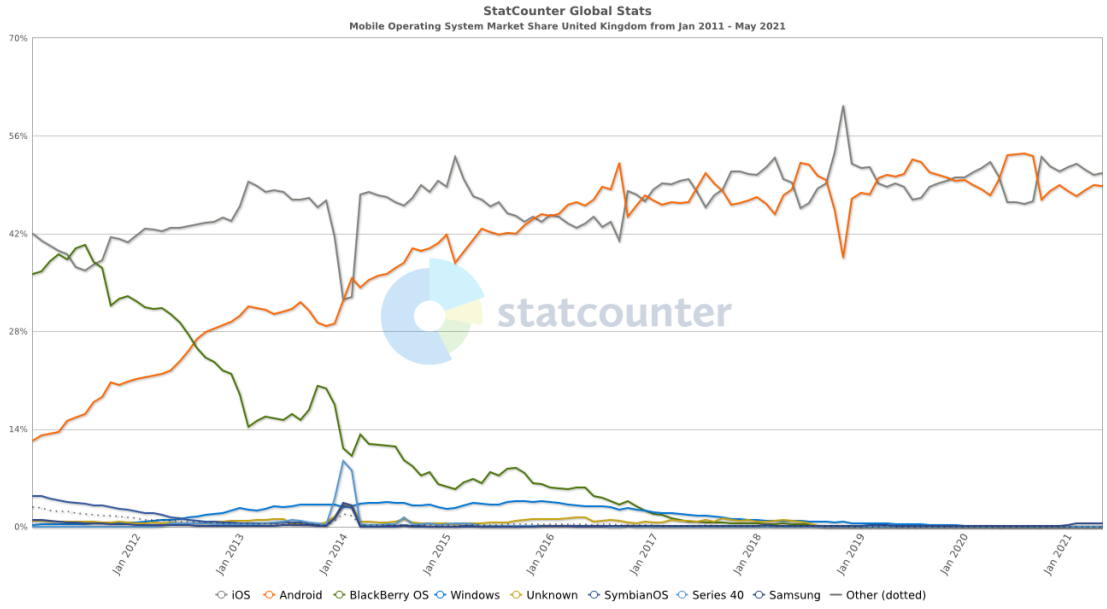



Figure 1.4: iOS vs Android Market Share Over The Last 10 Years [7]

With this change has brought with it a push towards frameworks that allow for development across multiple platforms, such as React Native [32] and Flutter [17]. For this reason, it was decided that a cross platform framework would be used, which is further discussed below.

## 1.5 Frontend: Programming Language

When deciding on the programming software, several metrics were taken into consideration, including cross-platform functionality, speed of development and perfor-

mance. For this reason, Dart and the corresponding Flutter software development kit SDK were chosen for the primary software. Flutter is a cross-platform development kit, meaning that it will natively run on both iOS and android applications created by Google [17]. Dart is compiled ahead-of-time into native ARM code giving better performance compared to other similar development kits, such as React Native and the user interface is implemented within a fast, low-level C++ library giving great speed to the application. Dart has also seen a large increase in usage within recent years, jumping up 532% from 2018 to 2019 [37] meaning that there is now a large community providing an extensive list of third-party plug-ins and support.

## 1.6 Back-end: SQL vs NoSQL database

An important decision when implementing the back end is to decide on whether to use a relational or a non-relational database. For the database, it was decided to use Google Firestore [10], a noSQL database that relies on nested documents within collections. This was chosen for several reasons. Firstly, as the chosen language Dart is run by Google, using firestore allows for greater integration and congruence with the platform and API. Compared to an SQL database, noSQL allows for repid scalability, which due to the future potential growth of the project is essential. The cloud firestore also integrates well with Firebases Authentication service, which is used throughout and discussed extensively in section 5

Another important feature of noSQL databases is the ability to easily modify the internal data in response to changing business requirements, in an interactive way that relinquishes the need for relationships between data. This means that as the project grows and requirements change, the database can easily adapt.

## 1.7 User Personas

The creation of user personas representing fictitious, archetypal users is an essential part of application development and UCD [28], allowing a deep understanding of the target user to be sought and implemented within the features and design of the application [8]. There are, however, some shortcomings to qualitative persona generation. Two of which are validity concerns and user bias [9] and although they are addressed by other methods, such as data-driven personas [25], require a broad user base. Therefore, here we have decided to stick with qualitative methods, which allow for enough brevity and depth for the scope of the project and the desired outcome.

Here we create 4 user personas that are used extensively in creating the software requirements set out within the next chapter and are discussed in detail below.

Persona 1 - Sarah Johnson
Profile:

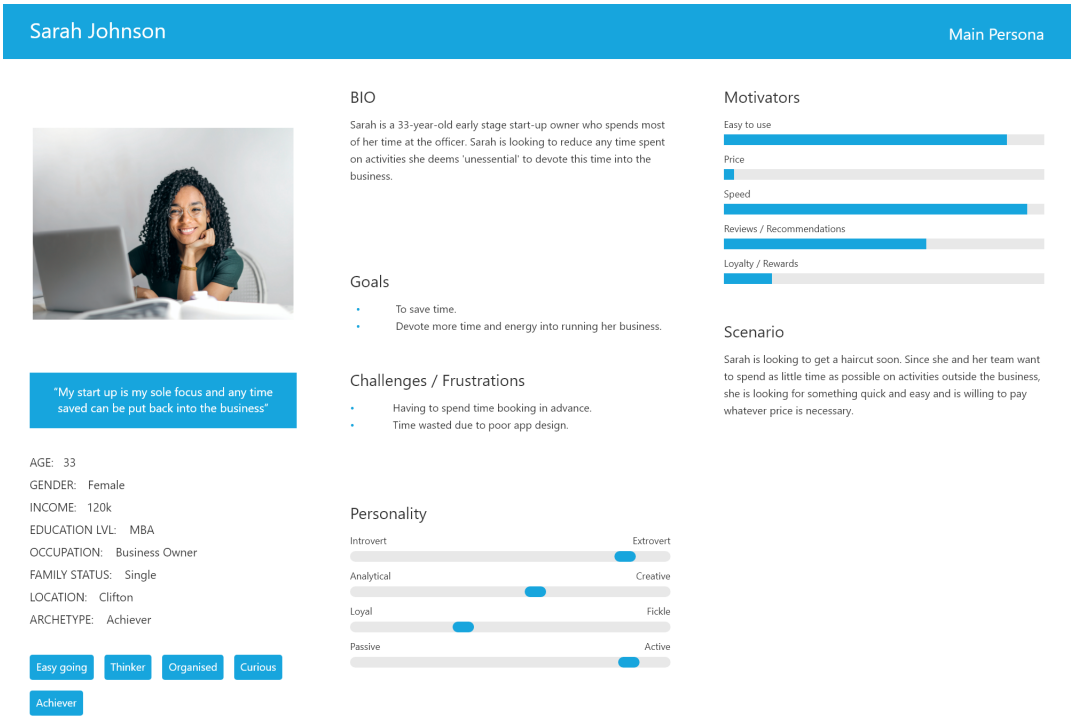



Figure 1.5: Persona 1

Persona 2 - Juan Smith
Profile:

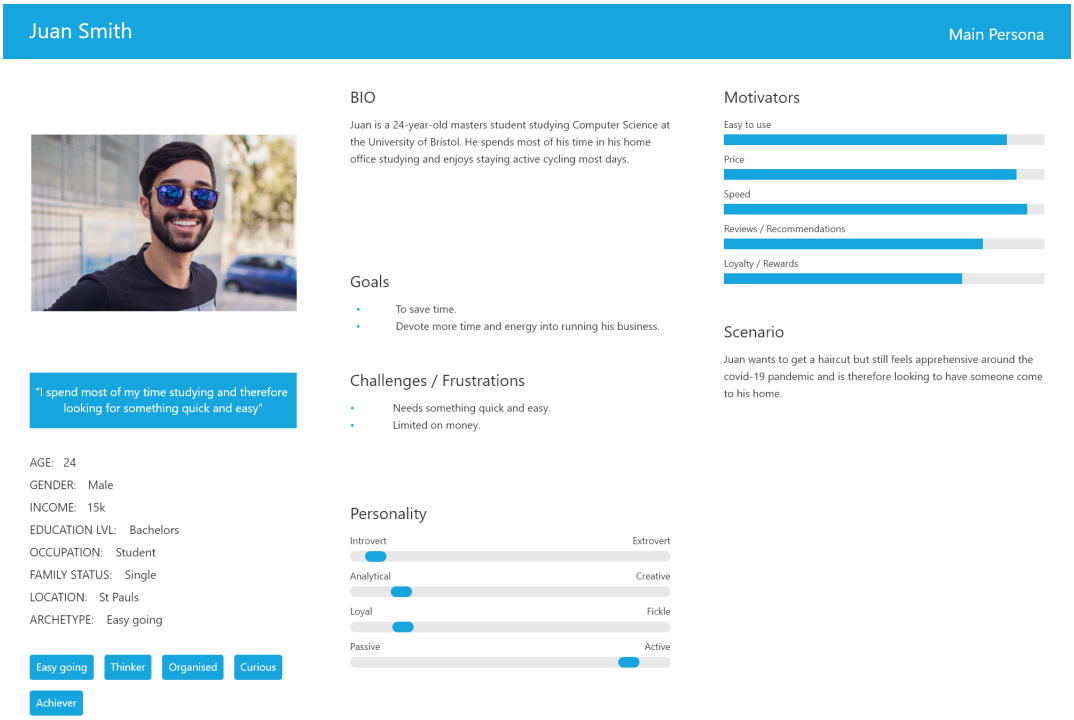



Figure 1.6: Persona 2

## Persona 3 - Emily White
## Profile:

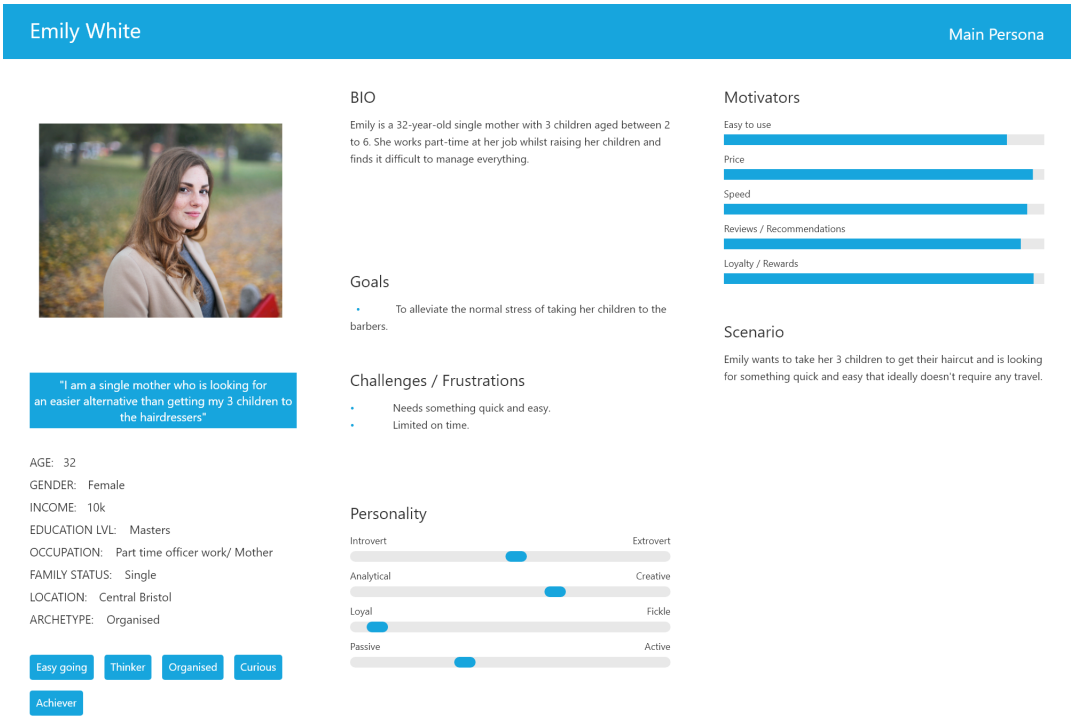



Figure 1.7: Persona 3

Persona 4 - Alastair Craig
Profile:

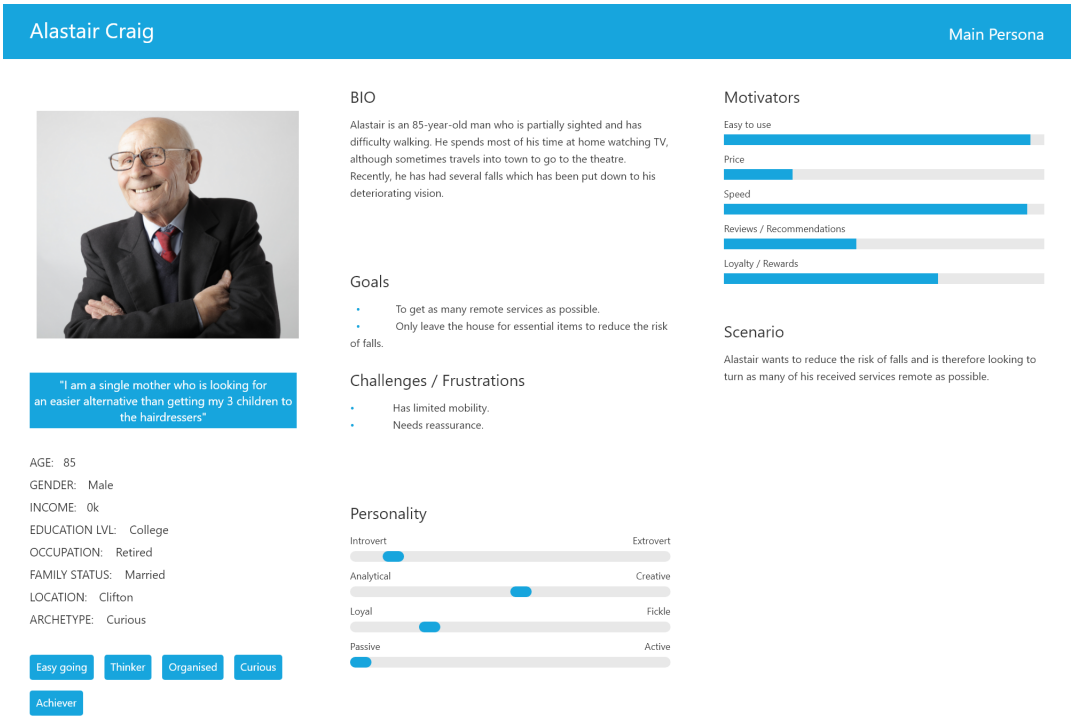



Figure 1.8: Persona 4

# 2 Software Requirements

Here we discuss and outline the software requirements, for which this section is similar to that found within an SRS document and lays the framework for the entire project. We define the application scope, user needs, functional and non-functional requirements, along with use cases.

## 2.1 Application Scope

The scope of the project was to create a fully working and functional barber application with several features, which are discussed in the User Needs below and were previously defined with the partner meeting in subsection 1.1. To further aid in scoping the project, epics were created, which were further split into stories that could be carried out. Although it could be argued that this type of agile methodology is more relevant when working within a team of developers, it helped to determine a stringent workflow and timeline and aided in project delivery. The scope was then further defined when wire-framing in Adobe XD, which allowed for the first tangible design to be made.

## 2.2 User Needs

• Allow the application to run on a mobile device.
• Allow the user to book a beauty treatment to receive at their home address.
• Allow a barber to set up an account and specify their product details.

## 2.3 Requirements

### 2.3.1 Functional Requirements

- Customer-side Application
  - The application shall allow a customer to create an account and login
  - The application shall provide the user with basic account management capabilities
  - The application shall allow for the user to pick from a range of relevant products and add them to their cart
  - The application shall allow the user full management of their shopping cart
  - The application shall provide only geographically relevant barbers and products to the user
  - The application shall allow the user to view their past orders
- Barber-side Application
  - The application shall allow a parent barber to create an account and login
  - The application shall allow for integrated back-end management of its barbers and products
  - The application shall allow for the parent barber to view its orders

### 2.3.2 Non-Functional Requirements

- Performance
  - The application shall take no longer than 3 seconds to load the users home screen
- Data
  - The application shall cache data where possible
  - The application shall minimise calls to the database and make them only when relevant
- Use-ability
  - The application shall follow Nielsens usability heuristics and be easily usable for the user without any guidance or help
- Security
  - The application shall ensure that all app data be secured and encrypted
  - The application should use OAuth for access delegation
- Operating System
  - The application shall run on both iOS and android devices
  - The application shall run on all devices newer than android 5.0 (API 21) - around 94.1% of android devices
  - The application shall run on all devices newer than iOS 9.0 - around 99.6% of iOS devices

## 2.4 Use Cases

Here the use cases are presented, which are described by Ivar Jacobson as "a description of a set of sequences of actions and variants that a system performs that yield an observable result of value to an actor." (Jacobson, et. al., 1999, p.41). Use cases are useful in the sense that they provide a structure for collecting customer requirements and setting the scope [23]. They also allow for validation of the project through post-production testing, which can be seen in section 6.

To produce the use cases we reference both the user personas created in subsection 1.7 and the functional and non-functional requirements discussed in the previous section. Below lists the most pertinent use cases.

- Use Case 1 - Sign Up
- Use Case 2 - Login
- Use Case 3 - Book a Haircut
- Use Case 4 - Search for a Barber

- Use Case 5 - Checkout
- Use Case 6 - View Orders
- Use Case 7 - Sign Out
- Use Case 8 - Add a Barber
- Use Case 9 - Add a Product

*The above use cases are further clarified in Appendix A.*

The next step is then to formalise these use cases into defined system behaviour, which is discussed below.

# 3 System Design

In this section, we discuss the structure of the project, the system and software architectures and data and state management given the previously specified requirements.

## 3.1 System Architecture

System Architecture can be broadly defined as a conceptual model which outlines the structure, behaviour and interactions between internal and external components of the system. Modelling and creating a structured and well-defined architecture allows for the development of a sustainable, scalable and stable software product which can easily grow relevant to the demands placed on its features.

### 3.1.1 Deciding on a Framework

There exists a variety of software architectural frameworks, although not exhaustive this includes Client-server, Peer-to-peer, Micro-services and Model-view controller (MVC). For this project, we have decided to split the architecture into 3 separate layers according to clean architecture principles[24]; the presentation layer, which implements an MVC design, the domain layer and the data layer, all of which are discussed below.

### 3.1.2 Modelling the Architecture

The overall system architecture can be seen below in Figure 3.1.

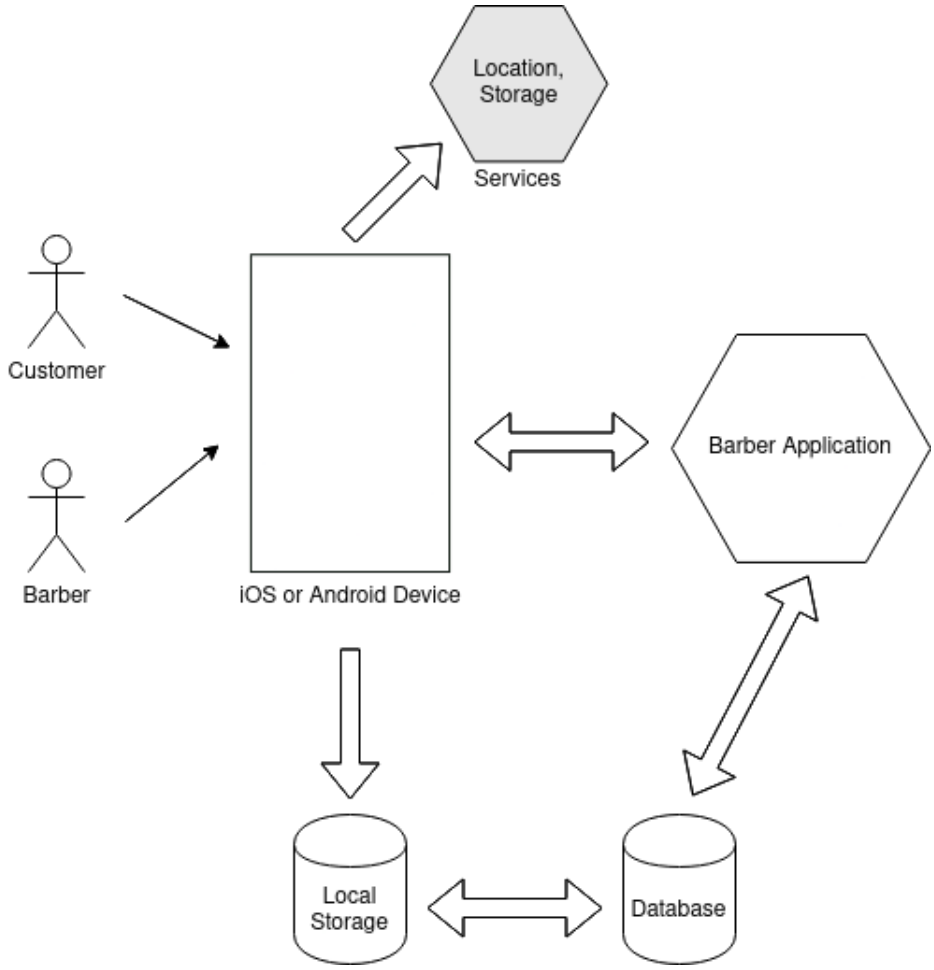



Figure 3.1: High-level System Architecture

When designing the architecture, core business logic was kept separate from the UI, database and network. For example, when interacting with the database the User Interface (UI) called upon the utilities package and any API calls where contained within the providers package. The project was also split into 3 layers, according to clean architecture principles [24] as follows:

- Presentation layer
  The presentation layer implements an MVC model, whereby they contain no logic, the screens act as the view, the models act as the model and the widgets act as the controller, containing basic logic that feed into the UI.
  - Screens - Contains unique UI elements that serve as the view
  - Models - Contains the models
  - Widgets - Serving as the controller; elements that are used to create the UI and feed into the screens

- Domain layer
  The domain layer contains more complicated business logic that is not specific to the UI. For example, in getTopRatedParents() within filter_list.dart, we find logic that pertains to filtering a list of ParentBarberModels based on rating.

- Logic - Business logic that does not implement the UI
- Providers - State management tools that are called on throughout the application.

- Data layer
  The data layer contains all the code for which data is passed through the application such as when making database calls or interacting with external APIs. For example, in the method getBarberById() found in barber_firestore.dart we make a call to the database to fetch the given barber from the relevant collection.
  - Utilities - Contains business logic and elements that are used to make calls to the database or interact with any external APIs.

This generally follows clean architecture principles [24] and each layer is represented in Figure 3.2 below.

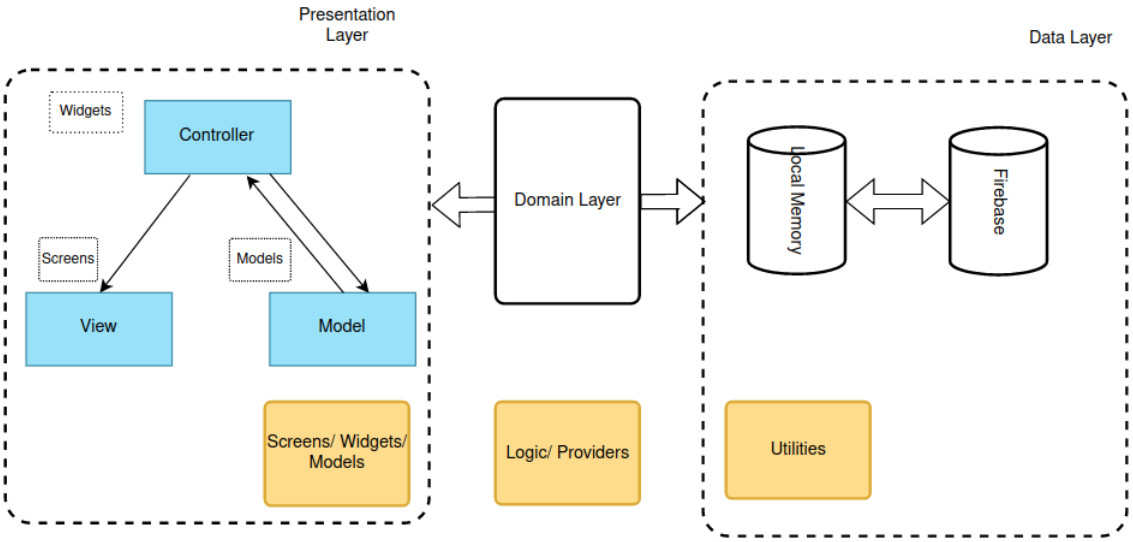



Figure 3.2: Clean Architecture (adapted from [24])

From this architecture and the previously devised specifications, an activity diagram was created, to detail the minimum required activities within the application, which can be seen in Figure 3.3 below.

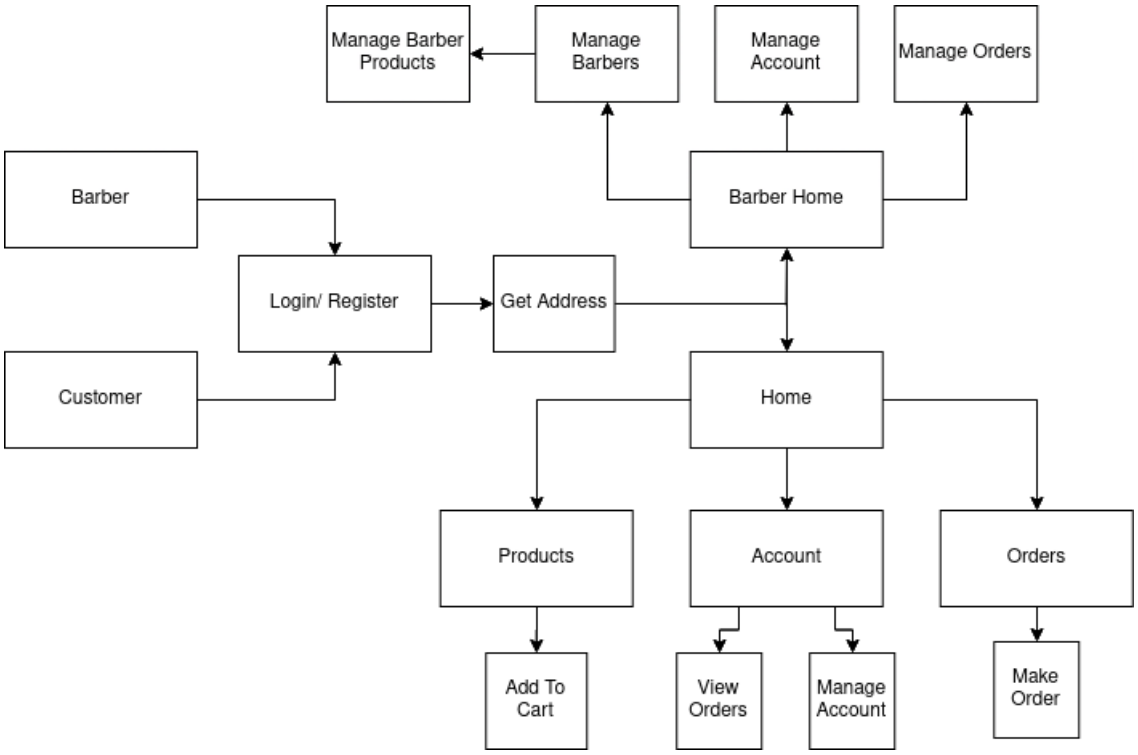



Figure 3.3: Application Minimal Activity Diagram

## 3.2 State Management

State management is an important feature within flutter. As flutter is declarative, rather than allow for changes in the widget or UI, each time a change is required the UI is rebuilt to reflect the applications current state. Within flutter there exists a variety of different methods for managing the state of the app, of which Provider, BLoC and GetX are some of the most popular and widely used. Here we will discuss the merits and pitfalls of each before settling on a framework for the application for both the local and global scope.

Provider

By far the most commonly used state management framework is Provider, a wrapper for the InheritedWidget class, which works by exposing all of the relevant daughter widgets to a value, so that data can be created, listened to and disposed of globally. To do this, a class is first created that extends ChangeNotifier, which allows classes subscribe to the senders data. Then, through using the method notifyListeners all of the daughter widgets will be updated with the current value and the UI rebuilt.

One negative of Provider is that it is only optimised for relatively few listeners, with it being $\mathcal{O}(n^2)$. With large applications and where speed is important this could be an issue.

BLoC

BLoC is a state management tool within flutter that is based on event driven states. For example, when adding to the basket you could trigger and AddBasketState, before

checking out and triggering a CheckoutState. The benefit of this is that the code becomes fairly inflexible, which in a team working environment could reduce the risk of accidental bug implementation.

BLoC is also beneficial in acting to separate the logic from the widgets, which aligns with the previously discussed architecture principles.

GetX

A large benefit of GetX is that, unlike Provider and BLoC, it acts not only as a state management tool, but as a ”micro framework”. For example, one does not need to access the widget tree and context to navigate between routes, allowing for seamless navigation between pages and also finding the object anywhere using Get.find(), allowing for more separation between the business and presentation logic. Due to not needing access to the widget tree, GetX removes the need for stateful widgets, allowing for dependency injection separate from using inheritedwidget, which removes unnecessary boiler plate code and aids in speed.

### 3.2.1 Provider

For locally managing state, setState() can be utilised within a statefulWidget, which was used extensively throughout the project, for example in the Checkout screen when updating the total to match the given products.

For managing the global state within the project it was decided that Provider would be used as the primary global state management framework for several reasons. Firstly, it is fairly simple to understand, and when used with ChangeNotifier allows for the required red state management for the proposed application. Secondly, Provider lends itself to a clean architecture with the logic separated from the UI, meaning that a simple widget tree can be designed and implemented, which can be seen in Figure 5.3 bellow, which displays the widget tree diagram for the application. Unlike BLoC, Provider allows for good flexibility, which as there was only 1 developer on the project makes more sense, giving greater scope for changes throughout the project. Despite this, future growth could lend itself towards also implementing BLoC alongside Provider, to improve structure and increase growth potential, a good example can be seen with Ebays Motor App [36].

Finally, although GetX makes an excellent framework for larger projects, due to its ”micro framework” that encompasses several features not limited to controlling state management, the fairly simplistic nature of Provider better suited the relatively small project and hence was chosen here. The overall state management can be seen in Figure 5.3.

# 4 Prototyping

An essential component of UCD and more generally UX design is prototyping, which involves making mock-ups of the application that act as early prototypes to influence later development [8]. Mobile app prototyping has many benefits to the project, including:

- Validates strategic design decisions of the product
- Saves time by discovering any constraints early in the project
- Allows for early, interactive user testing
- Acts as a template for the UI during the implementation phase

The prototyping for the application involved two distinct stages; firstly, an initial sketch was done with pen and paper to discern the layout and overall themes associated with the app, before a wireframe was created to further define the prototype and allow for early testing.

## 4.1 Initial Sketches and Brainstorming

Initial sketches involve using pen and paper to elucidate problems and brain storm ideas for the project. During this phase, several design and UI features were considered, which allowed us to generate many ideas, using a fast and simple approach.

The sketches representing the Login and Sign Up screens can be seen in Figure 4.1 below.

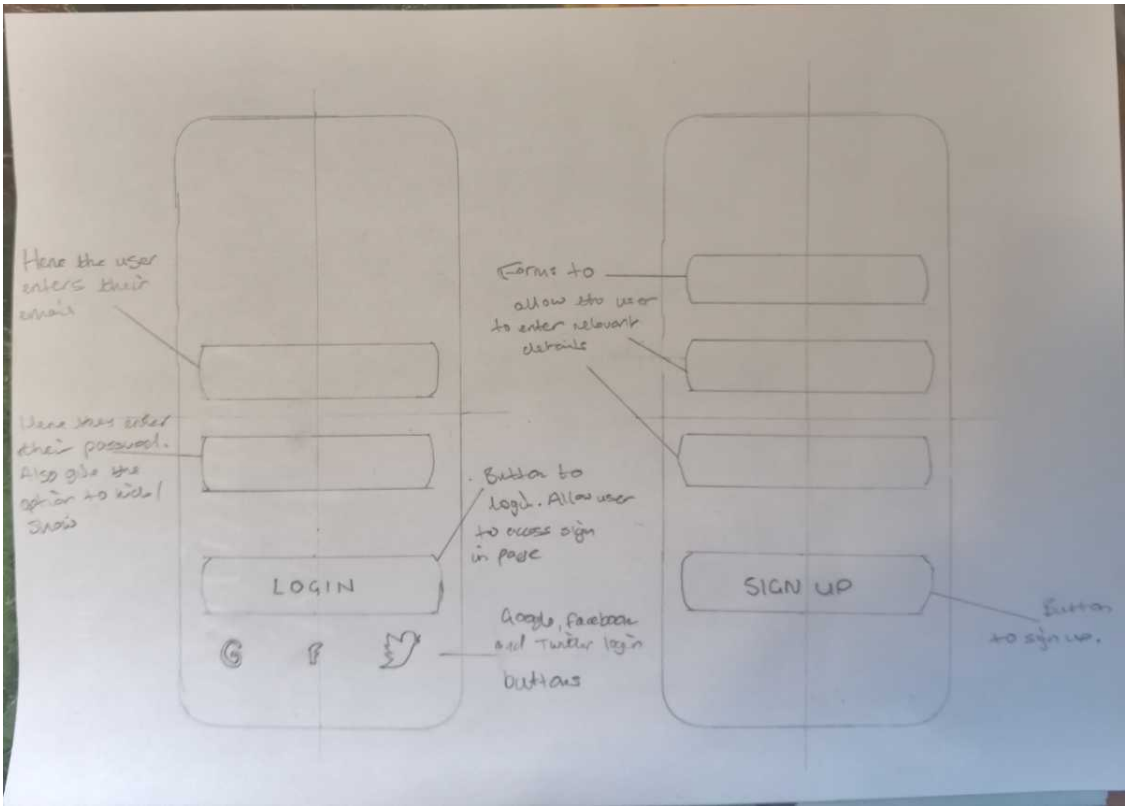



Figure 4.1: Pen and Paper Sketches and Brainstorming

## 4.2 Wire-framing

Once the sketches were completed we moved on to wire-framing the application, which involved taking the best sketch variants and creating a more detailed, lower level prototype.For the wire-framing application Adobe XD was chosen for several reasons. Firstly, it has strong prototyping functionality, allowing the user to click around the application through the use of 'components'. This interactivity means that early testers can get a real feel for how the application works. An illustration of this can be seen below, whereby each arrow represents a state change in the form of a trigger/ action pair, whereby for example a user could click on 'Available Right Now' and be taken to the 'Checkout' as seen in Figure 4.2 below.

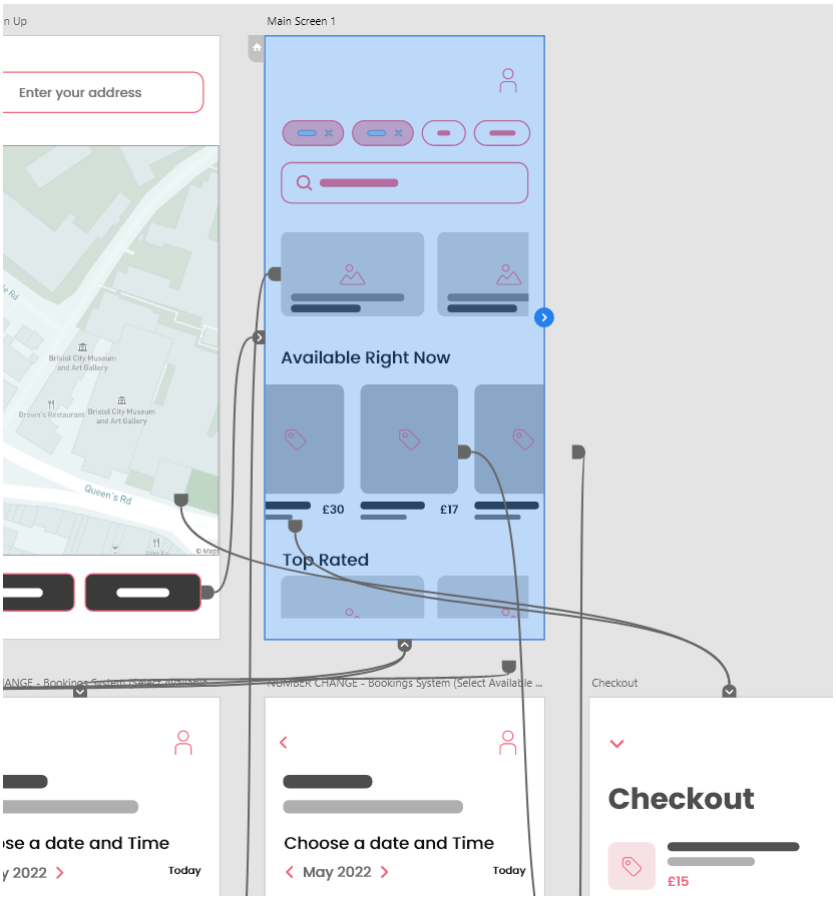



Figure 4.2: Component Interactivity within Adobe XD

Adobe XD also allows for easy distribution of the prototype in the form of a shareable link that opens in the browser and encompasses the same functionality and components that can be found within the application itself, meaning that anyone with access to a browser can test the prototype. Along with this, the prototype also allows for comments to be made, which are fed back to the owner. This comment capability was used early on during beta testing when it was sent out with the early questionnaire and influenced initial design decisions [34].

When designing the screens there was a strong focus on user experience following Nielsons 10 Heuristics for User Interface Design [15] . For example, the functionality was kept as minimal as possible to avoid clittering and avoid cognitive load on the user, the user was given control to go back and forward between previous screens to

allow for user control and freedom and simple and self-explanatory language was used to apply recognition over recall. For example, the Sign In screen below extraneous text was kept to a minimum by using images for the login items, such as Google, Facebook and Twitter, a sign up button was included to allow the user to access the application through creating a new account and large, clear sign in forms and buttons were used. The full interactive Adobe XD wireframe can be found here.

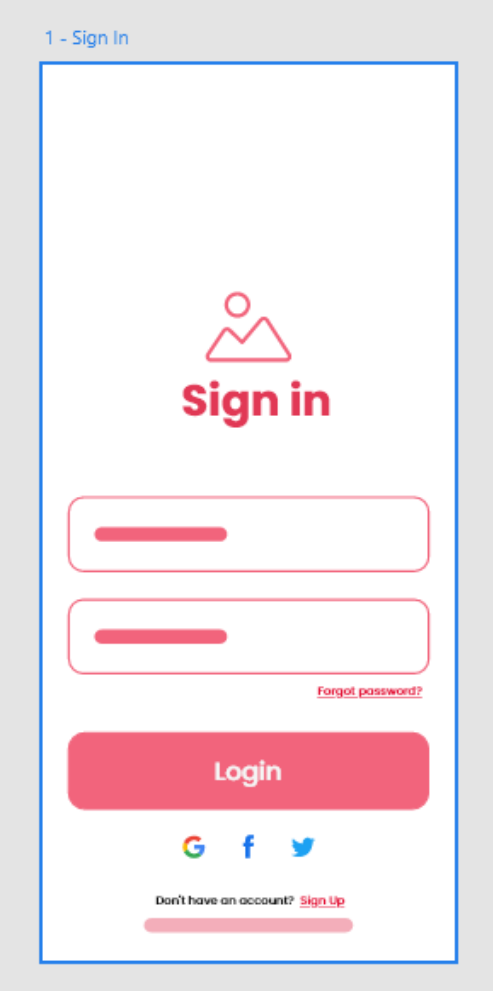



Figure 4.3: Sign In Page Made With Adobe XD

# 5 Implementation

Here we discuss the implementation of the application, which is divided into separate sprints, each of which pertains to a relevant feature within the application. Finally, we address the implementation each use case individually.

## 5.1 Setup

Before beginning the development sprints, the first task involved setting up the developer environment. To code the application it was decided that an integrated development environment (IDE) was used due to the comprehensive features it provides to aid in development and maximise productivity. For this, it was decided that the IDE Android Studio by JetBrains [14] was to be used, due to previous experience and familiarity with other JetBrains applications, along with Android Studios excellent built in features and seamless integration with flutter.

### 5.1.1 File Structure

Although there is no official recommendation for structuring the app, here we follow a commonly used scheme which aligns with the previously discussed architecture principle and includes models; the files that serve as collections of data that are used in conjunction with the widgets to form the user interface of the application; providers, which inhabit the state management tools of the application; screens, which display the UI of the app; utilities, which are used to connect to the back-end; and widgets, which contain the business logic of the app.

## 5.2 Sprint 1 - UI

The first sprint involved implementing the UI, which was previously wire-framed within Adobe XD. Flutter offer an Adobe XD plug-in to turn wireframes directly into code, however, this was not used for several reasons. In Adobe XD components are positioned absolutely, whereas in Flutter it is done relatively, leading to several issues with positioning that would not scale. Adobe XD also does not contain customer properties and therefore mapping these to components, such as title is not possible, therefore the UI was implemented manually.

Within Flutter, the UI is built through using widgets, which describe not only the look of the application, but also provide the state and can be rebuilt to reflect a change in state. As an example we can look at the forgot password screen, a segment of which is shown in Figure C. The UI here is built using a variety of widgets. For example, line 2 shows a Column widget, which allows for daughter widgets to be stacked alongside each-other vertically. The first daughter widget is a TextField that allows us to get text from the user and this is wrapped within a padding widget, which allows us to specify the required padding to aid in UI design. Finally, a button is placed within the application using the GestureDetector widget, which, by implementing the onTap function looks for input from the user and then navigates to the Login screen after using the AuthenticateProvider to send a password reset link.

### 5.2.1 Cupertino vs Material

A consideration when building the UI was on whether to model the application using Cupertino, which gives the app an iOS type look and feel or material, which is more generalised across android and iOS. As the project was designed to be multi-platform, it was decided that material would be used throughout.

Using material also lends itself well to the project for other reasons, For example, it gives us access to a number of useful widgets at the root of the application. For example, the Navigator allows us to keep a track of the users chosen screens as a stack and by using Navigator.pop() and Navigator.push() we can navigate between screens. This is implemented in the Navigate widget, which can be seen in Figure C and allows us to easily navigate the user around the application. Using material also gives us access to both a bottom navigation bar, which can be seen in Figure C and a side bar. Finally using material allows for easy styling of the app by using the built in class ThemeData. The implementation of material can be seen in the main.dart file in Figure C which serves as the root and entry point to the application.

## 5.3 Sprint 2 - Sign Up and Login

Once the UI was built the Login and Sign Up page logic was implemented. This involved creating an authentication page (authenticate.dart), which acted as a provider for the user and other authentication logic that could be injected into the UI along with using a database to store users so that user data could be persistent.

### 5.3.1 Authentication

Firebase has its own built in authentication library [16], which works by providing email and password based authentication along with OAuth 2.0 capabilities, both which were used extensively throughout. Firebase Authentication was used for the project due to several considerations -

- Excellent built in security features
  - Can easily restrict access to different specified groups within an organisation
  - Security is enforced by server-side rules, limiting unsafe usage within the app itself
  - Uses token generation to ensure confirmed data
- Integration with firebase database and storage
- Easy modification through Googles declarative language
- Excellent integration with OAuth 2.0
  - Using OAuth allowed for sign in methods with Google, Facebook and Twitter to align with the previously drawn wireframes
- 99.999% SLA

### 5.3.2 Sign Up

The user is first presented with the sign up screen whereby they can enter their name, email and password and click through to make an account. Once the user clicks the Sign Up button, a token is created within Firebase Authentication and the credentials are stored. Inside of authenticate.dart there is a signUp method which carries out the logic behind user sign up, which works in several steps.

- First, a UserCredential (which holds the return value of firebases sign up method) is returned from an attempt to create a new user with the email and password
- Next, a created model (UserModel, which can be seen in Figure C) is assigned to the user which holds all pertinent information, such as name, email, shopping cart items etc
- The Authenticate enum status is set to AUTHENTICATED as discussed below

### 5.3.3 Login

Once the user has been created and the credentials are stored, the user can login using the given name and password. This is then authenticated with Firebase and a response is returned to the client, before the user is then logged in for the session. To follow the authentication status of the application we created an enum seen in subsubsection 5.3.3 below. This can then be tracked by using Provider.of<AuthenticateProvider>(context), allowing us to manage and alter the state of the application based on the status of the user.

```
enum AuthStatus {
  UNINITIALISED,
  UNAUTHORISED_USER,
  UNAUTHORISED_BARBER,
  NOT_AUTHENTICATED,
  AUTHENTICATING,
  AUTHENTICATED,
  BARBER_AUTHENTICATED,
  AUTH_WITH_MAPS
}
```

Figure 5.1: AuthStatus Enum Found in authenticate.dart

The user can also be authenticated and signed in using Google sign in; for which the google_sign_in package was utilised, Twitter sign in; for which the package flutter_twitter_login was used and Facebook sign in; whereby the package flutter_facebook_auth was used. All of these work by returning a respective OAuth token and therefore is a secure way to authenticate the user. Although all follow similar methodologies, each was implemented separately, an example of which Google is discussed below.

To implement the Google login, we first set the AuthStatus to AUTHENTICATING. Next, we initiate the Google sign in widget by calling the .signIn method our instance of GoogleSignIn and returning this to a GoogleSignIn account, before returning an OAuth token in the form of a GoogleSignInAuthentication. The auth result of this is then

fed into a method which creates a user below. Together, this therefore authenticates the user using FirebaseAuth, creates a locally stored user with these credentials and persistently stores them within Firebase.

Similar to when signing up, on login, a User Model is created, which locally stores the details needed for the user, i.e. id, name, email, address and any other relevant data such as their cart items within the Authenticate class. In this way, the data contained within can be accessed using Provider and data such as cart items accessed globally. Part of the user model can be seen in Figure C.

### 5.3.4 Sign Out

To sign out, we call a method within navigate.dart, which carries out several functions. Firstly, as discussed in subsubsection 5.6.2 we store the users shopping cart items persistently, before we logout of each of the authentication methods, such as Google, Twitter etc. Within navigate.dart, we then clear the Navigator stack, which uses a RoutePredicate (route => false) that always returns false, hence will remove all of the routes on the stack, before pushing the UserType() screen. This therefore returns a blank stack and navigates the user to the home screen. This code can be seen below in subsubsection 5.3.4.

```
1      void clearNavigator(BuildContext context) {
2        Navigator.pushAndRemoveUntil(
3        context,
4        MaterialPageRoute(builder: (BuildContext context) =>
5        UserType()),
6        (Route<dynamic> route) => false
7        );
8      }
```

Figure 5.2: clearNavigator() function found in navigate.dart

### 5.3.5 State Management

As previously discussed, flutters built in Provider was used for the state management required throughout the application. As seen in Figure 5.3 - which displays a widget tree diagram for the application. Two providers, AuthenticateProvider and ParentBarbersFirestore are passed down the chain and are used to access relevant variables and data types globally. Within Authenticate provider the current user is stored, therefore giving access to their details, along with previous orders and current shopping cart items. Due to flutters declarative nature, this means that the state is kept above the widgets currently using it. The widget tree begins with MyApp, which is the entry point to the application and found in main.dart. We then initialise a MultiProvider, which allows us to instantiate several Providers to pass down the chain throughout the application.

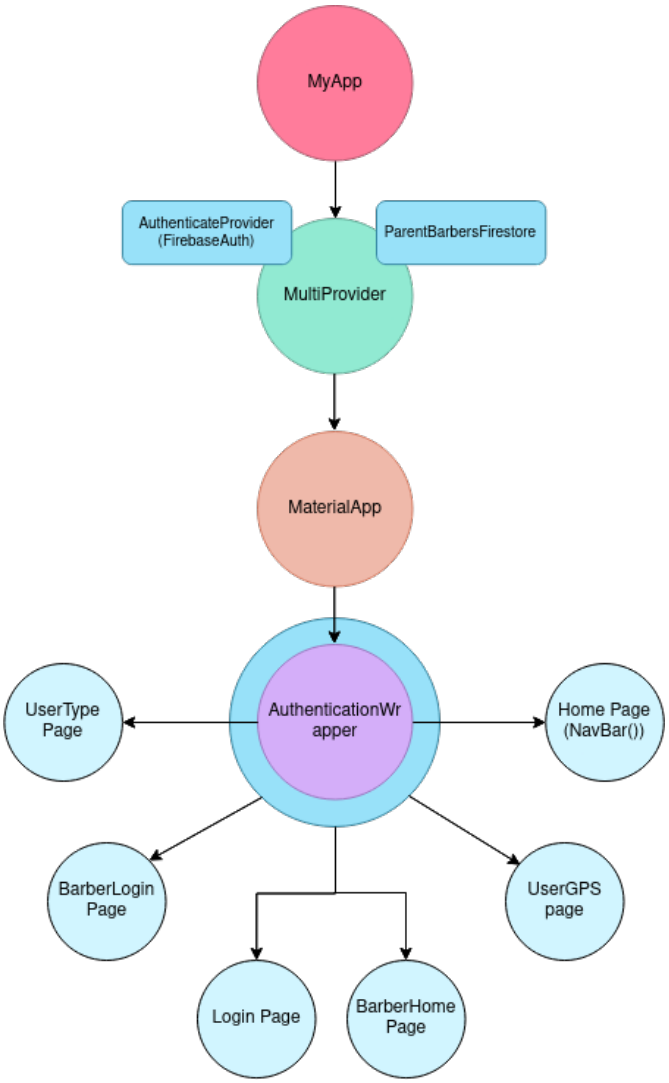



Figure 5.3: Widget Tree Diagram

## 5.4 Sprint 3 - Back end Setup and Modelling

The next sprint involved modelling and creating the database to suit the needs of the project. As previously discussed, it was decided that a noSQL database, instead of a relational one was to be implemented due to the aforementioned benefits it provided.

### 5.4.1 Setup

The setup for firebase was simple and involved several steps. First a project was created, initialised and registered on firebase.google.com. Next, the plugin cloud_firestore was installed with flutters built in pub get function. For security, firebase has its own language that allows us to implement rules that regulate access for both reading and writing to the database. This works by the given rules pattern matching against database paths, so for example in our database, if we only wanted to give read and write access to the /users paths to logged in users we would write

```
...
  "users": {
    "$\uid": {
      allow read, write:
      if "auth.\uid == $\uid"
    ...
```

As the project was only to show proof of concept it was decided that, rather than specify specific rules, only authenticated users could gain access to the entirety of the database. This is shown below, which displays code giving the user access to any documents within the database so long as they are logged in.

```
  service cloud.firestore {
    match /databases/{database}/documents {
      match /{document=**} {
        allow read, write:
        if auth != null
      }
    }
  }
```

### 5.4.2 Database Design

Although noSQL is most frequently used for non-relational data a database schema was constructed to constitute the parent barbers, barbers, users, orders and products as modelling this way added to readability with the added benefit of the features previously discussed. The schema is represented in Figure 5.4 below and shows each class with a primary key and foreign key where necessary.

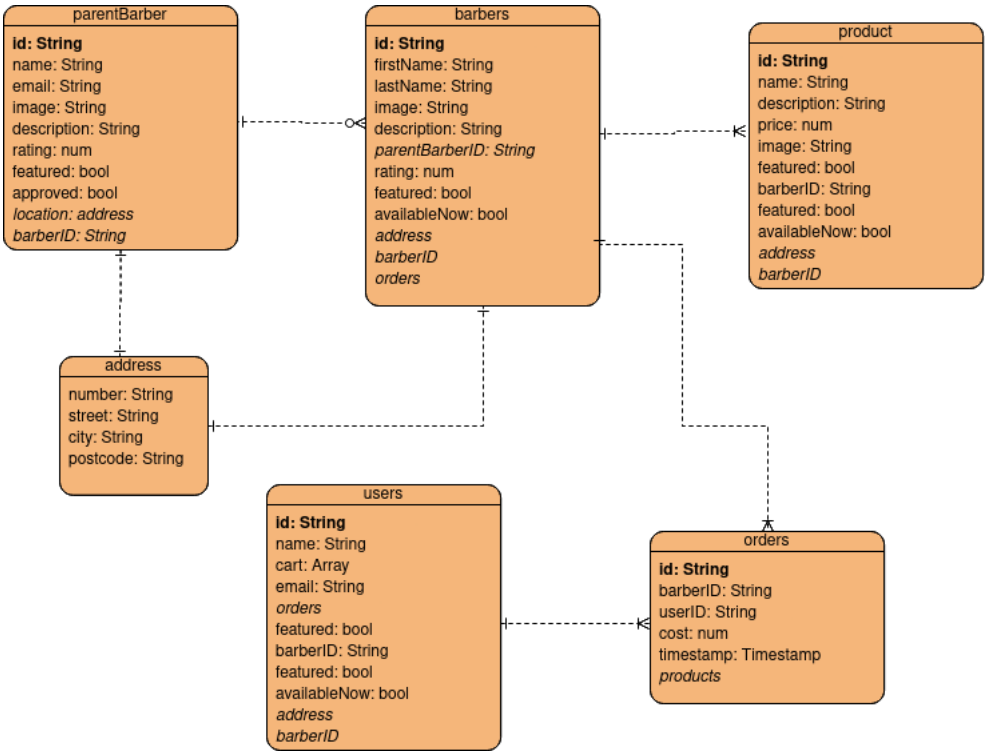



Figure 5.4: Database Schema

### 5.4.3 Creating the Models

As previously mentioned, within the project we used models to represent the data layer within our architectural structure. These were portrayed as individual classes, each of which contained minimal, only necessary logic relevant to its inherent functions. This is shown in Figure 5.5 below, which shows the model class diagrams and their relationships.

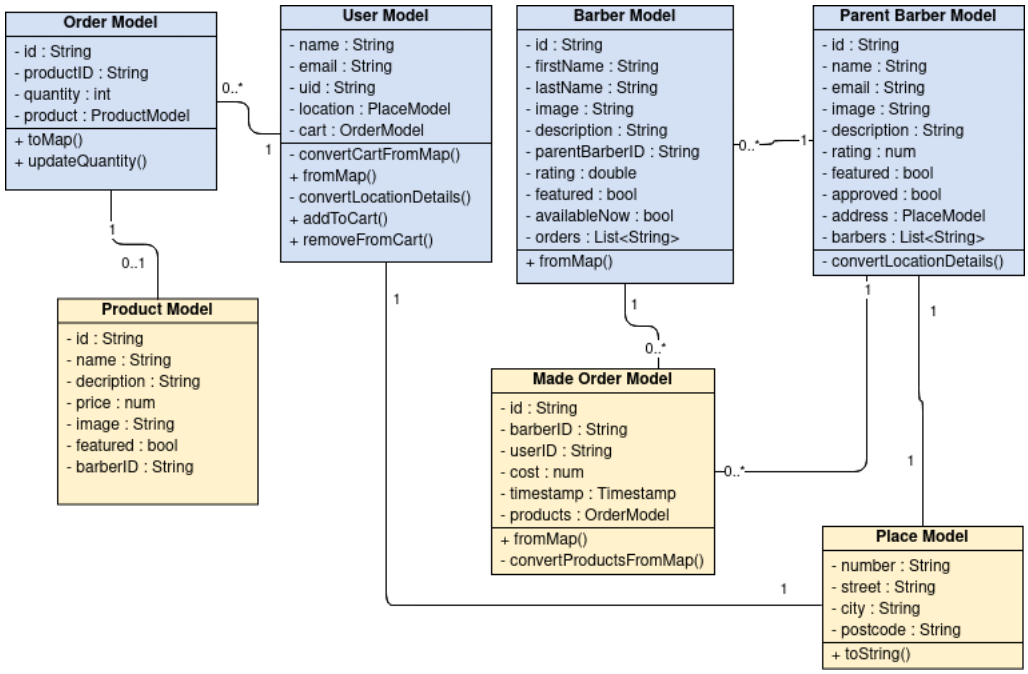



Figure 5.5: Model Class Diagrams

Each model also contains a .fromSnapshot function. DocumentSnapshots are data containers that hold undefined documents from firebase. To access this, we make a call using a CollectionReference, which is simply a reference to the data location within firebase, before defining the incoming data and specifying its type. Having a .fromSnapshot function within each model allows us to simply call the function on the DocumentSnapshot to explicitly define the data type and then load it into memory. An example of this can be seen in Figure C, which also uses 2 created methods; _convertOrderFromMap(), which converts the incoming data into an array of OrderModels and _convertLocationDetails(), which turns the data into a PlaceModel.

### 5.4.4 Avoiding Nested Data

A consideration when creating the database was to optimise queries for fast lookup time, as this was one of the main considerations in choosing a noSQL over SQL database. For this reason, when storing data, the id was stored as the document id. For example within the UserFirebase class, for the createNewUser() function, we pass through the authentication uid, which is then used as the document id. This means that rather than querying with a call such as

```
1     firebaseFirestore.collection(collection).where(\uid, .
          isEqualTo(givenID)).get()
```

which does not scale well due to a search time of $\mathcal{O}(n)$. Instead, we store the uid as the document id, allowing us to do a similar, although faster call such as

```
1     firebaseFirestore.collection(collection).doc(userId).get
          ()
```

which gives a search time of $\mathcal{O}(1)$.

## 5.5 Sprint 4 - Connecting the Frontend and Back end

Now that we have built a frontend UI, the database has been created, along with models to locally store data, it is time to connect the two.

### 5.5.1 Loading the Parent Barbers

After the user is logged in they are presented with the main screen containing 3 seperate widgets; Featured, which presents the user with 2 parent barbers; AvailableNow, which shows a list of the barbers that are immediately available; and TopRated, which displays a column of the 5 most top rated parent barbers. Each of the 3 elements is contained within its own widget as per the previously discussed clean architecture principles. All of the logic that connects the frontend and back-end are placed inside of the logic folder, with the logic only supporting the UI being contained within the widgets folder, which acts as the controller within the MVC component of our architecture.

When loading the parent barbers, initially this was done with 3 separate calls to the server. An example of this is when getting the parent barbers for the TopRated widget, the following database query was made:

```
1     firebaseFirestore.collection("parentBarber").orderBy("
          rating", Direction.DESCENDING).limit(5).get();
```

This made a call to the collection parentBarber, ordered by the top 5 rated parent barbers and fetched the resulting data. As there were 3 different widgets this resulted in 3 separate calls to the database. This was later changed in favour of fetching all of the parent barbers first before filtering within memory. This aided in not only increasing speeds due to local filtering, but also reduced the number of needed calls to the database, further reducing speeds and costs. An example of this filtering can be seen in the filter-list.dart file in Figure C.

This is done in the body of the parent_barbers.dart class and therefore is run when the class is instantiated. This is run in main.dart using a ChangeNotifierProvider as shown below and therefore means that the data is loaded as soon as the application is run and can also be accessed globally throughout the application along with the descendents being updated and rebuilt with any change in the widget.

```
1     ...
2     ChangeNotifierProvider<ParentBarbersProvider>(
3     create: (_) => ParentBarbersProvider(),
4     ),
5     ...
```

*Note: Loading the parents also involved filtering via location, which is discussed in detail in subsection 5.7.*

### 5.5.2 Loading the barbers

Once the parent barbers are loaded we must now load the individual barbers, each of which have 1 parent barber. This can be done in two ways; either through making a

database query based on the parent barber ID and fetching only the relevant barbers for each parent barber loaded in memory, or through loading every available barber into memory and then filtering in situ. Each method comes with pros and cons which are discussed briefly below.

Database Call on Parent Barber ID

As previously mentioned, firebase has a lookup time of $\mathcal{O}(1)$, therefore making individual calls to the database will scale well as it does not matter how many barbers there are within the system. Another benefit is that only the relevant barbers will be loaded into local memory, saving on valuable space. The disadvantageous side to this method of fetching the barbers, is that it will require several database calls, depending on how many parent barbers there are and therefore add to costs.

Loading all Barbers and Filtering Locally

Loading all the barbers and filtering locally is beneficial in that it is cheaper due to less database queries, however, the negative is that it does not scale well, due to having to load more and more barbers as the application grows. However, as further discussed in subsection 5.7, we filter first based on location and therefore greatly limit the amount of relevant barbers.

As the second method works out cheaper and due to the location restrictions scales well, we use this method to fetch the barbers.

## 5.6 Sprint 5 - Shopping Cart, Checkout and User Orders

Sprint 5 involved creating and implementing the logic for the shopping cart along with allowing the user to checkout.

### 5.6.1 Adding an Item To the Shopping Cart

As previously stated, within our UserModel class we store an array of type OrderModel, which represents the users cart. The OrderModel is simply a class with a price (so that each barber can set the price for the product individually), an int representing quantity, a product ID and a class of type Product Model to represent the product. The product_details.dart holds the UI for the selected item to be added. To add a product to the cart, the user first selects the desired quantity, which is implemented through a simple local setState() method, they then click the button 'Add Quantity To Basket'. Within the product_details.dart file, we use a Provider.of call to fetch the user and call the method addItemToCart, which first checks the current cart to see if the item already exists, before creating a new OrderModel from the item requested, and adding the order model to the array within the UserModel. All of this logic can be found within the order utility (order.dart).

### 5.6.2 Storing the Shopping Cart Items

When the user logs out the cart is automatically stored within local memory, however, here we also give the application the ability to persistently store the cart, so that each

user will be returned to the previously left cart state. An example of how we do this is found in Figure C, a method found within order.dart. To do this, we fetch all of the items within the cart, which are currently represented as an array of OrderModels. Next, we traverse the array and use a created method to turn the class items into a map, before uploading to firebase using the .update method, so as to only overwrite the cart document. Consequently, within getDatabaseCartItems() in orders.dart the cart items are then fetched when the user logs in doing the inverse and taking a map of items from firebase and assigning them to an OrderModel, before adding them to the users cart array.

### 5.6.3 Displaying The Shopping Cart Items

Throughout the application you can see a small icon in the top right corner representing a shopping cart, which when clicked will take you to the checkout screen (checkout.dart). Here, we implement a ListView builder, which, using a Provider call to AuthenticateProvider allows us to traverse the shopping cart and display certain elements of our cart item class at position position [index], for example the image and price etc. As we are using Provider, in this screen we are also able to change the quantity of the required product and delete items, which is reflected across the application.

### 5.6.4 Checking Out an Order

Similar to storing the cart items, each user had an array of items representing the orders. The type was of class MadeOrdersModel, which each contained a list of OrderModels, along with a timestamp and the user and barber IDs. When a new order was made in cart.dart, a call was made to the function createNewOrder(), within order.dart. Here, an ID was created using UuiD v4 (as recommended by Google), the order was then added to firebase within the orders collection. Similar to subsubsection 5.3.1 to create a search time of $\mathcal{O}(1)$ a separate orders collection was created, with each document representing all of the pertinent order details. Within each barber and user, the document (order) id was then added to each respective order arrays, for quick and easy lookup.

## 5.7 Sprint 6 - Location

The next sprint involved methods related to location, including fetching the users address and using it to filter barbers by location.

### 5.7.1 Autocomplete locations

After the user signs up, they are met with a map page which allows them to enter their address in the form of a postcode or first line. As a means for the user to autocomplete their address when signing up, the Place Autocomplete service within the Google Places API, which returns location predictions in response to HTTP requests was implemented using a request adhering to a set of parameters. The full list, along with details of the API can be found on the Google Developers website [29]. First, we enable the Places API within the Google console, before we then create a location

model which can hold the parsed data returned from the API. We create an API request using the below aforementioned API format.

```
https://maps.googleapis.com/maps/api/place/autocomplete/
   json?input=$input&types=address&components=country:uk
   &lang=en&key=$apiKey&sessiontoken=$sessionToken
```

For brevity, not every option is discussed, but those of importance include input, which is the user query, types, which determines the query returned, for which we specify address as we wish to fetch the users full address and a session token, which is required for each new query.

The returned results are in json format and after some minor error checking we parse using json.decode into a list with our previously created LocationModel class, whilst assigning a new UuiD for each query (Google recommends to use version 4 UuiD and so this is used here). The full code that is used to make the API call can be seen in Figure C.

For the content of the search page we use pass in newly created session token into the ShowSearchPage class, which in turn sends an API request and parses the json data to return a list of locations in the form of place ids and description using a Future-Builder that allows for asynchronous calls. From here, we pass through the location id to the getLocationDetails function to fetch the address details of each location and put into a PlaceModel object.

Next, we parse the data into JSON format by passing the PlaceModel object into the function createLocationMap, which creates a map using the location data. Finally we pass through this map to the addLocationDetails, which uses the given user id to update the database with the users location.

### 5.7.2 Filtering the Parent Barbers

The next step involves using the users location as a means to filter barbers. As is discussed below in subsection 5.9, once the barber creates an account they give their location in the form of their address. This is stored in geohashes, which are longitude and latitiude co-ordinates that are hashed into a single Base32 string. Each character presents a greater level of precision and therefore we opted for 9, which represents an area of 5 x 5 meters. With this, we can use the data to filter out parent barbers when a user first logs in after grabbing their location. User location access is granted through the following line in the ./android/app/src/main/AndroidManifest.xml file

```
<uses-permission android:name="android.permission.
   ACCESS_FINE_LOCATION"/>
```

Therefore, when the user first logs in, all of the parent barbers are fetched based on their geolocation relative to the user. For this, we use the plugin GeoFlutterFire [18], which was chosen due to it being open source, light-weight, but also having the ability to allow for queries instead of CollectionReferences, which means that it will allow for growth with the application. Within getLocalParents() in location_firestore.dart, we make the following query:

```
Stream<List<DocumentSnapshot>> stream = geoflutterfire.
   collection(collectionRef: _collectionReferenceParents
   ).within(center: center, radius: radius, field:
   location);
```

This call returns a stream of DocumentSnapshots, which, as previously mentioned are references to unspecified data within firebase, which we then parse to create a list of ParentBarberModels. This list will contain only the search results within *radius* and we then filter based on other relevant metrics i.e. rating, featured etc.

## 5.8 Loading the Users Orders

In user_orders.dart we run a nest ListView Builder where we fetch and display both the, order, which includes information such as the price and date, along with each of the products contained within each order. Alongside this we also display the barber, along with some details. To fetch this, we traverse through the barber ids and fetch the relevant barbers using a function getBarberIds() found in barber_widget.dart.

## 5.9 Sprint 7 - Barber Side Application

In order to create a comprehensive application the final sprint involved coding a barber side app, giving the ability for the barber to interact with the database and allow them to create an account, add and remove barbers and view orders.

### 5.9.1 Allow the Parent Barber To Create An Account

The first task was to allow the parent barber to sign up for an account. This was done by first splitting the application into two on login, so that the user specifies if they are a barber of a customer (future implementations will have these split into a separate application). The user then selects barber and is able to sign up in barber_registration.dart. Similar to the customer sign up, this involves the user filling in several text forms, but also allows for an image to be uploaded for the barber, the code for which can be found in Figure C. As the location functionality of the application requires both longitude and latitude co-ordinates for the barber, along with a geohash we needed to fetch this information from the barber. To do this, we first get the location of the user through the previously implemented Google places API call. With this data, we then convert this into Geohash data, before uploading to firestore, along with the other pertinent barber data.

### 5.9.2 Logging In and Adding a Barber and Their Products

Once the parent barber has created an account they are able to login similar to how a customer would. Within authenticate.dart there is a method named barberSignIn() which authenticates through firebase and then using a ListenableProvider in main.dart returns the barbers home screen. Within the home screen the barber has 4 options; add barber, delete barber, view orders and sign out. When the parent barber clicks on add barber they are taken to a screen similar to the sign up screen, with several input forms that allow for the creation of a barber. The parent barber also can choose from a list of barber products to add to the barber, along with being able to add a description and price to each. Once this is added, the product is parsed and turned into a ProductModel, before being added to an array. Once the parent barber then clicks create barber we use a previously created Uuid v4 that is assigned to the barber and

then used to link them to each product. The entered details, along with the products are then added to their respective collections within firebase.

## 5.10 Widgets, Common Items and Added Features

Here we discuss any items not covered within a specified sprint.

### 5.10.1 Widgets

Several widgets were used to increase readability and brevity of code. For example, return_text.dart allows for access to the main components of the Text function and return_image.dart allows for easy use of the NetworkImage function, giving brevity and readability to the code.

### 5.10.2 Common Items

The common items contains global variables that were accessible throughout the project. Initially this included structures and arrays that served as objects to test the functionality of the frontend, for example a barber shop class with a nested list of barbers classes, each with a name, age, description etc. As back-end functionality was added these items were removed. A theme class was then added which contains dart files that can be implemented. Doing it in this way meant that the application could be easily styled, without any unnecessary refactoring of code.

### 5.10.3 Notifying the User

Throughout the application, the user is notified of any pertinent error or status message, for example when their login fails, or adding an item already existing within the basket. To do this we use a ScaffoldMessenger widget, which can be applied on a scaffold widget. This briefly pops up a window that we can specify specific text to the user as shown in subsubsection 5.10.3 below.

```
1    ScaffoldMessenger.of(context).showSnackBar(
2    SnackBar(
3      content: ReturnText(text: "Login failed!", color: white,
4        )
5      )
6    );
```

Figure 5.6: Code Representing How a Notification is Given to the User

### 5.10.4 Launcher Icons

For the launcher icons we used the open source imaging editing application GNU Image Manipulation Program [19] (GIMP), which provides a range of creation tools to design the barber pole which is used as the logo and launcher icons. To then install this we used the plugin flutter_launcher_icons which allows us to use the created icon for both iOS and android devices.

### 5.10.5 Discount Codes

Within the checkout screen cart.dart we gave the user the ability to add discount codes to their order. This simply involved creating a new document within firebase discountCodes and then adding the code name as the document ID, a bool on whether it was active and the discount percentage. This was then called on in cart.dart, which used setState() to manage the state of the widget and two methods, getTotalPrice() and getDiscount(), acting as the logic to work out the discount.

### 5.10.6 Hide and Show Password

To add a layer of security the text form that represented the box to enter a users password obscured and the user is given the ability to toggle this through pressing an icon. This was implemented by using a local state management setState() function, which allows the obscureText function within the TextField widget to be updated, along with the icon colour when the user presses the icon. This code can be seen in Figure C.

# 6 Testing

## 6.1 Beta Testing

Beta testing was carried out through providing end users with a copy of the application and asking them to fill out a feedback form, which continually changed throughout development. The final form, which contains questions relevant to the entirety of features can be found in Figure 7.3. The form allowed for validation of the application, but also heavily influenced the 'future work' section of the application, for example, one user wrote: "It would be useful to add a side bar where I could access other details, such as my account and orders etc". This comment influenced the development in that previously there was no intention to implement a side bar, which was then later added. The survey also exposed some flaws and bugs related to the current version of the application, for example, one user wrote: "After creating an account I am unable to logout and I need to exit the application". This, along with many other comments throughout development exposed bugs within the application which allowed for fixes to be made.

## 6.2 Acceptance Testing

It is recommended that acceptance testing is used throughout development as a metric to ensure a continuous link between the customer and development and is an important tool within UCD [20]. To implement this, here we analyse against the previously defined use cases as a success metric.

### 6.2.1 Use Case Implementation

Throughout development, the previously thought out use cases were used as a measure of performance and to keep development directed. When production was completed, they were again consulted to make sure that each required use case could be implemented fully. Here we display this analysis, along with providing relevant screen shots of the application.

(*Here user refers to both a regular customer and barber as each application presents similar use cases. If instead we wish to refer to a specific case, customer or barber will be used explicitly.*)

Use Case 1 - Sign Up

- The customer can sign up through Google (Figure B), Facebook and twitter
- The user can sign up through email and password (Appendix B)
- The barber can sign up through email and password (Figure B)
- The user is notified when they enter an incorrect username or password (Figure B)

Use Case 2 - Login

- The customer can login through Google (Figure B), Facebook and twitter
- The customer can login through email and password (Figure B)
- The barber can login through email and password (Figure B)
- The user is able to reset their password (Figure B)

Use Case 3 - Add a Product To The Cart

- The customer is able to navigate to the required product
- The customer is able to chose the desired quantity and add it to the cart (Figure B)
- The customer is not able to add an item already in the basket

Use Case 4 - Search for a Barber

- The customer is able to search for a barber using the search bar
- The system presents the user with a list of relevant barbers (Figure B)

Use Case 5 - Checkout

- The customer is able to navigate to the shopping cart by pressing the cart icon
- The customer is able to checkout the order by pressing the checkout button (Figure B)
- The customer is notified if their basket is empty (Figure B)

Use Case 6 - View Orders

- The user is able to view their previously made orders (Figure B)

Use Case 7 - Sign Out

- The user is able to sign out (Figure B)

Use Case 8 - Add a Barber

- The parent barber is able to add a barber (Figure B)

# 7 Conclusion and Further Work

## 7.1 Analysis of the Required Objectives

The motivation for this project was to create a cross-platform working (MVP) of a delivery barber app that attained the required objectives as stated in the introductory chapter. Here we discuss how these objectives were met, any difficulties faced and what would be done differently.

### 7.1.1 Conduct Initial User Research to elucidate any market gaps

I begin the thesis by discussing the methodologies that will be utilised throughout the project, for example new product development - which encompassed the stages involved to bring the app from ideation to market and agile software development, which was used as a more general project management tool to guide the work. I found using lean to be extremely effective for the type of project. Breaking down the project into epics and then sprints allowed for well-defined targets to be aimed at, meaning that I could jump back and forth as I gained ability in the language.

In subsection 1.3 I then go on to discuss existing applications in the field, any market gaps and characterise the novelty of the proposed application. I therefore believe that this first objective was met well.

### 7.1.2 Through a user centric design methodology plan and prototype the user interface of the application

After discussing market gaps I then go on to decide which platform best suits the application, which can be seen in subsection 1.4, where I also discuss the most suitable software modalities, including frontend and back-end platforms. I then move on to presenting the target user and the conception of user personas, an integral component of UCD. I believe that using flutter was a wise choice for the project. Along with allowing for cross-platform development, the language follows syntax similar to C, Java and JavaScript, all of which I have some previous exposure too meaning that it was relatively easy to pick up. I am also happy with the use of Provider as the state management tool for the project. Despite its relatively simplistic nature, it served the simple state management of the project well.

The next chapter I present as a software requirements specification document, outlining the scope, user needs and requirements of the application, along with any pertinent use cases. Although I believe I met the minimum requirements for a user centric design approach, I feel this could have been improved through iterative development of further MVPs and more regular user testing, as although feedback was garnered throughout, this was minimal and therefore did not heavily influence development.

### 7.1.3 Create a minimum viable product (MVP) using new product development (NPD) methodologies

The next section discusses the implementation of the application, which are carried out during sprints and analysed here against the functional and non-functional requirements as set out in subsection 2.3.

Functional Requirements

All of the functional requirements for the application were met at least partially during the project, most of them fully including; allowing the customer to create an account and login, allowing the user to pick from a range of products and add them to their cart, along with full management of the cart, providing geographically relevant barbers and products and view their past orders. On the barber side application the barber was able to; create an account and login, manage their barbers and products and view their orders. Despite meeting all of the requirements, relatively little account management is provided and limited to changing email and resetting their password. However, I believe that this had little impact on the usability and functionality of the application and propose that this requirement was still met to an appropriate standard.

Non-functional Requirements

The application met all of the non-functional requirements set out.

## 7.2 Application Deployment

Here, we discuss how both the android and iOS apps are built and deployed to their respective application stores.

Android Application

To deploy the android application there are several steps that must be carried out. First a launcher icon must be added, which was previously implemented and discussed in subsubsection 5.10.4. Next, to publish the android version of the app, a digital signature must be created, first involving creating a 2048 bit RSA key through running

```
1     keytool -genkey -v -keystore ~/upload-keystore.jks -
         keyalg RSA -keysize 2048 -validity 10000 -alias
         upload
```

before the key is referenced from the app by creating a file and storing in

```
1     mobile-barber-app/android/key.properties
```

. This creates a private key that allows the developer to upload to the Google Play Store. Next, gradle [2], which is used as a build tool within android studio must be configured to use the key in the following steps which involve the build.gradle file found in ../android/app/build.gradle:

- The keystore information is added

- Within the buildTypes block setup information is added so that the app will be signed automatically

Finally, an android app bundle [1], which is the preferred format for the Play Store is created using

```
flutter build appbundle
```

which creates the app bundle that can be found at

```
../build/app/outputs/bundle/release/app.aab
```

and this can then be uploaded to the Google Play Store.

iOS Application

To deploy the iOS application, first we must create a developer account on apple.developer.com. Next, we should register our bundle ID [3], which acts as a unique identifier that every iOS application has. Next, an application record is created using App Store Connect, which registers the application. Now, an application icon is added, which was previously implemented in subsubsection 5.10.4. Next, a build archive, which is used to release the application is created using XCode which involves editing the Runner.xcworkspace file by adding a version number, along with the previously created build identifier. Finally, we create a build archive, which can be uploaded to the iOS store by running

```
flutter build ipa
```

which builds an ipa (iOS App Store Package), serving as an application archive file, that can be found in

```
build/ios/archive/MyApp.xcarchive
```

## 7.3 Future Work

As previously stated, the the goal of the project was to create an MVP of a mobile barber application, for which the initial required features were outlined in subsection 1.1. The project met all of the requirements except being able to schedule in a time to book a barber. As the developer, although it would have been relatively easy and quick to implement a simple booking system which used the Timestamp feature within Firebase, together with a list of times that each barber had for their availability drop-down menu, I felt that it would dilute the quality of the code, for which a long time was spent designing the UI and architecture and I therefore felt that this was something better left until after the project. The initial design in Adobe XD of this can be seen in subsection 7.3 below.

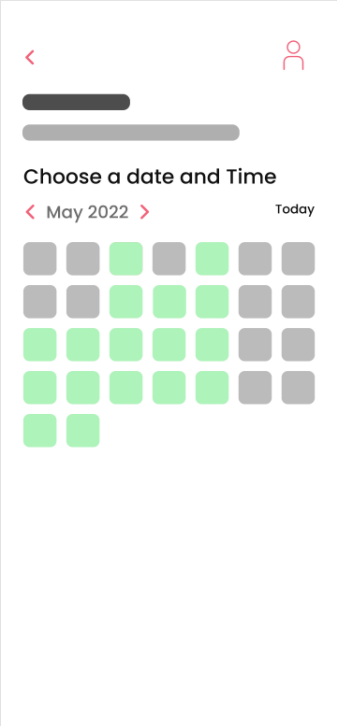



Figure 7.1: Booking System Design in Adobe XD

The final meeting with the external partner discussed releasing the MVP, which we hope to do once the booking system is created, along with a greater feature set that will be implemented when it goes live. The feature set is laid out below:

- Booking system which allows the user to select from a list of dates the relevant barber is available from
- Payment system
- Greater customisation for the users including:
  - Allowing for specific discount codes
  - Setting up a membership scheme
- Admin application which allows for:
  - Manage complaints
  - Manage admin profile
  - Manage users
  - Manage informational pages
- Complaints system which implements a ticket system that links with the admin application

# References


[1] Android. *Android App Bundle*. Android Developers. 2021. URL: https://developer.android.com/platform/technology/app-bundle (visited on 14/09/2021).

[2] Android. *Android Gradle Plugin Release Notes*. Android Developers. 2021. URL: https://developer.android.com/studio/releases/gradle-plugin (visited on 14/09/2021).

[3] Apple. *Bundle IDs — Apple Developer Documentation*. Bundle IDs — Apple Developer Documentation. 2021. URL: https://developer.apple.com/account/ios/identifier/bundle (visited on 14/09/2021).

[4] Mckinsey. *COVID-19 Digital Transformation & Technology — McKinsey*. 2020. URL: https://www.mckinsey.com/business-functions/strategy-and-corporate-finance/our-insights/how-covid-19-has-pushed-companies-over-the-technology-tipping-point-and-transformed-business-forever (visited on 15/09/2021).

[5] Mckinsey. *The Evolving Consumer: How COVID-19 Is Changing the Way We Shop — McKinsey & Company*. 2020. URL: https://www.mckinsey.com/about-us/covid-response-center/mckinsey-live/webinars/evolving-consumer-how-covid-19-has-changed-us-shopping-habits (visited on 15/09/2021).

[6] *Agile versus Waterfall*. 2021. URL: https://www.pmi.org/learning/library/agile-versus-waterfall-approach-erp-project-6300 (visited on 13/09/2021).

[7] *Android vs iOS Market Share 2023*. Statista. 2021. URL: https://www.statista.com/statistics/272307/market-share-forecast-for-smartphone-operating-systems/ (visited on 15/03/2021).

[8] Jonathan Arnowitz, Michael Arent and Nevin Berger. "Chapter 15 - Wireframe Prototyping". In: *Effective Prototyping for Software Makers*. Ed. by Jonathan Arnowitz, Michael Arent and Nevin Berger. Interactive Technologies. San Francisco: Morgan Kaufmann, 1st Jan. 2007, pp. 272–292. ISBN: 978-0-12-088568-8. DOI: 10.1016/B978-012088568-8/50016-3. URL: https://www.sciencedirect.com/science/article/pii/B9780120885688500163 (visited on 27/05/2021).

[9] Christopher N Chapman and Russell P Milham. "The Personas' New Clothes: Methodological and Practical Arguments against a Popular Method". In: (Apr. 2005), p. 6.

[10] *Cloud Firestore — Firebase*. URL: https://firebase.google.com/docs/firestore (visited on 16/08/2021).

[11] cosmeticsdesign europe.com. *Trend Reflections: 9 Ways COVID-19 Changed Beauty Consumption in 2020*. cosmeticsdesign-europe.com. URL: https://www.cosmeticsdesign-europe.com/Article/2020/12/22/Beauty-trends-during-COVID-19-changed-including-hygiene-safety-and-digital-focus-says-CosmeticsDesign-Europe (visited on 15/09/2021).

[12] *Desktop vs Mobile Market Share United Kingdom*. StatCounter Global Stats. 2021. URL: https://gs.statcounter.com/platform-market-share/desktop-mobile/united-kingdom/ (visited on 17/08/2021).

[13] *Desktop vs Mobile vs Tablet Market Share Worldwide*. StatCounter Global Stats. 2021. URL: https://gs.statcounter.com/platform-market-share/desktop-mobile-tablet/ (visited on 17/08/2021).

[14] *Download Android Studio and SDK Tools — Android Developers*. URL: https://developer.android.com/studio (visited on 16/08/2021).

[15] World Leaders in Research-Based User Experience. *10 Usability Heuristics for User Interface Design*. Nielsen Norman Group. URL: https://www.nngroup.com/articles/ten-usability-heuristics/ (visited on 16/08/2021).

[16] *Firebase Authentication*. Firebase. 2021. URL: https://firebase.google.com/docs/auth (visited on 25/08/2021).

[17] *Flutter - Beautiful Native Apps in Record Time*. URL: https://flutter.dev/ (visited on 21/05/2021).

[18] *Geoflutterfire — Flutter Package*. Dart packages. 2021. URL: https://pub.dev/packages/geoflutterfire (visited on 03/09/2021).

[19] *GIMP - The GNU Image Manipulation Program: The Free and Open Source Image Editor*. GIMP. 2021. URL: https://www.gimp.org/ (visited on 15/09/2021).

[20] Jez Humble and David Farley. *Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation*. 1st edition. Upper Saddle River, NJ: Addison-Wesley Professional, 27th July 2010. 512 pp. ISBN: 978-0-321-60191-9.

[21] Shortcut Mobile Inc. *Shortcut™ — In-Home Haircuts and More from Top Local Salons*. Shortcut. URL: https://www.getshortcut.co/ (visited on 15/08/2021).

[22] Berry Kieromin. *Introduction to Change Management*. Work Smarter Together. URL: http://worksmartertogether.ucd.ie/introduction-to-change-management/ (visited on 15/09/2021).

[23] Richard Larson and Elizabeth Larson. *Use Cases - What Every Project Manager Should Know*. 2004. URL: https://www.pmi.org/learning/library/use-cases-project-manager-know-8262 (visited on 18/08/2021).

[24] James Martin. *Rapid Application Development*. 1991. URL: https://books.google.co.uk/books/about/Rapid_Application_Development.html?id=o6FQAAAAMAAJ&redir_esc=y (visited on 13/01/2021).

[25] Jennifer (Jen) McGinn and Nalini Kotamraju. "Data-Driven Persona Development". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '08. New York, NY, USA: Association for Computing Machinery, 6th Apr. 2008, pp. 1521–1524. ISBN: 978-1-60558-011-1. DOI: 10.1145/1357054.1357292. URL: https://doi.org/10.1145/1357054.1357292 (visited on 20/05/2021).

[26] *Mobile Percentage of Website Traffic 2021*. Statista. 2021. URL: https://www.statista.com/statistics/277125/share-of-website-traffic-coming-from-mobile-devices/ (visited on 17/08/2021).

[27] *Monday - Home*. 2021. URL: https://monday.com/ (visited on 15/08/2021).

[28] *[PDF] Personas, Participatory Design and Product Development: An Infrastructure for Engagement — Semantic Scholar*. URL: https://www.semanticscholar.org/paper/Personas%2C-Participatory-Design-and-Product-An-for-Grudin-Pruitt/7ea9505694f1d444a330b1947109c7268a9704d2 (visited on 20/05/2021).

[29] *Place Autocomplete Requests — Places API*. Google Developers. URL: https://developers.google.com/maps/documentation/places/web-service/autocomplete#place_autocomplete_requests (visited on 06/07/2021).

[30] Mary Poppendieck and Tom Poppendieck. *Lean Software Development: An Agile Toolkit*. Boston: Addison-Wesley Professional, 8th May 2003. 240 pp. ISBN: 978-0-321-15078-3.

[31] Owen Ray. *28 Statistics Home Services Marketers Need to Know in 2021*. 2021. URL: https://www.invoca.com/blog/home-services-marketing-stats (visited on 11/08/2021).

[32] *React Native · Learn Once, Write Anywhere*. URL: https://reactnative.dev/ (visited on 15/08/2021).

[33] Shopify. *What Is Product Development? Learn The 7-Step Framework Helping Businesses Get to Market Faster*. Shopify. URL: https://www.shopify.co.uk/blog/product-development-process (visited on 21/05/2021).

[34] *Sketch vs Figma, Adobe XD, And Other UI Design Applications*. Smashing Magazine. 11:00:16 +0200 +0200. URL: https://www.smashingmagazine.com/2019/04/sketch-figma-adobe-xd-ui-design-applications/ (visited on 27/05/2021).

[35] *State of Agile Report — State of Agile*. 2021. URL: https://stateofagile.com/#ufh-i-661275008-15th-state-of-agile-report/7027494 (visited on 13/09/2021).

[36] eBay TechBlog. *eBay Motors & State Management*. eBayTech. URL: https://medium.com/ebaytech/ebay-motors-state-management-bd85cfc602a2 (visited on 24/08/2021).

[37] *The State of the Octoverse*. The State of the Octoverse. 2019. URL: https://octoverse.github.com/2019/ (visited on 15/08/2021).

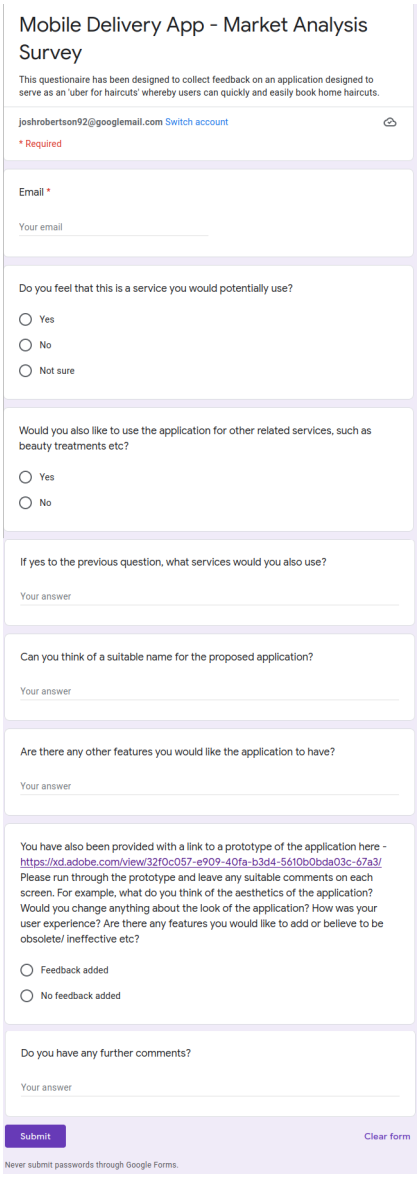
[38] *TRIM-IT Mobile Barbershops*. URL: https://trimit.app/ (visited on 15/08/2021).

[39] *TrimCheck - Home Haircuts on-Demand*. URL: https://get.trimcheck.com/haircut/ (visited on 15/08/2021).

[40] *Will Ride-Hailing Profits Ever Come? – TechCrunch*. URL: https://techcrunch.com/2021/02/12/will-ride-hailing-profits-ever-come/?guccounter=1&guce_referrer=aHR0cHM6Ly93d3cuZ29vZ2xlLmNvbS8&guce_referrer_sig=AQAAALI-wmBgq89Iy5-xw3IGxisaXnb2-j1jMukBMZr0fb_p2OE4cobBWyJa_VB8ySIkFyuVC5nWlLio8Y-SQmmd0NJw68rs3Ek9nhL9d6W4Ycu_pzIwjxa948bQiwzH4HAiSNMcEz0r0UxnxbOD_RHZM2VX2LLZ8F_II9lS_bqK (visited on 12/08/2021).

# Appendices

## Surveys



Mobile Delivery App - Market Analysis Survey

This questionaire has been designed to collect feedback on an application designed to serve as an 'uber for haircuts' whereby users can quickly and easily book home haircuts.

joshrobertson92@googlemail.com Switch account

* Required

Email *

Your email

Do you feel that this is a service you would potentially use?

- Yes
- No
- Not sure

Would you also like to use the application for other related services, such as beauty treatments etc?

- Yes
- No

If yes to the previous question, what services would you also use?

Your answer

Can you think of a suitable name for the proposed application?

Your answer

Are there any other features you would like the application to have?

Your answer

You have also been provided with a link to a prototype of the application here - https://xd.adobe.com/view/32f0c057-e909-40fa-b3d4-5610b0bda03c-67a3/ Please run through the prototype and leave any suitable comments on each screen. For example, what do you think of the aesthetics of the application? Would you change anything about the look of the application? How was your user experience? Are there any features you would like to add or believe to be obsolete/ ineffective etc?

- Feedback added
- No feedback added

Do you have any further comments?

Your answer

Submit Clear form

Never submit passwords through Google Forms.

Figure .2: Market Analysis Survey Using Google Forms

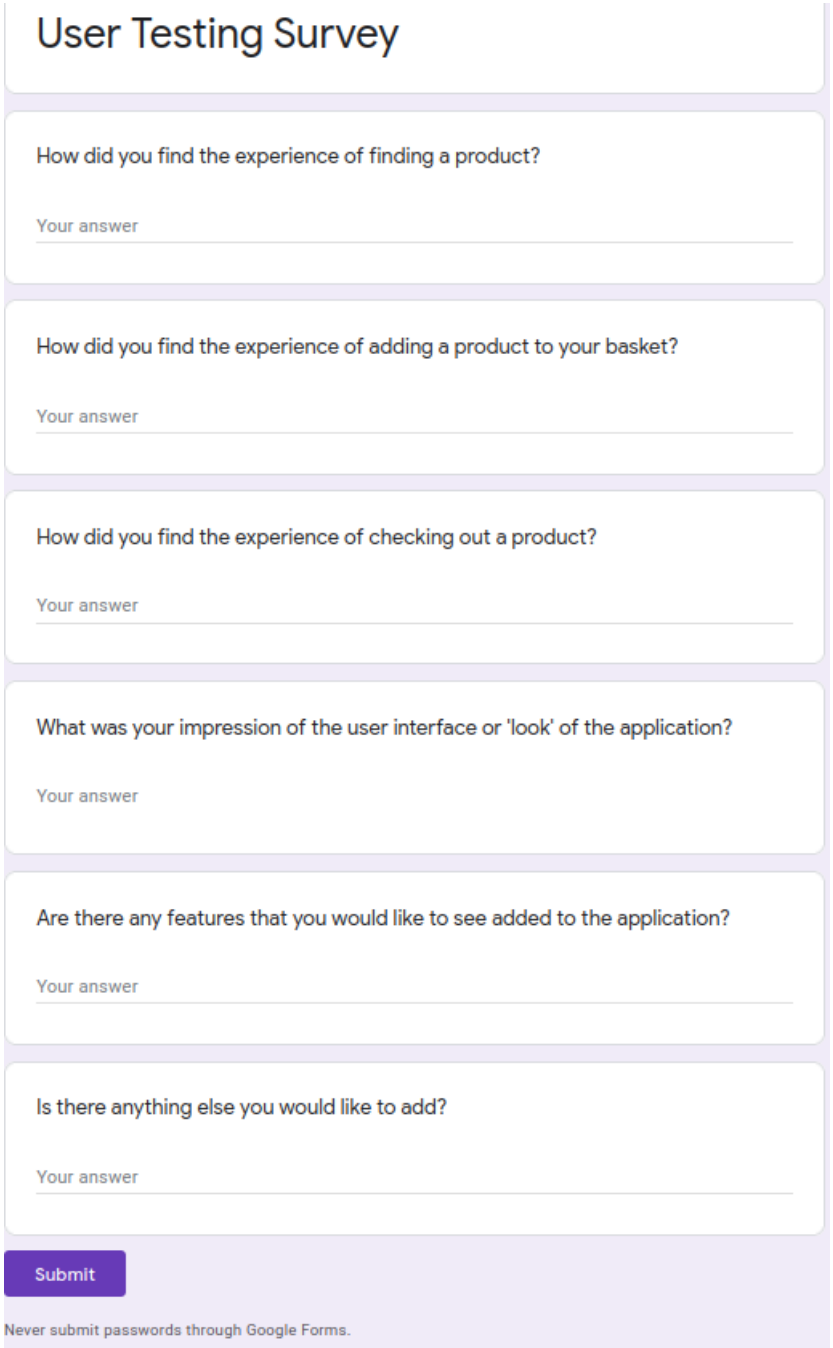



Figure .3: User Testing Survey Using Google Forms

# A Use Cases

| **Use Case 1** | **Sign Up** |
|---|---|
| *Description* | *Allow the user to sign up and create an account* |
| Pre-conditions | The user must not be signed in |
| Basic Flow | 1a) The user enters their sign-up details<br>1b) The user clicks on one of the OAuth sign-in buttons<br>2) The system creates and authenticates the new user and signs them in<br>3) The user is signed-in |
| Alternative Paths | 1) The user enters an email address in use and is notified of this<br>2) The user enters an invalid password or email and is notified of this |

Table A.1: Use Case 1 - Sign Up

| **Use Case 2** | **Login** |
|---|---|
| *Description* | *Allow the user login* |
| Pre-conditions | The user must be signed in |
| Basic Flow | 1a) The user enters their login details<br>1b) The user clicks on one of the OAuth sign-in buttons<br>2) The system validates the authentication request<br>3) The user is signed-in |
| Alternative Paths | 1) The user enters the wrong login details and is notified of this<br>2) The user has forgotten their password and is able to reset it |

Table A.2: Use Case 2 - Login

| **Use Case 3** | **Book a Haircut** |
|---|---|
| *Description* | *Allow the user to add a required product to their shopping cart* |
| Pre-conditions | The user must be signed in |
| Basic Flow | 1) The customer navigates to the required product<br>2) The customer chooses the required quantity and clicks Add To Cart<br>3) The system adds the item to the users cart |
| Alternative Paths | 1) The user tries to add a product already in the basket and is unable to |

Table A.3: Use Case 5 - Book a Haircut (*customers only*)

| **Use Case 4** | **Search for a Barber** |
|---|---|
| *Description* | *Allow the user to enter a search term to find a relevant barber* |
| Pre-conditions | The user must be signed in |
| Basic Flow | 1) The customer enters a search term within the search bar on the Home screen<br>2) The system presents the user with a list of relevant barbers |
| Alternative Paths | none |

Table A.4: Use Case 4 - Search for a Barber

| **Use Case 5** | **Checkout** |
|---|---|
| *Description* | *Allow the user to checkout their basket and create an order* |
| Pre-conditions | The user must have items in their basket |
| Basic Flow | 1) The user navigates to the basket<br>2) The user clicks checkout<br>3) The system creates an order within the database |
| Alternative Paths | 1) The user does not have any items in their basket and is notified of this |

Table A.5: Use Case 5 - Checkout (*customers only*)

| **Use Case 6** | **View Orders** |
|---|---|
| *Description* | *Allow the user to view their placed or confirmed orders* |
| Pre-conditions | The customer must have made an order or the barber must have customers orders |
| Basic Flow | 1) The user requests to see their orders<br>2) The system fetches them and displays them to the user |
| Alternative Paths | 1) The user does not have any orders and is notified of this |

Table A.6: Use Case 6 - View Orders

| **Use Case 7** | **Sign Out** |
|---|---|
| *Description* | *Allow the user to sign out of their account* |
| Pre-conditions | The customer must be signed in |
| Basic Flow | 1) The user requests to sign out<br>2) The system signs out the user |
| Alternative Paths | none |

Table A.7: Use Case 7 - Sign Out

| **Use Case 8** | **Add a barber** |
|---|---|
| *Description* | *Allow the parent barber to add a new barber* |
| Pre-conditions | None |
| Basic Flow | 1) The parent barber enters the details of the barber<br>2) The parent barber requests to create a new barber with the given details<br>3) A new barber is created |
| Alternative Paths | none |

Table A.8: Use Case 8 - Add a Barber (*barbers only*)

# B Application Images

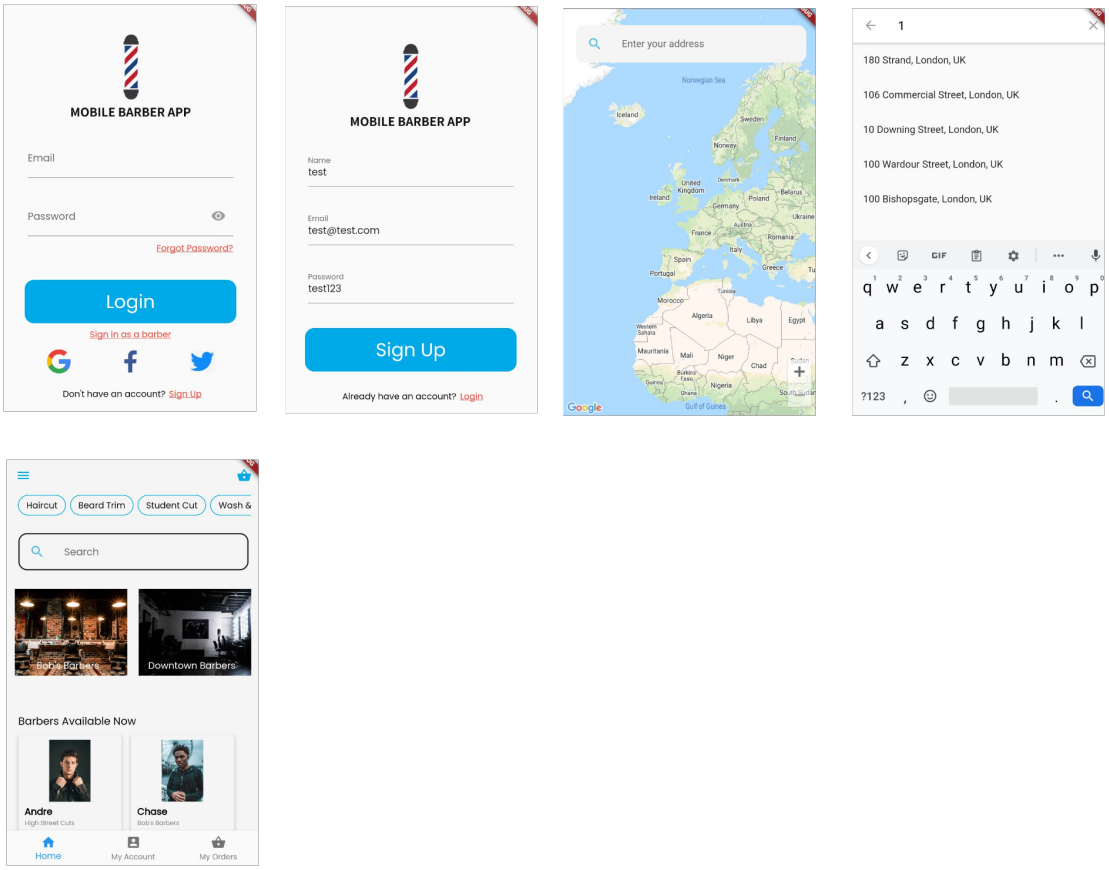



Figure B.1: Customer Sign Up With Email Application Image

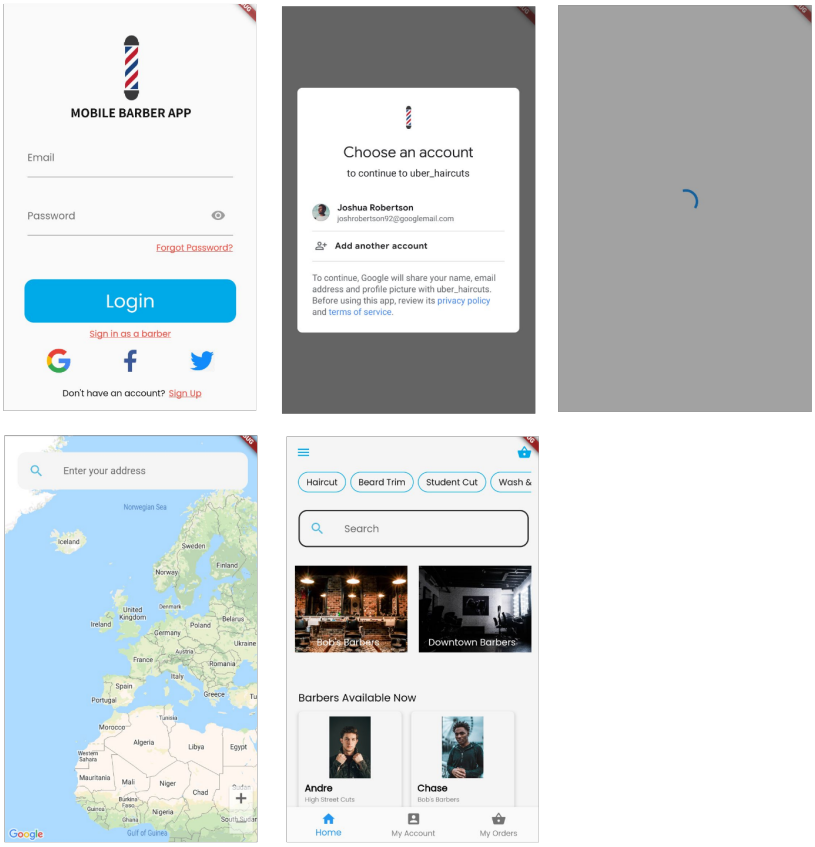



Figure B.2: Customer Sign Up With Google Application Image

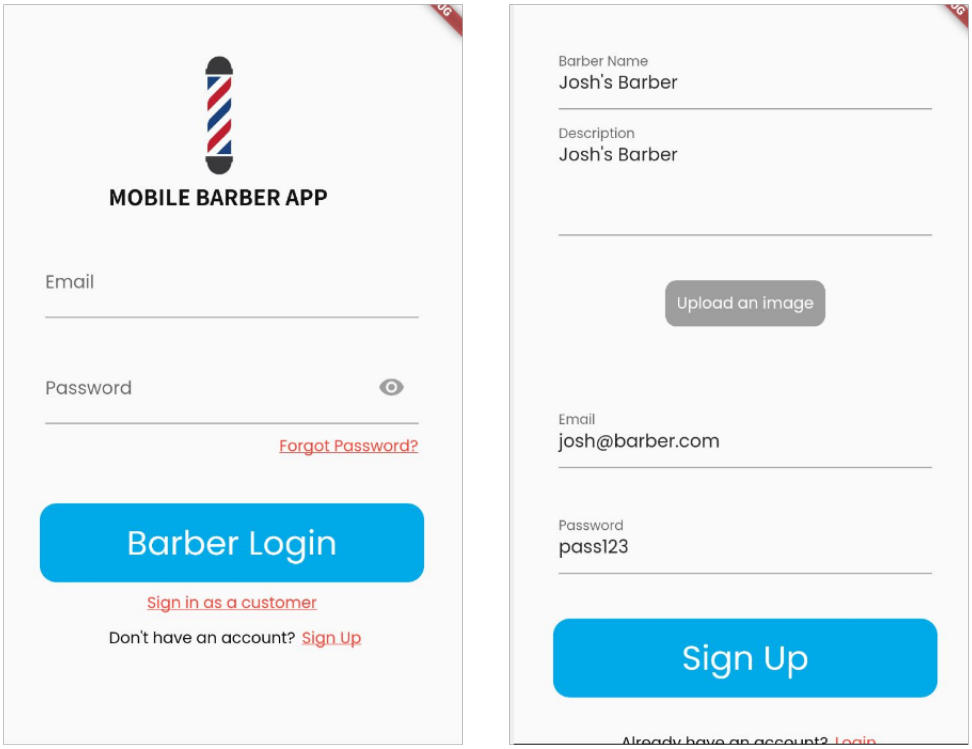



Figure B.3: Barber Sign Up Application Image

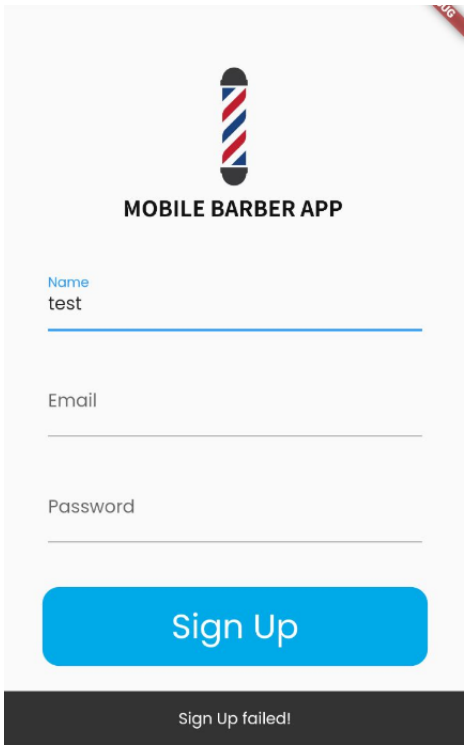



Figure B.4: Sign Up Failed

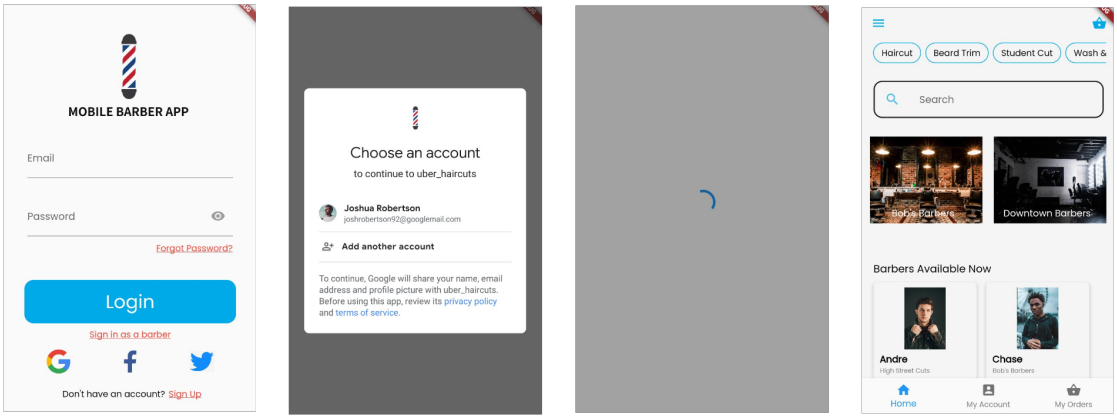



Figure B.5: Customer Login with Google Application Image

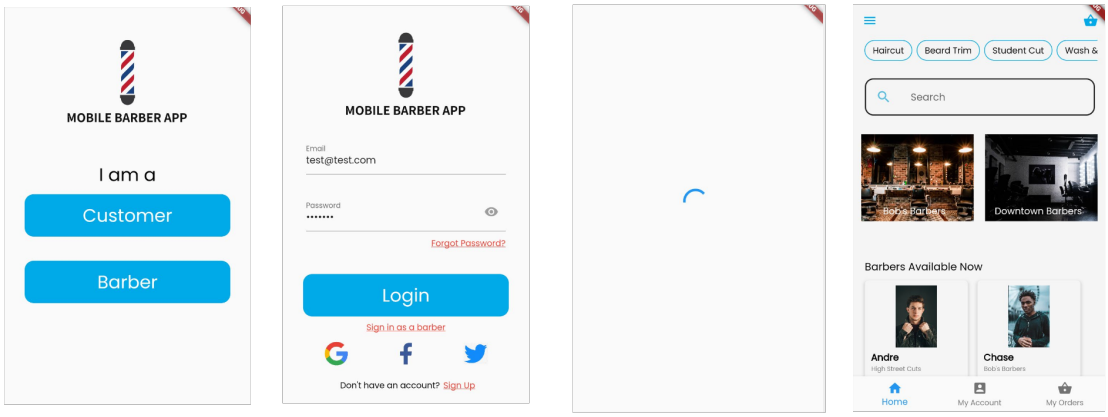



Figure B.6: Customer Login with Email and Password Application Image

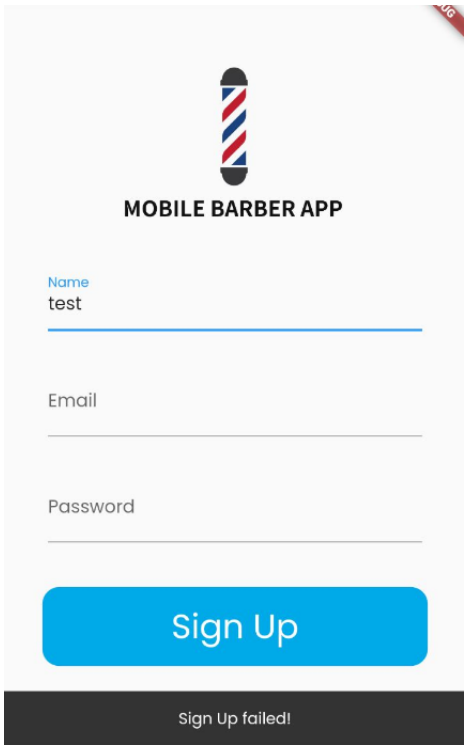



Figure B.7: Wrong Login Details Application Image

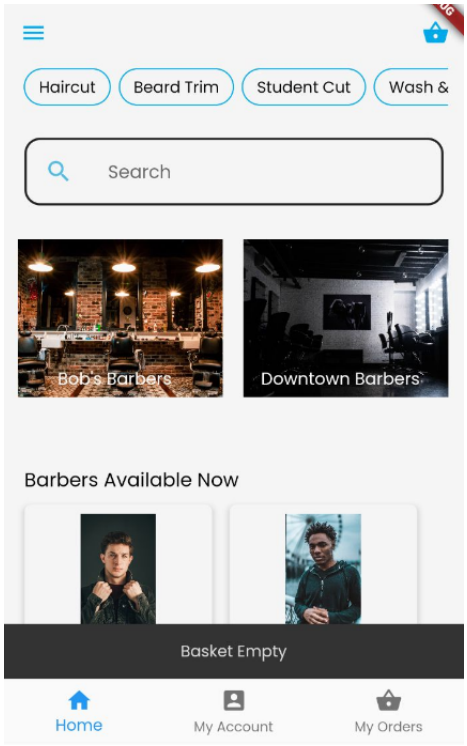



Figure B.8: Basket Empty Application Image

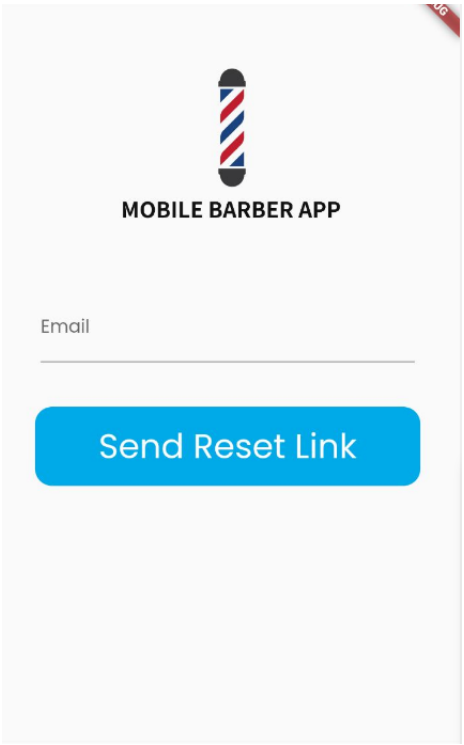



Figure B.9: Reset Password Application Image

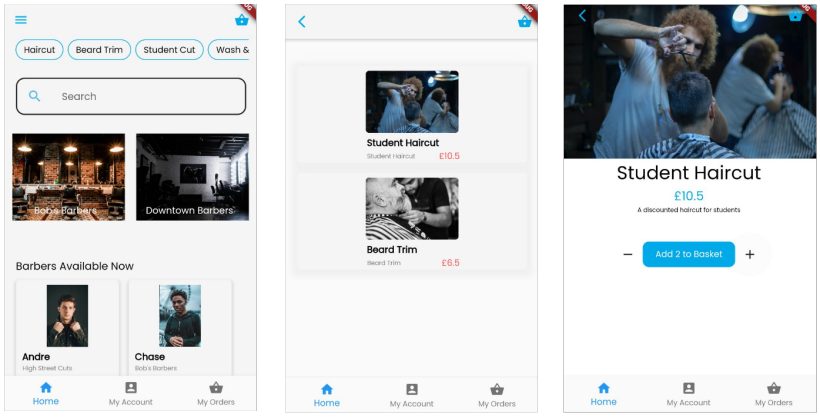



Figure B.10: Add Product to the Cart Application Image

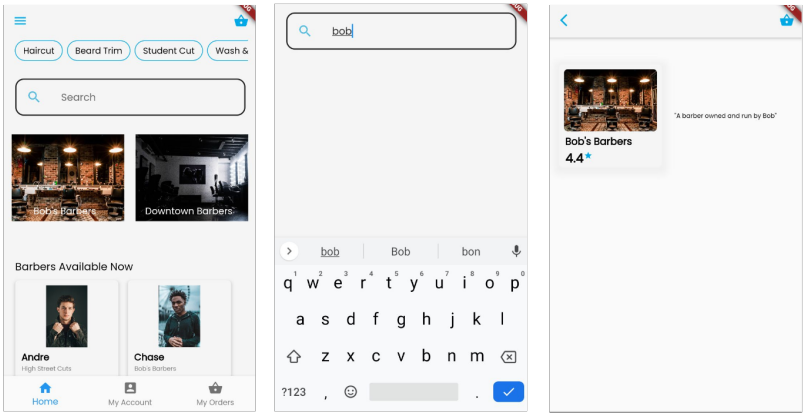



Figure B.11: Search for a Barber Application Image

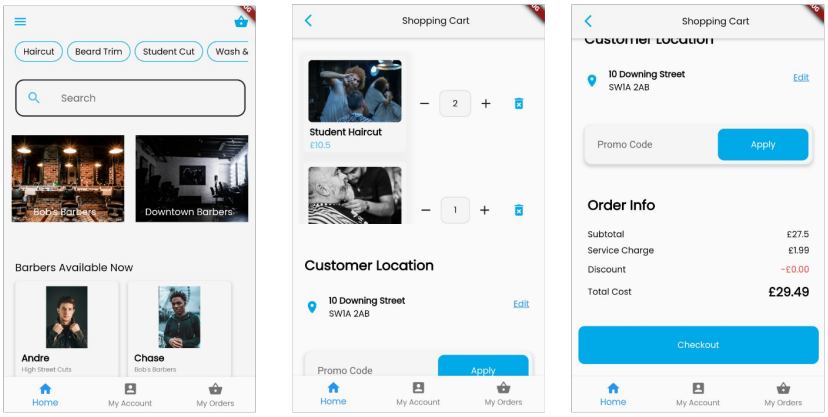



Figure B.12: Checkout Application Image

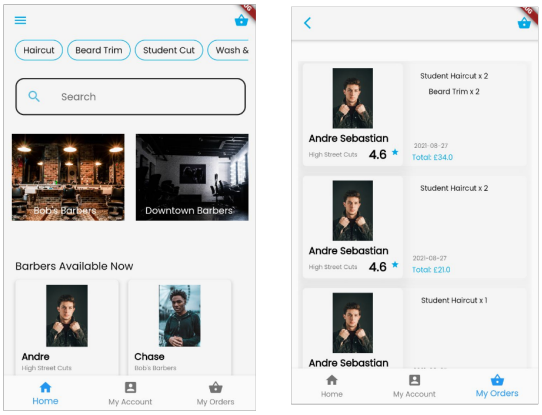



Figure B.13: View Orders Application Image

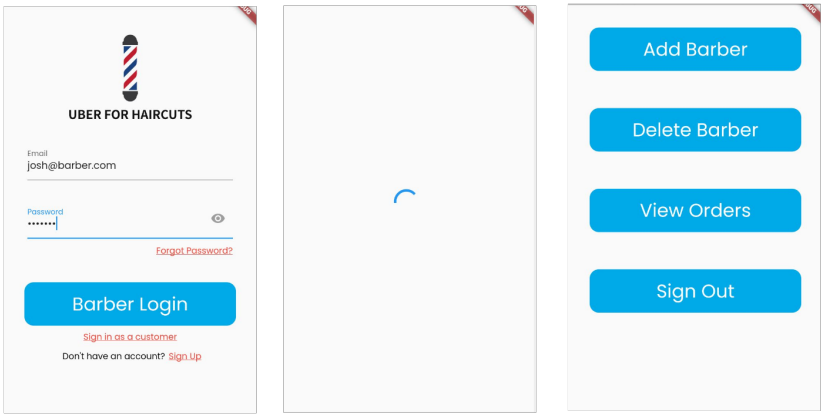



Figure B.14: Barber Login Application Image

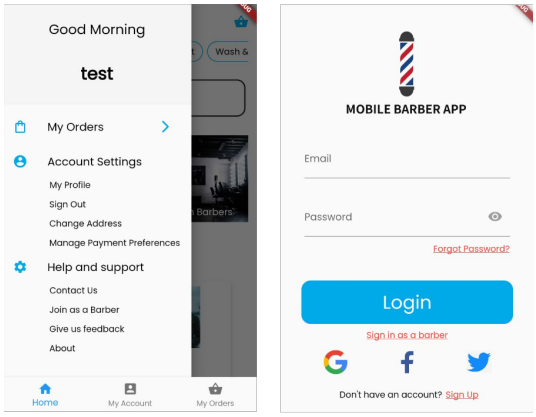



Figure B.15: Sign Out Application Image

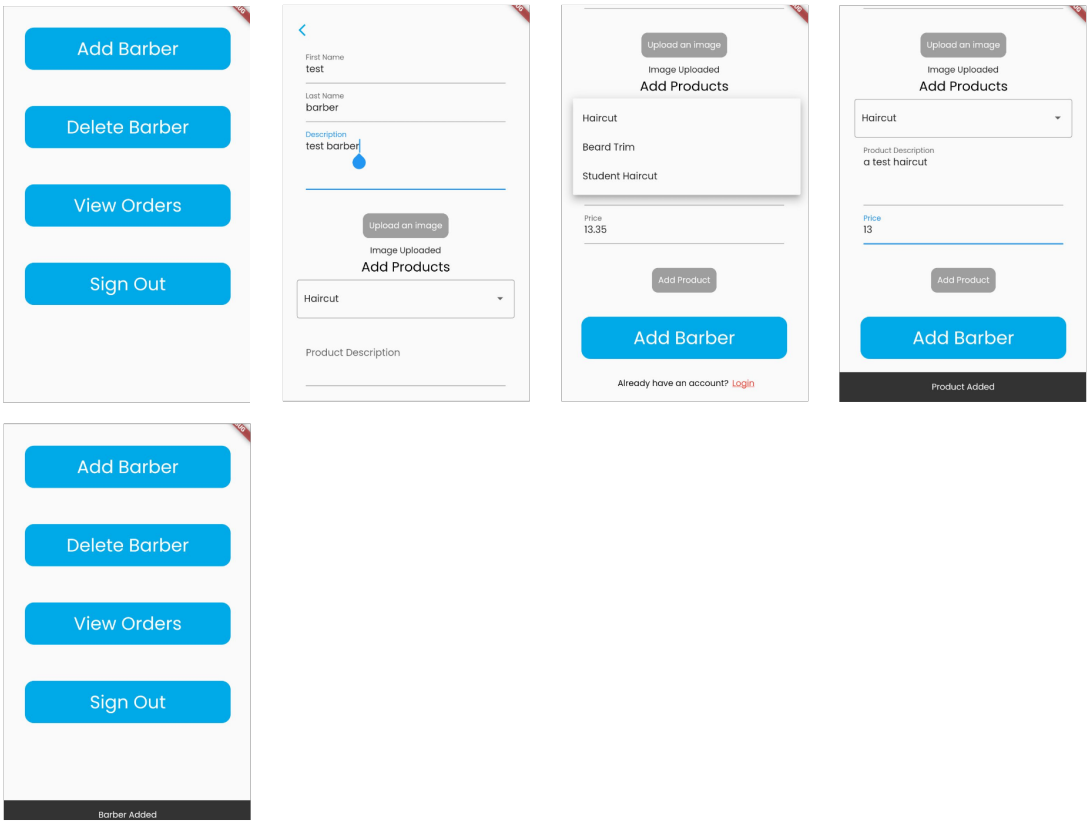



Figure B.16: Add Barber Application Image

# C Code Snippets

```dart
Future<bool> signIn({String email, String password}) async {
  try {
    // Change status to authenticating
    _authStatus = AuthStatus.AUTHENTICATING;
    // Update state management
    notifyListeners();
    final UserCredential _authResult = await
    _firebaseAuth.signInWithEmailAndPassword(email: email,
    password: password);
    print("signed in " + email);
    // Parse the data and create a new model
    userModel = await
    _orderUtility.getUserById(_firebaseAuth.currentUser.uid);
    List<OrderModel> orders = [];
    // Fetch the users stored database orders
    orders = await _orderUtility
    .getDatabaseCartItems(_authResult.user.uid);
    userModel.cart = orders;
    _authStatus = AuthStatus.AUTH_WITH_MAPS;
    notifyListeners();
    return true;
  } on FirebaseAuthException catch (e) {
    print(e);
    // Not authenticated so set status to display an error message
    _authStatus = AuthStatus.NOT_AUTHENTICATED;
    notifyListeners();
    return false;
  }
}
```

Figure C.1: Partial Code for Authenticate.dart Showing the Email and Password Sign In

```
    Container(
      child: Column(
        children: [
          Padding(
            padding: const EdgeInsets.fromLTRB(40, 30, 40, 0),
              child: TextField(
                controller: emailController,
                decoration: InputDecoration(
                labelText: "Email",
              )
            ),
          ),
      Padding(
        padding: const EdgeInsets.fromLTRB(35, 40, 35, 0),
        child: SizedBox(
        height: 70,
        child: Material(
          borderRadius: BorderRadius.circular(15),
          shadowColor: theme,
          color: theme,
          // Detect input from the user
          child: GestureDetector(
            onTap: () {
              // Allow the user to reset their password
              context.read<AuthenticateProvider>().resetPassword(
              emailController.text.trim()
            );
            // Take the user to the login screen once the password is reset
            navigateToScreen(context, Login());
          },
          child: Center(
            child: ReturnText(text: Send Reset Link, fontWeight:
            FontWeight.w500, size: 30, color: white,),
            ),
          ),
        ),
      ),
    ),

```

Figure C.2: Partial Code for forgot_password.dart

```dart
// Bidirectional screen navigation
void navigateToScreen(BuildContext context, Widget widget) {
  Navigator.push(context, MaterialPageRoute(builder: (context)
  => widget)
  );
}

// One way screen replacement
void replaceScreen(BuildContext context, Widget widget) {
  Navigator.pushReplacement(context, MaterialPageRoute(builder:
  (context) => widget)
  );
}
```

Figure C.3: Code for navigate.dart

```dart
class FilterList {

  // Order by highest rated and return the top 5
  Future<List<ParentBarberModel>> getTopRatedParents(List<Parent
  BarberModel> parents) async {
    parents.sort((a, b) => b.rating.compareTo(a.rating));
    return parents.take(5).toList();
  }

  // Find the featured parents, although this should always
  // only be 2, filter just in case
  Future<List<ParentBarberModel>> getFeaturedParents(List<Parent
  BarberModel> parents) async =>
  parents.where((element) => element.featured == true).take(2)
  .toList();

}
```

Figure C.4: Code for filter-list.dart

```
Future<void> updateCartFirestore({String userId, UserModel user}) async {
  List<OrderModel> orderItems = user.cart;

  try {
    List<dynamic> newOrders = [];
    for (OrderModel cartItem in orderItems) {
      newOrders.add(cartItem.toMap());
    }
    await _firestore.collection(USERS).doc(userId).update({
      "cart": newOrders
    });
  } catch(e) {
    print("Error deleting item from firestore cart: " + e.toString());
  }
}
```

Figure C.5: updateCartFirestore() Method Found in order.dart

```
Future<List<LocationModel>> getLocations(String input) async {
  // Create an api request using the session token/ api key
  final apiRequest =
  https://maps.googleapis.com/maps/api/place/
  autocomplete/json?input=$input&types=address
  &components=country:uk&lang=en&key=$apiKey
  &sessiontoken=$sessionToken;
  // Send a get request to the server and return a response
  final response = await client.get(Uri.parse(apiRequest));
  final result = json.decode(response.body);
  if (result[status] == OK) {
    // If we are returned results, put them into a list
    return result[predictions]
    .map<LocationModel>((p) => LocationModel(p[place_id],
    p[description]))
    .toList();
  }
  // Successful call but no results
  if (result[status] == ZERO_RESULTS) {
    return [];
  }
  else {
    throw Exception(result[error_message]);
  }
}
```

Figure C.6: getLocations() Method Found in maps.dart

```dart
class NavBar extends StatefulWidget {
  @override
  _NavBarState createState() => _NavBarState();
}

class _NavBarState extends State<NavBar> {
  int _currentIndex = 0;
  List<Widget> _pages = <Widget>[
  Home(),
  Account(),
  UserOrders(),
  ];
  // Set the state when an item is selected
  void _onItemTap(int index) {
    setState(() {
      _currentIndex = index;
    });
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: Center(
        child: _pages.elementAt(_currentIndex),
      ),
      bottomNavigationBar: BottomNavigationBar(
        currentIndex: _currentIndex,
        items: const <BottomNavigationBarItem>[
          BottomNavigationBarItem(
            icon: Icon(Icons.home),
            label: Home,
          ),
          BottomNavigationBarItem(
            icon: Icon(Icons.account_box_rounded),
            label: My Account,
          ),
          BottomNavigationBarItem(
            icon: Icon(Icons.shopping_basket),
            label: My Orders,
          ),
        ],
        onTap: (index) {
          setState(() {
            _currentIndex = index;
          });
        },
      )
    );
  }
}
```

Figure C.7: Code for navigation_bar.dart

```dart
class MyApp extends StatelessWidget {
  // This widget is the root of your application.
  @override
  Widget build(BuildContext context) {
    return MultiProvider(
      providers: [
      // Implement listenable provider to access AuthenticateProvider
        ListenableProvider<AuthenticateProvider>(
          create: (_) => AuthenticateProvider(FirebaseAuth.instance),
        ),
        // Used to access changed data as a stream
        StreamProvider(
          create: (context) =>
          context.read<AuthenticateProvider>().stateChanges,
          initialData: null,
        ),
        ListenableProvider<ParentBarbersProvider>(
          create: (_) => ParentBarbersProvider(),
        ),
      ],
    child: MaterialApp(
      title: Chop Chop,
      theme: ThemeData(
        primarySwatch: Colors.blue,
        fontFamily: Poppins
      ),
    home: AuthenticationWrapper(),
    ),
  );
}
```

Figure C.8: Partial Code for main.dart

```
class UserModel {
  static const NAME = "name";
  static const EMAIL = "email";
  static const UID = "uid";
  static const LOCATION = "location";
  static const CART = "cart";
  static const ORDERS = "orders";

  String _name;
  String _email;
  String _uid;
  List<OrderModel> cart;
  PlaceModel locationDetails;
  List<OrderModel> orders;

  String get name => _name;
  String get email => _email;
  String get uid => _uid;

  UserModel.fromSnapshot(DocumentSnapshot documentSnapshot) {
    _name = documentSnapshot.data()[NAME];
    _email = documentSnapshot.data()[EMAIL];
    _uid = documentSnapshot.data()[UID];
    cart = _convertOrderFromMap(documentSnapshot.data()[CART]);
    orders = _convertOrderFromMap(documentSnapshot.data()[ORDERS]);
    locationDetails = _convertLocationDetails(documentSnapshot.
    data()[LOCATION]);
  }

  UserModel();
```

Figure C.9: Partial Code for user_model.dart

```
Padding(
padding: const EdgeInsets.fromLTRB(40, 30, 40, 0),
child: TextField(
obscureText: _obscurePasswordText,
controller: _passwordController,
  decoration: InputDecoration(
    labelText: "Password",
  suffixIcon: IconButton(
    // Change the colour of the icon and set the obscure
    // password bool
    onPressed: () {
      setState(() {
        // Toggles the icon colour/ obscures text
        if (_hidePasswordColour == Colors.grey) {
          _hidePasswordColour = Colors.blueAccent;
        } else {
          _hidePasswordColour = Colors.grey;
        }
        _obscurePasswordText = !_obscurePasswordText;
      });
    },
    icon: Icon(Icons.remove_red_eye, color: _hidePasswordColour,
        ),
      )
    )
  ),
),
```

Figure C.10: Hide and Show Password Code Found In login.dart

```
// Upload an image file to firebase and return the link
Future<String> uploadImage(File imageFile, String barber) async {
  String filename;
  FirebaseAuth.instance.signInAnonymously();
  FirebaseStorage firebaseStorage = FirebaseStorage.instance;
  Reference firebaseRef = firebaseStorage
  .ref()
  // DateTime.Now used to improve randomness
  .child(barber + /${basename(imageFile.path)}
  + DateTime.now().toString());
  UploadTask uploadTask = firebaseRef.putFile(imageFile);
  await uploadTask.whenComplete(() =>
  print(basename(imageFile.path) +  uploaded')
  );
  await firebaseRef.getDownloadURL().then((fileURL) {
    filename = fileURL;
  });
  return filename;
}
```

Figure C.11: uploadImage() Method Found in files.dart

# D Structure of Included Compressed Folder - thesis.zip

```
thesis
├── latex
├── source-code
│   └── README.md
├── demonstration
│   ├── demonstration.webm
│   └── README.md
└── thesis.pdf
```