

MSC PROJECT: MOBILE HAIRDRESSER
APPLICATION



JOSHUA ROBERTSON

“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)

Contents

1	Introduction	6
1.1	Ideation and Concept	7
1.2	Project Management	8
1.2.1	Vision and Scope	8
1.2.2	Agile Methodology	8
1.3	Research and Market Analysis	9
1.3.1	Existing Applications	9
1.4	Deciding on a Platform	10
1.4.1	Mobile vs Desktop	10
1.4.2	Frontend: Android vs iOS	10
1.5	Frontend: Programming Language	11
1.5.1	Programming Language	11
1.6	Backend: SQL vs noSQL database	11
1.6.1	The Target User	11
1.7	User Personas	11
2	Prototyping	15
3	System Design	18
3.1	Application Scope	18
3.2	User Needs	18
3.3	Requirements	18
3.3.1	Functional Requirements	18
3.3.2	Non-Functional Requirements	18
3.4	System Architecture	18
3.5	State Management	19
4	Production	19
4.1	Version Control	19
4.2	Sprint 1 - Setup	19
4.2.1	File Structure	19
4.3	Sprint X - UI	19
4.3.1	Cupertino vs Material	19
4.4	Sprint X - Login and Sign Up	20
4.5	Sprint X - Backend Design	20
4.6	Connecting the Frontend and Backend	20

4.6.1	Loading the barbers	20
4.6.2	Shopping Cart	21
4.7	Sprint X - Location	21
4.7.1	Autocomplete locations	21
4.8	Sprint X - Payment	22
4.9	Widgets and Common Items	22
4.9.1	Widgets	22
4.9.2	Common Items	23
5	Testing	23
5.1	User Testing	23
5.2	Acceptance Testing	23
6	Random stuff	23
6.1	Launcher Icons	23
A	Use Cases	24
B	Figures	24
C	Application Images	24
D	Code Snippets	24

Abstract

Acknowledgements

Author's 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.

SIGNED: Joshua Robertson DATE: 14.09.2021

1 Introduction

The recent COVID-19 pandemic has brought with it a shift in perceptions around leaving the home, with the global home services market expected to grow by almost 20% per year until 2026 [?]. With this change brings an opportunity for applications, such as the one proposed here to capitalise through providing remote services to the public.

There already exists a range of applications suited towards providing home haircuts. For example, Shortcut [?], TRIM-IT[?] and TrimCheck[?] all provide bookable home haircuts. Despite this, none of the aforementioned applications follow the "uber" model, of allowing for immediate booking and delivery of home haircuts.

This application will therefore aim to facilitate this, along with exploring any other market gaps through early research and will aim to achieve this through the following objectives:

- Conduct research to elucidate any market gaps
- Through a user centric methodology design and prototype the user interface of the application
- Create a minimum viable product (MVP) using new product development

The research aspect of the project will aim to analyse the strength and weaknesses of existing applications within the field, therefore exposing any market gaps that may be present.

For the implementation of the application, a heavy focus on the end user was taken through using a user centric design (UCD) methodology, along with New Product Development, which refers to the entirety of processes leading to bringing a product to market and encompasses several steps as seen in figure 1 below and discussed throughout the proceeding chapters.



Figure 1: The 7 Steps of New Product Development
[?]

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. Through these meetings and discussion with other students, family and friends the following core functionality was created:

The application should allow the user to login, select from a range of barbers and book a home haircut.

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 Vision and Scope

1.2.2 Agile Methodology

To carry out the project, an agile methodology was utilised. Although this approach is usually 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 Backend
- Write the Dissertation

These were then used to create 'stories' which were placed into a timeline. For this, the project management tool monday.com [?] was used, which can be seen in figure 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.

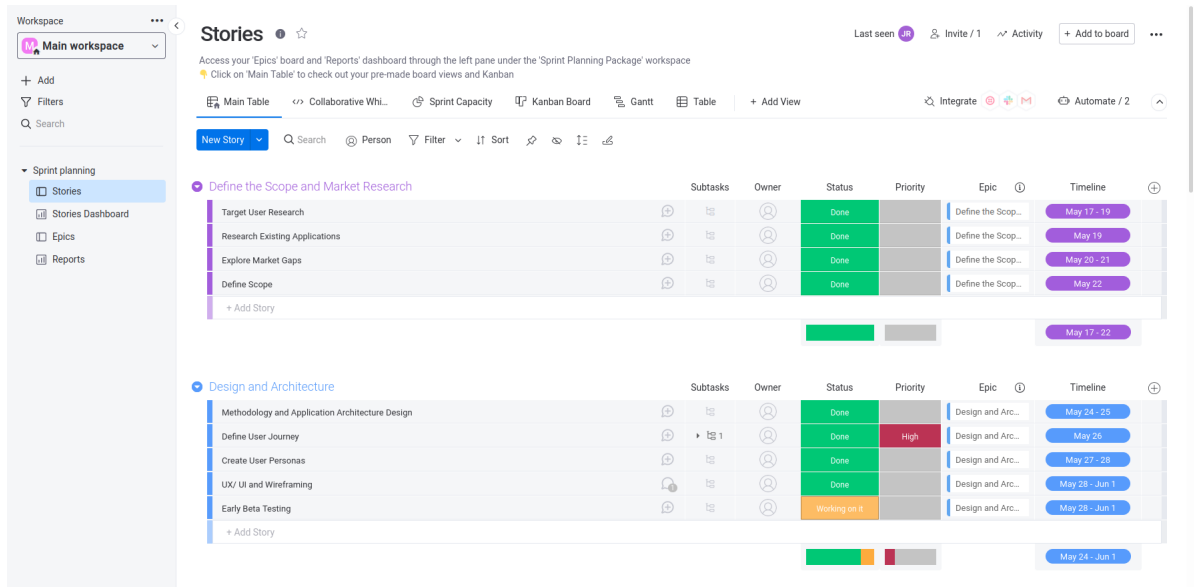


Figure 2: Screenshot of Monday.com displaying the project stories
[?]

Finally, using the aforementioned timeline, a gantt chart was created, which gave an overarching view of the project, with tasks performed represented along the vertical axis and the timeline represented along the horizontal axis. This can be seen in the

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.

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

1.4 Deciding on a Platform

1.4.1 Mobile vs Desktop

1.4.2 Frontend: Android vs iOS

An important consideration when creating a mobile application is deciding on which platform to choose. The two largest mobile 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 3). 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%.

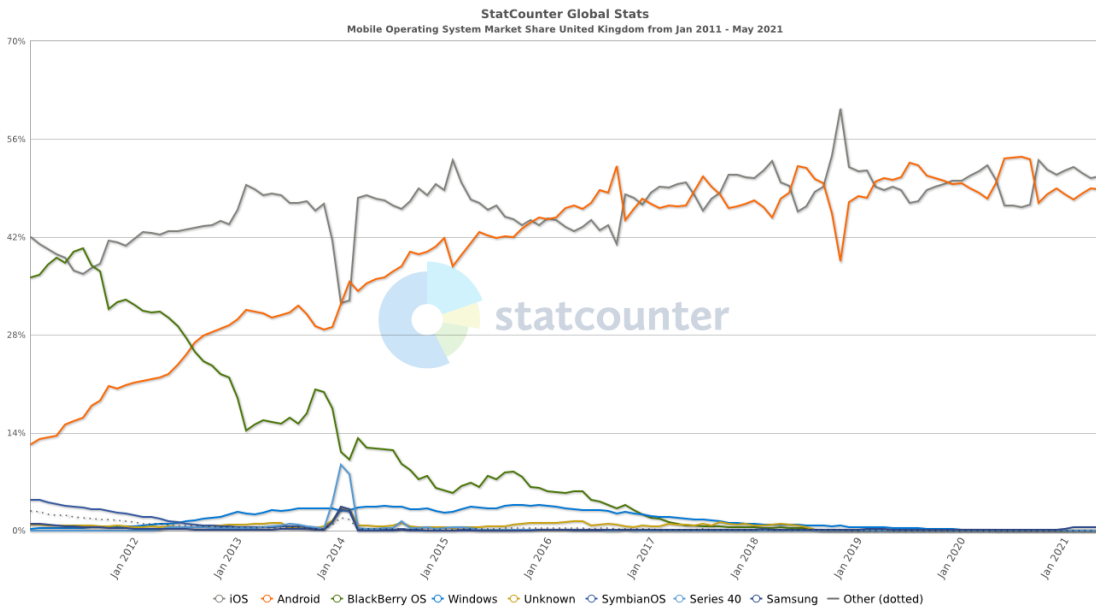


Figure 3: iOS vs Android Market Share Over The Last 10 Years
[?]

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

1.5 Frontend: Programming Language

1.5.1 Programming Language

When deciding on the programming software, several metrics were taken into consideration, including cross-platform functionality, speed, speed of development and performance. 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 [?]. 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 [?, ?] meaning that there is now an extensible list of third-party plugins available and a large community.

1.6 Backend: SQL vs noSQL database

Initially a relational database model was created, which can be seen below (ADD DATABASE). This was later changed to use Google Firestore ([?]), 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 APIs. Firestore also allows for rapid scalability, along with using Googles excellent cloud platform.

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 allows for do you use relationship data in firebase stackoverflow modification throughout the application lifestyle and therefore easy scaling.

1.6.1 The Target User

1.7 User Personas

The creation of user personas representing fictitious, archetypal users is an essential part of application development [?, ?] and allows a deep understanding of the target user to be sought and implemented within the features and

design of the application [?, ?]. There are, however, some shortcomings to qualitative persona generation, such as validity concerns and user bias [?, ?] and although they are addressed by other methods, such as data-driven personas [?, ?], these require a broad user base and therefore we have decided to stick with qualitative methods, which allow for enough brevity and depth for the scope of the project.

Here 3 user personas were created, which are discussed in detail below.

- Persona 1:

Name: Sarah Johnson

Profile:

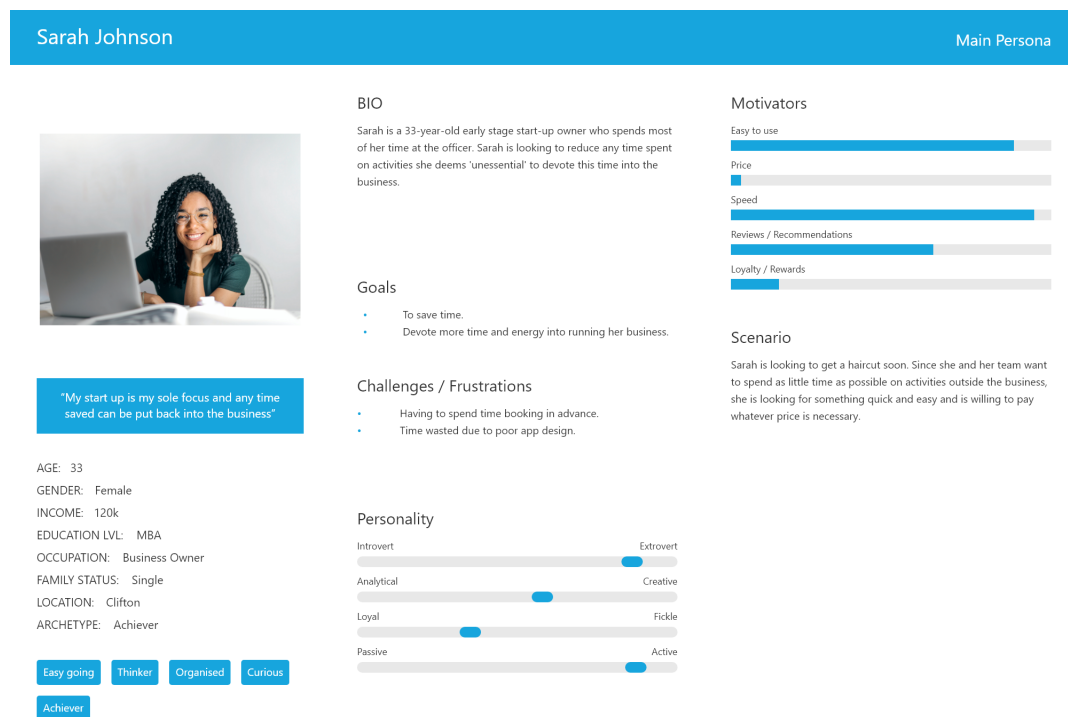


Figure 4: Persona 1

- Persona 2:

Name: Claire Sheppard

Profile:

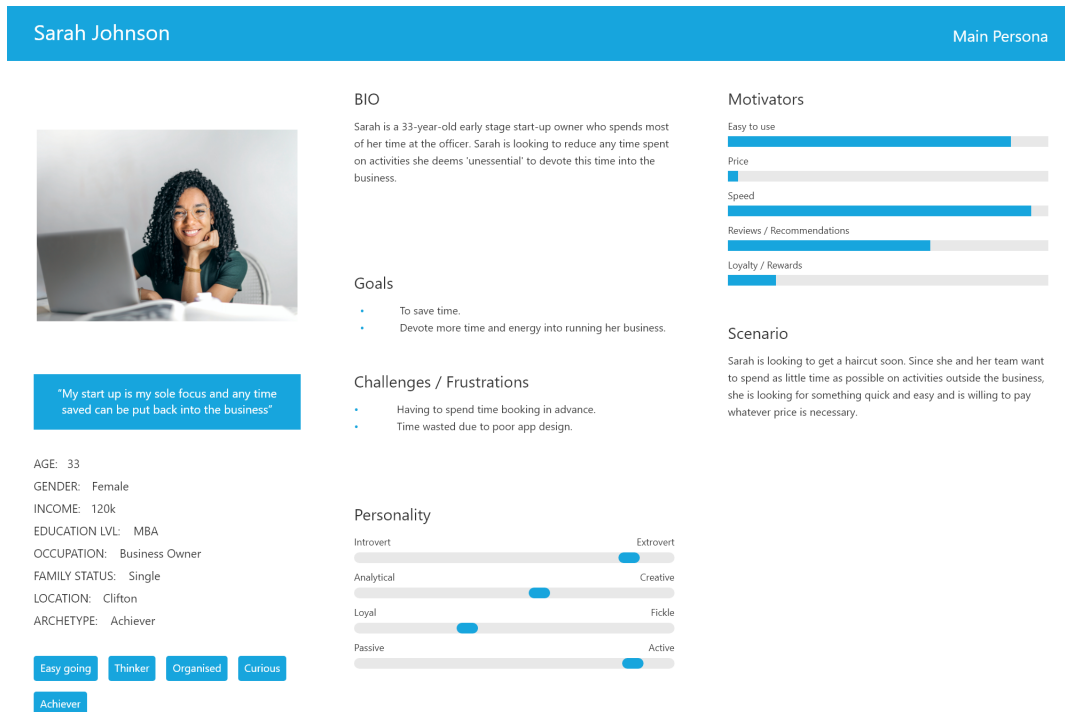


Figure 5: Persona 2

• Persona 3:

Name: Emma Bradford

Profile:

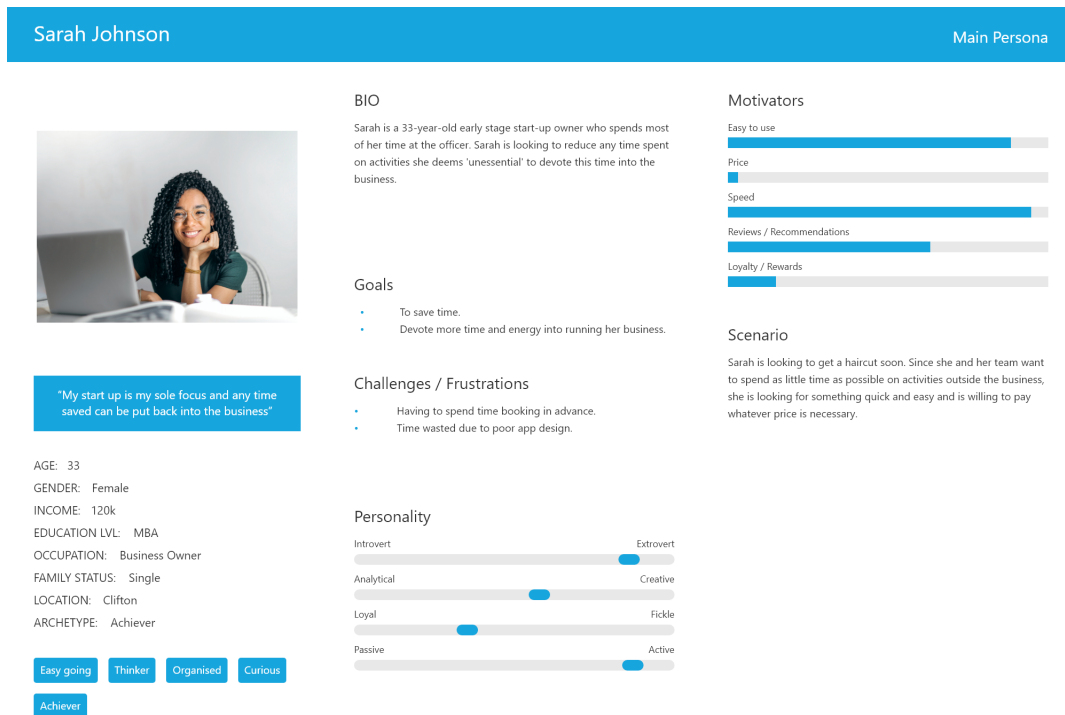


Figure 6: Persona 3

- Persona 4:
 Name: Claire Sheppard
 Profile:

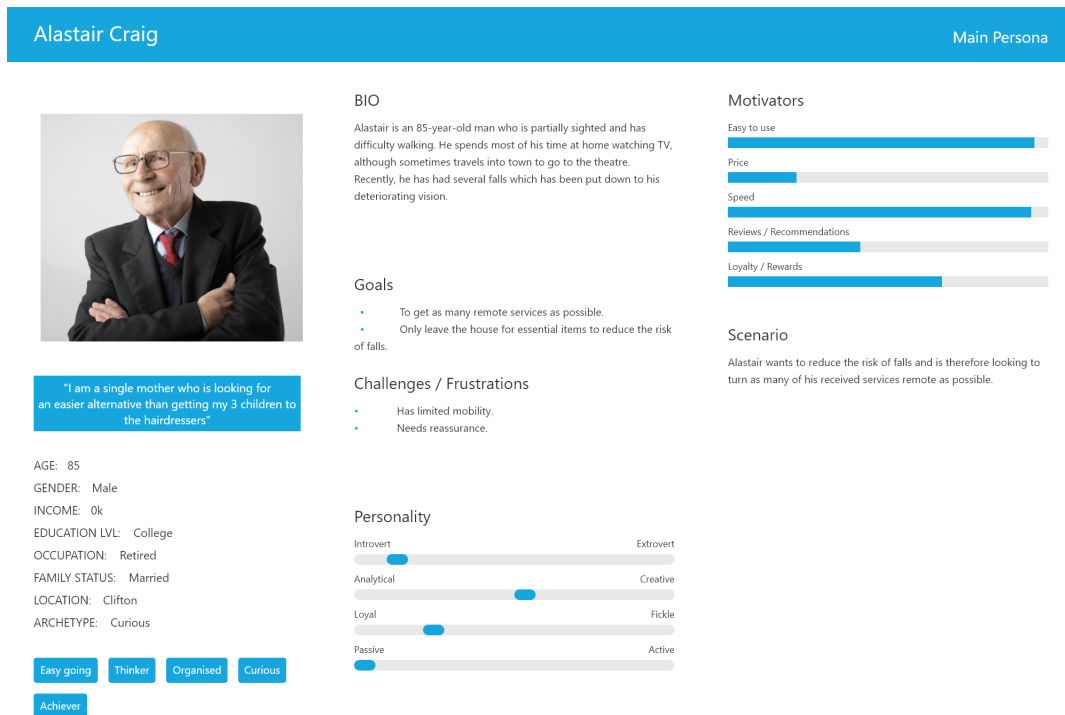


Figure 7: Persona 4

2 Prototyping

An important component of UCD and more generally UX is wireframing, which involves making a mock-up of the application that acts as an early prototype to influence later development, along with allowing for early beta testing [?]. For the wireframing 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 8 below.

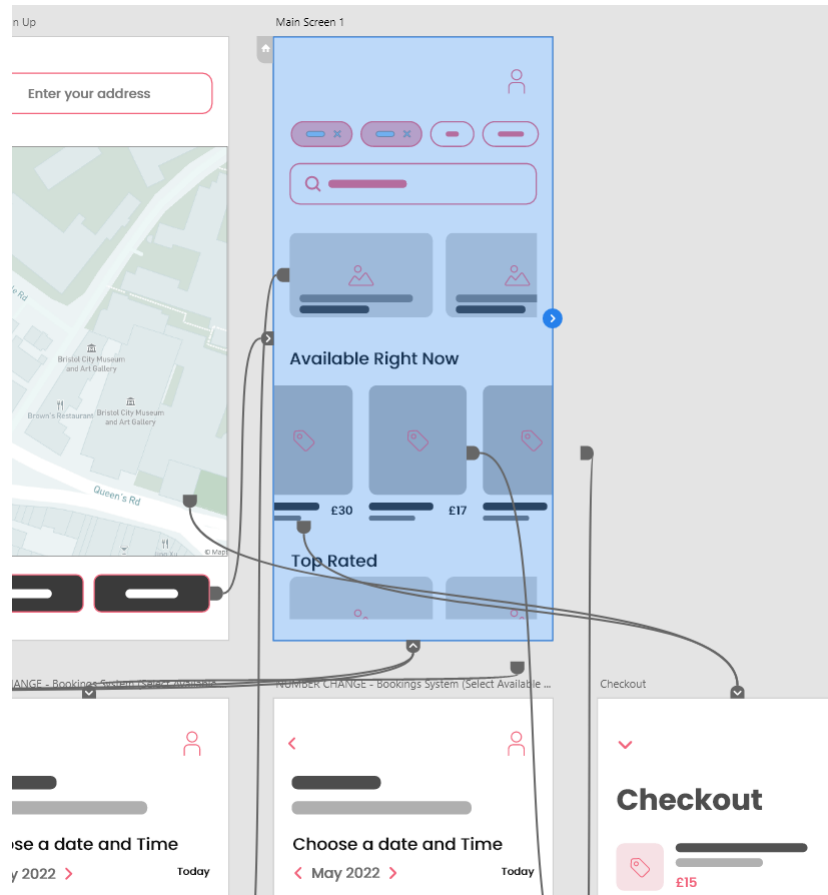


Figure 8: Component Interactivity within Adobe XD

Adobe XD also allows for easy distribution of the prototype in the form of a sharable 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 [?].

When designing the screens there was a strong focus on user experience following Nielsen's 10 Heuristics for User Interface Design [?] . For example, the functionality was kept as minimal as possible to avoid cluttering 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](#).

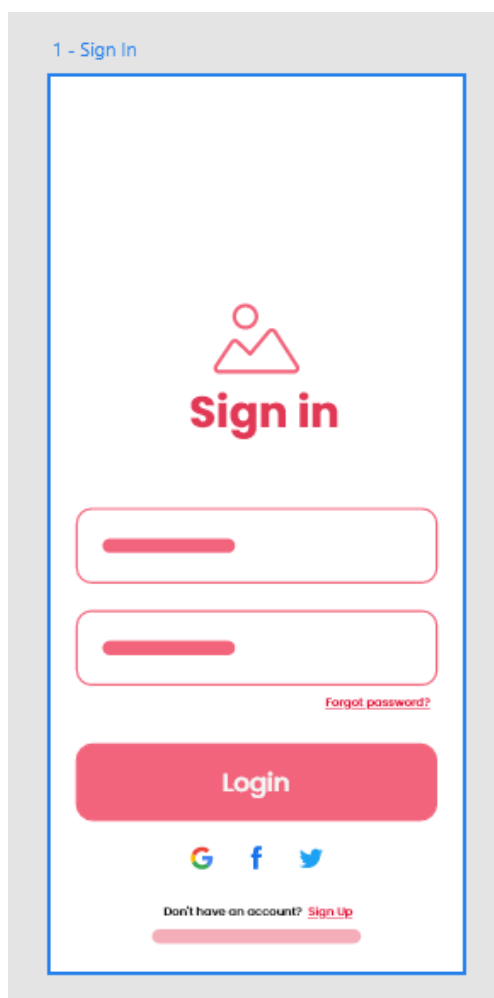


Figure 9: Sign In Page Made with Adobe XD

3 System Design

This section discusses the process of the ideation and creation of the system design, including the initial scope, requirements, application architecture and state management.

3.1 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. 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 further defined when wire-framing in Adobe XD, which allowed for the first tangible design to be made.

3.2 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 own product details.

3.3 Requirements

3.3.1 Functional Requirements

- The application shall allow a barber to create an account and login
 - The application shall allow a customer to create an account and login
- The

3.3.2 Non-Functional Requirements

3.4 System Architecture

(INSERT SYSTEM ARCHITECTURE DIAGRAM) <https://www.researchgate.net/profile/Wei-Jung-Shiang/publication/267418345/figure/fig1/AS:295656959823872@1447501520198/Architecture-for-XML-based-data-exchange-model.png> -

https://www.researchgate.net/figure/Overview-of-the-schema-matching-system_fig2_267418345
Good example

3.5 State Management

Simple state management used (Provider) - involves: ChangeNotifier and Provider.of <https://flutter.dev/docs/development/data-and-backend/state-mgmt/simple>

4 Production

4.1 Version Control

4.2 Sprint 1 - Setup

During the first sprint, the task involved setting up the environment in preparation to begin development. For the editor, Android Studio was TALK ABOUT SCREENS/ MODELS/ HELPER etc

4.2.1 File Structure

Although there is no official recommendation for structuring the app, here we follow a commonly used structure which 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; screens; utilities; and widgets; .

4.3 Sprint X - UI

Flutter offer an Adobe XD plugin 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. Adobe XD also does not contain customer properties and therefore mapping these to components, such as title is not possible.

4.3.1 Cupertino vs Material

The final project was built using material due to..

4.4 Sprint X - Login and Sign Up

Within the `UserDatabase` class, for the `createNewUser` function, we pass through the authentication uid, which is then used as the document id, so that future calls can refer to this and therefore fetch the document, without doing a call such as

```
1  _firebaseFirestore.collection(collection).where
    ('uid', .isEqualTo('givenID')).get()
```

which does not scale well due to a search time of $\mathcal{O}(n)$, compared to

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

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

When authenticating through Google, Facebook and Twitter, a user is created so that data can be persistently stored alongside the credentials.

4.5 Sprint X - Backend Design

Loading all parentBarbers/ barbers and products - When entering the app it is more efficient to make one call to the server rather than multiple due to GPS restrictions. Therefore ParentBarbers, barbers and all of their products are loaded within a certain radius, rather than loading parentBarbers and barbers separately throughout the app.

Don't nest as calls will fetch all of the parent and nested structured so better to have separate data.

(INSERT DATABASE SCHEMA)

4.6 Connecting the Frontend and Backend

To get the list of items within a range 1) Get the users current location 2) Query firestore with the radius

4.6.1 Loading the barbers

When loading the barbers either they were loading through making a database query based on the parentBarberId once the parent barbers were loaded, or every barber was loaded into memory and this was the filtered in memory.

Pros of first method - Scales well as does not matter how many barbers there

are

Cons of first method - more costly as there are more requests to the db

Pros of 2nd - cheaper

cons - does not scale well

4.6.2 Shopping Cart

Chose to store the cart items in a nested array as this adds to readability and structure and although this restricts look up time, this is not relevant due to the limited number of items a user will order.

4.7 Sprint X - Location

<https://firebase.google.com/docs/firestore/solutions/geoqueries> Once the user logs in they give their location in the form of their address. This is stored in 'geohashes', which are longitude and latitude co-ordinates that are hashed into a single Base32 string.

User location access is granted through the following line in the ./android/app/src/main/AndroidManifest.xml file

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

For the search results we use a drop down menu in the form of flutters built in 'showSearch' function loosely following a guide on medium [?] to display a search page and 'SearchDelegate' to define the content of said search page.

A TextEditingController is used to collect the inputted data from the user and pass through to the showSearch function.

4.7.1 Autocomplete locations

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 [?]. First, we enable the Places API within the Google console, before we then create a location model which can hold the data returned from the API. We then create an API request using the above aforementioned API format. 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 query can be seen here:

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

The returned results are in json format and after some minor error checking we parse using `json.decode` into a list with our `LocationModel` class, whilst assigning a new UUID for each query (Google recommends to use version 4 UUID and so this is used here). Without our 'user_gps.dart' file

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 id's' and 'description' using a `FutureBuilder`. 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.

4.8 Sprint X - Payment

4.9 Widgets and Common Items

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

4.9.1 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.

4.9.2 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 backend 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 Testing

5.1 User Testing

5.2 Acceptance Testing

[?, ?, ?]

6 Random stuff

6.1 Launcher Icons

Created using GIMP. Plugin flutter_launcher_icons used to install icons across android and iOS

Appendices

A Use Cases

B Figures

C Application Images

D Code Snippets