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)

1 Introduction

Plan - 1) Covid has caused an increase in homeworking 2) People want greater access to remote services 3) People also want quick access to products 2 reasons for application is that people want fast and easy access to services, and it is expected now with companies i.e. amazon now offering same day delivery. and also peoples views around covid

There is little dispute that the recent COVID-19 pandemic has had a significant impact on our daily lives. Along with having a devestating both on a societal and individual level, it has brought with it a shift in perceptions around leaving the home. Subsequently, this has led to a desire for homeworking and easier access to remote services, with employers . Even with a reduction in COVID cases following a wave of vaccinations, it is likely that home working will continue to a greater extent than was seen previously, with home workers showing an increase in job satisfaction [**?**, **?**], [**?**, **?**], productivity, mental health [**?**]flexjobs, 2020) and even making more money (ADD CITE). Although there already exists several apps that allow for home bookings, none follow the "uber" model, of allowing for immediate booking and delivery of home haircuts. This app therefore will facilitate this, along with aiming to explore any other market gaps through early research.

New Product Development refers to the entirety of processes leading to bringing a product to market and encompasses several steps as seen in figure 1 below.

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

1.1 Ideation and Concept

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like "Huardest gefburn"? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

1.2 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.

1.2.1 Existing Applications

As previously discusses there exists a variety of similar applications, for which the most prominent will be discussed below, along with

1.3 Deciding on a Platform

1.3.1 Mobile vs Desktop

1.3.2 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 2). 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%.

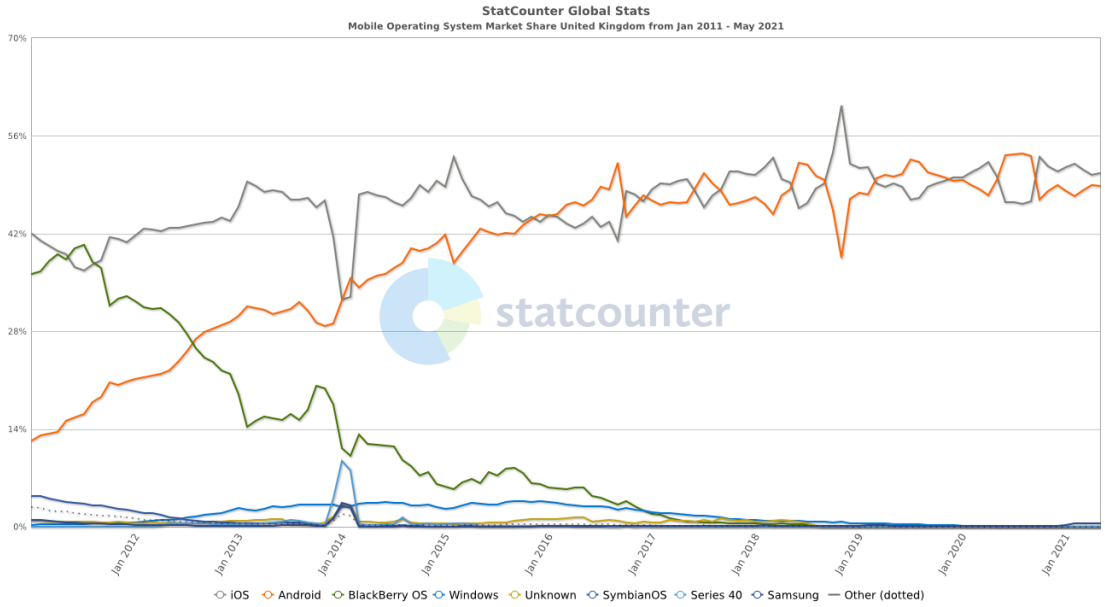

Figure 2: iOS vs Android Market Share Over The Last 10 Years [?]

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

1.4 Frontend: Programming Language

1.5 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 interative way that allows fordo you use relationship data in firebase stackoverflow modification throughout the application lifestyle and therefore easy scaling. Finally, the current model requires that each seperate barbers will need similar, although different data. For example, alough two barbers will likely share the product 'haircut', the description of each is likely to be entirely different making a normal SQL database difficult to implement.

1.5.1 The Target User

1.5.2 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 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 [**?**, **?**], there 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:

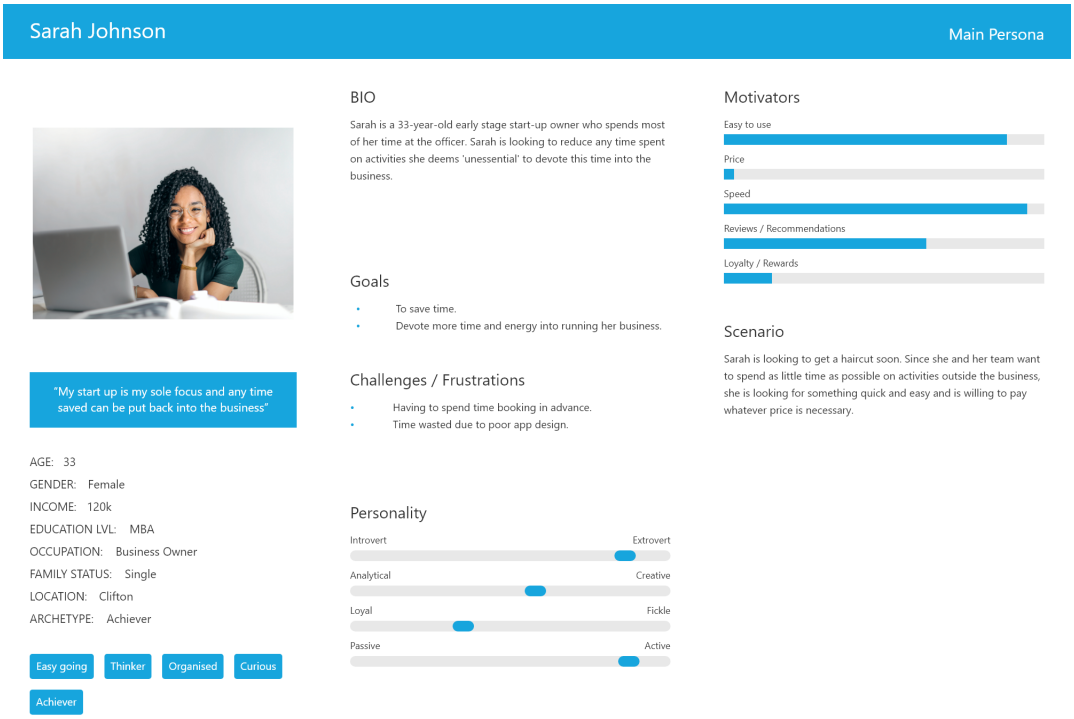

Figure 3: Persona 1

- Persona 2:

  Name: Claire Sheppard

  Profile:

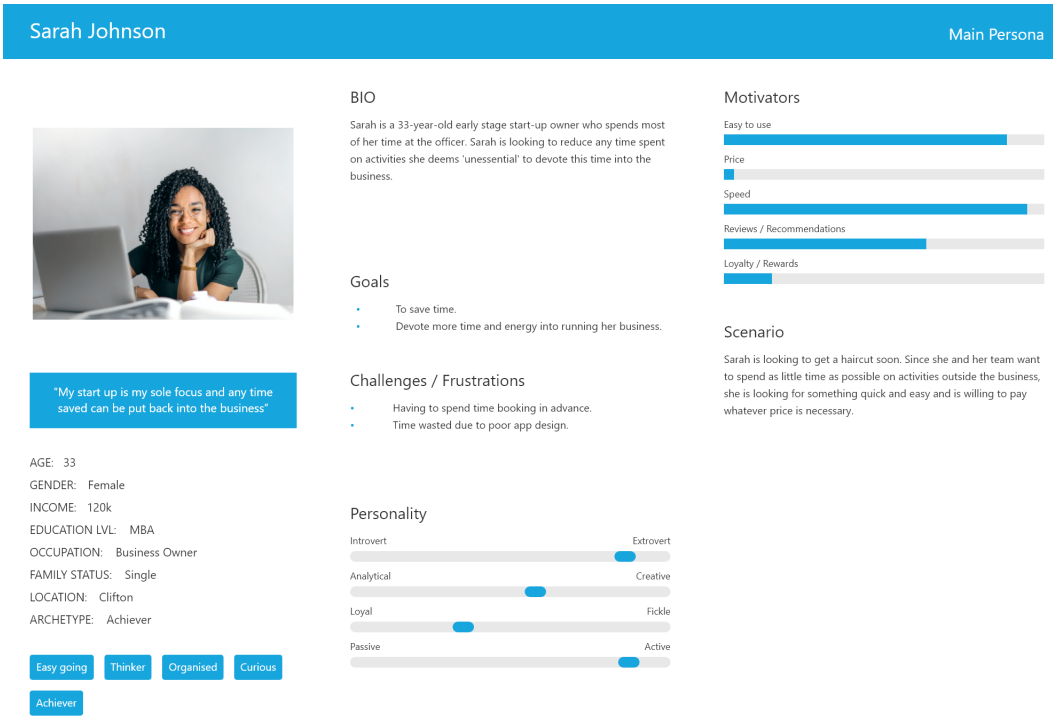

Figure 4: Persona 2

- Persona 3:

  Name: Emma Bradford

  Profile:

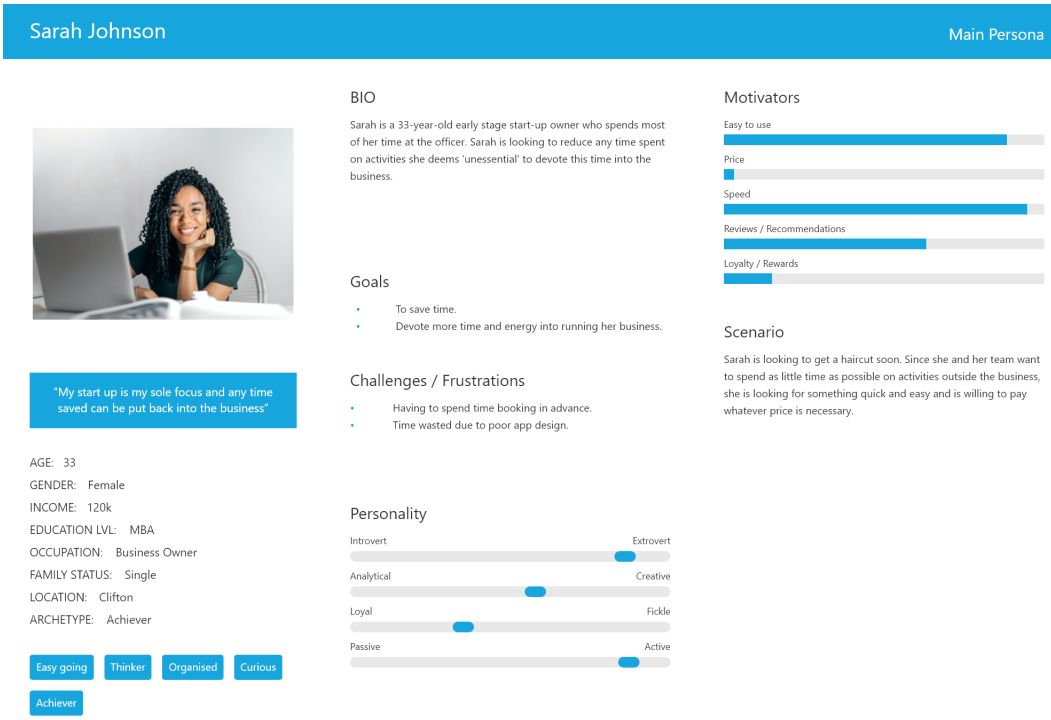

Figure 5: Persona 3

- Persona 4:

  Name: Claire Sheppard

  Profile:

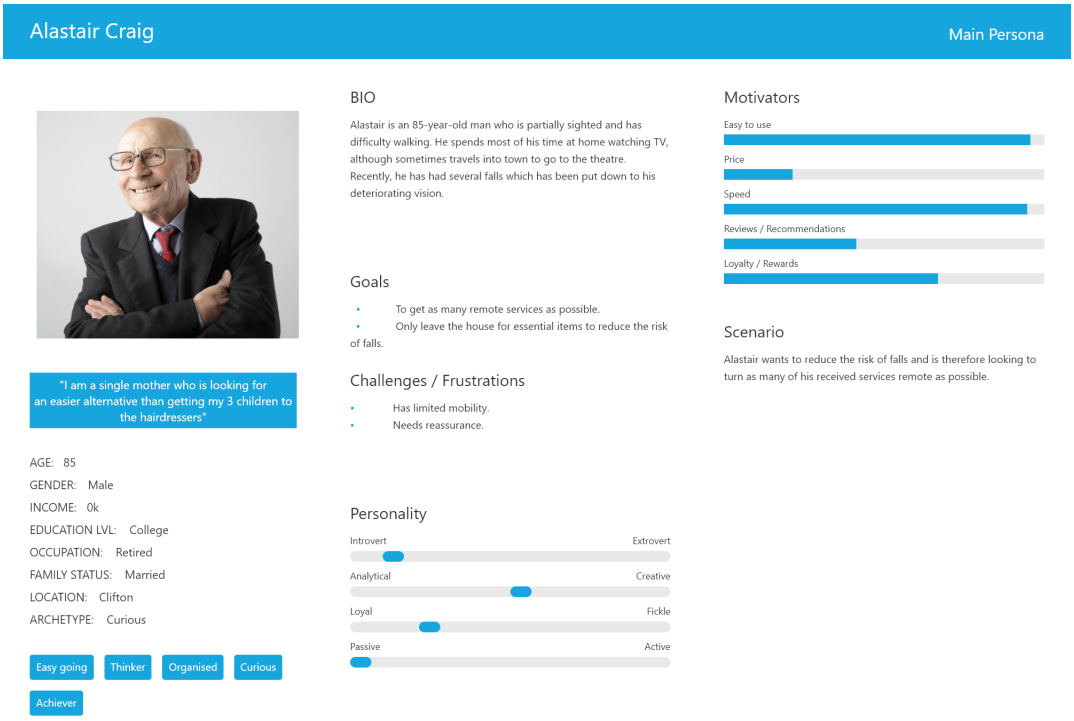

Figure 6: Persona 4

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 7 below.

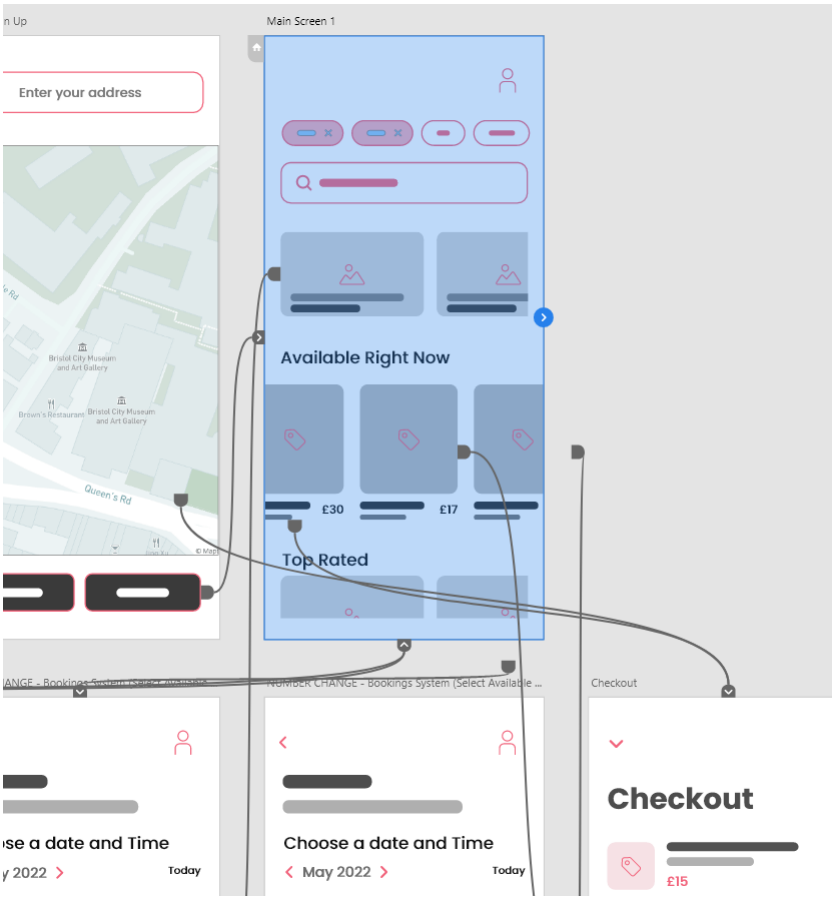

Figure 7: Component Interactivity within Adobe XD

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 Nielsons 10 Heuristics for User Interface Design [**?**] . 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.

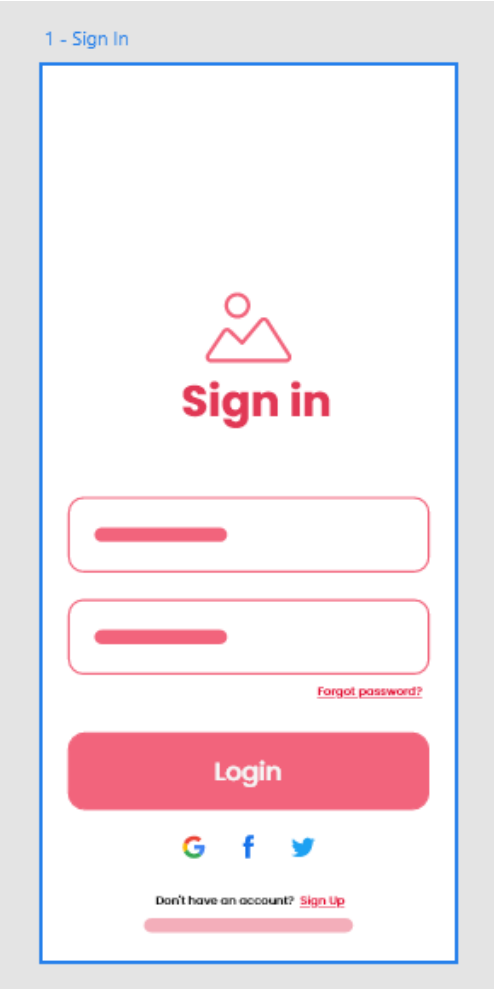

Figure 8: Sign In Page Made with Adobe XD

3 Production

3.1 Version Control

3.2 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

3.2.1 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; .

3.3 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.

3.3.1 Cupertino vs Material

The final project was built using material due to..

3.4 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 the user logs in/ signs up the current location is stored in the database so that the nearest barbers can be loaded.

3.5 Sprint X - Backend Design

Loading all parentBarbers/ barbers and products - When entering the app it is more effecient 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.

3.6 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 latitiude 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.

3.6.1 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.

3.7 Widgets and Common Items

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

3.7.1 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.

3.7.2 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.