Sweet Spot

A Web Application to Optimise Meeting Points in Greater London

Zahra Kanji

1403922

Supervised by Dr. Yulia Timofeeva

Year of Study: 2017-2018

Abstract

With a population of over 8 million people and an area of 1,572 km^2, Greater London is by far the largest city in the United Kingdom. As a result, residents frequently plan to meet with people who are based significant distances away and have to sift through a huge number of places to find a fair meeting point.

Existing applications for making this task easier fail to take into account the extensive and popular public transport network of Greater London. By considering the inconsistent relationship between time and distance in London, this project investigates the definition of 'fairness'. This research is then applied through a web application which recommends optimal meeting points for a group of people based on their starting locations and modes of transport.

Keywords

cost functions, fair meeting location, fair meeting point, meeting location, meeting points, optimal meeting location, optimal meeting point, public transport, travel time, web application

Acknowledgements

First and foremost, I would like to extend my gratitude to my supervisor, Dr. Yulia Timofeeva, for her constant support, feedback, and insight throughout the project. I would also like to thank Michael Gale for taking the time to observe my presentation and provide useful feedback on my work.

I would like to express appreciation for the support, both pastoral and academic, provided by Dr. Jane Sinclair and Dr. Matthew Leeke throughout the year. In addition, the suggestions from Professor Artur Czumaj and Professor Alexander Tiskin on the algorithmic approach to this project have been invaluable, as has the feedback from colleagues and friends who kindly tested the web application.

This project would not have been possible without the support and help of Harjot Singh who spent a great deal of time introducing me to the technologies used in this project, as well as Amy Preston whose proofreading and suggestions have greatly improved the quality of my writing.

I would like to give recognition to my friends Charlotte Taylor, Ellen Rose Wootten, and Gaggun Chaggar for their feedback and moral support throughout the project. And finally, I cannot thank my family enough for their patience, kind words of motivation, and support from the start of my degree to the end.

Contents

1 Introduction

The frustration of finding somewhere to meet up, be it to spend time with friends or to meet clients for a business meeting, is a common one. An ideal destination is usually somewhere in the middle that is fair for all members of a group to travel to. However, locating such a place can be inconvenient and difficult for many reasons: defining 'fairness' is subjective; considering the mode of transport used by each individual in a group is complicated; and analysing potential destinations is time-consuming and requires knowledge of the area and familiarity with it.

1.1 Background and Motivation

Existing theory behind finding optimal meeting points typically defines one as either the minimisation of the total cost for all people in a group, or the minimisation of the maximum cost [1]. An example of this is finding points on a map (or road network) with the smallest sum of distances for all people [2]. While this can be a helpful way of finding somewhere to meet, these approaches do not usually consider the concept of 'fairness' to be a part of the definition of 'optimal'.

Systems which do exist to find fair meeting points typically rely on using distance as the cost of the journey and try to incorporate fairness by finding equidistant locations (locations of an equal distance) for each person in the group. However this approach is not always appropriate for a number of reasons, especially in a city like Greater London.

Firstly, it is the largest city in the UK, both in terms of its area, 1,572 km^2 [3], and its population, 8,173,941 [4]. This means that it is common for people to live considerably far away from one another and there are a huge number of places to search through to find somewhere reasonable. There are over 65,000 restaurants, takeaways, or food shops in London [5], for example, and at least 3,000 pubs, bars, and nightclubs [6].

Secondly, and arguably more importantly, is London's extensive and complicated public transport network. Figure 1.1 illustrates the popularity of public transport amongst Londoners as well as their reliance on it. The percentage of Londoners who use public transport for commuting, 44.63%, is significantly higher than in other cities, the next of which is Edinburgh at 27.61%.

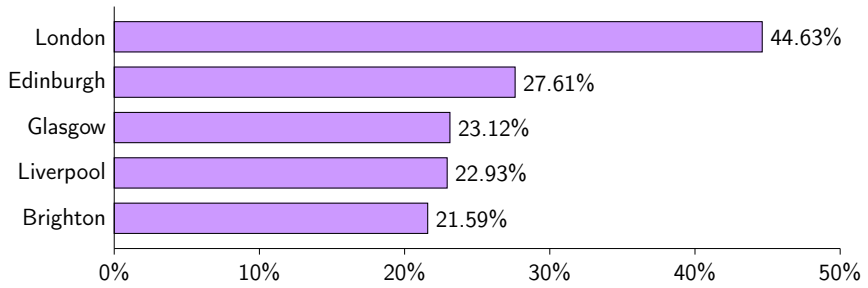


Figure 1.1: Commuting by bus, train or metro by city (2011, Great Britain) [7].

We can expect to see a rise in the popularity of London's public transport network as continuous improvements are made and show no signs of slowing down. In recent years, the bus network has been substantially invested in. Since the introduction of the Night Tube services [8], for example, weekend-only night bus services have been running more frequently. Furthermore, bus routes have been extended and improved, and by 2020, the streets of London will have around 3,250 Zero Emission and Ultra Low Emission buses [8]. Future work to the London Underground services include the opening of the Elizabeth Line in 2019 [9], and commuters on the tube network having access to 4G mobile coverage [10].

This popularity is not limited to public transport. The Santander Cycles Scheme in London has grown hugely since its introduction in June 2010 [11] and saw a record-breaking 600,000 journeys made in January 2017 alone. Furthermore, 54% of households in London have at least one car. [12]

Despite the strong possibility that members of the same group will be travelling using different modes of transport, very few of the existing applications for finding fair meeting points take this into account. In fact, only one even considers that group members may be using different modes of transport, and this application, Roudle [13], is not available for use in the United Kingdom.

One of the developers of Roudle said "Apps or websites that can help solve this problem are virtually non-existing, even in today's world. In the few cases they have been developed, these tools simply refer to the middle point on the map, not taking into account the complexity of the roads and real travel times." [14]

Another of these applications, Meetways, was created as the direct result of a team member considering "how many hours she had wasted in the car driving more than her fair share of miles to meet up with clients." [15]

These quotes capture the essence of the current problems very well. Solving these, as well as filling the gaps left by existing applications which will be discussed in the report, is the motivation behind this project.

1.2 Project Aims

The aim of the project can be summarised best as finding optimal and fair destinations for a group of people to meet at by considering the durations of their journeys. Defining one specific location as the 'fairest' is neither possible nor sensible as this depends on the opinions of the users. So rather than focusing on finding one such location, the emphasis will be on filtering out bad results and highlighting desirable places.

Figure 1.2 illustrates that the internet is a common medium for arranging transport, so a web application has been chosen as the platform to display the results of the algorithms which will be developed.

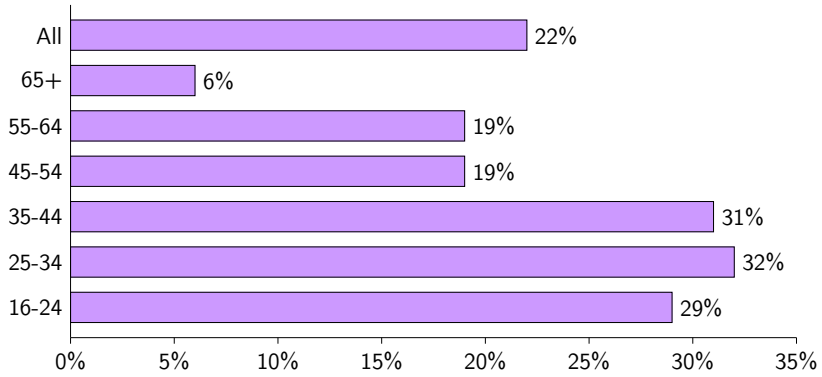


Figure 1.2: Use of intermediary websites or apps dedicated to arranging transport services by age (2017, Great Britain) [16].

This format will be beneficial for users as it provides the opportunity to present helpful information to aid their decision making process. It will also allow the user to interact with different definitions of fairness and analyse their options in more detail.

2 Ethical, Legal, Professional, and Social Issues

As this project will make use of existing APIs and software packages alongside collecting various data from people, it is necessary to recognise in advance any potential ethical, legal, professional and social issues. This chapter highlights important considerations to ensure that they are respected and acknowledged throughout the execution of the project.

Ethical Issues

The University of Warwick requires ethical consent if data is gathered from people outside the the "social circle, family, peers, or colleagues" of the developer [17]. This will mainly be relevant during the testing of the project's user interface and results once development has been completed.

Every effort will be made to avoid storing personal information, and only necessary data will be collected to begin with. This is especially important as this software will be collecting sensitive data from users such as their location, time of travel, and destination.

Legal Issues

Various software and APIs will need to be selected for accessing maps, public transport data, and front end development. These will come with terms and conditions which should be respected throughout the development of the project. Since the APIs will be used within the free allowances, usage limits must be respected. This will require the developer to work to a carefully planned schedule which allows testing and adjustments to take place over an extended period of time. Other considerations may include the storage of data collected from APIs and attributing work to its author based on copyright licenses. Where possible, open-source software packages will be chosen as they are freely available and may be redistributed and modified to suit the needs of the project with no restrictions or reservations.

Professional Issues

In order to maintain the highest professional standard throughout this project, care will be taken to evaluate existing systems and research fairly by discussing only factual information, both positive and negative. This is to ensure that work undertaken by other developers and researchers is not unfairly disrespected or defamed. Furthermore, any content will be correctly cited so that readers can find the source of the information and due credit is given to the rightful owners.

The software developed will need to be well-structured, well-written, and tested extensively so that further development can take place in the future and to ensure a smooth and pleasant user experience. This should include good design choices as well as good documentation.

The British Computing Society has a code of conduct [18] which outlines desirable behaviour such as having due regard for the legitimate rights of Third Parties and seeking and accepting honest criticisms of work. Throughout the project, the behaviour of the developer and decisions made will abide by this code of conduct.

Social Issues

As this project aims to eventually provide results to users of the software through an interface, it will be important to investigate what features and layout will be most beneficial. It is also vital to keep track of what data is collected; how it will be used; whether it will be shared; and the level of optimality (if any) guaranteed by the results. This is so that the information is ready to be formally expressed to any users who wish to access it.

3 | Existing Systems

To understand the need for this project and gain an insight into existing solutions, their usability, and their limitations, an extensive review of seven existing systems has been completed on web applications with similar aims to this one – that is, to find fair places for groups of people to meet. This research is split into four categories; the User Interface (UI); the search options available to the users; the algorithmic approach to solving the problem; and the quality of the results returned.

Although these systems discussed independently of this project specifically, the aim is to use this review to establish project requirements based on the positive and negative aspects discovered. These will ensure that the developed system addresses the issues where existing systems fall short; is completed to a high standard; and gives the intended audience what they need.

Table 3.1: Features available in existing systems.

Key		Roudle	Geo Midpoint	What's Halfway	Meetways	A Place Between Us	Where To Meetup	Shall We Meet in the Middle
✓✓	Feature implemented well							
✓	Feature implemented							
	Feature unavailable							
aesthetically pleasing		✓		✓	✓✓		✓✓	✓✓
different modes of transport		✓		✓	✓			✓
equal travel time				✓				
equal travel distance		✓	✓✓	✓✓	✓✓		✓	✓✓
arrival/departure time								
destination ratings		✓	✓✓	✓✓	✓✓		✓✓	
directions			✓	✓	✓✓	✓		✓
can be used in London			✓✓	✓✓	✓✓	✓✓	✓	✓
group size greater than two		✓✓	✓✓	✓✓		✓✓		
responsive display		✓			✓✓	✓	✓	✓
destination type can be chosen		✓	✓✓	✓✓	✓✓	✓✓	✓✓	

Table 3.1 shows how the different systems compare to one another. These will be discussed in more detail in the next sections.

3.1 Roudle

Roudle [13], the "meeting location planner and optimizer" was created as the solution to the frustration of "organizing meetings with people from different areas" [19]. This web application is in Dutch and works within the Netherlands, Belgium, and Luxembourg. To analyse the site fairly and accurately, friends who have lived in the Netherlands were asked to test the website and give comments based on probing questions such as "How do you find the User Interface?", "Would you use this web application?", and "Do you find the results accurate and fair?".

User Interface

Although the User Interface in Figure 3.1 is reasonably clear and straightforward to follow, the map is relatively small and can be difficult to look at. Furthermore, there is a lot of arguably unnecessary text on the page which is all in a similar font and size, making it difficult for the user to know what to read first. This has the potential to make using the web application time-consuming and frustrating which adds to the inconvenience of finding a place to meet up. A small glitch present on the User Interface is that a location cannot be selected from the autocomplete dropdown using the up and down keyboard keys when a user inputs their address.

Search Options

In the free version of the application, up to six starting addresses can be entered. Users in the Netherlands can also individually specify whether they will be driving or using public transport. Users can suggest the type of place they would like to meet at, such as a coffee shop or restaurant, however this can only be done after the initial search and, according to testers, does not consistently return good results.

Algorithmic Approach

The creators of the website have discussed in interviews that they use "smart algorithms" [14] to suggest locations to users. To discover more about their application, the developers were

contacted but they were unable to share these algorithms due to the commercial nature of their product. They did however explain that their application is more focused on finding 'efficient' meetup points rather than 'fair' ones. In the Netherlands, this meant the minimum travel time. But when the service was expanded to Belgium, they discovered that people there preferred the most equal travel times. The conclusion they reached was that cultural and infrastructural preferences vary from region to region.

Quality of Results

Firstly, this application produces results which are zones rather than actual places (shops, cinemas, parks). They appear to be ranked by firstly prioritising distance, then price. On top of these values, users can also view the duration of their journey, as well as the unique feature of their carbon dioxide emissions. One of the website testers commented that it is unclear how this information is calculated and whether it is accurate. For example, public transport costs seem to be calculated as though distance and cost are related linearly, which is not necessarily the case.

Once the user has selected a region, they can find more specific places by entering a search term or choosing from the list available. When these locations are selected by their markers on the map, further information is displayed such as the address, phone number, website, twitter, and the option to view it on OpenStreetMap or email the location.

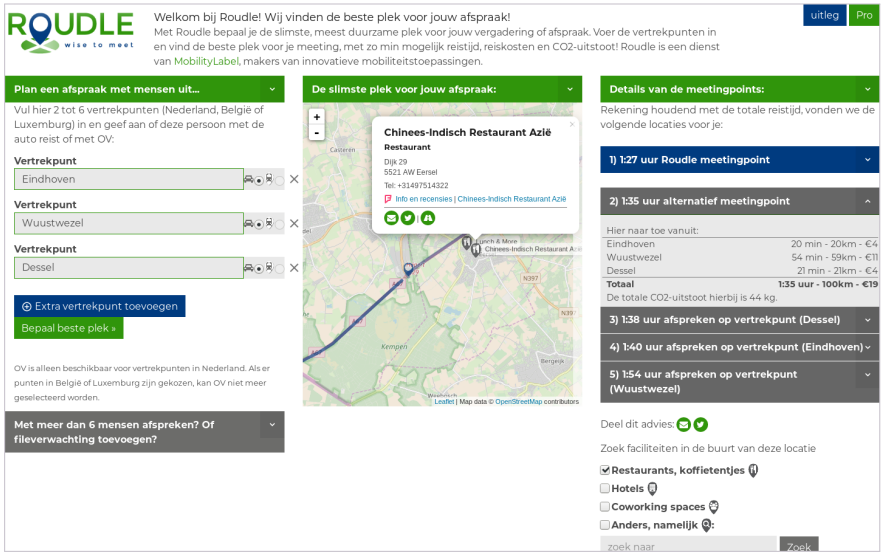


Figure 3.1: Roudle.

3.2 Geo Midpoint: Let's Meet in the Middle

Geo Midpoint is a website with many features and tools such as a random point generator and a bearing calculator. One of these tools is 'Let's Meet in the Middle' [20] which finds some ideal points of interest between two or more addresses.

User Interface

As we can see in Figure 3.2, the User Interface of this website is quite basic and, at times, unintuitive. One example of this is that it is not immediately obvious how to view directions for the different group members once a place has been suggested. This is both time-consuming and inconvenient, as well as frustrating for the user. In general, the elements of the page are quite small and fail to make good use of screen space, giving it a crowded feel and impacting the user experience negatively. There is no special layout for smaller screen resolutions, but it is not too difficult to use on mobile.

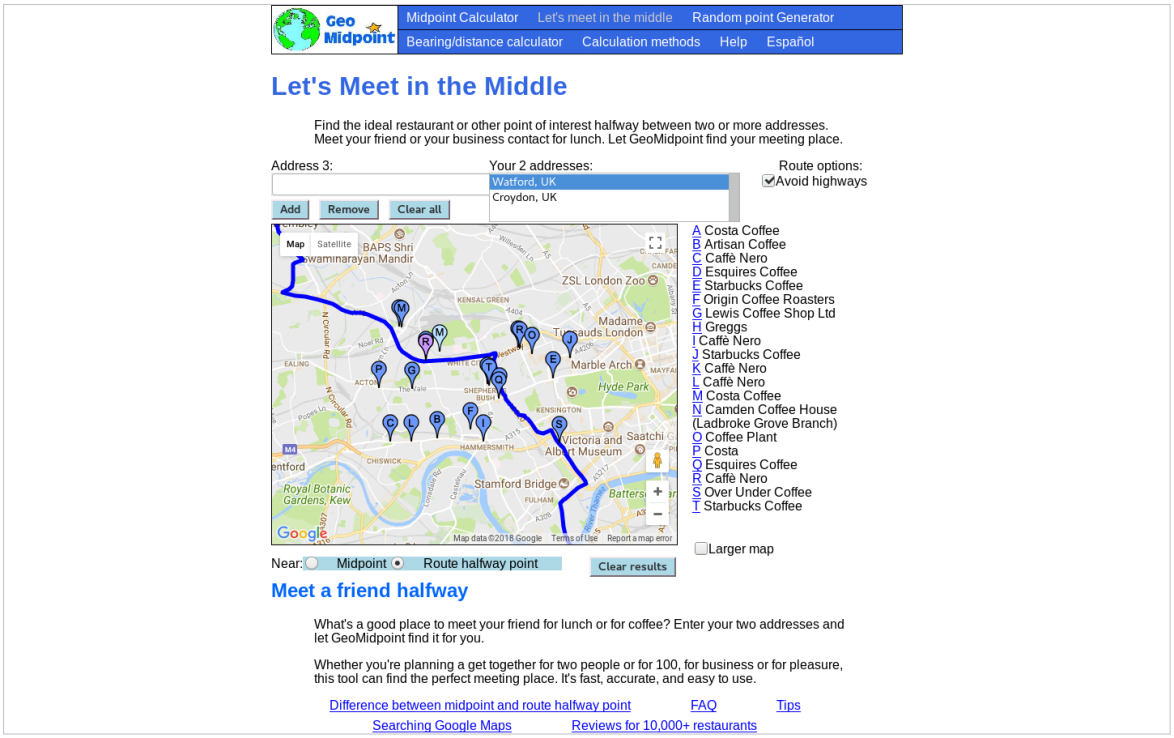


Figure 3.2: Geo Midpoint: Let's Meet in the middle.

Search Options

A positive feature of the application is that there is no upper limit on the number of people in the same group who want to meet up together. Categories of destinations can be selected by the user entering a keyword, and we can look for places near the geographic midpoint of all users, or near the route halfway point if there are two users.

The option to avoid toll roads is not available on this web application which could be a problem for users who are trying to avoid these regions. However, highways can be avoided as long as the user remembers to select this option before entering their addresses; selecting it afterwards has no impact on the results.

Algorithmic Approach

The 'route halfway point' uses real-time Google Maps data to find a route between the two users as though one were driving to meet the other, and finds its midpoint in terms of distance. The 'midpoint' option uses the centre of gravity of all the users as the midpoint. The results then suggest 20 places located in the radius around this midpoint.

Quality of Results

There is no formal ranking of results, just Google Maps results are shown. There are directions available to and from each destination for each user. The address and rating of the location can be seen, however the 'more information' option and 'sending' options for each suggested location do not actually work.

3.3 What's Halfway

What's Halfway [21] is a website that offers two main features: a meeting planner and a journey planner. Their work is extensive and they are currently developing APIs for their services with the intention for partnering websites to embed the various functionalities on their own sites.

User Interface

At first glance, the User Interface in Figure 3.3 is bright with colourful graphics and a mascot. Initially it is easy to begin a search, however when results appear, it is easy to miss them as they appear far below the map on the screen and are not within the immediate view of the user. Although the option is available to make the map full-screen, the page is quite congested as a result of the large logo; the small and difficult to navigate map; and the abundance of adverts. There is no specific mobile friendly version of the website, but again it is still usable on smaller screen resolutions without too much difficulty.

Upon display of the results, there are occasionally glitches in the User Interface and it is unclear what selecting a location actually does. For example: choosing a place from the suggested list of locations does not select it on the map, and selecting a location on the map does not present the information available about it in the list of locations. This results in quite a chaotic and confusing experience for the user, especially the novice user.

Search Options

Up to ten starting locations can be entered and the type of place to meet up at can be chosen. The user can also choose whether they would like to meet up at the halfway point in terms of time, or at the halfway point in terms of distance. This makes it the only application to consider time as part of the definition of fairness. When there are only two users, further options such as choosing a mode of transport and adding waypoints (places to travel via) are available. However, this only seems to work occasionally as both users are assumed to be travelling using the same transport method and Google Maps cannot always find the required route using public transport.

Algorithmic Approach

When there are only two people travelling, a route is mapped between them using real-time Google Maps data, and the midpoint is found based on the user's preferences (i.e. time or distance). A radius is then drawn around this point and destinations which match the chosen description are found. When there are more than two people, the geographic midpoint is found and the radius is drawn around this point instead.

Quality of Results

The results here are almost identical to those found by Geo Midpoint: Let's Meet in the Middle, most likely on account of the fact that they use the same algorithm. A benefit for users of this website is the opportunity to refine results by increasing or decreasing the size of the radius. They can also move the centre of this radius to suit their requirements or experiment. The address, website, number and a review of each location is given, and users can choose to email themselves directions to the place from each starting location which may be helpful to users who will not have access to mobile data during their journey.

Figure 3.3: What's Halfway.

3.4 Meetways

The Meetways [22] website displays lots of information convincing people to use it, for example to find the best place for a business meeting or a date, or discovering a restaurant between your home and office. These explanations of how the website can be used and why it is so helpful add a pleasant dimension to the system for the user and implies to the user that a great deal of consideration has gone into the system and its.

User Interface

We can see that the User Interface in Figure 3.4 has a professional aesthetic with a clean layout and a traditional blue and white colour scheme, giving us the impression that it was built with care and will do its job well. Once results have appeared and we choose to look at further details about a location, it is initially difficult to work out how to return back to the list of recommended locations without repeating the entire search. This usability issue may be small, but it is irritating for the user and prevents them from being able to undo mistakes easily.

The mobile version looks great, however the map on mobile screens is very difficult to navigate due to its small and unusual shape. A small but noticeable mistake throughout the UI is the failure to display the correct icon of the chosen mode of transport when directions are shown. The car symbol is always shown, even when walking directions are displayed, confusing the user.

Search Options

The users of this website are limited to two starting addresses and can select one mode of transport between them. They can also suggest the type of place they would like to meet up at. These limitations can be problematic if more than two people are trying to meet as a group, however the website does not advertise itself as aiming to solve that specific problem.

Algorithmic Approach

The algorithm very simply maps a route between the users using real time data from Google Maps and finds the midpoint in terms of distance, around which it recommends places.

Quality of Results

The place results are powered by Yelp and show the name, address, rating and a review of ten places in the vicinity matching the description provided. It is unclear if and how the results are ranked but directions are neatly presented for each user to arrive at their destination.

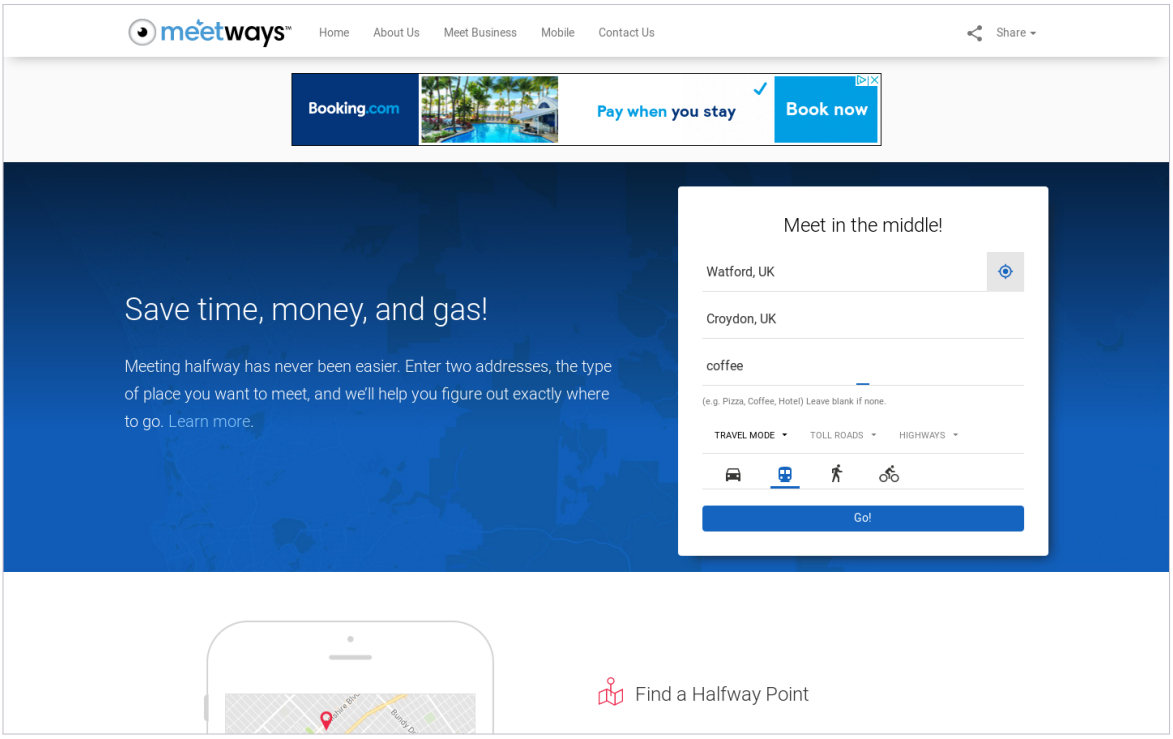


Figure 3.4: Meetways.

3.5 A Place Between Us

A Place Between Us [23] is a simple web application which uses Google Maps APIs to find a place between a group of people.

User Interface

Although the User Interface in Figure 3.5 seems quite basic, it resizes surprisingly well on smaller screen resolutions with the exception of the map. On mobile, it is easy to get trapped in the map which means the user has to refresh the page to get back to the list of results.

Users are given a letter of the alphabet as their label. When the number of users exceeds 26, users start being labelled using symbols instead of letters which is a little unusual, but also probably not how the website was intended to be used. Regardless, it is slightly confusing and could be considered a usability issue. A further issue is that deleting user B, for example, does not rename user C to user B. This leaves a user A and a user C but no user B.

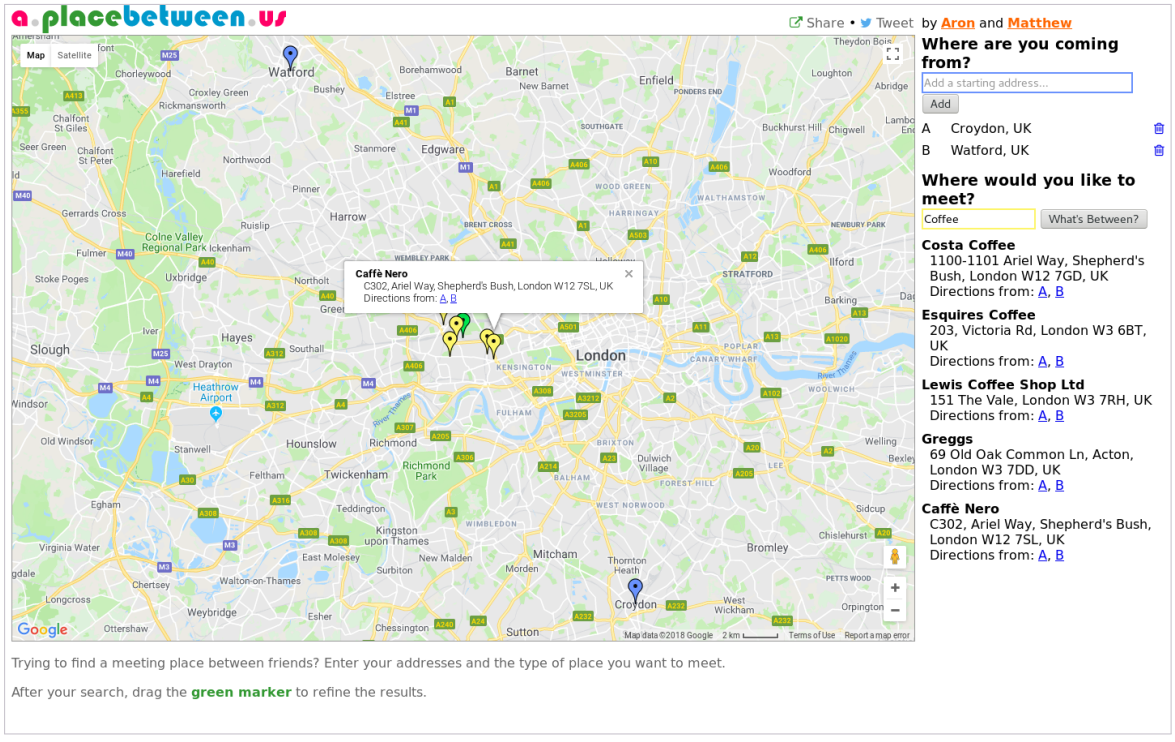


Figure 3.5: A Place Between Us.

Search Options

There is no upper limit for how many starting addresses can be entered, and the users can also type a keyword to find a relevant place.

Algorithmic Approach

This website very simply finds the geographic midpoint of all the addresses and displays five results matching the keyword around this midpoint.

Quality of Results

One of the strongest aspects of this web application is its speed; results appear almost instantly. The application also allows the user to refine their results by moving a marker to redefine the midpoint, giving them the freedom to experiment and interact with the list of results. Lastly, when the user clicks the link to find directions, they are redirected to the Google Maps website or app to see the journey. This is a really nice solution as it gives users the opportunity to easily browse through different routes, modes of transport, and calendar dates on a familiar platform.

3.6 Where to Meetup

Where to Meetup [24] is a simple application is powered by Yelp and MapBox. It has a very pleasant feel to it as a result of the informal language and bright colours used.

User Interface

Although the User Interface in Figure 3.6 is easy to use, simple to understand, and clear to navigate on the desktop version, there are some significant issues. After a search has been completed, the map which shows up appears not to be correctly implemented as it breaks when trying to zoom in and out. Furthermore, there is a lack of integration between the search results list and the markers on the map, meaning that clicking a map marker provides incomplete results with key pieces of missing data such as routes and journey distances.

The mobile version also has has some limitations. It is easy to get stuck in the map section of the application, and the list of results are not connected to the place markers on the map at all. Furthermore on the web application in general, the polylines (lines which show the journey for the user) do not clear, nor does the map resize to show the entire journey. Lastly, the place markers do not stand out particularly well on the map on account of being blue like the rivers with no borders.

Search Options

Two addresses can be entered and there are predefined categories of destination types to choose from. Although in theory it gives the user the chance to filter their search results, this feature is very confusing. There are two drop-down menus. The first allows the users to say the kind of place they would like to go to such as ‘meet for golf’ or ‘meet for dinner’. The second gives a list of different types of food places and seems like it should only be enabled if the user has chosen a place to eat as their preferred location. The ability to use both these menus at the same time means we can accidentally choose “meet for golf, I feel like sandwiches” and the search then ignores the original category, i.e. it recommends places to get sandwiches rather than places to play golf. This application also allows users to decide whether they would like to find places close to the first person’s address, the second person’s address, or in the middle of the two addresses.

Algorithmic Approach

Depending on the user's choice, a small radius is drawn around the route's midpoint or one of the two starting locations and places within this radius are suggested. The size of this radius appears to be fixed meaning that if no places are found within the radius, it is not increased and so the list of results could be empty.

Quality of Results

Although the website is only supposed to work in the United States, for two addresses, some results can be found in other parts of the world. The results show location addresses, pictures, a rating, and the distance of each journey. However, a noticeable glitch is that the distance of each journey does not display correctly when a place is chosen by its marker on a map. These places can be saved and emailed to other people which is helpful to share the list of recommendations with a friend.

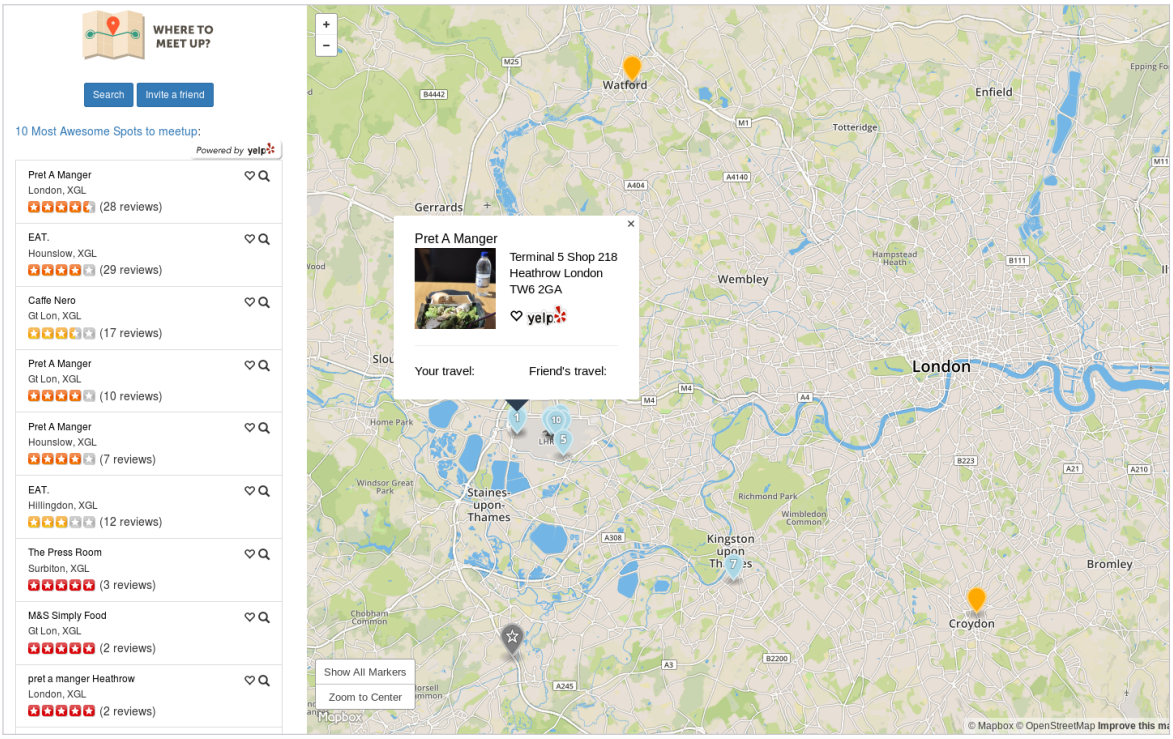


Figure 3.6: Where To Meetup.

3.7 Shall We Meet in the Middle

Shall We Meet in the Middle [25] was released in 2014 and works in a similar way to the other systems discussed in this review.

User Interface

The User Interface in Figure 3.7 is easy and comfortable to work with as there is a big clear map in the background. At first glance, the site is responsive on smaller screen resolutions, however search results do not actually get displayed on the mobile site which is a significant issue. The results on the desktop version appear in list form on the left hand side and can be clicked to be viewed on the map.

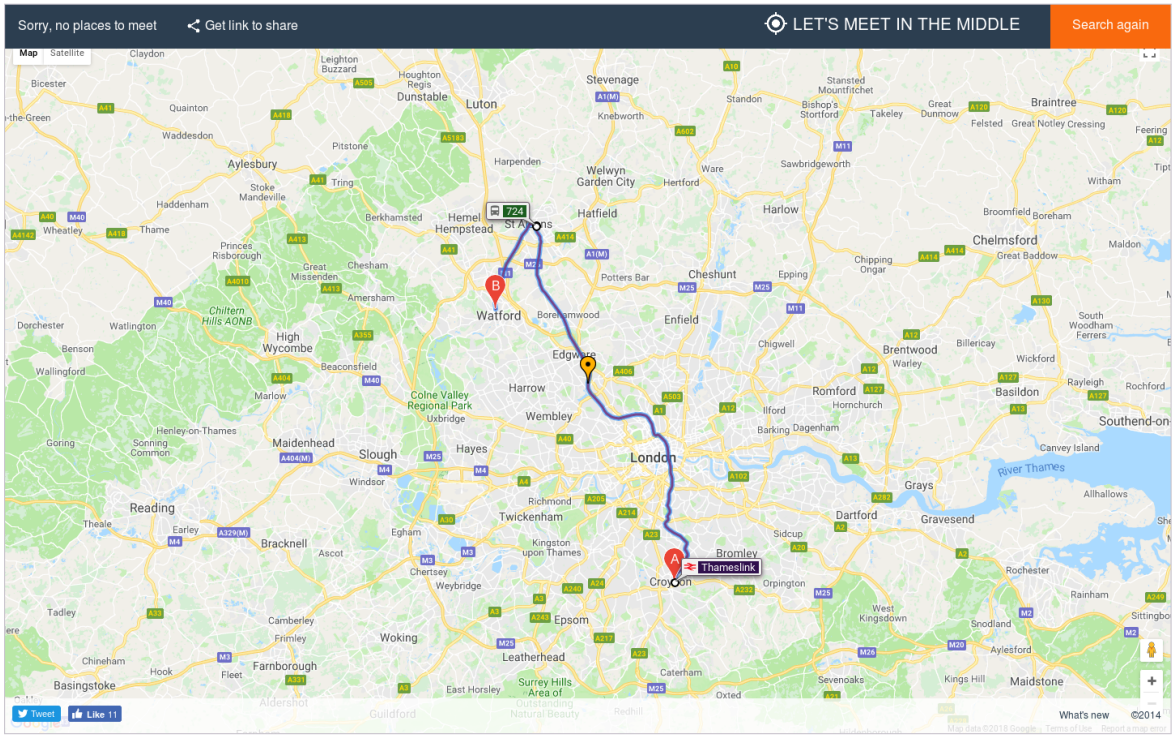


Figure 3.7: Shall We Meet in the Middle.

Search Options

Two addresses can be entered and a method of transport can be chosen, however no category or keyword can be entered to filter the results.

Algorithmic Approach

A route between the two addresses is found using the selected mode of transport, the midpoint is identified, and a search is done for places of interest within a radius of 5% of the total distance between the two users.

Quality of Results

Interestingly, although the application was developed by someone based in the United Kingdom [26], it works quite poorly in the UK. Most searches had zero or one suggested place making the application ineffective in many areas. Suggested locations show address, phone number, website. Like with most of the other systems, there is no ranking of results by any specific parameter. The application does not display directions, distances, or durations of journeys, leaving users with extra work to do to find the right place to hang out.

3.8 Summary

As anticipated, one of the biggest issues found in these applications was the inability for group members to select different modes of transport from the start. The result is that no matter how accurate the calculations made are and no matter how 'fair' the recommended destinations are, they are incorrect. The journey between two places by car and the journey between the same two places by train, for example, will almost always be different.

Another issue was the focus on distance as the main parameter of fairness rather than time. Time and distance in London do not have a constant relationship because of the huge variation in transport services and other factors like congestion [27]. So when an application only considers distance, it finds locations which have the potential to be unfair, at least with regard to journey durations.

Continuing this discussion on variation, driving times and public transport schedules especially vary significantly over the course of a day. None of these applications request the time that a group would like to meet at and simply use Google Maps real-time data. So if a group are making their plan late at night but are aiming to travel in the afternoon, for example, the results presented will be quite inaccurate.

The final problems which will impact how this project is designed were the glitches within the User Interfaces and the poor design choices. However small, they can impact the user experience negatively. In these systems, problems ranged from the unintuitive placement of features and poor layout on mobiles to the lack of seamless integration between UI elements and user interactions with them.

This project seeks to learn from these limitations as well as the positive aspects to make the process of finding fair places to meet up as smooth as possible. A summary of features which would make a good web application was presented in Table 3.1 and it is clear from this table that no existing system is perfect for Londoners.

4 | Specification

Having identified what makes a good web application and what improvements need to be made on existing systems, the project should achieve all the features displayed in the introduction in Table 3.1, with the exception of equal travel distance. However not all of these are required to meet the overall aim of the project - to optimise meeting points.. Therefore the requirements have been split into Primary Requirements, those which must be completed; Secondary Requirements which will improve the user experience but are not vital; and possible extensions. Functional Requirements are labelled **FR**; Non-Functional requirements are labelled **NFR**; and extensions are labelled **E**.

4.1 Primary Objectives

FR1: The User Interface should allow up to four starting addresses to be entered.

FR2: The User Interface should allow a mode of transport for each person to be selected.

FR3: The User Interface should allow a type of destination to be chosen.

FR4: The User Interface should allow either arrival or departure time to be selected.

FR5: Users should be able to minimise their total time travelled.

FR6: Users should be able to minimise their total distance travelled.

FR7: Users should be able to find the fairest location with respect to the duration of their journeys.

FR8: The suggested destinations should be shown in relation to the starting addresses on a map to visualise the journey for each user.

4.2 Secondary Objectives

FR9: Include ratings of the suggested meeting points.

FR10: Rank the suggested locations.

FR11: Show directions for each person travelling.

NFR1: The User Interface should adapt and respond well to different screen resolutions.

The following non-functional requirements are based on Nielson's Usability Principles for User Interface Design [28].

NFR2: The system status should always be apparent to the user so they know what is going on.

NFR3: The system should speak the user's language by using familiar words and concepts which appear in a natural order.

NFR4: The system should provide the user with the control and freedom to undo and redo actions.

NFR5: The system language, layout, and actions should be consistent throughout.

NFR6: The system should be able to prevent and recover from errors.

NFR7: The user should be able to recover from errors easily.

NFR8: Users should be able to navigate the system by recognition of objects and options without relying on memory.

NFR9: The system should be flexible and efficient to use for both the expert and novice user.

NFR10: The User Interface should have aesthetic and minimalist design which considers the display of information carefully.

4.3 Possible Extensions

An informal aim of the project is to create a scalable application which can be easily extended for future work. Some examples of further extensions include:

E1: The User Interface could produce a link (to Google Maps) for each traveller's directions.

E2: The User Interface could save midpoints and starting addresses for later use.

E3: The User Interface could display the price of each journey using Transport for London data and petrol prices in the UK.

E4: Users could be able to find the fairest places based on journey prices.

4.4 Constraints

4.4.1 Hardware Constraints

Although the project will be hosted locally on machines in the Department of Computer Science, the application will be designed in such a way that it could be deployed and hosted by a live web server for multiple users to access at once. The recommended browser will be Google Chrome and all testing will take place using this browser.

4.4.2 Software Constraints

The use of APIs to access public transport data and other journey data will apply some constraints to this project too. As discussed in the project background, there are large quantities of places to meet at in Greater London. Given that data will be required for each user and each location, to keep this problem solvable within the limits of these APIs, a maximum of 200 places and four users will be considered at any time. However, this will not influence structure and design of the back end which will be able to handle any arbitrary n number of people and locations.

4.4.3 'Fairness' Constraints

A project which aims to explore and capture a mathematical definition of fairness, which is a subjective concept, has the potential to consider a wide range of human factors. For example, many people would not consider it unfair for one person in their group to travel for 10 minutes longer than everyone else. A further example is that it might be fairer for people who are walking or cycling to travel for less time than people driving. To make this problem solvable, a constraint is applied such that any algorithms implemented will consider the duration of journeys independently of the mode of transport, i.e. that no special considerations will be made for people travelling by foot over people travelling by car.

5 Project Management

5.1 Methodology

The Agile Manifesto outlines many principles, but two emphasised aspects are producing working software and being able to respond to change [29]. One way of implementing these is through the planning of short, adjustable sprints for focused work over a brief period of time.

More specifically, Feature Driven Development (FDD) is an incremental and iterative process based on Agile which concentrates on the requirements of the client. The basic outline of FDD is typically as follows [30]:

1. Develop an overall model
2. Build feature list
3. Plan by feature
4. Design by feature
5. Build by feature

This combination of focusing on the whole product to produce software based on the needs of a user, whilst maintaining an adaptable schedule, suits the nature of this project quite well. For a sole developer, not every aspect of Agile Software Development is applicable, and there are other types of Agile besides FDD which will be beneficial. As a result, this order of events will be adapted and various techniques will be combined to produce the final plan outlined below.

Firstly, a basic working system outlined by the primary requirements in the specification will be built. This will be followed by considering a list of features based on the secondary requirements and selecting the most appropriate one to implement given the current state of the project. They will then be designed, implemented, tested, and integrated into the system. The goal is to have a complete and functional system at the end of each cycle before the next feature is developed. This process is illustrated in Figure 5.1.

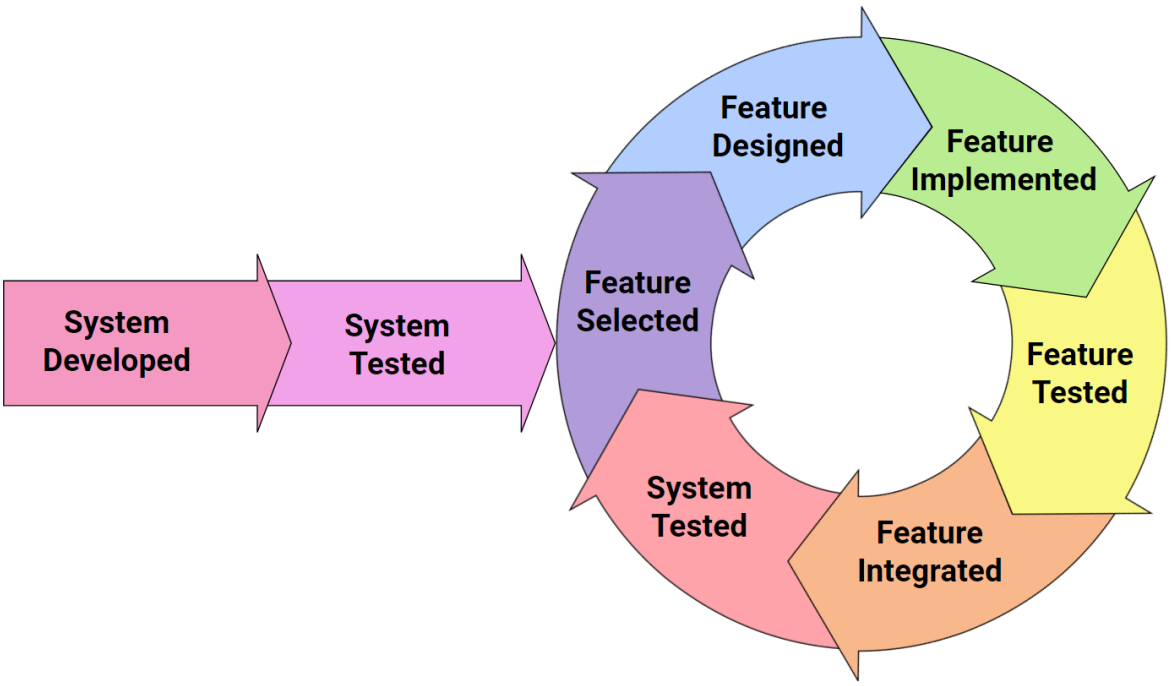


Figure 5.1: Adapted Feature Driven Development methodology.

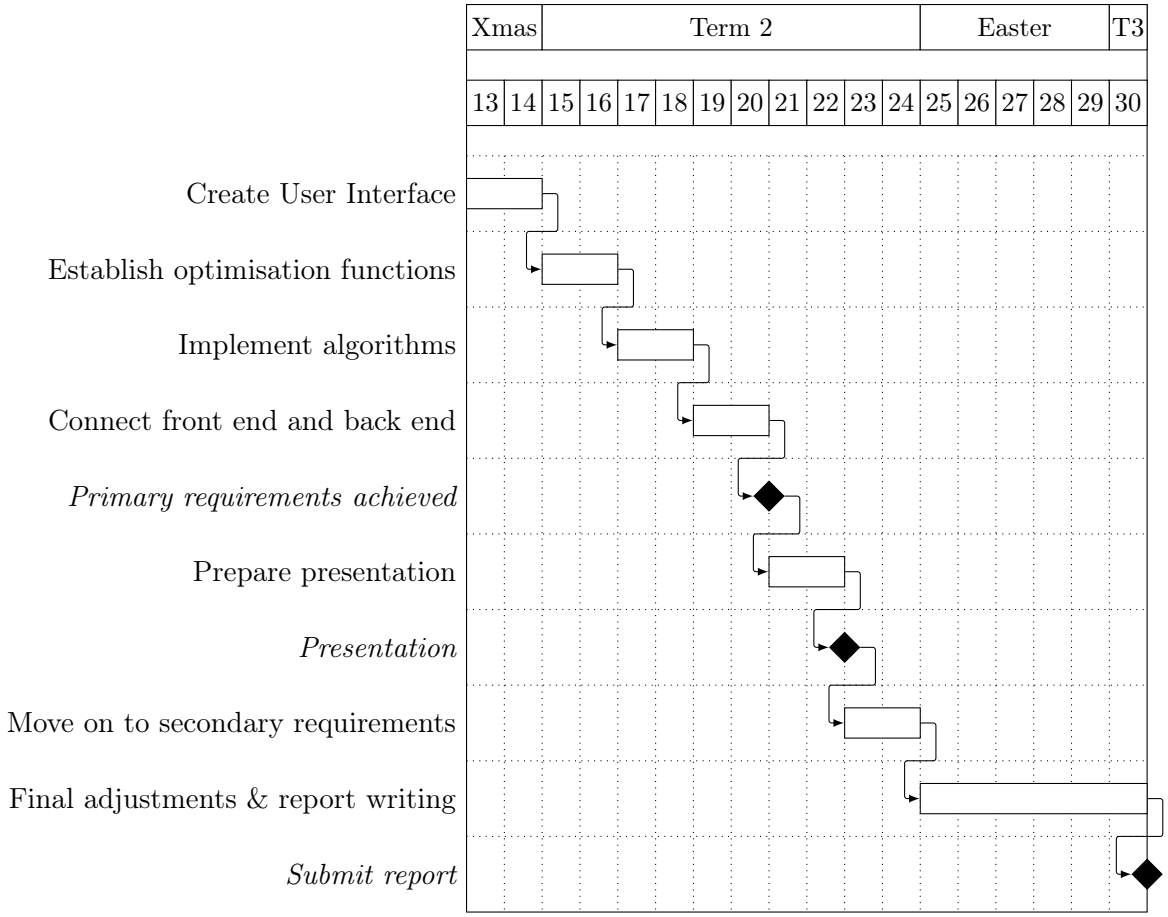
5.2 Schedule

As the developer's first software development project, a significant amount of time has been and will be required to conduct research; explore the technologies available; and learn how to use them. Once this is complete, the project aims to follow the plan outlined in Table 5.1.

Although change to this plan is expected and will most likely be required given the unknowns of the project, it is reasonable to expect that, at least approximately, the project will fit into the slots allocated.

The schedule is split into several two week sprints, each with a specific outcome. The assigned time includes considerations for slack time; the chance for the order of events to be altered; time for other commitments and coursework; and the opportunity to mitigate any unforeseen circumstances which may occur throughout the course of the project.

Table 5.1: Gantt chart showing project schedule.



6 Design and Implementation

This chapter discusses in detail the processes which took place to achieve the final product. We begin with an understanding of what users would like through User Stories and turn this into an overview of the system that will be produced. Using this system overview, technologies that can help achieve the required system will be researched and finalised.

After this, we will go into the initial design of the User Interface, the final UI, and the implementation of this UI. Next, data collection and processing are covered before the algorithms which use them are explained. Once all these have been completed, system integration is shown and the final system architecture is expanded upon.

6.1 User Stories

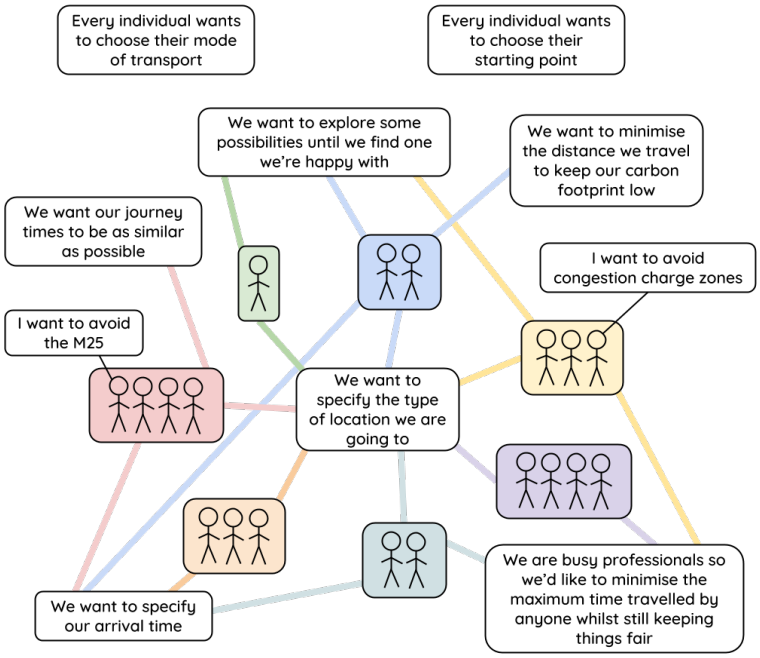
Since the project aims to give users the ability to access and explore fair locations through a User Interface, User Stories were composed. These are short descriptions of a feature told from the perspective of the user [31] and are beneficial for a number of reasons: keeping the developer focused on what users actually want when decisions need to be made about the system, as well as helping with planning and tracking the features which should be in the final product.

Although typically there is one user of a web application, this web application considers there to be up to four users at once who form a group. These User Stories are represented in diagrams showing groups of people who will be using the application. They are split into two sections, one which discusses what information they should be able to give the system and one which explains what information they expect to get out of the system.

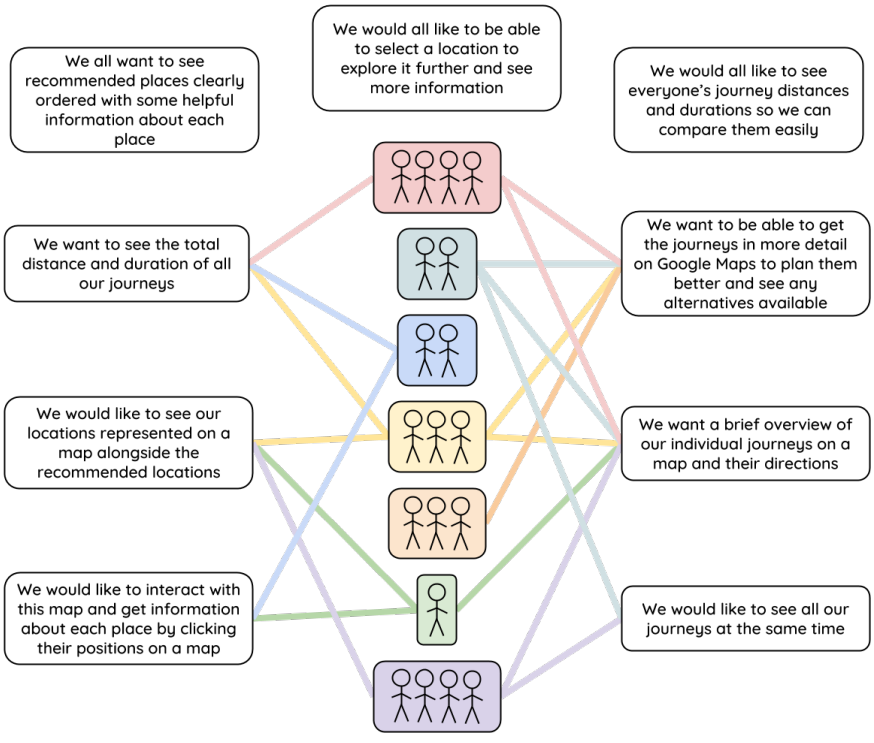
In Figure 6.1a, we can see the aspects users want the system to take into consideration before giving them results. This ranges from avoiding congestion charge zones to specifying their starting points and modes of transport.

In Figure 6.1b, the same groups are shown describing what they expect as the outcome of their search. For example, the people in the red (first) and yellow (fourth) groups who wanted to avoid the M25 and congestion charge zones respectively would both like a brief overview of their journeys and to see them on a map. Another example is that every group

would like to be able to see individual journey distances and durations for easy comparison.



(a) Search.



(b) Results.

Figure 6.1: User stories.

6.2 System Overview

Taking into consideration the Specification and User Stories mentioned previously, the high level overview in Figure 6.2 has been produced. It shows how the system should work from start to finish.

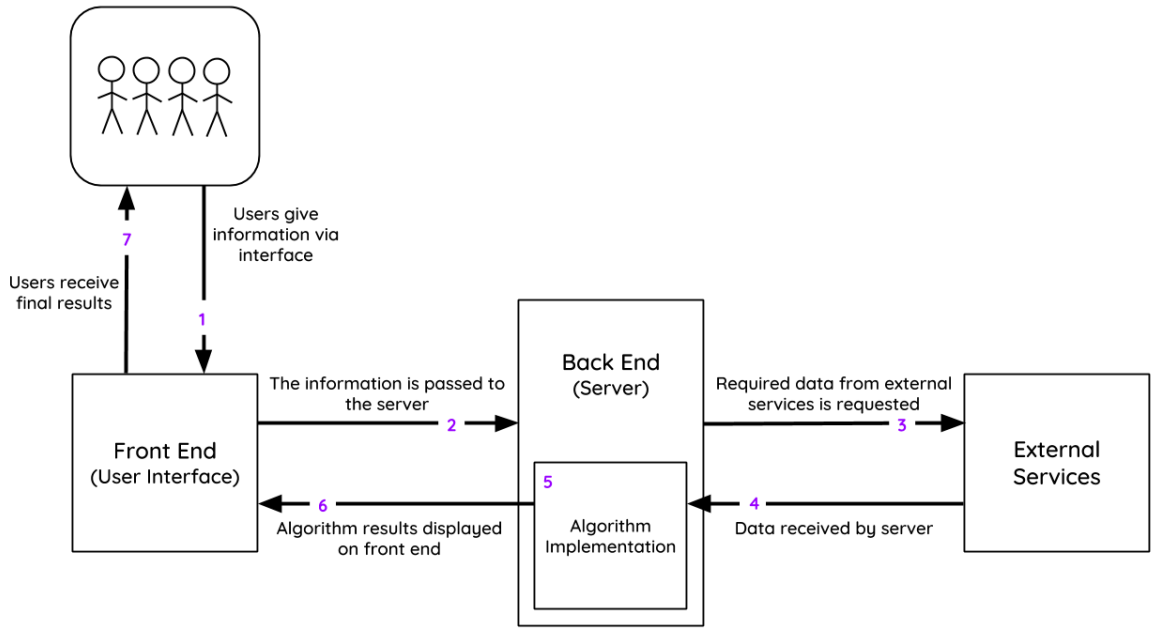


Figure 6.2: High level system overview.

Using this diagram, the key aspects of design and implementation required can be seen clearly. A User Interface will be required to receive information from the users (stage 1) and present results back to them (stage 7). This data will need to be passed to a server (stage 2) and returned to the UI (stage 6). Before the algorithms can be implemented at stage 5, further information is required, such as places data and journey distances times. This information will come from external APIs after being requested at stage 3 and returned at stage 4.

6.3 Development Technologies

Now that the overview has been designed, what follows is a discussion of various technologies which will allow the successful implementation of the system.

6.3.1 Google Maps

In order for the algorithms to be implemented in this project, a large amount of data will be required: finding places and their details; finding journey times; and finding directions.

To find places, there are a number of APIs available such as FourSquare, Yelp, and TripAdvisor [32]. Yelp has developed a reputation for being damaging for small businesses [33], at times being called a "bully" [34]. Moreover, Yelp and FourSquare's search capabilities only return up to 50 places, and Yelp only returns locations which have customer reviews. Google Places on the other hand can return up to 200 places at once and has a generous free usage limit, so it has been chosen for this project.

Displaying results on a map on the User Interface can be done through a range of options including OpenStreetMap and MapBox [35]. However, the Google Places API policies quite clearly state that if an "application displays Google Places API Web Service data on a map, that map must be provided by Google." [36] hence Google Maps has also been chosen for displaying information.

As well as this, public transport data can also be accessed through a number of available APIs like TransportAPI, CityMapper, and the Transport for London Unified API. [37] Whilst these are reasonable choices, Google Maps has been chosen for public transport data too as it has fantastic data available for public transport journeys [38] and is well known for having accurate real-time traffic data [39]. Additionally, their services take into account toll roads and highways, making it the obvious choice for directions, journey times, and journey details.

Google Maps is a hugely popular service and has a global rank of 467 [40] compared to OpenStreetMaps' global rank of 3,047 [41] and as a result, most users will have a strong sense of familiarity when navigating a Google Map. In order to take advantage of this and give the users a pleasant experience, the Google Maps JavaScript API will be used for the seamless integration of features required by users on the front end.

Specifically for this project, some required features will be custom map markers (which show specific locations on a a map), info windows (which appear above a marker and display information about it), and polylines (which visualise a journey). An example of how simple it is to implement some of these features can be seen below in Figure 6.3:

```
// A point on the map given by its latitude and longitude
var uluru = new google.maps.LatLng(-25.363882,131.044922)
// A map is created centred around this point
var map = new google.maps.Map(document.getElementById('map'), {
    center: uluru
})
// A marker is created on the map positioned at the point
var marker = new google.maps.Marker({
    position: uluru,
    map: map,
})
```

Figure 6.3: Google Javascript API example code [42].

6.3.2 Python and Flask

Since the Google Maps APIs will be relied on heavily throughout the process, a Client Library for Google Maps Web Services will be required. The options available include Java, Python, Go and Node.js [43].

Not only has Python been ranked the top programming language of 2017 [44], it allows for fast and efficient development with excellent frameworks for many tasks such as unit testing [45]. Furthermore, given the time limit imposed by the project, Python has been chosen as the language to limit the number of new technologies which will need to be learned by the developer, who already familiar with Python.

To build a Python web application, we also need a web framework. The two most popular options are Flask and Django. Flask is becoming increasingly popular and is known for its simplicity and flexibility [46]. Django, on the other hand, has a higher learning curve so as a newcomer to web application development [47], Flask has been selected as the web framework for this project.

6.3.3 Semantic UI

The secondary objectives of this project are to create a web application which abides by Nielson's usability principles; i.e. a minimalist and aesthetic web application which has a sense of familiarity to the user. To give the web application a better chance of achieving these goals, a CSS Framework will be used to enhance its appearance. A CSS Framework is a common structure with standardised features that many websites have such as buttons and forms, and in many cases automatically apply adjustments for website elements on smaller screen resolutions, such as mobiles [48].

Bootstrap is the most popular CSS Framework [49] and provides many features which are easy to pick up, however its popularity means that, without customisation, websites using the Bootstrap Framework start to look very similar. Customisation in Bootstrap is difficult and its reliance on HTML classes can make development messy [50].

On the other hand, Semantic UI [51] has a huge range of elements available which can be customised very easily. Ultimately, both frameworks offer good documentation and have great features, but in this case, Semantic UI has been chosen along with some additional modules such as a calendar [52].

6.3.4 JavaScript, jQuery and AJAX

For front end development, JavaScript and some additional libraries will be used. Moment.js [53], for example, will allow the journey durations returned by Google Maps in seconds to be displayed in a user-friendly way (hours and minutes). The Semantic UI components which will be used can be customised using JavaScript too.

This project aims to produce a dynamic web application. Here, this means one page that the user sees which updates quickly and smoothly without needing to be refreshed. In order to achieve this, server-side processing is required [54]. AJAX (Asynchronous JavaScript and XML) allows data to be exchanged in the background between the front end, where the user sends or receives it, and the back-end, where it is required for processing [55]. It will be necessary to change the HTML Document Object Model (DOM) frequently to show the different information requested by the user. Although this can be done using JavaScript, jQuery [56] will be used. jQuery is a popular JavaScript library which makes the manipulation of HTML documents quicker and easier.

6.4 User Interface Initial Design

These mockups of initial designs show the plan made for the layout of the web application based on the User Stories. Although these do not include every feature required by the users just yet, the layout is adaptable enough to have new elements added easily. These aim to allow the developer to start building the front end which can then be analysed and more features and information can be added as appropriate.

6.4.1 Welcome and Search Screen Mockup

In Figure 6.4a, the user sees a map as the background (depicted by blue) and a search panel in the centre. The search panel contains a form which allows user input. The users can enter addresses and modes of transport for four people, along with the option for the group to choose where and when they would like to go. At this stage of design, it was still unclear how many algorithms there would be or how to best provide the option to avoid tolls and highways, so these two elements are not present in the initial mockups.

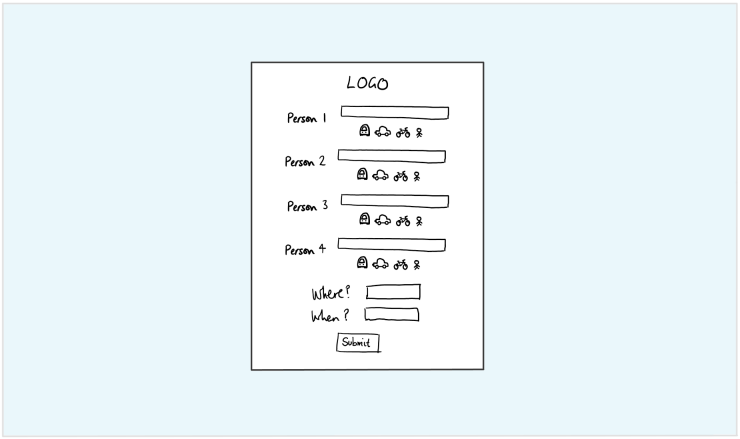
6.4.2 Results Screen Mockup

Figure 6.4b shows the original idea for the results screen. This was inspired by a combination of the existing systems and Google Maps. The main goal is to make optimum use of the screen space to give the user a pleasant experience. On the right hand side of the screen, users can see a large map displaying all the people in the group and the search results. On the left hand side, there is a panel showing results in a clearly labelled list. Once it is clear what data is available from Google Maps, the exact details to be displayed will be chosen, but the overall aim is to keep things minimal and helpful.

6.4.3 Selected Location Screen Mockup

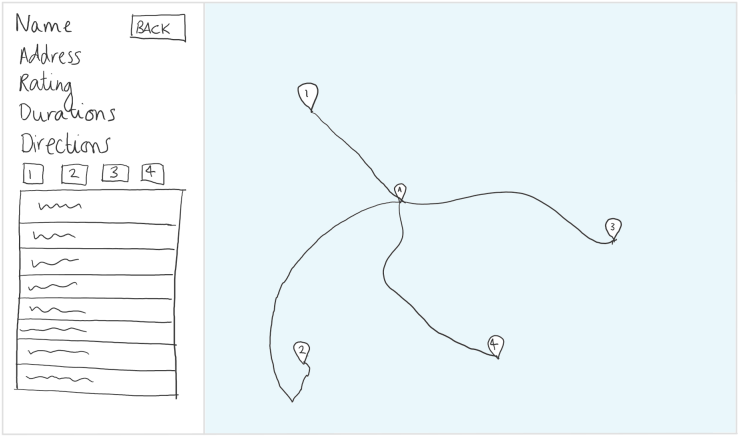
Once a location has been selected, shown in Figure 6.4c, the map should zoom in to show that location in relation to the four users, as well as their journey. The panel should remove the list of results and show further information about the location such as the duration of each group member's journey and their directions. These will be chosen by clicking a button

corresponding to the user which will then open their specific journey instructions.



(a) Welcome screen.

(b) Results screen.



(c) Selected location screen.

Figure 6.4: User Interface mockups.

6.5 Final User Interface

The initial mockups were followed closely when creating the final UI. This will be discussed separately for desktop and mobile layouts, and within each of these sections, the Welcome Screen, Results Screen and Selected Location Screen will be individually explained.

6.5.1 Desktop

Welcome Screen

When the user opens the web application, they see a big map covering the whole screen, centred around Greater London as shown in Figure 6.5. As the screen resizes, the map continues to stay centred around London to emphasize that the web application is specifically for use in London.

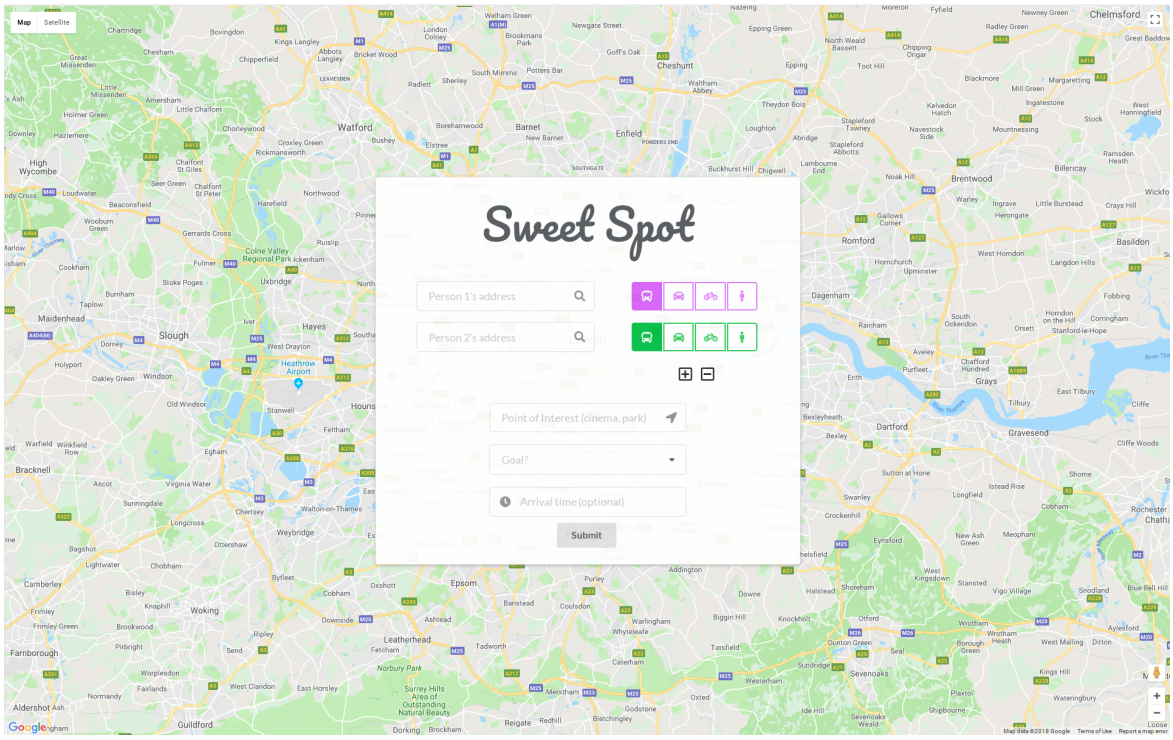


Figure 6.5: Desktop welcome screen.

In the centre of the screen, they also see a box containing a big clear logo saying "Sweet Spot" along with their available search options. Initially, there is space for two people to input their information as seen in Figure 6.7, however this can be decreased to a minimum

of one like in Figure 6.6 or increased to a maximum of four like in Figure 6.9.

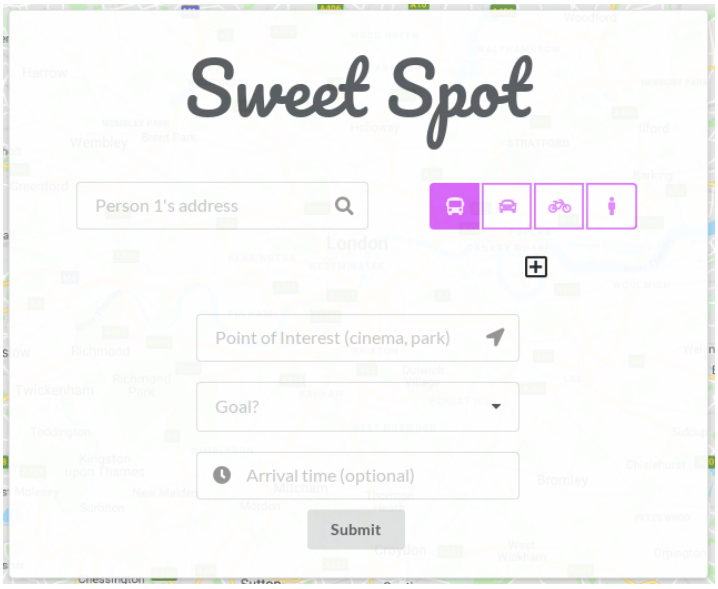


Figure 6.6: User input section for one person.

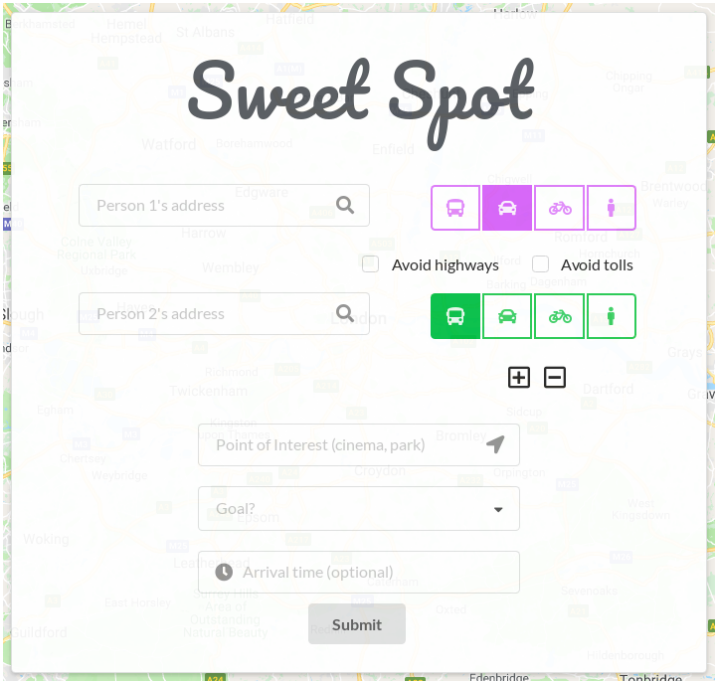


Figure 6.7: User input section for two people.

Group members can be added or removed using the '+' and '-' buttons available. This works by storing an HTML template for user input containing a user id, the address input and the transport buttons. This is then fetched using jQuery and cloned every time a user

is added. When hovering over a '+' or '-' button, the button changes colour to indicate to the user that it can be clicked as shown in Figure 6.8. Once clicked, the user receives immediate feedback as the next row of information appears for another person's information to be entered.

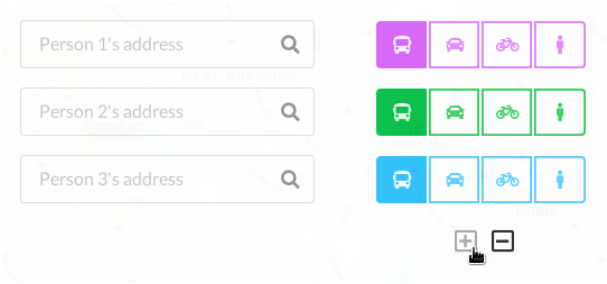


Figure 6.8: Hovering over '+' button.

As this happens, users are implicitly assigned a colour (purple, green, blue, orange) and throughout the rest of their use of the web-application, information directly related to them such as the journey or duration of travel is displayed in this colour for clarity. The four colours are seen in Figure 6.9.

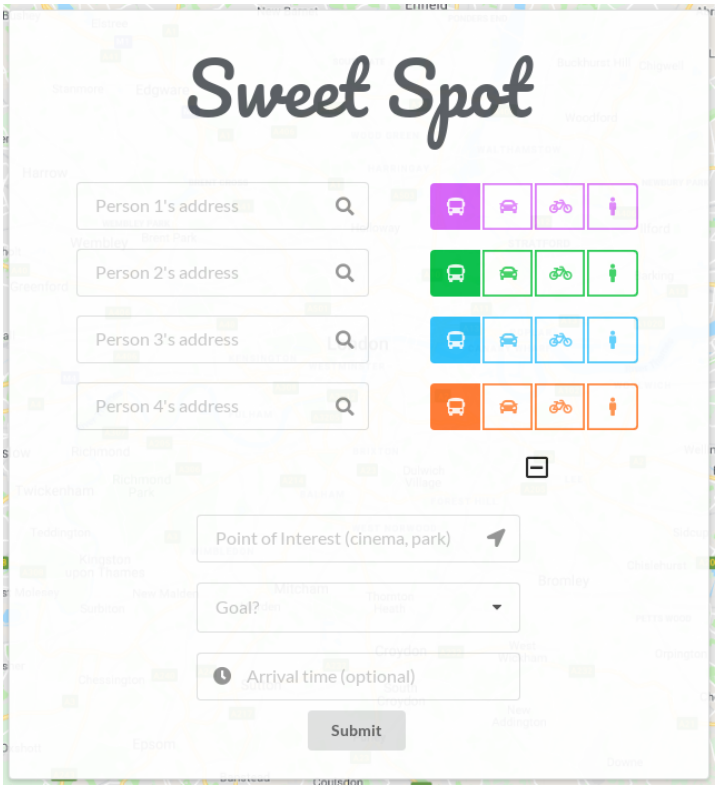


Figure 6.9: User input section for four people.

Beside the input field for the addresses of the people who will be meeting up, there is a small magnifying glass icon (seen in Figure 6.11) to emphasise that a search will be taking place. This is because the User Interface provides Google Autocomplete suggestions for addresses as the user types an address. An example of the results displayed is shown in Figure 6.10. This is beneficial for a number of reasons: the application is easier and quicker for the user and reduces the chance of an error since all places in Google Autocomplete are recognised by Google Maps.

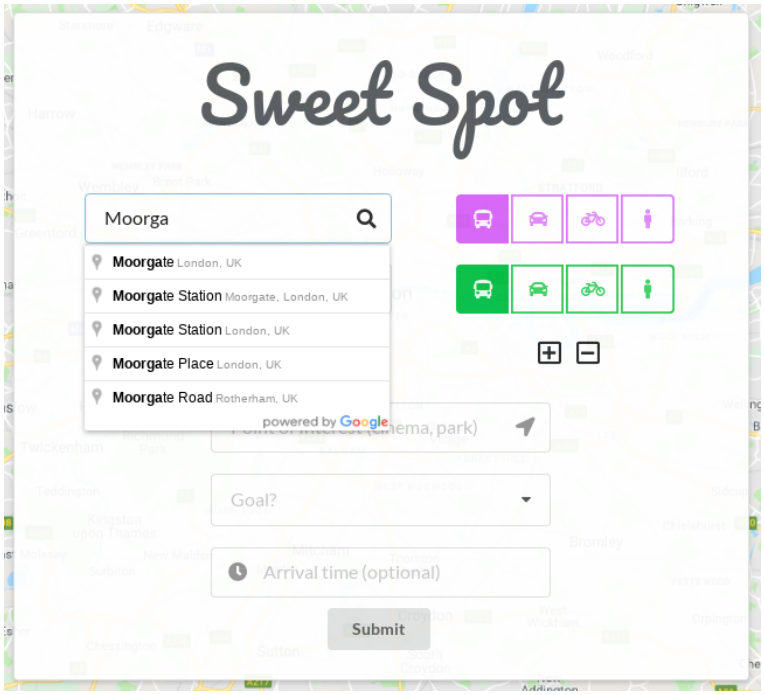


Figure 6.10: Example of Google Autocomplete options appearing as user types.

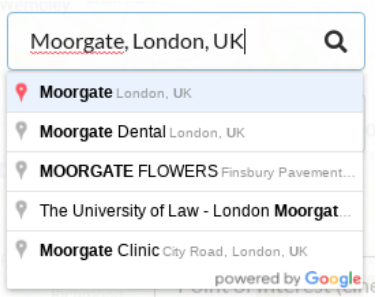


Figure 6.11: User selection of Autocomplete option.

On the right of each of the people's addresses is a set of four familiar icons representing public transport, driving, cycling and walking. These are there for the users to select how they will be travelling as this is a key part of the web-application. The feedback for the

user is good as the buttons shade in when they are hovered over, and change colours once clicked. In Figure 6.12, the dark green public transport button has been selected; the driving and walking options are white and unselected; and the cycling option is light green as it is being hovered over. Figure 6.7 shows us the additional options presented to a user if they are driving. The option allows them to specify whether they would like to avoid any tolls or highways.

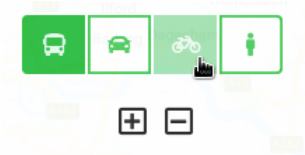


Figure 6.12: Set of transport options, public transport selected, bicycle being hovered over.

The user also types in a point of interest just like they would in Google Maps, which is typically a keyword indicating the type of location they would like to meet at. This indicated by the arrowhead icon symbolising a place and target and can be seen in Figure 6.13.

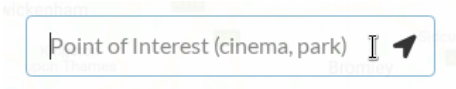


Figure 6.13: Point of interest input field.

In Figure 6.14, a dropdown menu can be seen displaying the various algorithms available to the user.

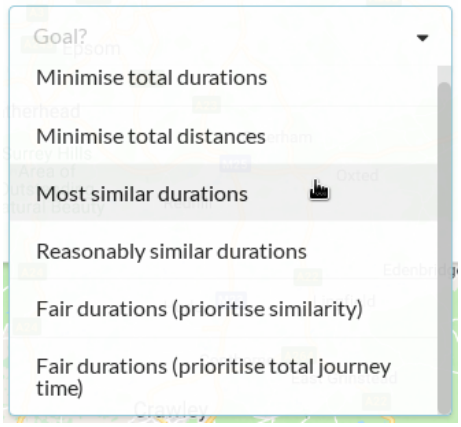
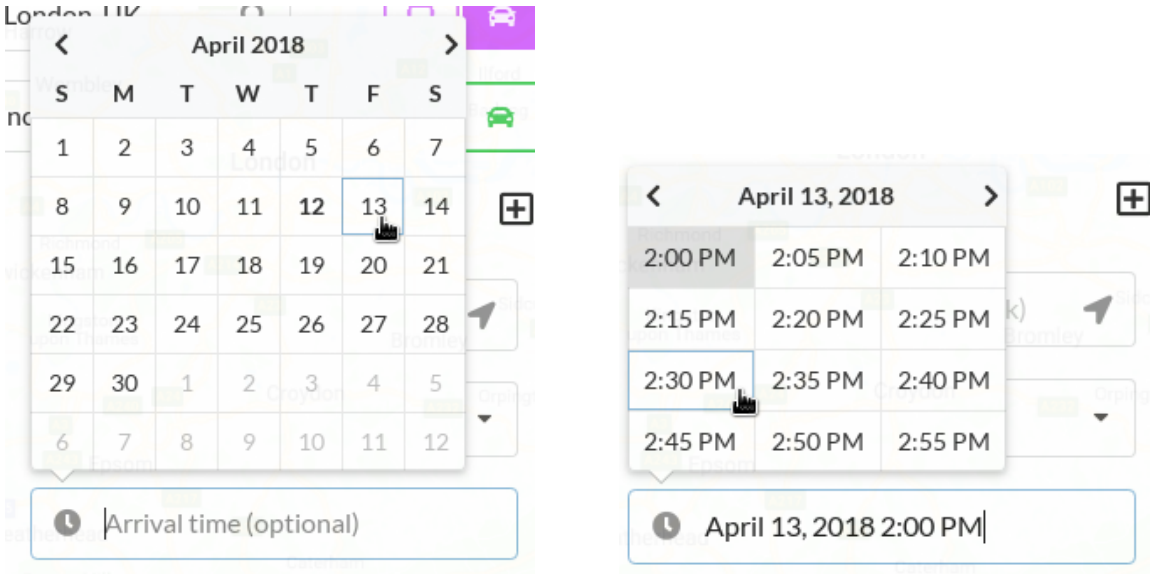


Figure 6.14: Dropdown menu of available algorithms.

Lastly there is a calendar and list of times available for the user to select the day and time that the group will be meeting at. These can both be seen in Figure 6.15



(a) Calendar.

(b) Time.

Figure 6.15: Arrival time and date selection.

Once selected, the time is written in a user friendly format for the user to confirm, an example of which can be seen in Figure 6.16.

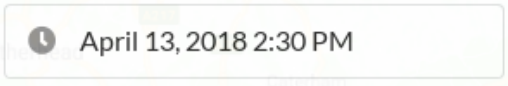


Figure 6.16: Final arrival time and date.

Provided there are no errors in user input, when the submit button is pressed, the button is disabled to prevent being clicked again, and a loading indicator appears to inform the user that their results will be available soon. In Figure 6.17, the submit button can be seen alone, whilst being hovered over, and once clicked.

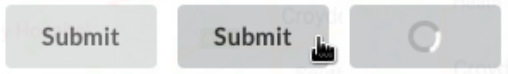


Figure 6.17: Unclicked submit button, submit button being hovered over, loading icon.

Form validation occurs when the submit button is pressed to ensure that no details have been left out. If there are errors in user input, any missing fields are highlighted in red and an error message appears explaining the problem as seen in Figure 6.18. This is not required for modes of transport as they are set to public transport by default.

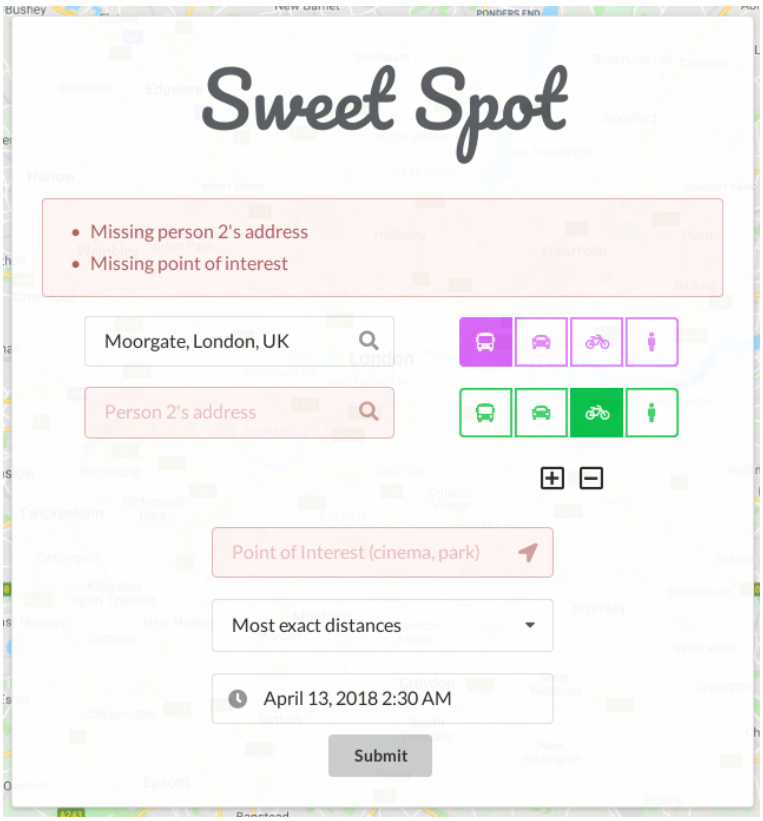


Figure 6.18: Form validation.

User validation takes places separately once the results have been submitted. If there is a problem with a user, i.e. their address is invalid or not recognised by Google Maps, a further error message is displayed as shown in Figure 6.19.

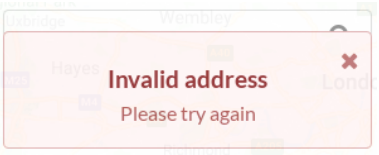


Figure 6.19: Form validation.

Results Screen

This section will look at the results of the example search in Figure 6.20. Four people coming from Croydon, Edgeware, Ilford, and Richmond are all travelling by public transport. They want to find a church to attend at 9am on Sunday April 14th 2018, and wish to see results which minimise the overall duration of their journeys. Note, although these are not very accurate addresses, they are used here to show the layout of the web application and

use Google Maps results to find latitude and longitude.

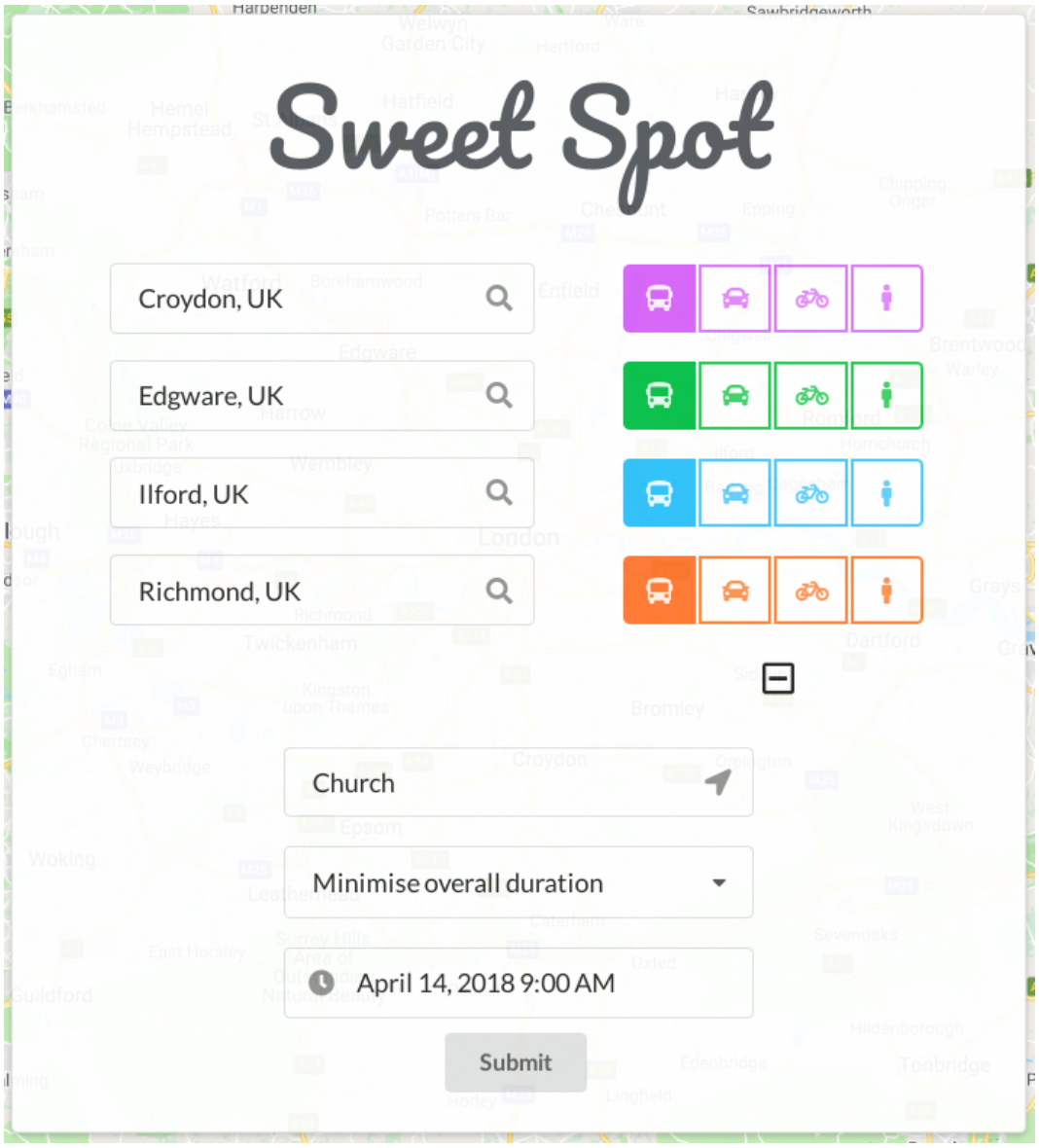


Figure 6.20: Example Search.

Once the search has been completed, a sidebar appears on the left of the screen displaying the ten results shown in Figure 6.22 and the map takes up the remainder of the screen on the right hand side, as seen in Figure 6.21. At the bottom of every set of results, the "powered by Google" logo is shown.

Initially, when looking at the map, the user can see numbered purple, green, blue, and orange markers showing the starting addresses that were entered for each user. They can also

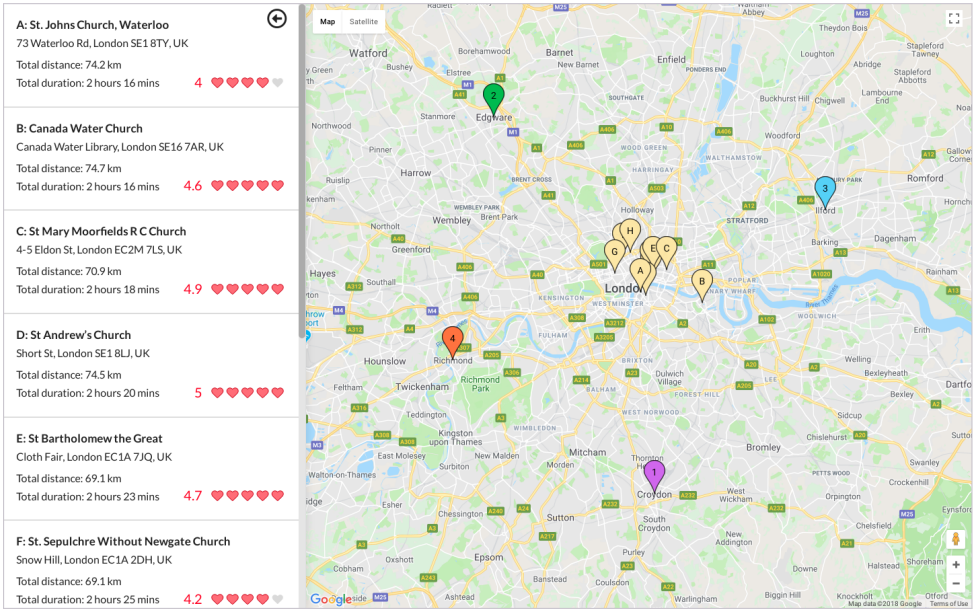


Figure 6.21: Results Screen.

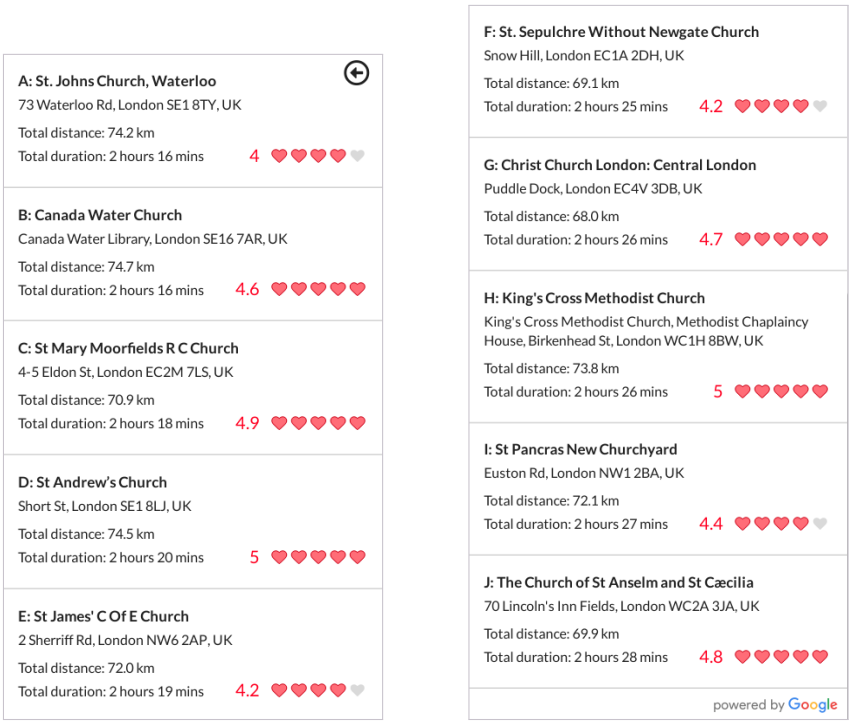


Figure 6.22: The ten results.

see up to ten yellow markers labelled A to J, each corresponding to a search result. These can be seen more clearly in Figure 6.23. Markers representing people are considered the most

important, followed by marker A, then B, then C, and so on. They are z-indexed so that when there is overlap, the most important ones appear above the others to give the user the best possible experience.

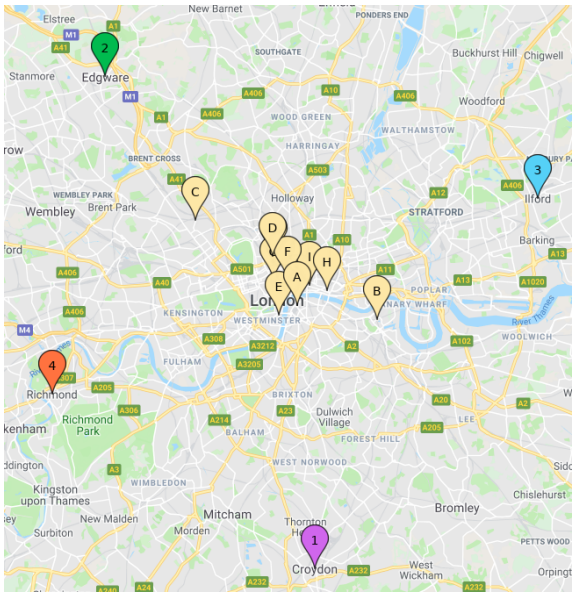


Figure 6.23: All map markers.

If a user wants to see more information about a place, they can hover over markers on the map to make an InfoWindow such as the one in Figure 6.24 to see more information. They can also hover over the markers representing the four starting locations to be reminded of the address of that person. Furthermore, these markers can be clicked to select the location and open the selected result screen, in the same way it would if the location was selected from the sidebar.

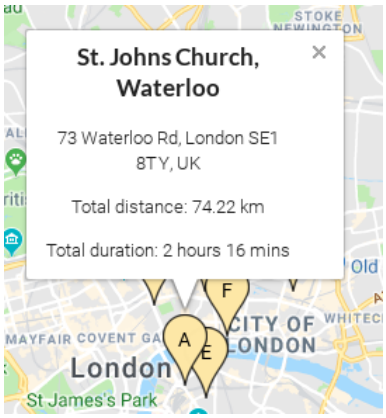


Figure 6.24: InfoWindow.

The user can zoom in easily by scrolling with their middle mouse button, using the buttons available on the map, or using pinch-to-zoom if they have a touchscreen device. This gives them a clearer view of the area like in 6.25 and they can move the map around as they can in Google Maps.

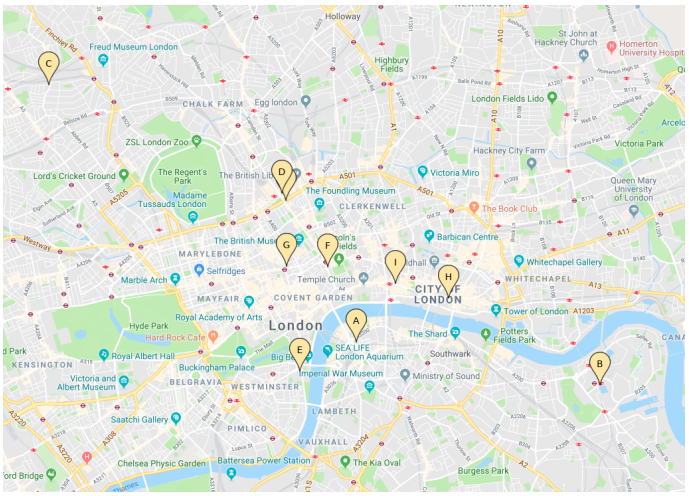


Figure 6.25: Place markers.

Looking more closely at the results sidebar, each suggested location, for example Figure 6.26, contains helpful data for the user. This includes its name, its address (73 Waterloo Rd), and its Google rating (if available) shown as a number (out of five) and indicated by red hearts. Furthermore, the total distance and durations of the journeys for all people travelling can be seen clearly as extra information which may be useful or interesting to the user. This is also achieved through the use of an HTML template which stores a placeholder. The placeholder is cloned and the information is updated using the actual data returned from the Google Places API.

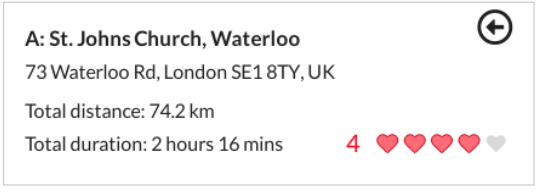


Figure 6.26: First result from sidebar.

Each place in the sidebar can be selected by hovering and clicking as shown in Figure 6.27. To keep the user interface intuitive and help the user know what to do next, the background of the location is shaded grey.

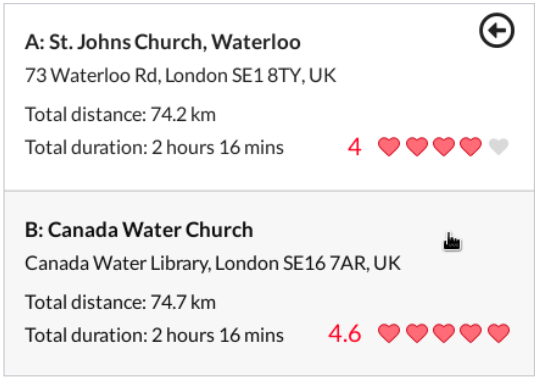


Figure 6.27: Hovering over a result.

The back button, represented by the arrow shown in Figure 6.28, is located in the top right corner of the sidebar. The left-pointing arrow makes the functionality clear to the user, and they know that it is a button as it changes colour to a light grey when hovered over. Clicking it takes the user back to the welcome screen with their information still present in the input fields in case they need to make any adjustments, for example changing the mode of transport of a person or the point of interest.

Figure 6.28: Hovering over back button.

On top of allowing the user to see and interact with the map, the use of the sidebar is beneficial for a number of reasons. Sidebars are a familiar element which are likely to have been seen and used by most internet users previously, for example when looking at search results on Google Maps. As a result, it is intuitive for users to navigate without previous exposure to the web application. It also allows information to be structured clearly and consistently which is important for an application which portrays a lot of information in a small space.

Selected Location Screen

The structure of the selected location screen is similar to that of the results screen to ensure a seamless transition with a familiar layout. However, jQuery is used to change the content of the page to provide the user with further useful information that might help inform them about the best destination for their group.

Once the user has selected a location, the sidebar clears the other results of the search and focuses on the chosen destination. Before this happens, the sidebar results are stored so that they can be recompiled when a user presses the back button. The map markers showing all the search results are also cleared, leaving only the marker of the relevant place and the markers showing the starting addresses of the people in the group. Figure 6.29 below shows location 'A' from the list in Figures 6.22.

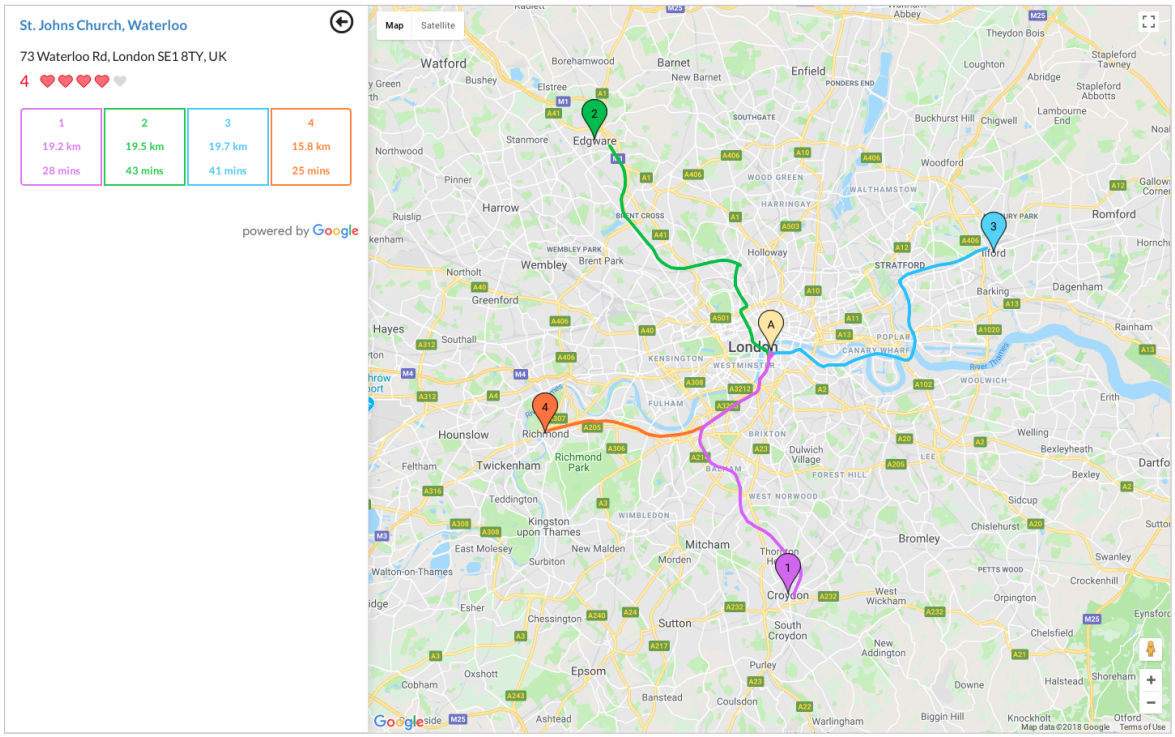


Figure 6.29: Location 'A' selected.

When looking closer at the sidebar on the left, new information can be seen about the location. Although the name of the location, the address of the location, and its Google rating are all available, Figure 6.30 shows that the name of the destination has now been linked to its website to allow the user to navigate to it and find out more information such as prices, facilities, and opening times.

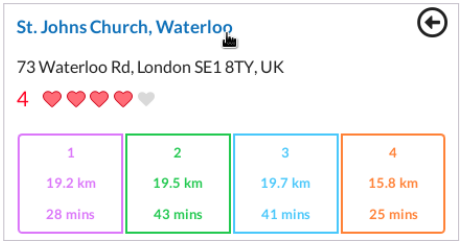


Figure 6.30: Sidebar closeup of place details.

Figure 6.30 also shows four buttons, one for each person added on the welcome screen search. These are labelled '1' to '4' and coloured purple, green, blue, and orange, matching the numbers and colours assigned to each person on the welcome screen. Examples with fewer people can be seen in Figure 6.31.

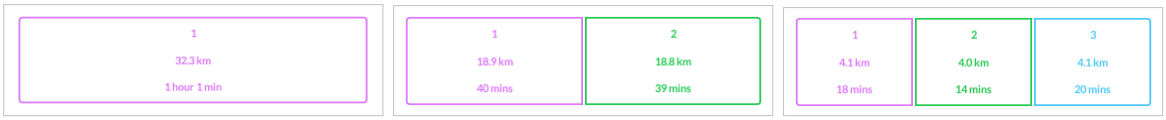


Figure 6.31: Example buttons for different numbers of people.

This consistency is key in creating an atmosphere of familiarity and clarity so that the user has a pleasant experience. These buttons show the user the distance (in km) and duration (in hours and minutes) of the journey recommended for them by Google Maps. This allows for quick and easy comparison of travel distances and times which is a key aspect of this project as it gives the group of people the information they require to decide for themselves which location is most fair.

The aforementioned routes recommended by Google Maps are illustrated in Figure 6.29, and more clearly in Figure 6.32 by the four lines connecting each person's map marker to the map marker of the destination. These lines, known as 'polylines', are also the same colour as the marker for each individual, enabling the journeys of all four people to be visualised clearly and interacted with simultaneously.

When a person's button from Figure 6.30 is selected, the map markers for the other three people are cleared, and the map zooms into their route, whilst also displaying a brief overview of the directions from Google Maps in the sidebar, as shown in Figure 6.33. The journeys of the other three people can be seen in Figure 6.34.

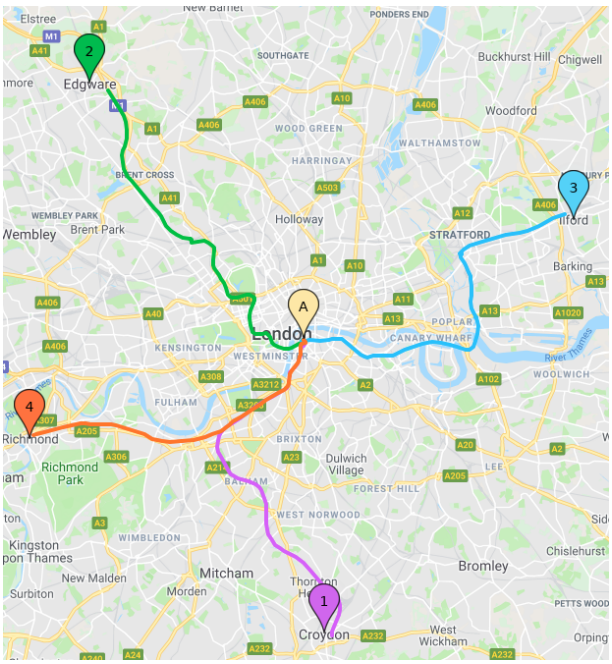


Figure 6.32: Closeup of polylines.

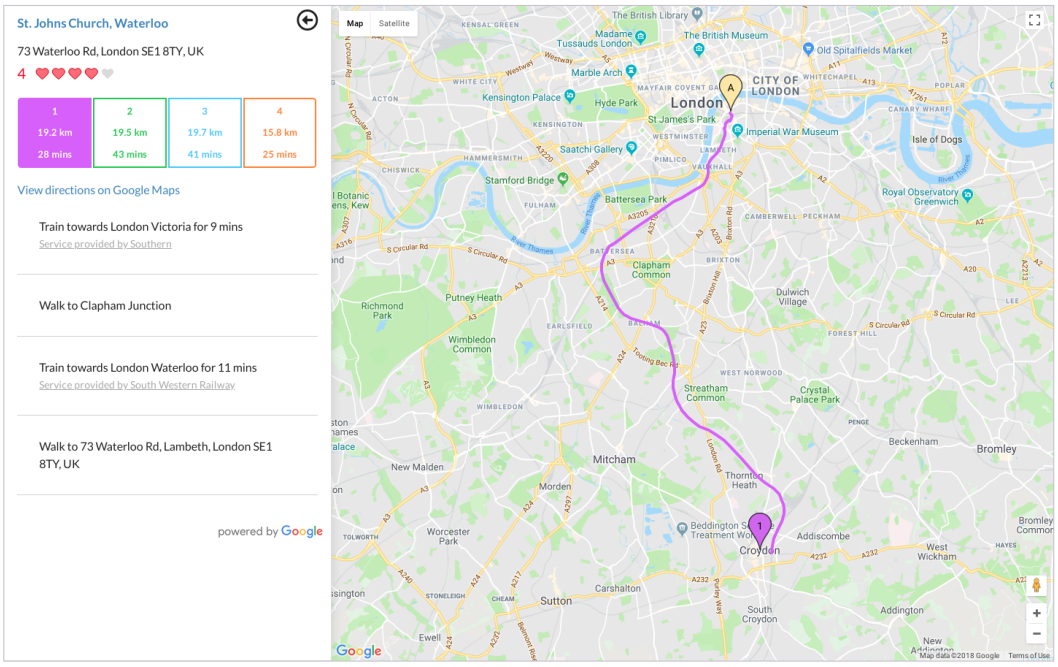


Figure 6.33: Person 1's journey.

For public transport journeys, the name and URL of the service provider is made available to the user as required by the Google Directions terms and conditions. We can see this in Figure 6.33 in grey underneath each train.

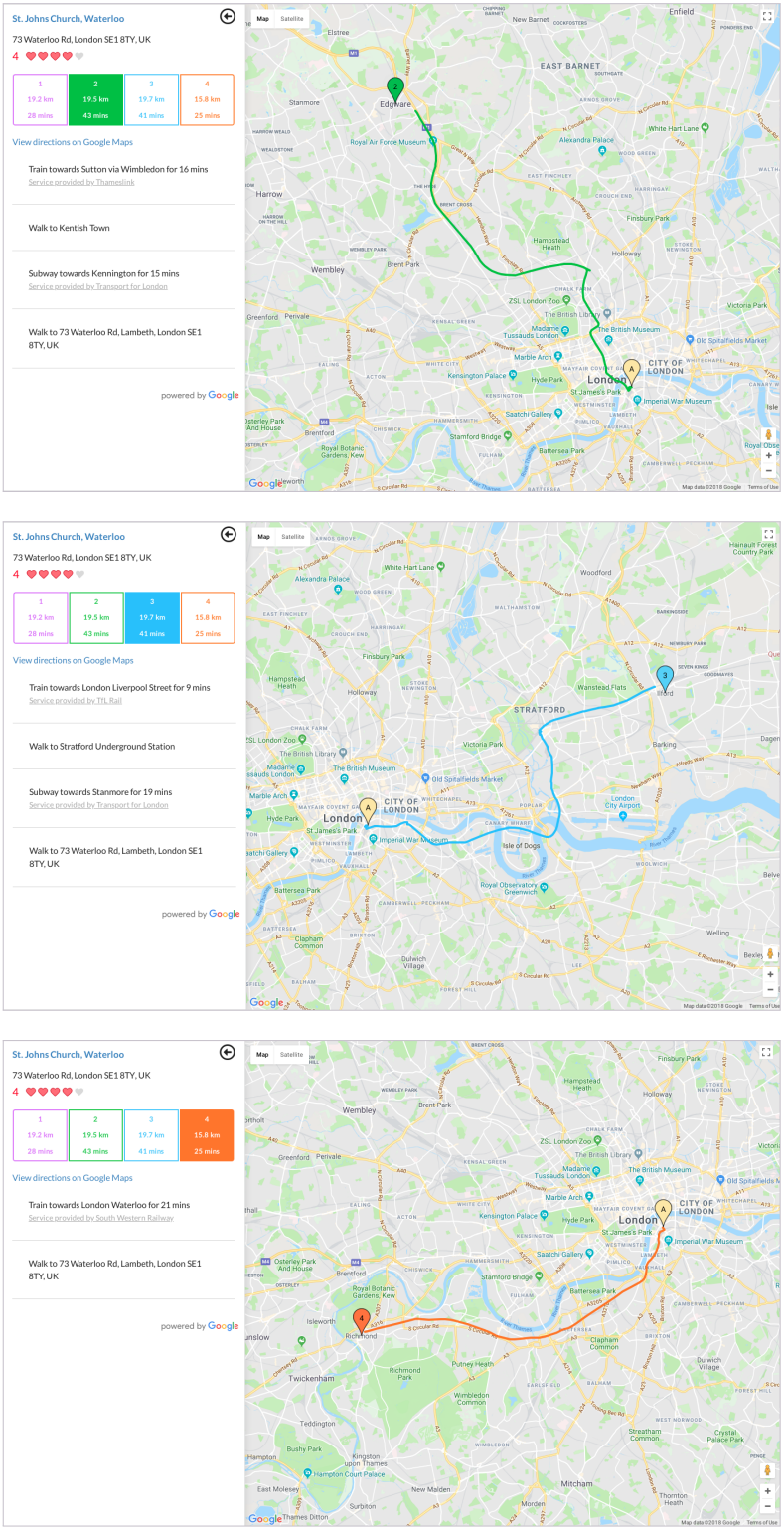


Figure 6.34: The other three journeys.

The directions are intentionally brief to keep the User Interface uncluttered whilst giving an overview of the legs of each journey. However, as illustrated by Figure 6.35, the option is available to view the directions on Google Maps. This works by redirecting the user to the Google Maps website or application, as shown in Figure 6.36, using a custom URL which gives them the opportunity to explore other available routes options with minimal effort.

Figure 6.35: Hovering over 'View directions on Google Maps'.

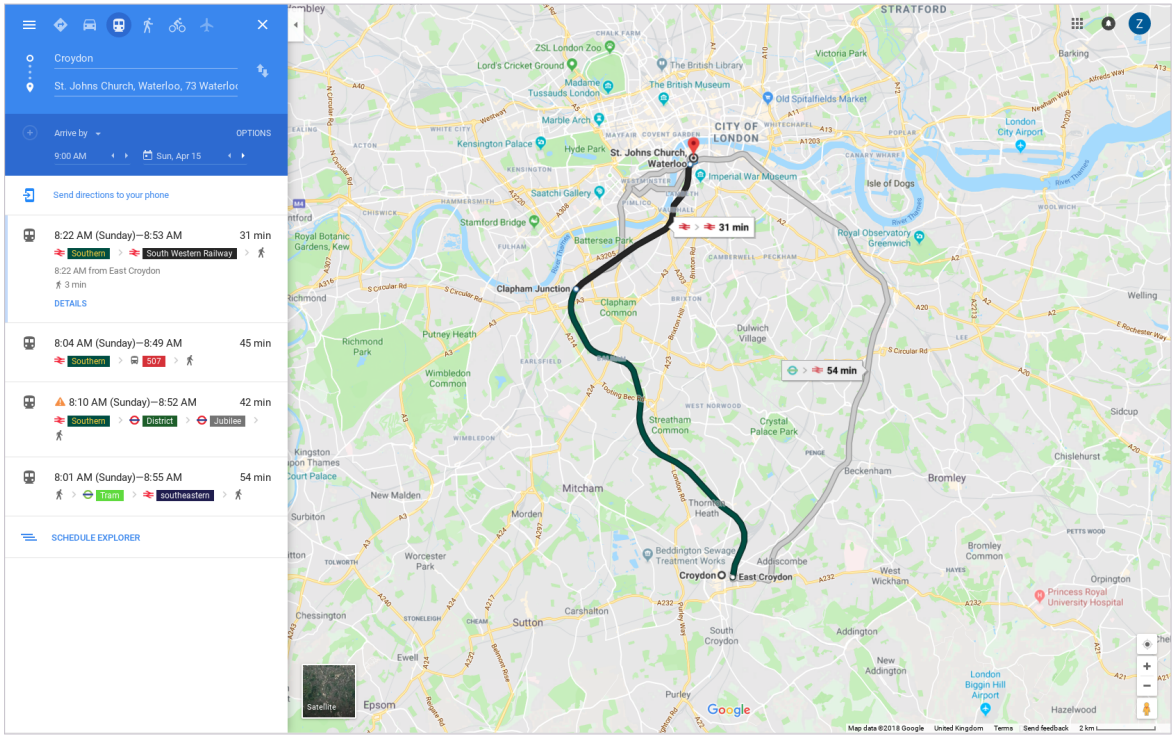


Figure 6.36: Directions on Google Maps.

To view the original list of results, the user simply clicks the back button shown in Figure 6.37 which is located in exactly the same position as on the results screen. This refills the sidebar with the recommended meeting points ready for the user to continue looking through.

Figure 6.37: Hovering over back button.

6.5.2 Mobile

The development of a web application that responds well to smaller screen sizes can be done in two ways. One is to have a completely separate site for these smaller screens, and the other is to use additional CSS Media Queries to explicitly tell the UI how to present itself on these smaller screens. The latter method was chosen as it has the additional benefit of transitioning smoothly between different screen resolutions without any loss of user data when the user manually resizes their window. The main shortcoming of mobile views on other applications was the poor use of screen space and the lack of seamless integration of UI features, so these were kept in mind when creating the mobile views for this application.

Welcome Screen

On the welcome screen in Figure 6.39, we see the same options but the fields are stacked and can be scrolled through when the content does not fit on the page. The view on an iPad is almost identical to on a desktop and can be seen in Figure 6.38.

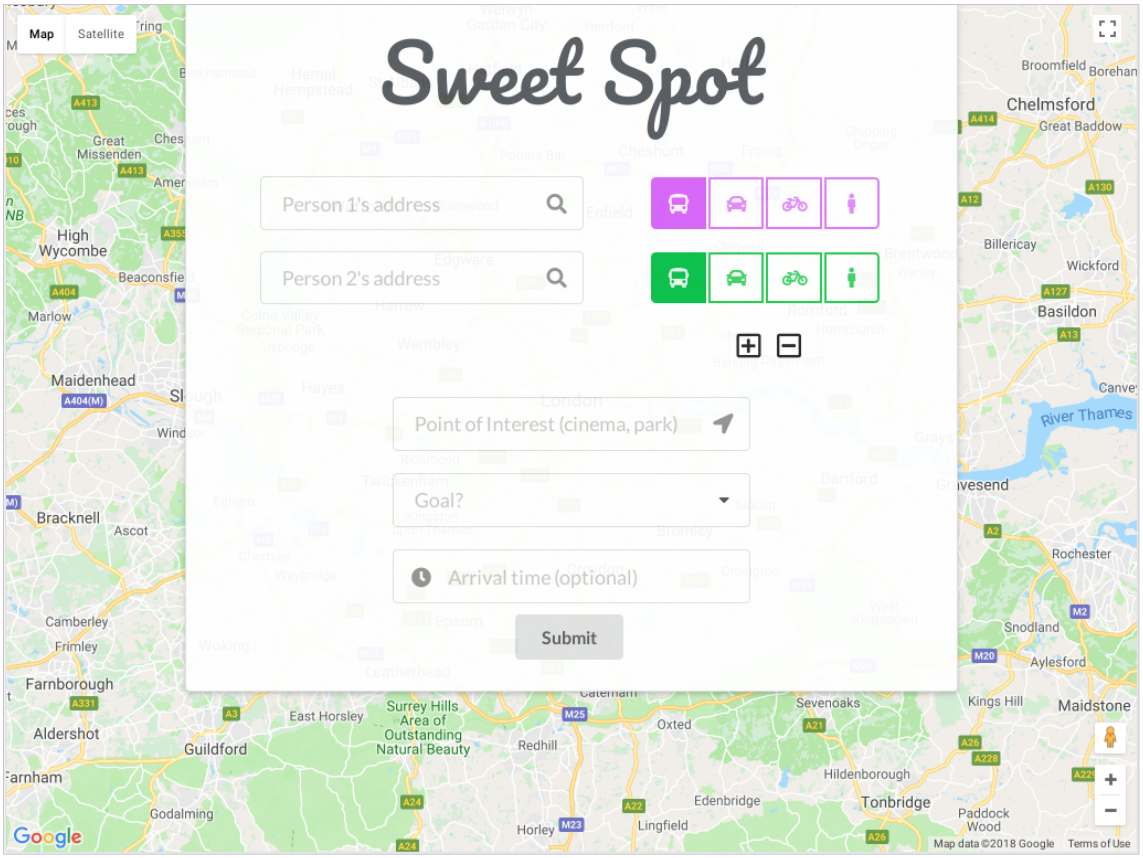


Figure 6.38: iPad welcome screen.

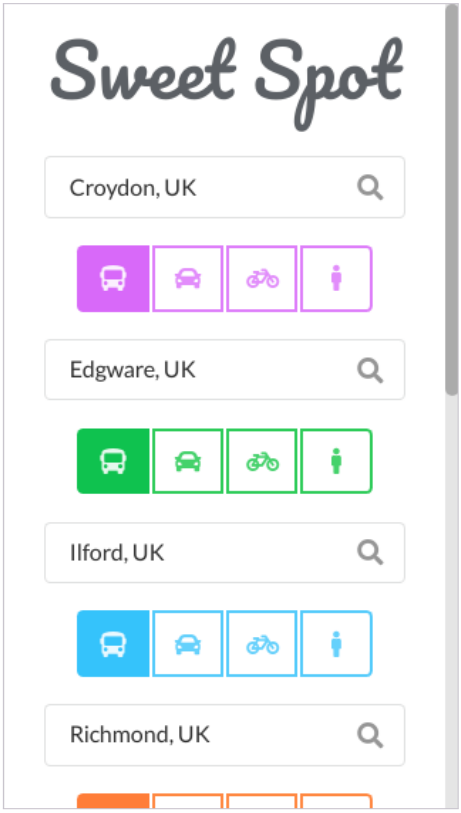


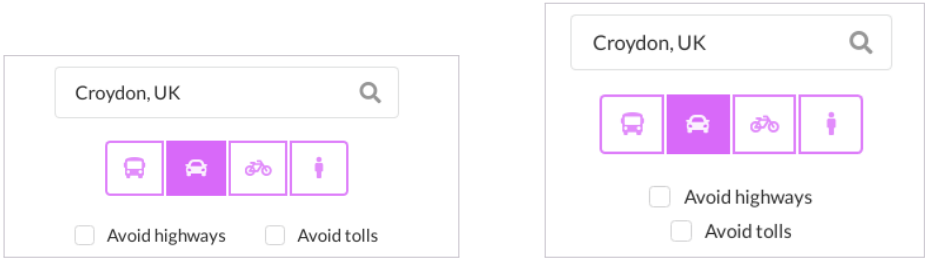
Figure 6.39: Mobile layout.

Once the screen becomes too small, the background map disappears to keep the screen as clear as possible. This is achieved using a CSS Media Query, an example of which can be seen in Figure 6.40.

```
1 /* When the screen is less than 414px wide, hide the map */
2 @media only screen and (max-width: 414px) {
3   #map {
4     display: none;
5   }
6 }
```

Figure 6.40: CSS Media Query example.

Even among mobile phones, there are a wide range of resolutions, so part of having a good responsive display is catering for different device widths. One example of this is the avoid highways and tolls checkboxes which appear slightly differently depending on the screen size, as can be seen in Figure 6.41.



(a) Wide screen.

(b) Narrow screen.

Figure 6.41: Mobile views for travelling by car.

Form and user validation, shown in Figure 6.42 works in exactly the same way on mobile. Error messages were intentionally designed to be shown above the input fields so that problems can be seen immediately without needing to scroll up and down the page.

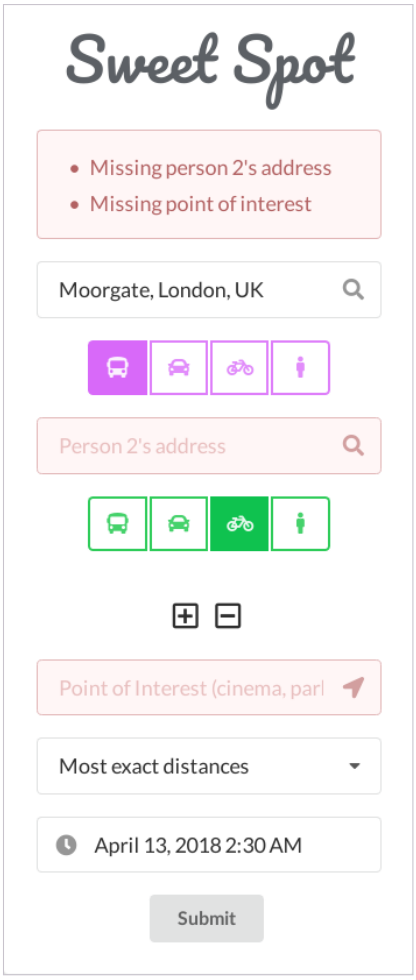


Figure 6.42: Form validation on a mobile.

Results Screen

A map showing the markers of every location on a mobile is a poor use of already limited screen space. So for narrow screens such as in Figure 6.43, the sidebar fills the width of the device and the rating appears on a new line, allowing the user to scroll through the results easily with no distractions. Since there is no Google Map showing these results, the "powered by Google" logo is especially important because of Google's guidelines.

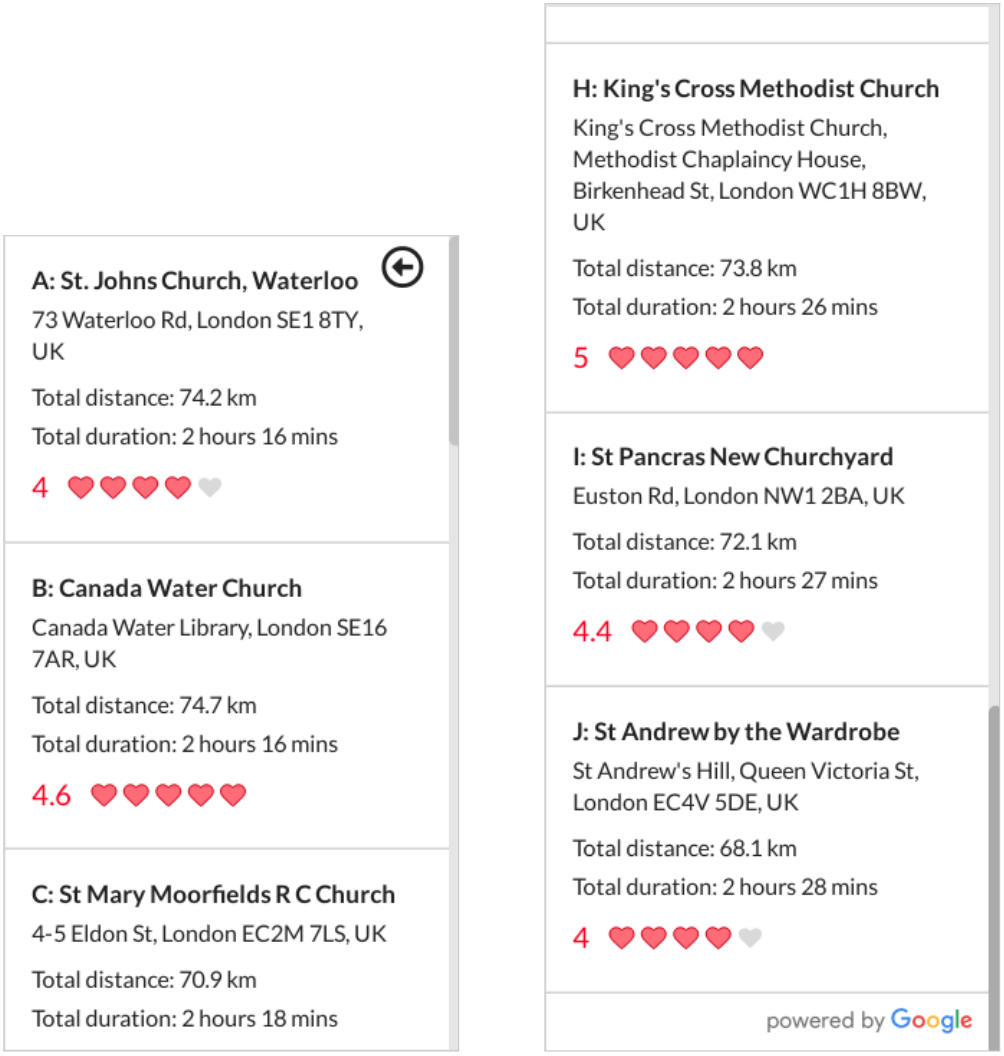


Figure 6.43: Mobile views of results.

Everything else works in the same way; the back button can be clicked to get back to the search screen, and selecting a location presents further information about it.

Selected Location Screen

The selected location screen has more space available than the results screen, but still not enough to have a big map like the desktop version of the site. As expected, once a destination has been chosen, the sidebar clears all other results and the total time and total distance. As illustrated by Figure 6.44, a map is still available for the user, however it is a small one within the sidebar to save space. There is a gap around the outside so that the user does not get trapped in the map when using it on a touchscreen device.

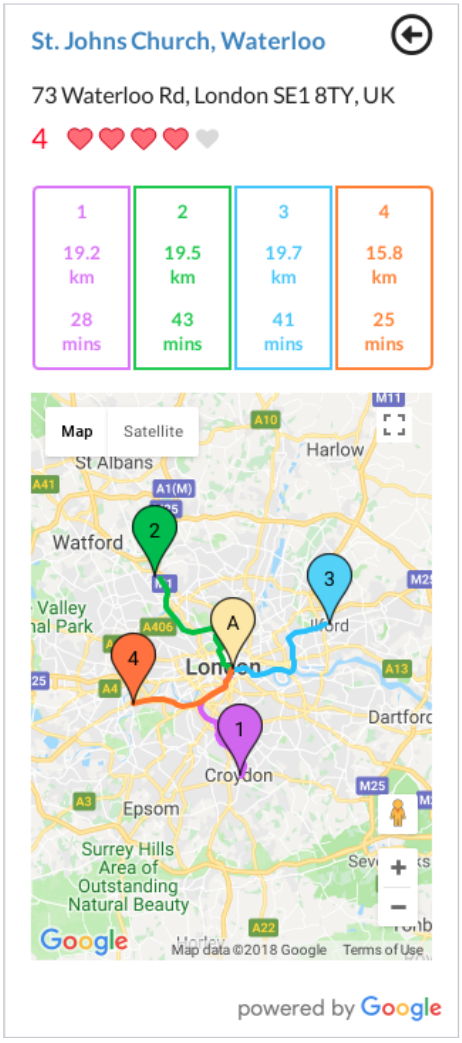


Figure 6.44: A selected location on mobile.

This small map is still interactive and shows the journeys of all the users simultaneously. The markers can be clicked to see InfoWindows such as the one in Figure 6.45.

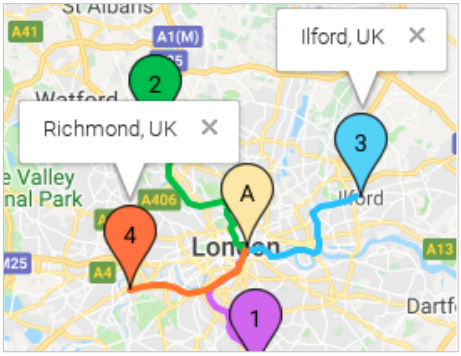


Figure 6.45: InfoWindows on the mobile map.

However, this time, when a directions button is clicked, directions are not printed beneath it. Instead, the custom Google Maps URL redirects the user to the Google Maps website or the Google Maps application where available as in Figure 6.46.

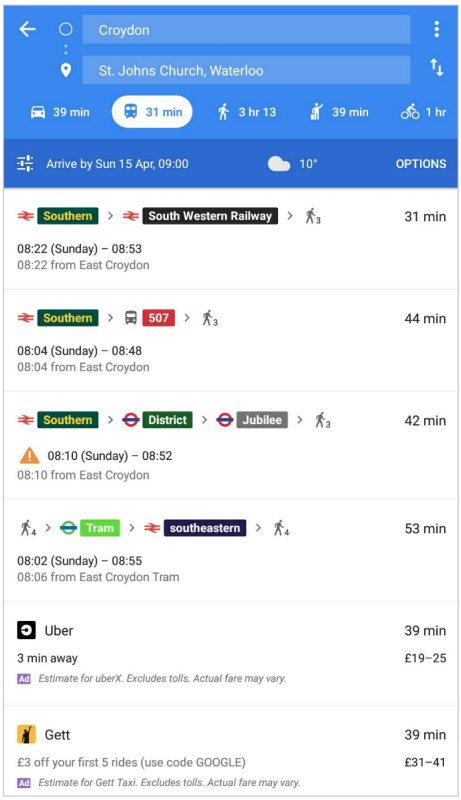


Figure 6.46: Person 1's journey opened in the Google Maps Android application.

All in all, the mobile version of the application is easy to use and provides the users with what they need regardless of the limited screen space.

6.6 Data Collection and Processing

In order for the application to run successfully, certain data must be available based on the information provided by the user. The information provided by the user consists of:

- The starting addresses of all members of the group.
- The mode of transport being used by each member of the group.
- Where they would like to meet.
- (Optionally) the time they would like to meet up at.

In this section, we will see how this information is used to collect and process important data for use by the underlying algorithms and the front end of the web application. The required data is as follows:

- The geographic location of each user.
- Places in Greater London matching the description provided by the user and their geographic location.
- More detailed data about these places.
- The distances and durations of the journeys between every user and every place using the selected mode of transport.
- Removing faulty data.
- Calculating various values which will be used during the implementation of the algorithms.

While discussing the collection of data, we will use a small example of a city represented as a graph to aid understanding. In Figure 6.47, the nodes of the graph represent all the locations in the city, and the red dot in the centre represents the centre of the city.

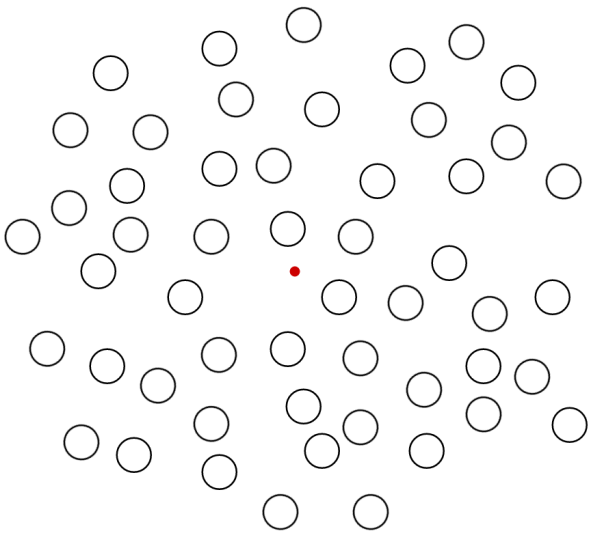


Figure 6.47: A city represented as a graph.

6.6.1 The geographic location of each user

Firstly, user objects are created for each person who provided their details. This process includes verifying that the addresses given by the users are valid to avoid problems later during the implementation. A request is made to the Google Maps Geocoding API, as shown in Figure 6.48. If valid, the address will be converted into latitude and longitude which is stored for each user along with their mode of transport. This address will also be used later to indicate the location of each user with a map marker.

```
1 # Google Maps setup
2 gmaps = googlemaps.Client(api_key)
3 # User's address is passed as a parameter
4 def getLngLat(address):
5     # Request to Geocoding API
6     return gmaps.geocode(address=address)
```

Figure 6.48: Google Maps Geocoding example.

Looking at the example graph, the location of each person in the group is now known and is represented by the coloured, numbered nodes in Figure 6.49.

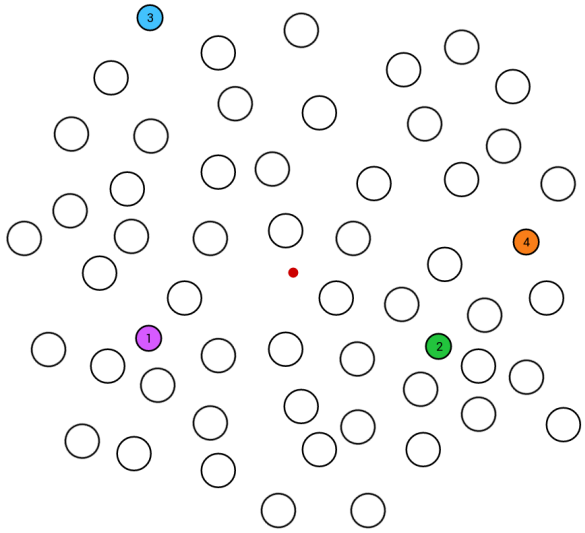


Figure 6.49: The city with the location of people.

6.6.2 Places matching the description provided by the user and their geographic location

After this, up to 200 places are found matching the keyword or point of interest submitted by the user. This is achieved using the Google Maps Places API's Radar Search. The Radar Search aims to "help users identify specific areas of interest within a geographic area." [57] Along with the keyword provided by the user, the centre of the circle used to search for relevant places, and the radius of this circle are taken as parameters in Figure 6.50.

```
1 # Places matching the keyword are found within a 35000km radius
2 # from the centre of London
3 def getPlaces(destination):
4     return gmaps.places_radar((51.508840, -0.126772), keyword=destination,
5     radius=35000)
```

Figure 6.50: Google Maps Radar Search example.

In this application, the centre of London [58] is used with a 35,000 km radius that captures locations up until just outside the M25 motorway. The output is given in JavaScript Object Notation (JSON), making it easy to work with. From this result, the Google Places ID, latitude and longitude of each location is extracted and stored as a list of Place objects.

Referring back to our example, Figure 6.51 shows a visual representation of the radar search taking place. The red arrow represents the radius used, and we can assume here that the ten yellow nodes labelled from A to J are the results obtained by this query.

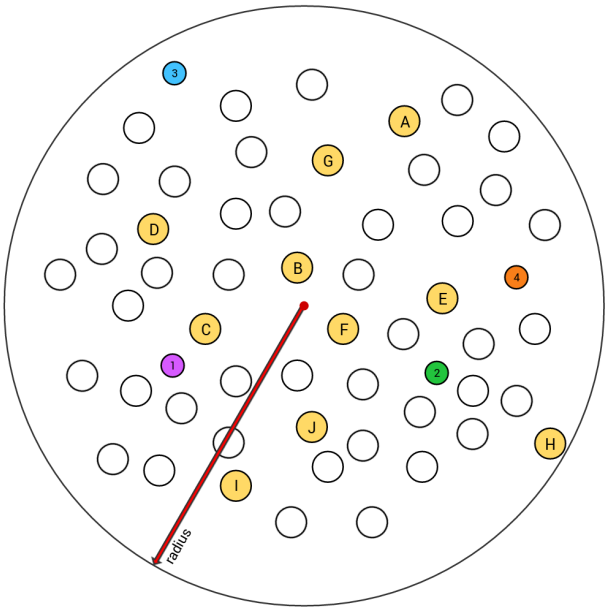


Figure 6.51: The radar search implemented on the city.

6.6.3 More detailed data about these places

In order to provide the user with more details about these places so that they can see information such as its address and rating, the query in Figure 6.52 is performed using the Google Places ID of each location to get this information from the Google Places API. The address, ratings, and url of each location are stored. In reality, this data is not requested until just before results are shown to avoid making unnecessary API calls. Rather than getting extra details about all possible places, this is only done for the top ten locations recommended by the algorithm.

```
1 # More details are found about a place
2 def getPlaceDetails(place):
3     # Use the previously stored Google Places ID
4     return gmaps.place(place_id=place.get_id())
```

Figure 6.52: Google Maps Place Details Search example.

6.6.4 The distances and durations of the journeys between every user and every place using the selected mode of transport

The journey for each user to each of the potential destinations must be found so that their distance and durations can be calculated. This is done using the user's chosen mode of transport and the arrival time if there is one. These are given to the Google Maps Distance Matrix API using the query in Figure 6.53. The Distance Matrix then returns the duration and distance of each journey based on a route recommended by Google Maps.

```
1  def calculateDistancesAndDurations(user, places_list, time=None):
2      # A list of destinations is made using the place ids
3      destinations = ["place_id:" + place.get_id() for place in places_list]
4      # A call is made to the Google Maps Distance Matrix with the user's
5      # address as the origin, the destination list, the mode of transport,
6      # the arrival time chosen by the users, and any extra options the user
7      # chose if they are travelling by car
8      queryresult = gmaps.distance_matrix(user.get_address(), destinations,
9      units="metric", mode=user.get_mode(), arrival_time=time,
10     avoid=user.get_avoid_string())
```

Figure 6.53: Google Maps Distance Matrix example.

Although there are only a maximum of 200 places considered (the number returned by the Google Places Radar Search), the assumption is made that this should work for any number of locations in the future. Since the Google Maps Distance Matrix can only handle 100 matrix entries in a single query, the implementation splits the list of places into smaller sublists of maximum size 100.

Returning to our example, the Distance Matrix query allows weighted edges to be drawn with the cost of the journey between each user and each of the ten yellow locations. In the example in Figure 6.54, we assume the cost represented by the edges is the duration of each journey.

Location A, for example, is 15 minutes away from person 1; 15 minutes away from person 2; 95 minutes away from person 3; and 5 minutes away from person 4.

The storage of this information in a meaningful way is vital for the successful and clean implementation of the algorithms. The algorithms will be using functions that require these

costs to be stored for each user in vectors for each location. In this project, the cost parameters are duration and distance however, in theory, this could also include the price of journeys or the carbon footprint of journeys for example.

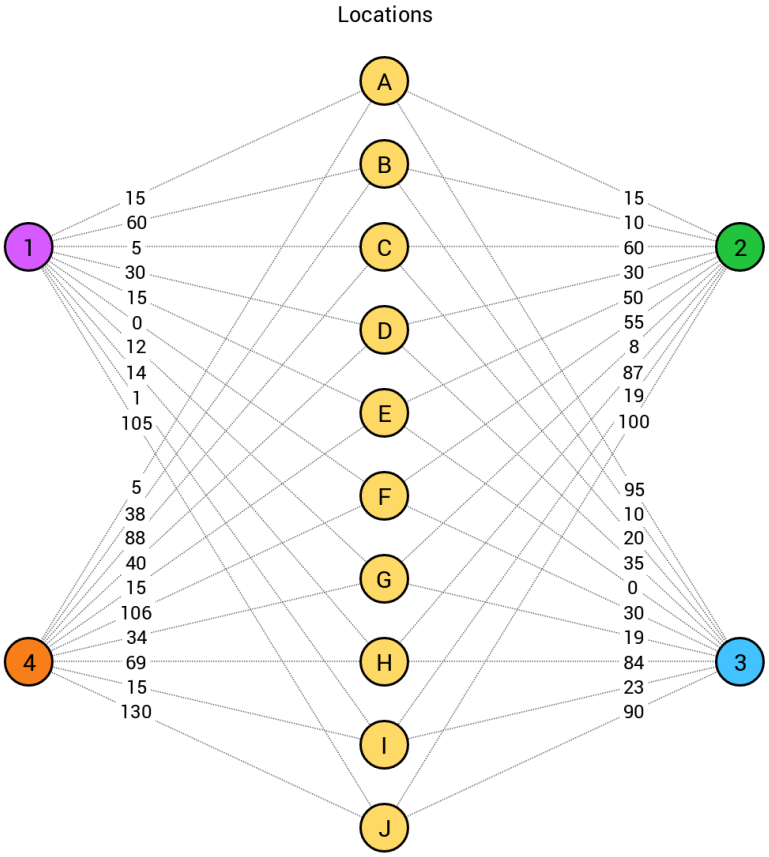


Figure 6.54: The durations of the journeys between each person and each location.

The use of Python lists means the algorithms can work for multiple users, data for a specific individual can be easily accessed through list indexing, and Built-In Python functions can be used on them to assist the implementation of the algorithms. Below a location's vector with its corresponding Python list can be seen. (Note: this notation will be used interchangeably for the remainder of the report.)

$$vector = \begin{pmatrix} cost\ of\ person\ 1's\ journey\ to\ the\ location \\ cost\ of\ person\ 2's\ journey\ to\ the\ location \\ cost\ of\ person\ 3's\ journey\ to\ the\ location \\ \vdots \\ cost\ of\ person\ n's\ journey\ to\ the\ location \end{pmatrix}$$

Python list = [*cost of person 1's journey to the location, cost of person 2's journey to the location, . . . , cost of person n's journey to the location*]

The final lists for the example in Figure 6.54 would be:

A = [15, 15, 95, 5]
B = [60, 10, 10, 38]
C = [5, 60, 20, 88]
D = [30, 30, 35, 40]
E = [15, 50, 0, 15]
F = [0, 55, 30, 106]
G = [12, 8, 19, 34]
H = [14, 87, 84, 69]
I = [1, 19, 23, 15]
J = [105, 100, 90, 130]

6.6.5 Removing faulty data

There was one significant issue encountered whilst using the Google Maps Distance Matrix; when using the place id of a location, occasionally no results were returned by the query. But results were found when repeating the query using the latitude and longitude (latlng) of the location instead like in Figure 6.55 .

```
1 def distanceMatrixQueryByLatLng(user, place, index, time=None):
2     # The destination's latitude and longitude are used this time
3     destinations = [(place.get_lat(), place.get_lng())]
4     queryresult = gmaps.distance_matrix(user.get_address(), destinations,
5     units="metric", mode=user.get_mode(), arrival_time=time,
6     avoid=user.get_avoid_string())
```

Figure 6.55: Google Maps Distance Matrix example.

However, when using the latlng of a location, occasionally no results were found by the query, but using the place id was successful. And occasionally, neither the place id nor the latlng of the location returned any results. These issues were considered by the implementation, and when no results were returned after exhausting all options, the destination was removed from the set of places considered as no meaningful data was found.

6.6.6 Calculating various values which will be used during the implementation of the algorithms

The implementation of the algorithms requires some values which are very quick to calculate using Python, so they are also done at this stage. One example of this is the cost of the maximum journey shown by red edges in Figure 6.56.

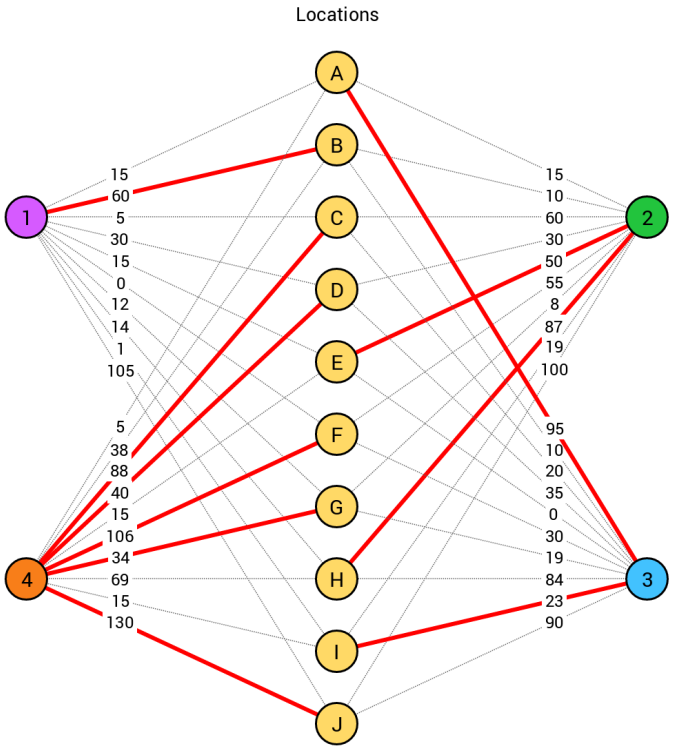


Figure 6.56: The longest journey to each location.

Other values calculated and stored in the place objects are the total distance and total duration of all the journeys; the average distance and duration; and the total deviation from the average which is explained in more depth later in the report.

6.7 Algorithms

'Fairness' is a subjective concept which makes it difficult if not impossible to formally define. It depends on the individual values of the person being asked, and the same person can change their mind depending on their circumstances and what is most important to them in a given moment.

For example, if a group of people are all driving to their destinations, they may want to minimise the overall distance travelled to ensure the lowest amount of money is spent on petrol. If a group of people are environmentally friendly, they may wish to minimise the quantity of harmful gases being released into the atmosphere.

Looking at the the example in Figure 6.57, we see that the journey times to location A for person 1, 2 and 3 are 40 minutes, 30 minutes, and 25 minutes respectively; 40 minutes, 40 minutes, and 50 minutes to location B; and 40 minutes, 40 minutes, and 25 minutes to location C.

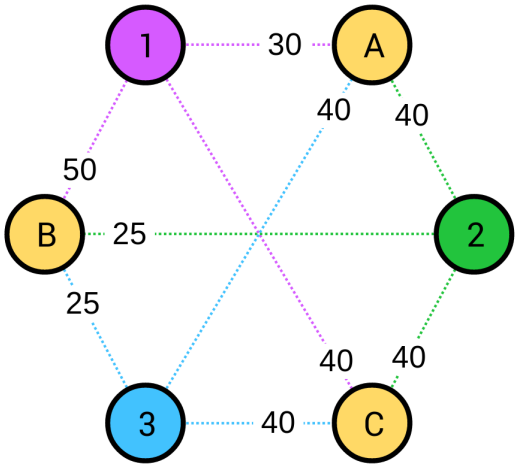


Figure 6.57: The length of each user's journey to each location in minutes.

The corresponding vectors are:

$$A = \begin{pmatrix} 30 \\ 40 \\ 40 \end{pmatrix}, \; B = \begin{pmatrix} 50 \\ 25 \\ 25 \end{pmatrix}, \; C = \begin{pmatrix} 40 \\ 40 \\ 40 \end{pmatrix}$$

Which is the fairest place to meet? Some would say that everyone should travel for the same length of time no matter what, meaning location C is fairest. Others may argue that that it makes sense to choose location A because person 1's journey will be shorter yet the other journey times are unaffected. Someone else may say that, actually, place B is where the group should meet because person 1 travelling for ten minutes longer than they would to place C saves half an hour between the other two people. There may even be groups of people who believe it is fairest to minimise the maximum cost because they don't mind who has the longest journey, they just want it to be as short as possible.

Since the aim of this project is to provide the users with an interface to explore filtered results which have the potential to be the most fair and optimal for their requirements, a number of algorithms have been implemented to give the users a variety of options to choose from.

A quick overview of the way most of these upcoming algorithms work is by using a function to assign a score to each location. These are then used to rank the places in order from 'best' to 'worst'. The lower the score, the 'fairer' and more optimal the place.

6.7.1 Notation

This is a reference for some notation that will be referred to later in the report.

Let the vector v be defined as:

$$v = \begin{pmatrix} v_1 \\ v_2 \\ v_3 \\ \vdots \\ v_n \end{pmatrix}$$

The sum of the vector v is defined as:

$$sum(v) = \sum_{i=1}^{n} v_i \tag{6.1}$$

The average of the vector v is defined as:

$$average(v) = \frac{sum(v)}{n} = \frac{\sum_{i=1}^{n} v_i}{n} \tag{6.2}$$

The deviation of a single vector element v_i of the vector v is defined as:

$$deviation(v_i) = |v_i - average(v)| \tag{6.3}$$

The total deviation of a vector v is defined as:

$$\begin{aligned} total\_deviation(v) &= sum(|v - average(v)|) \\ &= \sum_{i=1}^{n} deviation(v_i) \\ &= \sum_{i=1}^{n} |v_i - average(v)| \\ &= \sum_{i=1}^{n} \left| v_i - \frac{\sum_{i=1}^{n} v_i}{n} \right| \end{aligned} \tag{6.4}$$

6.7.2 Minimising Overall Duration

The first choice for the user is minimising the total duration travelled by the group. This algorithm simply calculates the total duration of all the journeys to a place and orders them from lowest to highest.

Using this algorithm on the example in Figure 6.57, the order given is B = [50, 25, 25] with a total duration of 100 minutes; then A = [30, 40, 40] with a total duration of 110 minutes; and lastly C = [40, 40, 40] with a total duration of 120 minutes.

In a case where there are multiple locations whose total duration is equal, there is no absolute way of defining the best order of locations. For example, the locations in the yellow column of Table 6.1 all have a total duration of 90 minutes.

Looking at locations E = [50, 40, 0] and F = [90, 0, 0], is it fair for person 1 to travel 40 minutes longer to decrease the journey time for person 2?

How about locations E = [50, 40, 0] and G = [30, 50, 10]? Is it more optimal for someone to travel for 0 minutes if the maximum journey does not increase?

Ideally, all these options will be presented to the user for them to decide amongst their group which is the best location for them. However, to assist the experience, some assumptions are made to process the data and provide some form of ranking.

Table 6.1: Example locations whose total duration for each user is 90 minutes.

Original locations	Minimum duration	Sorted (by minimum duration)	Maximum duration	Sorted (by maximum duration)	Final order of locations
A = [30, 30, 30]	30	E (0)	50	A (30)	A = [30, 30, 30]
B = [10, 40, 40]	10	F (0)	90	I (33)	I = [33, 27, 30]
C = [30, 40, 20]	20	B (10)	40	B (40)	B = [10, 40, 40]
D = [20, 50, 20]	20	G (10)	50	C (40)	C = [30, 40, 20]
E = [50, 40, 0]	0	H (10)	70	E (50)	E = [50, 40, 0]
F = [90, 0, 0]	0	C (20)	40	G (50)	G = [30, 50, 10]
G = [30, 50, 10]	10	D (20)	50	D (50)	D = [20, 50, 20]
H = [70, 10, 10]	10	I (27)	33	H (70)	H = [70, 10, 10]
I = [33, 27, 30]	27	A (30)	30	F (90)	F = [90, 0, 0]

Before the locations are ordered by overall duration, they are first ordered by minimum duration (minimising the shortest journey) as shown in the blue columns. After this, they are ordered by maximum duration (minimising the longest journey) which can be seen in the orange columns. We discuss the reasoning behind these in more detail in the explanation of the Most Similar Durations algorithm. The red column of Table 6.1 shows the final order of locations which is a reasonable ranking of the original locations.

6.7.3 Minimising Overall Distance

The concept behind each of the algorithms implemented in this section is that the parameters are modifiable, although small changes in implementation may be required. Some examples of this were discussed earlier, however in order to show the feasibility, an algorithm to minimise total distance was also implemented to show the feasibility and simplicity of changing parameters.

6.7.4 Most Similar Durations

This algorithm caters for people who do not care how long each journey is, as long as the journey times for everyone in the group are as similar as possible. For example, in the example in Figure 6.57, location C = [40, 40, 40] would be 'fairest' by this definition because all the journeys are 40 minutes long. This is followed by A = [30, 40, 40] and then B = [50, 25, 25] which has the most variation in journey times.

There are many ways to calculate how wide the variation between journey durations in a group is. For example, the range of the vector could be taken to show the difference between the longest and shortest journeys to a location. Alternatively, the difference between everyone's journey times could be calculated.

The approach taken by this algorithm, however, is considering how far each journey deviates from the average duration of the journeys. This is defined in Equation 6.4 as the *total_ deviation*. The lower the value, the smaller the deviation from the average.

We can find the *total_ deviation* for place A from the example for clarity. We start by finding the *sum* of A as defined in Equation 6.1:

$$sum(A) = 30 + 40 + 40 = 110$$

Next we find the *average* as defined in Equation 6.2:

$$av(A) = \frac{sum(A)}{n} = \frac{110}{3} = 36.\dot{6}$$

Lastly, we find the *total_deviation* as defined in Equation 6.3:

$$\begin{aligned}
total_deviation(A) &= sum(|A - av(A)|) \\
&= sum\begin{pmatrix} |30 - 36.\dot{6}| \\ |40 - 36.\dot{6}| \\ |40 - 36.\dot{6}| \end{pmatrix} \\
&= sum\begin{pmatrix} |-6.\dot{6}| \\ |3.\dot{3}| \\ |3.\dot{3}| \end{pmatrix} \\
&= 6.\dot{6} + 3.\dot{3} + 3.\dot{3} \\
&= 13.\dot{3}
\end{aligned}$$

The same can be done for locations B and C:

$$\begin{aligned}
sum(B) &= 50 + 25 + 25 = 100 \\
av(B) &= \frac{sum(B)}{n} = \frac{100}{3} = 33.\dot{3} \\
total_deviation(B) &= sum(|B - av(B)|) \\
&= sum\begin{pmatrix} |50 - 33.\dot{3}| \\ |25 - 33.\dot{3}| \\ |25 - 33.\dot{3}| \end{pmatrix} \\
&= sum\begin{pmatrix} |16.\dot{6}| \\ |-8.\dot{3}| \\ |-8.\dot{3}| \end{pmatrix} \\
&= 16.\dot{6} + 8.\dot{3} + 8.\dot{3} \\
&= 33.\dot{3}
\end{aligned}$$

$$
\begin{aligned}
sum(C) &= 40 + 40 + 40 = 120 \\
av(C) &= \frac{sum(C)}{n} = \frac{120}{3} = 40 \\
total\_deviation(C) &= sum(|C - av(C)|) \\
&= sum \begin{pmatrix} |40 - 40| \\ |40 - 40| \\ |40 - 40| \end{pmatrix} \\
&= sum \begin{pmatrix} |0| \\ |0| \\ |0| \end{pmatrix} \\
&= 0 + 0 + 0 \\
&= 0
\end{aligned}
$$

These values can then be ordered from smallest to highest, which leave the final ranking as C, then A, then B, as expected.

If we consider the more complicated example highlighted in yellow in Table 6.2 under the 'Original locations' column, we see eight places and the duration of the journeys of each user to them. The average journey times and deviations per journey (defined in Equation 6.3) are shown in the blue columns. The total deviations for each location and the locations sorted by total deviation from lowest to highest can be seen in the orange columns. The thick black borders show the locations grouped by their deviation which will be discussed later.

Table 6.2: The stages of the Most Similar Durations algorithm.

Original locations	Average journey duration	Deviation per journey	Total deviation	Sorted (by deviation)	Longest journey	Sorted (by longest journey)	Final order of destinations
A = [20, 20, 20]	20	[0, 0, 0]	0	A (0)	20	H (2)	H = [2, 2, 2]
B = [48, 51, 54]	51	[3, 0, 3]	6	F (0)	75	A (20)	A = [20, 20, 20]
C = [13, 12, 11]	12	[1, 0, 1]	2	H (0)	2	F (75)	F = [75, 75, 75]
D = [3, 0, 6]	3	[0, 3, 3]	6	C (2)	13	C (13)	C = [13, 12, 11]
E = [7, 1, 4]	4	[3, 3, 0]	6	B (6)	54	D (6)	D = [3, 0, 6]
F = [75, 75, 75]	75	[0, 0, 0]	0	D (6)	6	E (7)	E = [7, 1, 4]
G = [22, 21, 17]	20	[2, 1, 3]	6	E (6)	7	G (22)	G = [22, 21, 17]
H = [2, 2, 2]	2	[0, 0, 0]	0	G (6)	22	B (54)	B = [48, 51, 54]

As can be seen, there are four locations whose total deviation is 6 minutes, however looking at the actual journey times, there is clearly a huge difference between location E = [7, 1, 4] and location B = [48, 51, 54] for example. Most people would prefer option E because it is much shorter for everyone. This can also be seen when considering locations A = [20, 20, 20] and H = [2, 2, 2]; clearly everyone travelling two minutes is better than everyone travelling 20 minutes. The ranking of locations should know which of these journeys is better.

If a group of people are aiming to make their journeys as exact as possible, it is reasonable to assume that they would not like anyone to have to travel for much longer than the other people. In order to accommodate for this, the algorithm aims to minimise the maximum durations. The longest journeys to each of the locations, shown in the green column, are found.

We consider a deviation bracket to be a set of locations grouped by a specific rule. In this case, the deviation brackets consist of any locations with the same total deviation and can be seen by the thick black borders in Table 6.2. Within each deviation bracket, the locations are sorted in ascending order of longest journey. This means that the most similar durations are dealt with, followed by the next most similar durations, and so on and so forth.

In the implementation of the algorithm, before the maximum journeys are minimised, there is one prior step: the locations are sorted in ascending order by minimum journey duration. This is done under the assumption that, if the length of the maximum journey will not be increased, the length of the minimum journey may as well be as low as possible. An example of this is taking two locations whose duration vectors are as follows:

$$\text{A} = [50, 49, 54]$$
$$\text{B} = [51, 48, 54]$$

The total deviation for both locations is 6 minutes. Is it better to choose location A or B? Well, in both cases, the longest journey (54 minutes) is the same, so if it is possible for someone to have the shortest possible journey (48 minutes) without this affecting the longest journey, this is beneficial.

The final ranking of destinations is shown in the red column. Using this algorithm, the users are left with a ranking of the most similar journey times, where whoever has the longest journey has the shortest possible longest journey.

Using real data returned by the Google Maps Distance Matrix API, we can apply this algorithm to a new set of test cases, shown in Table 6.3.

Table 6.3: The stages of the Most Similar Durations algorithm using real data.

Original locations	Average journey duration	Deviation per journey	Total deviation	Sorted (by deviation)	Longest journey	Sorted (by longest journey)	Final order of destinations
A = [86, 68, 50]	68.0	[18.0, 0.0, 18.0]	36.0	N (1.3)	57	N (57)	N = [56, 57, 56]
B = [79, 52, 50]	60.3	[18.7, 8.3, 10.3]	37.3	L (5.3)	52	L (52)	L = [50, 47, 52]
C = [39, 50, 41]	43.3	[4.3, 6.7, 2.3]	13.3	G (7.3)	47	G (47)	G = [43, 47, 40]
D = [79, 57, 53]	63.0	[16.0, 6.0, 10.0]	32.0	F (10.0)	61	J (55)	J = [55, 45, 50]
E = [65, 66, 57]	62.7	[2.3, 3.3, 5.7]	11.3	J (10.0)	55	F (61)	F = [61, 53, 60]
F = [61, 53, 60]	58.0	[3.0, 5.0, 2.0]	10.0	E (11.3)	66	E (66)	E = [65, 66, 57]
G = [43, 47, 40]	43.3	[0.3, 3.7, 3.3]	7.3	C (13.3)	50	C (50)	C = [39, 50, 41]
H = [53, 56, 33]	47.3	[5.7, 8.7, 14.3]	28.7	M (15.3)	52	M (52)	M = [37, 52, 44]
I = [33, 50, 46]	43.0	[10.0, 7.0, 3.0]	20.0	I (20.0)	50	I (50)	I = [33, 50, 46]
J = [55, 45, 50]	50.0	[5.0, 5.0, 0.0]	10.0	K (20.0)	55	K (55)	K = [55, 38, 42]
K = [55, 38, 42]	45.0	[10.0, 7.0, 3.0]	20.0	O (22.0)	63	O (63)	O = [63, 42, 51]
L = [50, 47, 52]	49.7	[0.3, 2.7, 2.3]	5.3	H (28.7)	56	H (56)	H = [53, 56, 33]
M = [37, 52, 44]	44.3	[7.3, 7.7, 0.3]	15.3	D (32.0)	79	D (79)	D = [79, 57, 53]
N = [56, 57, 56]	56.3	[0.3, 0.7, 0.3]	1.3	A (36.0)	86	A (86)	A = [86, 68, 50]
O = [63, 42, 51]	52.0	[11.0, 10.0, 1.0]	22.0	B (37.3)	79	B (79)	B = [79, 52, 50]

Note: All values rounded to one decimal place.

6.7.5 Reasonably Similar Durations

Since journey times can be affected by the smallest things, from a delayed bus to unexpected traffic, it is impossible to truly predict the actual duration of a journey. Furthermore, for many people, travelling 5 or 10 minutes longer than their friends is neither here nor there and it does not affect them. This algorithm therefore considers reasonably similar journey times and operates in a very similar way to the previous Most Similar Durations algorithm.

The first steps are the same: calculating the total deviation for each location and sorting the locations by this total deviation. Previously, locations were grouped into deviation brackets based on having the exact same total deviation. In this algorithm, the rule which determines which bracket a location falls into is altered. The total deviation lies within specific ranges:

$$\text{lower bound} < \text{total deviation} \leq \text{upper bound}$$

Although in theory, the range of these bounds can be decided by the users, for the purposes of implementation, the difference between the upper and lower bound is 10:

$$\begin{aligned} 0 &\leq \text{total deviation} \leq 10 \\ 10 &< \text{total deviation} \leq 20 \\ 20 &< \text{total deviation} \leq 30 \\ &\vdots \end{aligned}$$

We find that the first bracket considers locations with a maximum deviation of up to 5 minutes, the second bracket considers locations with a maximum deviation of up to 10 minutes, then 15 minutes, then 20, and so on. This can be seen clearly in the purple column of Table 6.4 where the new deviation brackets are surrounded by thick black borders.

Table 6.4: The stages of the Reasonably Similar Durations algorithm.

Original locations	Average journey duration	Deviation per journey	Total deviation	Sorted (by deviation)	Max deviation	Shortest journey	Sorted (by shortest journey)	Longest journey	Sorted (by longest journey)	Final order of destinations
A = [86, 68, 50]	68.0	[18.0, 0.0, 18.0]	36.0	N (1.3)	0.7	56	G (40)	47	G (47)	G = [43, 47, 40]
B = [79, 52, 50]	60.3	[18.7, 8.3, 10.3]	37.3	L (5.3)	2.7	47	J (45)	55	L (52)	L = [50, 47, 52]
C = [39, 50, 41]	43.3	[4.3, 6.7, 2.3]	13.3	G (7.3)	3.7	40	L (47)	52	J (55)	J = [55, 45, 50]
D = [79, 57, 53]	63.0	[16.0, 6.0, 10.0]	32.0	F (10.0)	5.0	53	F (53)	61	N (57)	N = [56, 57, 56]
E = [65, 66, 57]	62.7	[2.3, 3.3, 5.7]	11.3	J (10.0)	5.0	45	N (56)	57	F (61)	F = [61, 53, 60]
F = [61, 53, 60]	58.0	[3.0, 5.0, 2.0]	10.0	E (11.3)	5.7	57	I (33)	50	I (50)	I = [33, 50, 46]
G = [43, 47, 40]	43.3	[0.3, 3.7, 3.3]	7.3	C (13.3)	6.7	39	M (37)	52	C (50)	C = [39, 50, 41]
H = [53, 56, 33]	47.3	[5.7, 8.7, 14.3]	28.7	M (15.3)	7.7	37	K (38)	55	M (52)	M = [37, 52, 44]
I = [33, 50, 46]	43.0	[10.0, 7.0, 3.0]	20.0	I (20.0)	10.0	33	C (39)	50	K (55)	K = [55, 38, 42]
J = [55, 45, 50]	50.0	[5.0, 5.0, 0.0]	10.0	K (20.0)	10.0	38	E (57)	66	E (66)	E = [65, 66, 57]
K = [55, 38, 42]	45.0	[10.0, 7.0, 3.0]	20.0	O (22.0)	11.0	42	H (33)	56	H (56)	H = [53, 56, 33]
L = [50, 47, 52]	49.7	[0.3, 2.7, 2.3]	5.3	H (28.7)	14.3	33	O (42)	63	O (63)	O = [63, 42, 51]
M = [37, 52, 44]	44.3	[7.3, 7.7, 0.3]	15.3	D (32.0)	16.0	53	A (50)	86	B (79)	B = [79, 52, 50]
N = [56, 57, 56]	56.3	[0.3, 0.7, 0.3]	1.3	A (36.0)	18.0	50	B (50)	79	D (79)	D = [79, 57, 53]
O = [63, 42, 51]	52.0	[11.0, 10.0, 1.0]	22.0	B (37.3)	18.7	50	D (53)	79	A (86)	A = [86, 68, 50]

Note: All values rounded to one decimal place.

Within each of these deviation brackets, the worst case scenario is that the deviation of someone's journey time from the average journey duration is at most $\frac{upper\ bound}{2}$ minutes. Although this does not necessarily include every location with journeys that have a deviation of less than $\frac{upper\ bound}{2}$ minutes, every journey included in the bracket has a maximum deviation of $\frac{upper\ bound}{2}$ minutes.

To help understand this, we can consider a deviation bracket:

$$0 \text{ minutes} = \text{lower bound} < \text{total deviation} \leq \text{upper bound} = 10 \text{ minutes}$$

Since the sum of the individual deviations (before taking the absolute values of them) must add to 0, in order to have a total deviation of 10, the highest possible individual deviation is 5 minutes. It cannot be any higher because if there is an individual deviation of > 5, there must be (at least one) other individual deviation(s) whose sum is $<$ -5 in order for the total sum to be 0. If there are individual deviations of ≥ 6 and $\leq$ -6, the total deviation is at least 12 minutes, which is greater than the upper bound (10 minutes).

The same thing can be seen in the next deviation bracket:

$$10 \text{ minutes} = \text{lower bound} < \text{total deviation} \leq \text{upper bound} = 20 \text{ minutes}$$

To have a total deviation of 20, the highest and lowest possible deviations are 10 minutes and -10 minutes. Again showing that the maximum deviation is 10 minutes which is at most $\frac{upper\_bound}{2}$ minutes.

The following sorting is done within the deviation brackets to rank the locations which will be recommended to the users. The blue columns show the locations being ordered by minimising the shortest journeys. This affects locations C and I in particular as they have the same maximum journey duration (50 minutes) but different minimum journey durations (39 minutes and 33 minutes respectively). We can see that they swap their order of appearance in this column.

The green columns highlight the longest journeys and them being sorted in ascending order, and finally the red column shows the algorithm's recommended order of destinations.

6.7.6 Duration Fairness Functions

The last algorithms implemented aim to find a balance between the total deviation and the total cost of the journeys to each location. Although there are many ways to decide how to balance these based on which is more important, this algorithm assigns a 'score' to each location l using the following formula:

$$\begin{aligned} &\alpha(total\_deviation(l)) + \beta(sum(l)) \\ &\qquad where\ \alpha + \beta = 1 \end{aligned} \tag{6.5}$$

The reason for this is to easily allow the ratio of importance between total deviation

and total cost to be changed easily. During testing over a wide range of different data sets, adjusting α and β (for example $\frac{1}{2}$, $\frac{1}{3}$, $\frac{1}{4}$, $\frac{1}{5}$) produced very similar results with very slight differences in order. From this experimenting, two different algorithms were been selected to give the users the best possible experience. One of these prioritises the total deviation and one prioritises the total duration of the group members' journeys.

Prioritise Total Deviation

When total deviation is prioritised, α is set to $\frac{3}{4}$ and β is set to $\frac{1}{4}$ to give it three times as much importance.

Prioritise Total Duration

When total duration is prioritised, α is set to $\frac{1}{4}$ and β is set to $\frac{3}{4}$ to give total duration three times as much importance.

In theory, the users could have the ability to select which of these are more important to them and by how much to give them full control over α and β. The results of the chosen values of can be seen in Table 6.5 below. The green columns show the prioritisation of total deviation and red shows the prioritisation of total cost.

Table 6.5: The stages of the Fairness Function algorithms.

			Prioritise total deviation			Priotitise total cost		
Original locations	Total deviation	Total cost	Score = (0.75 x total deviation) + (0.25 x total cost)	Sorted (by score)	Final order of destinations	Score = (0.25 x total deviation) + (0.75 x total cost)	Sorted (by score)	Final order of destinations
A = [86, 68, 50]	36.0	204	78.0	G (38.0)	G = [43, 47, 40]	162.0	G (99.3)	G = [43, 47, 40]
B = [79, 52, 50]	37.3	181	73.2	L (41.2)	L = [50, 47, 52]	145.1	C (100.8)	C = [39, 50, 41]
C = [39, 50, 41]	13.3	130	42.5	C (42.5)	C = [39, 50, 41]	100.8	I (101.8)	M = [37, 52, 44]
D = [79, 57, 53]	32.0	189	71.2	N (43.2)	N = [56, 57, 56]	149.8	M (103.6)	I = [33, 50, 46]
E = [65, 66, 57]	11.3	188	55.5	M (44.7)	M = [37, 52, 44]	143.8	K (106.2)	L = [50, 47, 52]
F = [61, 53, 60]	10.0	174	51.0	J (45.0)	J = [55, 45, 50]	133.0	L (113.1)	K = [55, 38, 42]
G = [43, 47, 40]	7.3	130	38.0	I (47.2)	I = [33, 50, 46]	99.3	H (113.7)	J = [55, 45, 50]
H = [53, 56, 33]	28.7	142	57.0	K (48.8)	K = [55, 38, 42]	113.7	J (115.0)	N = [56, 57, 56]
I = [33, 50, 46]	20.0	129	47.2	F (51.0)	F = [61, 53, 60]	101.8	O (122.5)	H = [53, 56, 33]
J = [55, 45, 50]	10.0	150	45.0	E (55.5)	E = [65, 66, 57]	115.0	N (127.1)	O = [63, 42, 51]
K = [55, 38, 42]	20.0	135	48.8	O (55.5)	O = [63, 42, 51]	106.2	F (133.0)	F = [61, 53, 60]
L = [50, 47, 52]	5.3	149	41.2	H (57.0)	H = [53, 56, 33]	113.1	E (143.8)	E = [65, 66, 57]
M = [37, 52, 44]	15.3	133	44.7	D (71.2)	D = [79, 57, 53]	103.6	B (145.1)	B = [79, 52, 50]
N = [56, 57, 56]	1.3	169	43.2	B (73.2)	B = [79, 52, 50]	127.1	D (149.8)	D = [79, 57, 53]
O = [63, 42, 51]	22.0	156	55.5	A (78.0)	A = [86, 68, 50]	122.5	A (162.0)	A = [86, 68, 50]

6.7.7 Summary

A direct comparison of the results can be seen in Table 6.6 below. They all order the locations slightly differently, each with a different goal. There are some similarities between them, for example that location A is almost consistently at the bottom of each list, or in the 'Most Similar' yellow column, it is the penultimate location in the list. Similarly Location G is the first location in the last three algorithms, and is third in the first two.

Table 6.6: A comparison of the location ranking of each algorithm.

Final order of destinations				
Minimise total duration	Most similar	Reasonably similar	Prioritise total deviation	Prioritise total duration
I = [33, 50, 46]	N = [56, 57, 56]	G = [43, 47, 40]	G = [43, 47, 40]	G = [43, 47, 40]
C = [39, 50, 41]	L = [50, 47, 52]	L = [50, 47, 52]	L = [50, 47, 52]	C = [39, 50, 41]
G = [43, 47, 40]	G = [43, 47, 40]	J = [55, 45, 50]	C = [39, 50, 41]	M = [37, 52, 44]
M = [37, 52, 44]	J = [55, 45, 50]	N = [56, 57, 56]	N = [56, 57, 56]	I = [33, 50, 46]
K = [55, 38, 42]	F = [61, 53, 60]	F = [61, 53, 60]	M = [37, 52, 44]	L = [50, 47, 52]
H = [53, 56, 33]	E = [65, 66, 57]	I = [33, 50, 46]	J = [55, 45, 50]	K = [55, 38, 42]
L = [50, 47, 52]	C = [39, 50, 41]	C = [39, 50, 41]	I = [33, 50, 46]	J = [55, 45, 50]
J = [55, 45, 50]	M = [37, 52, 44]	M = [37, 52, 44]	K = [55, 38, 42]	N = [56, 57, 56]
O = [63, 42, 51]	I = [33, 50, 46]	K = [55, 38, 42]	F = [61, 53, 60]	H = [53, 56, 33]
N = [56, 57, 56]	K = [55, 38, 42]	E = [65, 66, 57]	E = [65, 66, 57]	O = [63, 42, 51]
F = [61, 53, 60]	O = [63, 42, 51]	H = [53, 56, 33]	O = [63, 42, 51]	F = [61, 53, 60]
B = [79, 52, 50]	H = [53, 56, 33]	O = [63, 42, 51]	H = [53, 56, 33]	E = [65, 66, 57]
E = [65, 66, 57]	D = [79, 57, 53]	B = [79, 52, 50]	D = [79, 57, 53]	B = [79, 52, 50]
D = [79, 57, 53]	A = [86, 68, 50]	D = [79, 57, 53]	B = [79, 52, 50]	D = [79, 57, 53]
A = [86, 68, 50]	B = [79, 52, 50]	A = [86, 68, 50]	A = [86, 68, 50]	A = [86, 68, 50]

The benefit of having these alternative algorithms is that they account for the fact that fairness cannot be captured in a single statement and the users have the opportunity to explore the variations themselves to find the best option for their group.

6.8 Integration and System Architecture

Now that the User Interface and Algorithm functions have been explained and described, in this section we elaborate on the diagram of the System Overview from Figure 6.2 and look more closely on how the front and back ends interact with one another.

There are two key points where the front and back ends of the system interact with each other and data is requested from the external services, i.e. the Google Maps APIs.

The first of these is when the user submits their data and receives a list of recommended locations. The diagram in Figure 6.58 shows the sequence of events when the 'submit' button is clicked. This diagram can be read by following the line down from the tip of each arrowhead to the next arrow.

Until the address given by a user is geocoded, the system does not know whether it is a valid input or not, so before allowing the user to 'submit' their data, the address given must be checked as an additional step to form validation.

Steps 3 and 6 show what happens when incomplete or invalid information is given by the user; an error message is displayed. Steps 4, 7, and 9 show the points when data is requested from the various Google Maps APIs. Step 12 is when the user's chosen algorithm is applied, and step 15 is when these results are returned to the front end for display to the user.

The second example highlighting the importance of smooth system integration is when a user selects a location to see more information about it. At this point, extra data is required, i.e. the directions of each user's journey to the selected location. Waiting until a location is selected to get the directions reduces the user's initial wait for results and the process can be followed in the sequence diagram in Figure 6.59

When the location is selected, the stored details about the location are displayed immediately, and at the same time, in step 2, a request is sent to the Google Directions API using the place id. These directions are then made available to the user to view once the query has been successfully completed after steps 3 and 4.

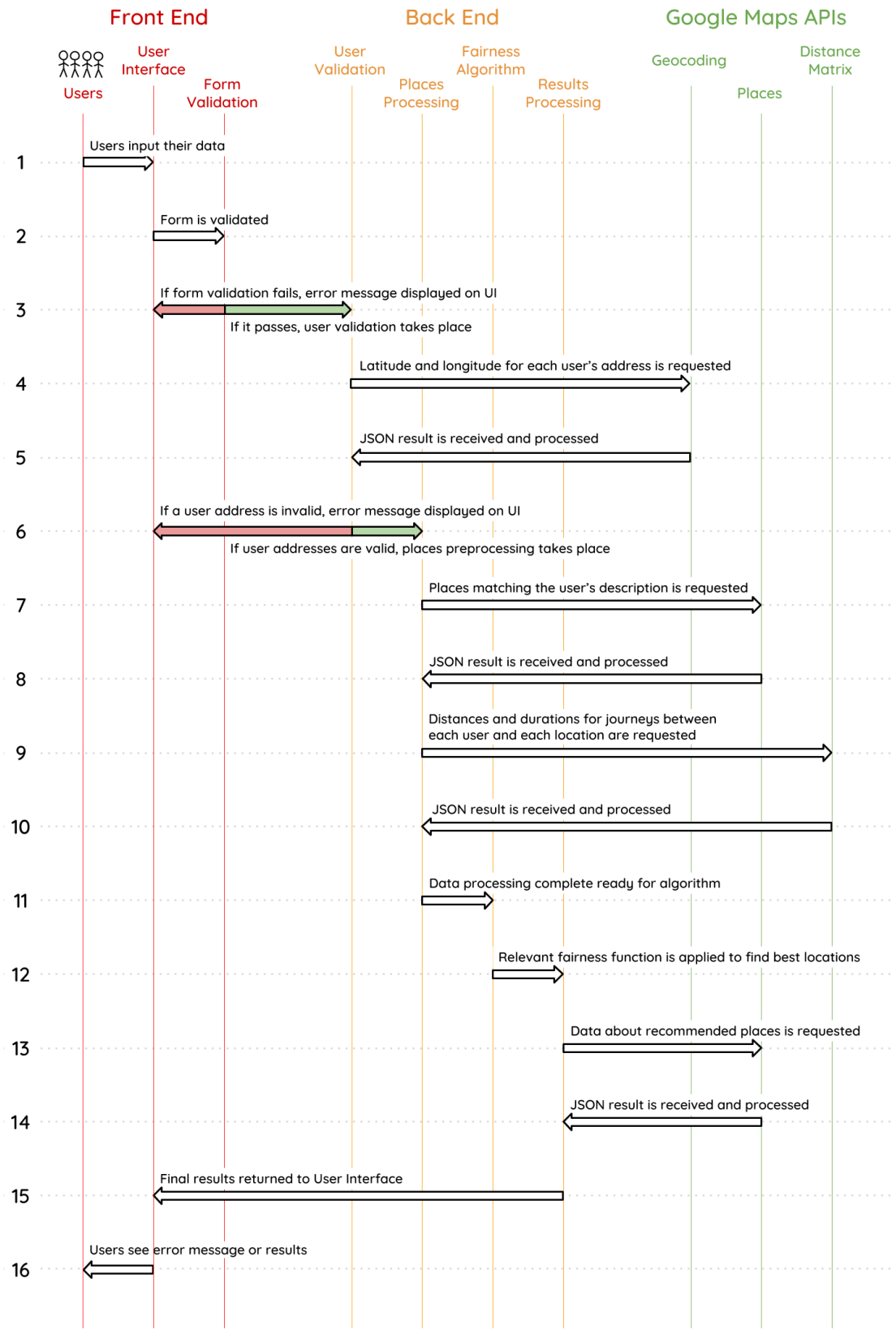


Figure 6.58: Sequence Diagram after submit clicked.

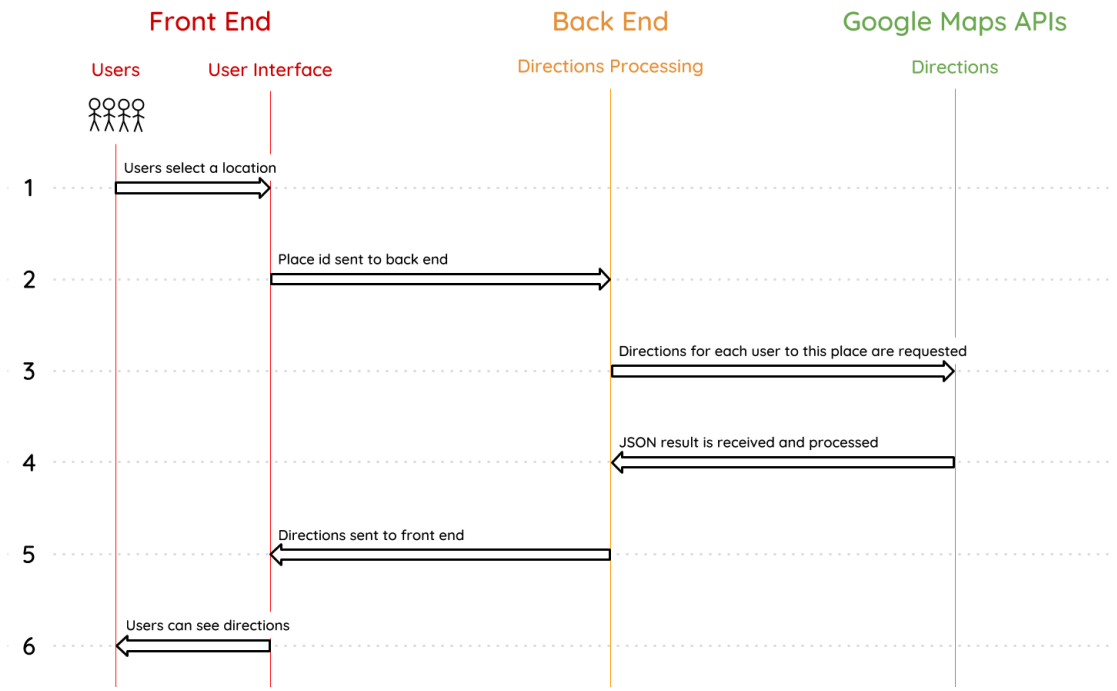


Figure 6.59: Sequence Diagram after location clicked.

7 | Testing

Testing is important throughout any software development project and this is especially the case for this project which follows the adapted Feature Driven Development methodology outlined earlier in Figure 5.1. The introduction of each new feature relies on the successful and accurate implementation of previous features. Furthermore, without an extensively tested foundation, finding the source of the problem when a new feature fails can take longer than it should. This has the potential to slow down development and require a cumulation of previously overlooked issues to be fixed.

Most of the testing in this project followed four formal stages: Unit Testing; Integration Testing; System Testing; and User Acceptance Testing [59]. For the User Interface, rather than Unit Testing, UI Testing was done [60]. Once the software itself was working, a further level of testing was required to check that results were actually reasonable. This could not necessarily be achieved through these five methods of testing, but systems with similar aims were used to compare results for the same test cases. This is discussed further in the Results section of the report.

7.1 Unit Testing

The underlying principle of the application is that users should receive a ranked list of results which has been compiled by an algorithm in the back end. It was important, therefore, to check each separate element of the back end was working correctly in isolation to make sure that there were no issues which would affect the list of results displayed to the user.

Python has a unit testing framework called `unittest` which makes it easy to test that individual functions are producing the expected output. In Figure 7.1, we can see an example of some code which finds the average of a list to one decimal place.

```
1 def getAverage(test_list):
2     total = sum(test_list)
3     return round(sum(test_list)/len(test_list), 1)
```

Figure 7.1: Python function that finds the average of a list of integers to one decimal place.

In Figure 7.2, we can see a unit test checking that this is calculated correctly.

```
1  import unittest
2  class TestAverage(unittest.TestCase):
3
4      def test_average(self):
5          list_to_check = [21, 21, 22]
6          expected_average = 21.3
7          actual_result = getAverage(list_to_check)
8          self.assertEqual(expected_average, actual_result)
9
10 if __name__ == '__main__':
11     unittest.main()
```

Figure 7.2: Python unit test example.

The outcome of this is: '`Ran 1 test in 0.000s OK`' indicating that the test ran successfully and passed.
This is the format of the unit tests for the back end and some examples of tests are outlined below. When one of these failed, debugging took place, followed by retesting.

1. JSON responses from Google Maps APIs are correctly processed and stored.
2. Maximum durations and distances are correctly identified.
3. The sum of a vector is correctly calculated.
4. The average of a vector is correctly calculated.
5. The places are ranked in ascending order by the correct key.
6. Lists are correctly split into smaller lists of maximum size 100.
7. 'Rogue' places (places with missing data) are correctly removed.
8. Invalid users are correctly detected and dealt with.
9. The final list of places is the correct length.
10. Specific test cases are ranked as expected (for example [10, 10, 10] is always better than [20, 20, 20]).

Some of these tests were added when specific issues were found during development. For example, initially data was being extracted from the JSON responses incorrectly which was causing errors.

7.2 UI Testing

The way in which the users receive the results on the User Interface is another important aspect of the project. A similar process to unit testing was used to check that individual elements of the front end were working correctly, which are outlined below:

1. Google Autocomplete works.
2. Google Autocomplete suggestions are limited to the UK.
3. Transport button groups work.
4. People travelling by car have additional options.
5. The calendar works.
6. Adding a user works.
7. Adding a user feature unavailable at maximum group size.
8. Removing a user works.
9. Removing a user feature unavailable at minimum group size.
10. The colours of the users are assigned correctly.
11. Form validation occurs.
12. The submit button shows a loading a symbol while waiting for results.
13. The submit button is disabled after being clicked.
14. The sidebar appears when results are found.
15. Results should have the name of the location.
16. Results should display the address of the location.
17. Results should display the total distances and durations of the journeys to it.

18. Ratings should be visible if and only if available.

19. Selected locations display the name of the location.

20. The name of the location is linked to the URL of the location.

21. Selected locations display the address of the location.

22. Selected locations display buttons for each user.

23. Each button has the correct colour corresponding to each person.

24. Directions are displayed in a list.

25. If there are no directions, this is made clear.

26. The map displays markers for people and locations.

27. The map markers are the correct colours.

28. The map displays polylines showing routes.

29. The map resizes to fit markers and polylines.

30. Infowindows with details about each marker should appear when hovering over one.

31. The Google Maps URLs are formed correctly.

These were done by the developer on Google Chrome as features were implemented. Further testing of these elements took place with slight alterations for the mobile version. To name a few:

1. The Google Maps URL opens in the Google Maps mobile application.

2. The markers on the mobile can be clicked rather than hovered over to view InfoWindows.

3. The sidebar expands to fill the entire width of the screen.

7.3 Integration Testing

After the elements of the front and back ends were tested, integration testing was performed to ensure that they work together correctly. This focused heavily on the smooth navigation between the different UI screens and can be split into two types: tests which checked that the client and server sides of the application were communicating correctly, and tests which confirmed whether the UI elements were working as expected.

An outline of some of the interactions:

1. User selections is sent to the back end correctly.
2. User validation takes place and results are sent back to the front end.
3. Directions are found correctly when the user clicks a location.

Some of the tests for UI elements working correctly;

1. The results list compiles correctly.
2. The correct location is displayed when selected.
3. The list of all markers, polylines and InfoWindows clear and appear at the right times.
4. Clicking markers triggers the correct location to be displayed.
5. Results are displayed correctly when the back button is pressed after viewing a specific place.
6. The submit button is enabled and the loading symbol is removed when the user returns to the search screen.

One of the issues encountered during this stage of testing was that directions were not being displayed. This was happening because the directions buttons were being clicked before the request to the server was complete. The solution was simply to only make the buttons visible once directions are available.

7.4 System Testing

System testing was the point at which the entire application was tested as a whole to confirm that the initial specification was satisfied, and that every designed feature of the application worked as expected for the optimum user experience. This was done in a number of ways on a wide range of popular screen resolutions. Initially, the developer tested the system using brief test cases to confirm that the flow of the system was as expected. These test cases included invalid users; locations with known issues; invalid points of interest; and journeys with non-existent directions to test the ability of the system to handle such cases.

After this, the previously composed User Stories were tested to see whether they were achieved successfully. In this stage, one of the issues found was that users could press the submit button multiple times. This resulted in unnecessary calls to the Google Maps APIs and, in some cases, elongated the user's wait for results. This was fixed by disabling the submit button once it had been clicked. Another small and easily fixable issue found was directions not being displayed consistently when the buttons were being clicked. This happened when a user clicked the text within the button (such as "17 mins") because the system was registering the text as clicked rather than the button.

7.5 User Acceptance Testing

The point of this web application is to make the experience of finding places to meet up easier which is why User Acceptance Testing (UAT) was so important. A wide range of people who have all lived in London used the application in real world situations. As the users navigated through the web application, any comments they had were noted down.

Most of the feedback received was positive and showed that the requirements had been achieved. An issue discovered during this phase (that was consequently fixed) was text overflow in buttons with long journey durations. Users of the application requested different colours to the originally available ones, additional UI elements such the Google Maps link, and the inclusion of total journey distances and durations. These were implemented and adjusted accordingly to make the application as suitable as possible for user needs. Testers also requested further information about each algorithm as it was not completely clear which option they should select from the dropdown menu. This is something to consider in the future work section, where other comments from UAT are discussed.

8 Results

Now that the development and testing has been completed, we can look at some specific case studies to measure the success of the project. To find these case studies and test the application fairly without bias, a random postcode generator [61] was used to find addresses for mock participants.

Each of these case studies will include a table of results where each column shows the top 10 locations recommended by each algorithm. In some of the case studies, this will include a comparison to existing solutions. It is also worth noting that these algorithms were run a few minutes apart. So in some cases when a location appears to be in one column but not in another where we might expect it to be, this is likely to be because traffic has changed since the last algorithm was run, or perhaps that the next bus time is different. These results were not changed to show real implementation of the algorithms.

Although the results will not be assigned a letter for clarity, the way in which to read the table is the same as it has been throughout the rest of this report. For clarity, a column showing '[54, 12, 36]' would mean that three people will be travelling to this location. Person 1 will be travelling for 54 minutes; person 2 for 12 minutes; and person 3 for 36 minutes. The same locations may appear in different columns in different positions; this is simply based on how each algorithm processes the locations.

Some key points will be described about each table, but as the nature of this project is very subjective, and to avoid repetition, the results are displayed clearly enough for us to explore individually. When looking at the recommended locations, we should decide for ourselves which algorithm's results we like best, and we are encouraged to consider why this is the case as it is an interesting question.

8.1 Case Study 1

This case study tested two people travelling from Woodgrange Avenue, Harrow, HA3 0XD and Esther Close, London, NW1 1AW, which are within ten miles of one another. Both users are travelling by public transport and they would like to meet for bowling.

The results shown in Table 8.1 show what we expect for each column. In the purple

column, where total duration is minimised, we see that there is significant variation in journey durations, but overall the sum of these is minimised for each location.

In the yellow column which shows the 'most similar' journey times, we see that the difference in journey times increases as we go down the column. This results in the location which is almost two hours away from both users being second in the list simply because both durations are equal.

The blue column showing 'reasonably similar durations' gives a much more reasonable order for fairness than the previous two algorithms. The journeys to each location are all within 5 minutes of each other and the maximum journey time has been minimised.

Prioritising the total deviation, in the orange column, shows what appears to be very similar results to the 'most similar' durations column with [111, 111] filtered out. None of the journey times to each location are more than one minute apart, so it seems to have found what might be described as the most similar but still sensible locations.

The green column shows the results for prioritising the total duration while of course still considering the total deviation. The results are very similar to the 'reasonably similar' results but are presented in a different order.

The final two columns show the results for two of the existing systems discussed in the report. Many of the results in these columns are quite acceptable, however there are some examples of journeys which have significantly different durations such as [68, 48] which appears in both results and [67, 50] which appears in What's Halfway's column.

Table 8.1: Results for case study 1.

Final order of destinations						
Minimise total duration	Most similar	Reasonably similar	Prioritise total deviation	Prioritise total duration	Meetways	What's Halfway
[56, 22]	[61, 61]	[49, 48]	[61, 61]	[49, 48]	[50, 51]	[50, 51]
[20, 65]	[111, 111]	[46, 51]	[49, 48]	[50, 51]	[58, 49]	[48, 48]
[84, 11]	[49, 48]	[46, 51]	[50, 51]	[53, 52]	[54, 51]	[56, 54]
[26, 70]	[50, 51]	[50, 51]	[53, 52]	[52, 49]	[56, 54]	[54, 51]
[49, 48]	[53, 52]	[52, 49]	[61, 60]	[53, 50]	[68, 48]	[58, 49]
[46, 51]	[61, 60]	[53, 50]	[69, 70]	[53, 50]	[56, 22]	[67, 50]
[46, 51]	[69, 70]	[53, 50]	[79, 78]	[54, 51]	[58, 57]	[68, 48]
[41, 57]	[79, 78]	[53, 52]	[83, 82]	[61, 61]	[62, 58]	[58, 57]
[31, 69]	[83, 82]	[54, 50]	[86, 85]	[46, 51]	[61, 61]	[51, 55]
[50, 51]	[86, 85]	[54, 51]	[86, 85]	[46, 51]	[53, 52]	[54, 56]

8.2 Case Study 2

Case study 2 looks more closely at people who are travelling by car. Person 1 is coming from Cheam Road, Sutton, SM1 2BT, and person 2 is coming from Greenstead Avenue, Woodford Green, IG8 7ET. They are going to the cinema.

The most immediate observation is that the first results in Table 8.2 returned by Meetways and What's Halfway require one of the users to travel for over an hour and a half. When compared to the results of every algorithm implemented by this project, none of the journeys are this long.

Most of the results in the 'most similar', 'reasonably similar', 'prioritise total deviation' and 'prioritise total duration' are quite similar, for example [52, 52] and [51, 52] appear in all four columns.

By most definitions of fairness, the implemented algorithms in this project appear to give a better ranking of locations than the two existing systems shown.

Table 8.2: Results for case study 2.

Final order of destinations						
Minimise total duration	Most similar	Reasonably similar	Prioritise total deviation	Prioritise total duration	Meetways	What's Halfway
[38, 51]	[52, 52]	[45, 50]	[52, 52]	[52, 52]	[93, 53]	[36, 93]
[30, 59]	[51, 52]	[45, 50]	[51, 52]	[51, 52]	[80, 45]	[53, 68]
[23, 68]	[52, 51]	[50, 46]	[52, 51]	[52, 51]	[65, 88]	[68, 70]
[65, 27]	[51, 52]	[41, 51]	[51, 52]	[51, 52]	[90, 48]	[75, 53]
[65, 27]	[54, 53]	[52, 47]	[54, 53]	[54, 53]	[60, 63]	[60, 75]
[41, 51]	[52, 54]	[51, 52]	[52, 54]	[52, 54]	[43, 98]	[58, 70]
[41, 52]	[58, 56]	[52, 51]	[58, 56]	[50, 46]	[68, 88]	[83, 45]
[55, 38]	[58, 60]	[51, 52]	[58, 60]	[45, 50]	[53, 68]	[65, 68]
[28, 66]	[64, 66]	[52, 52]	[64, 66]	[45, 50]	[73, 63]	[30, 88]
[59, 35]	[57, 54]	[53, 43]	[57, 54]	[58, 56]	[43, 98]	[75, 58]

8.3 Case Study 3

Case study 3 looks at four users who are travelling using different modes of transport by walking from University College London, taking public transport from Imperial College London, walking from King's College London, and taking public transport from City, University of London. This case allows us to look at some of the unique features of the application which are not offered by other systems such as different modes of transport and an arrival time. There is no comparison to existing systems' results as this would not be fair.

Most of the results returned are quite similar in each column and it is difficult to say which location is the best for the users specifically. Some factors which may be able to help them decide are the ratings displayed by the UI, or the individual circumstances of each member of the group. For example, they may choose location [9, 28, 20, 23] or perhaps location [22, 18, 12, 26] because they know person 1 and person 3 are travelling by foot and these journeys are shorter for them.

Table 8.3: Results for case study 3.

Final order of destinations				
Minimise total duration	Most similar	Reasonably similar	Prioritise total deviation	Prioritise total duration
[2, 27, 29, 18]	[21, 24, 33, 27]	[22, 18, 12, 26]	[21, 24, 33, 27]	[22, 18, 12, 26]
[22, 18, 12, 26]	[22, 18, 12, 26]	[11, 27, 21, 23]	[22, 18, 12, 26]	[11, 27, 21, 23]
[4, 28, 25, 22]	[11, 27, 21, 23]	[21, 24, 33, 27]	[11, 27, 21, 23]	[21, 24, 33, 27]
[9, 28, 20, 23]	[25, 19, 12, 27]	[27, 26, 7, 23]	[25, 19, 12, 27]	[25, 19, 12, 27]
[11, 27, 21, 23]	[25, 19, 12, 27]	[10, 27, 22, 24]	[25, 19, 12, 27]	[25, 19, 12, 27]
[2, 28, 26, 26]	[26, 20, 13, 28]	[25, 19, 12, 27]	[26, 20, 13, 28]	[9, 28, 20, 23]
[25, 19, 12, 27]	[10, 27, 22, 24]	[25, 19, 12, 27]	[10, 27, 22, 24]	[10, 27, 22, 24]
[25, 19, 12, 27]	[21, 33, 40, 33]	[9, 28, 20, 23]	[9, 28, 20, 23]	[26, 20, 13, 28]
[10, 27, 22, 24]	[9, 28, 20, 23]	[26, 20, 13, 28]	[21, 33, 40, 33]	[27, 20, 14, 30]
[27, 26, 7, 23]	[27, 20, 14, 30]	[27, 20, 14, 30]	[27, 20, 14, 30]	[27, 26, 7, 23]

Ultimately, we can see that different algorithms seem to work best for different cases and that other UI features like ratings can help users make their decision. This highlights the benefit of this application which lets the user explore the options available to find their sweet spot.

9 Evaluation

To begin with, we can see in Table 9.1 an updated version of Table 3.1 with an added column for the software produced in this project. The features available show the success of this application against other existing systems. The only unavailable feature is 'equal travel distance' and not including this was a conscious decision because existing applications already present results for this feature and journey durations were the main focus of this project.

Table 9.1: Features available in existing systems compared to 'Sweet Spot'.

Key		Roudle	Geo Midpoint	What's Halfway	Meetways	A Place Between Us	Where To Meetup	Shall We Meet in the Middle	Sweet Spot
✓✓	Feature implemented well								
✓	Feature implemented								
	Feature unavailable								
aesthetically pleasing		✓		✓	✓✓		✓✓	✓✓	✓✓
different modes of transport		✓		✓	✓			✓	✓✓
equal travel time				✓					✓✓
equal travel distance		✓	✓✓	✓✓	✓✓		✓	✓✓	
arrival/departure time									✓✓
destination ratings		✓	✓✓	✓✓	✓✓		✓✓		✓✓
directions			✓	✓	✓✓	✓		✓	✓✓
can be used in London			✓✓	✓✓	✓✓	✓✓	✓	✓	✓✓
group size greater than two		✓✓	✓✓	✓✓		✓✓			✓✓
responsive display		✓			✓✓	✓	✓	✓	✓✓
destination type can be chosen		✓	✓✓	✓✓	✓✓	✓✓	✓✓		✓✓

9.1 Specification Evaluation

All Primary and Secondary Requirements were achieved, as well as some of the extensions. Although most of this has been shown previously in the report, this evaluation of the specification will take each requirement and discuss briefly how it was achieved.

9.1.1 Primary Requirements

FR1: The User Interface should allow up to four starting addresses to be entered.

This was achieved by allowing users to add or remove group members until there were at most four or at least one.

FR2: The User Interface should allow a mode of transport for each person to be selected.

Each user is able to select a mode of transport from driving, walking, cycling, or using public transport. When they are driving, they may also choose to avoid highways or toll roads.

FR3: The User Interface should allow a type of destination to be chosen.

Users enter a keyword or point of interest which is passed to the Google Places API to find places that match it.

FR4: The User Interface should allow either arrival or departure time to be selected.

If they wish to, users are able to use the calendar to select the date and time they would like the group to meet at. This information is given to both the Google Maps Distance Matrix and the Google Maps Directions APIs for accuracy. Google Maps uses its scheduling and predicted traffic knowledge to produce results.

FR5: Users should be able to minimise their total time travelled.

This option is available to the user and once selected applies the appropriate algorithm on their data.

FR6: Users should be able to minimise their total distance travelled.

This option is also available in the dropdown menu on the search screen.

FR7: Users should be able to find the fairest location with respect to the duration of their journeys.

Users can achieve this by trying one of the many algorithms available: 'most similar durations'; 'reasonably similar durations'; or one of the two fairness functions.

FR8: The suggested destinations should be shown in relation to the starting addresses on a map to visualise the journey for each user.

Using polylines and map markers, the users can see the journeys of the whole group to a location illustrated simultaneously on the map in their assigned colours. They can also look more closely at the journey of a single user by selecting the appropriate button from the sidebar.

9.1.2 Secondary Requirements

FR9: Include ratings of the suggested meeting points.

These are shown as pink hearts when available from the Google Places search results. The actual value of the rating is also displayed.

FR10: Rank the suggested locations.

The locations are typically ranked based on a value assigned to them in the algorithm. For the minimum total 'costs', they are obviously ranked by total 'cost'. This includes the minimum total durations, distances and deviations. The reasonably similar durations are also ordered and then reordered according to the algorithm which is the final ranking shown to the user. Finally, the fairness algorithms assign each location a score which is used to rank them.

FR11: Show directions for each person travelling.

Directions are displayed to the user in a list once they have clicked the button for their specific journey. They can also view this directly on Google Maps. When they are using the web application on their phone, they can only view the directions on Google Maps and it should open in the application if they have it downloaded.

NFR1: The User Interface should adapt and respond well to different screen resolutions.

This was achieved through the use of Semantic UI which handles most of the switching between various screen resolutions. When creating the mobile version of the application, the choices available were to either create a separate mobile site, or to use CSS Media Queries to adjust elements based on screen size. CSS Media Queries were chosen to further customise the layout and ensure the best possible user experience. This allows the screen to be resized without any user data being lost or the user's process being interrupted.

As explained previously, the main differences on the mobile version are: no huge map showing all the locations in the results list; smaller maps showing user journeys within the sidebar; and directions open in the Google Maps application.

NFR2: The system status should always be apparent to the user so they know what is going on.

The only time the user has to wait for results is when the 'submit' button is initially clicked. While the data is collected and processed, the user sees a 'loading' animation so that they know results will be available soon. If results cannot be processed or displayed because of missing or incorrect data, error messages appear explaining the situation. This can range from a missing address to unavailable directions.

NFR3: The system should speak the user's language by using familiar words and concepts which appear in a natural order.

This was achieved by using language such as 'most similar durations' rather than 'smallest total deviation' when describing the algorithms to the user. The icons and buttons used are all as intuitive as possible using icons to help. For example, the back button which is a left facing arrow is also the same back button used to exit a Netflix video.

The system follows a similar order of events to other existing systems as well as popular applications such as Google Maps. That is, making a search, seeing a list of results, and being able to click each result.

NFR4: The system should provide the user with the control and freedom to undo and redo actions.

Back buttons are available on every screen. If a user gets search results and it turns out that had made a typing error or chose the wrong mode of transport, they can press 'back' to return to the search screen, where their information is still available and can be changed easily. On the selected location screens, pressing 'back' takes the user back to the original list of results.

NFR5: The system language, layout, and actions should be consistent throughout.

Clicking each result gives the user the same information about each location in the same position and the map updates consistently with the correct information. Throughout the web application, results are always shown in the sidebar on the left and the map information

is always displayed on the right hand side.

NFR6: The system should be able to prevent and recover from errors.

Form validation makes it easy to prevent future errors. Any missing information is requested from the user before a search takes place. User validation also prevents errors by checking a user's address is real. If it is not, an error message appears which must be fixed before the system will accept the input. Furthermore, when distances or durations are unavailable, the place is disregarded and when directions are unavailable, a message explaining this is displayed to the user.

NFR7: The user should be able to recover from errors easily.

Again, clear error messages help the user fix input errors such as entering invalid points of interest or invalid addresses.

NFR8: Users should be able to navigate the system by recognition of objects and options without relying on memory.

On every screen of the web application, options available to the user are made very clear and can be recognised easily. All clickable features have feedback when hovered over, such as button colours changing or extra information being made visible in a window over a marker.

NFR9: The system should be flexible and efficient to use for both the expert and novice user.

The novice user can navigate the website easily and quickly through placeholder text, such as 'Person 1's address' or general intuition such as icons showing different modes of transport on the buttons.

Because the web application is so simple to navigate, expert users can do so quickly without needing to stop to read anything. They can also use the 'tab' key on their keyboard and the 'up' and 'down' arrow keys to navigate the initial search page faster.

NFR10: The User Interface should have aesthetic and minimalist design which considers the display of information carefully.

Every piece of information displayed on the web application is done so for a reason and has been well-considered. The colours used and the layout are simplistic and easy to follow. On mobile, any unnecessary information is removed, such as the map displaying all locations

at once, to keep the user experience smooth and pleasant.

9.1.3 Possible Extensions

One extension was completed:

E1: The User Interface could produce a link (to Google Maps) for each traveller's directions.

This was achieved and can be seen on both the desktop and mobile versions of the application. When the user clicks it, the starting address, destination and mode of transport are automatically filled in on the Google Maps website or application for them to browse through easily.

9.2 Ethical, Legal, Professional, and Social Issues Evaluation

At the start of the project, a number of ethical, legal, professional, and social issues were highlighted for consideration throughout the project.

9.2.1 Ethical

During User Acceptance Testing, all participants were aware in advance that the information they provided was also used by the Google Maps APIs. The participants were all colleagues and friends so no formal consent was required.

The only pieces of personal information requested from the user are starting locations, modes of transport, and (optionally) arrival time. All of these are justified as they are required for the algorithms used by the back end of the software and none of this data is stored.

9.2.2 Legal

Whilst using Google Maps APIs, usage limits were respected throughout the project. During the initial phases and testing of the project, responses from requests to the Google Places API were stored in line with the terms and conditions for no longer than 30 days at a time [62]. Other considerations included only displaying Google Places results on a Google Map and

incorporating a "Powered by Google" logo [63] where a Google Map is not present, although this logo is visible on all pages apart from the Welcome screen.

The use of the Google Directions, Distance Matrix, and Geocoding APIs is prohibited without displaying the results on a Google Map which was upheld in any relevant places. A further requirement of the Directions API is that attributions must be stated for third-party content, meaning the application "must display the names and URLs of the transit agencies that supply the trip results" [64].

If this app were released to the public, a further requirement would be to make available a Terms of Use and a Privacy Policy [65] which explicitly state that users are bound by Google's Terms of Service [66] and notifies the users which APIs are used with reference to the Google Privacy Policy [67].

The other frameworks and libraries used such as Semantic UI [68] and Moment.js [53] were freely distributable under the terms of the MIT license [69].

9.2.3 Professional

A respectful and honest approach was taken to analyse existing systems which is clearly split into sections to accurately identify the positives and negatives of each one. Throughout the project, in every assessed part, citations and references have been provided. On top of this, the code of conduct of the British Computing Society was upheld to the best of the developer's ability.

The design of the produced software was achieved by making thoughtful choices and provides a clean, extendible base for further development. For example, although the number of places used each time was limited to a maximum of 200, the code works for any number of places. The same can be said for the number of people. The only shortcoming is one which has been acknowledged since the start of the project; the limitations with places data.

Documentation is provided in the form of this final report; a README; commented code; and a requirements.txt file containing project dependencies.

9.2.4 Social

Alongside an initial exploration of existing software to find helpful features and a clear layout, User Acceptance Testing was used to further cater to the opinions of the target audience. Responses from User Acceptance Testing were implemented and improvements were made as much as possible given the time constraints.

As discussed previously, collected data can be justified to the user and its use by both the application and Google can be explained. There is no guaranteed level of optimality of results because the project has acknowledged that it is limited by constraints such as the results returned by the Google Places API, and has expressed clearly that the intention is to filter out bad options and recommend optimal and desirable results where the definition of optimal is clearly defined by functions.

A further issue which became apparent when selecting APIs was considering the bad reputation of Yelp. To avoid this, Yelp was not chosen as an API for this project.

9.3 Project Management Evaluation

9.3.1 Original Plan and Deviations

Meeting all requirements from the specification and beginning work on extensions is a good indication that this was a successful and well-managed project. It was also completed to a high standard which is clear from the results of testing, especially User Acceptance Testing.

To summarise, the majority of term one was spent researching the problem and existing solutions, and researching and choosing the technologies required to develop the software. Term two focused more on applying these technologies and completing the development.

Although the final schedule strayed slightly from the one outlined in the Gantt chart in Table 5.1, the choice of using an adapted Agile methodology meant that development was still well-structured and completed in time. This was aided by regular meetings with the project supervisor.

9.3.2 Tools

To keep track of tasks that needed to be completed in each sprint and an overall view of the project, Trello [70] was an invaluable tool. An example of a board used in the project can be seen in Figure 9.1. Toby [71], a tab management system was also used to organise research and useful information.

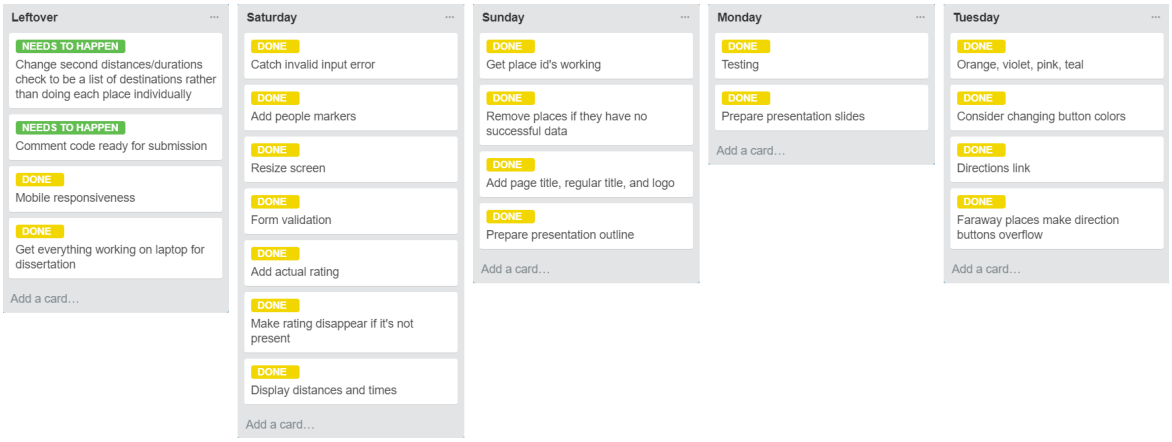


Figure 9.1: Example Trello Board.

For version control and backing up work, Git, Github [72], Google Docs [73], and Overleaf [74] were used. These proved to be invaluable on a number of occasions such as when the developer's laptop broke and when the developer accidentally deleted one of the project directories.

9.3.3 Hindsight and Lessons Learnt

Although the project was completed in time, more would have been achieved had development started sooner. Less time could have been spent trying to work out the best possible technologies to use and learning how to use Node.js, and instead could have been spent implementing the Google Places Nearby Search rather than the Radar Search. This would have potentially returned more data for the algorithms as well as avoiding the Radar Search deprecation notice. It would also have been better to take more advantage of mock and automated testing from the start of the project.

Despite this, the most valuable lesson learnt was how to manage time effectively and overcome difficulties with software development. Towards the start of the project, this was a

huge struggle. But as development progressed, it became easier to organise tasks and learn new things which built up confidence in the developer.

9.4 Author's Assessment of the Project

9.4.1 What is the technical contribution of this project? Why should it be considered relevant and important for the subject of your degree?

The system produced is a proof-of-concept solution for the real world problem of finding fair meeting points. Using existing papers and systems with similar aims, shortcomings in the options available were identified and solved appropriately through the implementation of six algorithms, four of which were not previously seen in available research.

Furthermore, drawbacks of the user experience of existing systems were highlighted and the User Interface produced solved these. This included careful design and thorough testing to avoid glitches, as well as a good use of screen space on a wide range of screen resolutions. The option for users to adjust their modes of transport and method of 'optimisation' provides them with the unique experience of interacting with different places based on different parameters.

As a Discrete Mathematician, this project has developed and tested many of the skills required on both sides of the degree. Specifically, the algorithms required Mathematical thinking and the development of the software required use of Computer Science skills. Relevant to both degrees was the project management and research aspects of the experience. All these skills were improved upon throughout the project and existing knowledge from modules such as Social Informatics was drawn from.

9.4.2 How can others make use of the work in this project?

The most obvious way that this project can be used is in the daily lives of people looking for places to meet in Greater London. Other ways include making the API created public so that people can use it to find data in their applications or embed the results in a similar way to the service offered by What's Halfway [75].

The work done in this project is a good starting point for people who want to research the topic further and experiment with new algorithms. The report also compiles helpful

references to existing systems and papers which are not immediately easy to find.

9.4.3 Why should this project be considered an achievement?

We can see throughout this report that there are a number of different elements which have come together as one successfully. This required the developer to apply themselves across a wide range of areas. Again, the project achieved all its requirements and some of the extensions laid out at the start. As seen by the results, it is objectively better than existing systems in many ways, although it has some limitations and shortcomings which are discussed in the next question.

9.4.4 What are the limitations of this project?

One of the biggest limitations of this project is the use of the Google Maps APIs, especially the Distance Matrix, because of the quantity of requests required for the successful implementation of the algorithms. Interestingly, one of the developers of Roudle said that the reason they did not use Google Maps APIs for calculating distances and durations was also due to the quantity of requests that would need to be made. Moreover, other existing systems ran out of API requests while they were being tested, implying that this is quite a common issue to encounter. Addressing this would remove the upper limit of four people per group which has been imposed on the application.

The application is not unreasonably slow, however it has the potential to be much quicker if the dependency on the Google Maps Distance Matrix is removed, or at least reduced. Some solutions to these problems are discussed in Future Work. Although not a huge issue, the Distance Matrix can only handle one parameter to avoid (either highways or tolls) which means that when both are selected, the results may be slightly inaccurate.

The other significant limitation is that the software only considers up to 200 places at once which is only a fraction of the number of destinations in Greater London, limiting the accuracy of the results to what the Google Places API returns.

10 | Conclusion

10.1 Summary

We have seen throughout this report the research, design, implementation and testing of a web application which finds fair meeting points for a group of people in Greater London. Although this project was successful in many ways, there are some limitations which could be solved. Along with these, the next section outlines future work to undertake to use the work provided by this project to its full advantage.

10.2 Future Work

Improving the speed of the web application and overcoming the API free limits would be a good next step. One of the ways of overcoming these limits would be to pay for a higher number of requests. Another could be to find a new way to implement the solution such as using free, open-source alternatives like OpenStreetMaps [76] and storing open Transport for London data [77].

Whilst this would increase the number of times the application can be used without running out of API requests, it might also reduce the accuracy of the application as Google Maps is well known for having real-time traffic data and up-to-date public transport data with over 17,000 different routes from the schedules of almost 1,500 different transport operators [78]. Without processing data manually to find public transport routes, another possible (but premium) solution would be to use an API such as the one provided by HERE WeGo Maps [79] whose application is one of the few which achieves route planning without access to the internet.

In terms of the speed, changing the language from Python or changing the implementation of vectors from lists to NumPy arrays has the potential to increase the speed [80]. This is because although Python is well-known for being a fast language to develop in, its performance is relatively poor.

Looking at the functionality of the web application, there are some further considerations and features which could be added. One of these is specifying the exact type of public

transport requested by the user, such as specifically only the bus, or specifically only the underground.

During User Acceptance Testing, a participant pointed out that when a friend has a car, they often volunteer to pick some members of the group up on the way to the meeting point. To accommodate for this, a 'waypoints' option could be added which will allow pickups to be specified.

Lastly, two small but helpful features that could be implemented are the presence of a progress bar so that the user can see how many places are left to decide between and the ability to load more results after the top ten places are shown.

Since a web application of this nature is likely to be used on the go, future work could Android and Apple mobile applications for the User Interface. For both this web application and any mobile applications, if they were to be released for general use, it will be necessary to include some further details for users such as a privacy policy; a terms and conditions page; and a description of each algorithm to enhance the user experience.

When considering the actual algorithms created, there was not much research done on defining 'fairness' so this is a good start, but future work on this problem might consider human definitions of fairness. For example, the fact that a 10 minute increase in journey time to save someone else more than 10 minutes is perfectly reasonable, however a 20 minute increase in journey time is, for some people, unacceptable. Or perhaps making adjustments based on how much effort is required for a journey, such as the difference between someone cycling and someone driving.

This exploration might also consider the extensions which were not achieved, such as changing the cost parameters to the price of journeys, or indeed a combination of parameters such as duration and price.

References

[1] Da Yan, Zhou Zhao, Wilfred Ng (The Hong Kong University of Science and Technology). *Efficient Algorithms for Finding Optimal Meeting Point on Road Networks*. Tech. rep. 11. Sept. 2011, pp. 968–979. URL: https://www.cse.ust.hk/~wilfred/paper/vldb11.pdf (Accessed: Apr. 23, 2018).

[2] Shivendra Tiwari and Saroj Kaushik (Department of Computer Science and Engineering, IIT Delhi). *Scalable Method for k Optimal Meeting Points (k-OMP) Computation in the Road Network Databases*. Tech. rep. 2013, pp. 277–292. URL: https://link.springer.com/chapter/10.1007%2F978-3-642-37134-9_21 (Accessed: Apr. 23, 2018).

[3] Wikipedia. *Greater London*. URL: https://en.wikipedia.org/wiki/Greater_London (Accessed: Dec. 04, 2017).

[4] John Elledge, Centre for Cities, New Statesman. *Where are the largest cities in Britain?* Sept. 16, 2015. URL: http://www.citymetric.com/skylines/where-are-largest-cities-britain-1404 (Accessed: Dec. 04, 2017).

[5] Food Standards Agency. *Food Hygiene Ratings - restaurants, takeaways or food shops*. URL: http://ratings.food.gov.uk/search-a-local-authority-area/en-GB/London (Accessed: Dec. 04, 2017).

[6] Food Standards Agency. *Food Hygiene Ratings - pub/bar/nightclub*. URL: http://ratings.food.gov.uk/enhanced-search/en-GB/%5E/London/Relevance/7843/%5E/%5E/0/1/10 (Accessed: Dec. 04, 2017).

[7] Centre for Cities, Office for National Statistics. *Commuting by Bus, Train or Metro 2011*. 2011. URL: http://www.centreforcities.org/data-tool/#graph=bar&city=show-all&indicator=commuting-by-bus-train-or-metro%5C%5Csingle%5C%5C2011&sortOrder=high&tableOrder=tableOrder%5C%5C1%7B%7D,1 (Accessed: Apr. 16, 2018).

[8] Transport for London. *Improving Buses*. URL: https://tfl.gov.uk/modes/buses/improving-buses (Accessed: Dec. 04, 2017).

[9] Transport for London. *Countdown to Launch*. June 4, 2017. URL: https://tfl.gov.uk/travel-information/improvements-and-projects/countdown-to-launch (Accessed: Dec. 04, 2017).

[10] Will Noble, Londonist. *4G Mobile Coverage on the Tube from 2019*. Nov. 27, 2017. URL: https://londonist.com/london/transport/4g-mobile-coverage-on-the-tube-from-2019 (Accessed: Dec. 04, 2017).

[11] Transport for London. *Record-breaking year for Santander Cycles*. Feb. 13, 2017. URL: https://tfl.gov.uk/info-for/media/press-releases/2017/february/record-breaking-year-for-santander-cycles (Accessed: Apr. 16, 2018).

[12] Transport for London. *How many cars are there in London and who owns them?* July 14, 2013. URL: http://content.tfl.gov.uk/technical-note-12-how-many-cars-are-there-in-london.pdf (Accessed: Apr. 16, 2018).

[13] Christiaan Rasch and Thijs Brentjens. *Vind de beste plek voor je afspraak met locatieplanner Roudle*. June 2, 2015. URL: https://www.roudle.com/ (Accessed: Dec. 04, 2017).

[14] Christiaan Rasch & Thijs Brentjes, Roudle.com, MobilityLabel. *Press Release Launch Roudle*. Apr. 29, 2015. URL: https://mobilitylabel.nl/press-release-launch-roudle/ (Accessed: Dec. 04, 2017).

[15] Kristen McCaffrey, Meetways.com. *Company Overview*. Jan. 18, 2017. URL: https://www.meetways.com/about (Accessed: Dec. 04, 2017).

[16] Office for National Statistics. *Use of the internet to arrange transport from another individual, by age group, 2017, Great Britain*. 2017. URL: https://www.ons.gov.uk/peoplepopulationandcommunity/householdcharacteristics/homeinternetandsocialmediausage/bulletins/internetaccesshouseholdsandindividuals/2017 (Accessed: Dec. 04, 2017).

[17] Department of Computer Science, University of Warwick. *Ethical Consent for Undergraduate Projects*. URL: https://warwick.ac.uk/fac/sci/dcs/teaching/material/cs310/ethics/ (Accessed: Apr. 16, 2018).

[18] The British Computing Society [GB]. *BCS Code of Conduct*. May 5, 2016. URL: http://www.bcs.org/category/6030 (Accessed: Apr. 16, 2018).

[19] Christiaan Rasch and Thijs Brentjens. *Roudle, the meeting location planner and optimizer*. Feb. 4, 2015. URL: https://www.roudle.com/meeting-location-planner-and-optimizer/ (Accessed: Dec. 04, 2017).

[20] *Geo Midpoint, Let's Meet in the Middle*. Jan. 18, 2010. URL: http://www.geomidpoint.com/meet/ (Accessed: Dec. 04, 2017).

[21] *What's Halfway? Find great places to meet or stop halfway between...* July 2, 2014. URL: http://www.whatshalfway.com/ (Accessed: Dec. 04, 2017).

[22] *MeetWays: Meet in the Middle - Find a Halfway Point!* Jan. 19, 2017. URL: https://www.meetways.com/ (Accessed: Dec. 04, 2017).

[23] Aron Atkins and Matthew Bellantoni. *A Place Between us.* 2007. URL: http://a.placebetween.us/ (Accessed: Dec. 04, 2017).

[24] *Where to Meet Up?* Dec. 14, 2009. URL: http://www.wheretomeetup.com/ (Accessed: Dec. 04, 2017).

[25] Emma Sax. *Let's Meet in the Middle.* Oct. 23, 2014. URL: http://www.shallwemeetinthemiddle.com/ (Accessed: Dec. 04, 2017).

[26] Emma Sax. *Finding meeting places between two points.* Nov. 11, 2014. URL: https://github.com/emmasax/meetinthemiddle (Accessed: Apr. 16, 2018).

[27] David Thorpe, Sustainable Cities Collective, World Economic Forum. *In which cities is it quicker to walk and cycle than drive a car?* May 27, 2015. URL: https://www.weforum.org/agenda/2015/05/in-which-cities-is-it-quicker-to-walk-and-cycle-than-drive-a-car (Accessed: Apr. 16, 2018).

[28] Jakob Nielsen. *10 Usability Heuristics for User Interface Design.* Jan. 1, 1995. URL: https://www.nngroup.com/articles/ten-usability-heuristics/ (Accessed: Apr. 16, 2018).

[29] Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas. *Manifesto for Agile Software Development.* 2001. URL: http://agilemanifesto.org/ (Accessed: Apr. 16, 2018).

[30] Lea Karam, Apiumhub. *FDD; ITS PROCESSES & COMPARISON TO OTHER AGILE METHODOLOGIES.* May 18, 2017. URL: https://apiumhub.com/tech-blog-barcelona/feature-driven-development (Accessed: Apr. 16, 2018).

[31] Mountain Goat Software. *User Stories.* Jan. 11, 2010. URL: https://www.mountaingoatsoftware.com/agile/user-stories (Accessed: Apr. 16, 2018).

[32] API Evangelist. *Overview of 11 Places Data APIs.* Mar. 4, 2012. URL: https://apievangelist.com/2012/03/04/overview-of-11-places-data-apis/ (Accessed: Dec. 04, 2017).

[33] Post Staff Report, New York Post. *Yelp admits smaller businesses are fed up with its site.* May 10, 2017. URL: https://nypost.com/2017/05/10/yelp-admits-smaller-businesses-are-fed-up-with-its-site/ (Accessed: Dec. 04, 2017).

[34] Sophia Harris, CBC News. *Yelp accused of bullying businesses into paying for better reviews.* Jan. 14, 2015. URL: http://www.cbc.ca/news/business/yelp-accused-of-bullying-businesses-into-paying-for-better-reviews-1.2899308 (Accessed: Dec. 04, 2017).

[35] Janet Wagner, ProgrammableWeb. *Top 10 Mapping APIs.* Feb. 23, 2015. URL: https://www.programmableweb.com/news/top-10-mapping-apis-google-maps-microsoft-bing-maps-and-mapquest/analysis/2015/02/23?utm_source=self&utm_medium=paralinks&utm_campaign=related-content (Accessed: Dec. 04, 2017).

[36] Google Places API. *Places API Policies.* 2018. URL: https://developers.google.com/places/web-service/policies (Accessed: Apr. 16, 2018).

[37] James Peckham, Techradar. *TfL will never make the ultimate travel app - but there is a very good reason.* Sept. 14, 2015. URL: http://www.techradar.com/news/phone-and-communications/mobile-phones/tfl-will-never-make-the-ultimate-travel-app-but-there-is-a-very-good-reason-1304247 (Accessed: Dec. 04, 2017).

[38] Matthew Sparkes, The Telegraph. *Google Maps adds every train, bus and ferry in UK.* May 14, 2014. URL: https://www.telegraph.co.uk/technology/google/10795012/Google-Maps-adds-every-train-bus-and-ferry-in-UK.html (Accessed: Apr. 16, 2018).

[39] Ravi Sharma, Gadgets360. *How Google Maps Gets Its Remarkably Accurate Real-Time Traffic Data.* Mar. 2, 2017. URL: https://gadgets.ndtv.com/apps/features/how-google-maps-gets-its-remarkably-accurate-real-time-traffic-data-1665385 (Accessed: Apr. 16, 2018).

[40] SimilarWeb. *Free Report on maps.google.com.* Apr. 16, 2018. URL: https://www.similarweb.com/website/maps.google.com (Accessed: Apr. 16, 2018).

[41] SimilarWeb. *Free Report on openstreetmap.org.* Apr. 16, 2018. URL: https://www.similarweb.com/website/openstreetmap.org (Accessed: Apr. 16, 2018).

[42] Google Maps JavaScript API. *Adding a Google Map with a Marker to Your Website*. July 17, 2016. URL: https://developers.google.com/maps/documentation/javascript/adding-a-google-map (Accessed: Apr. 16, 2018).

[43] Google Maps APIs. *Client Libraries for Google Maps Web Services*. URL: https://developers.google.com/maps/web-services/client-library (Accessed: Apr. 16, 2018).

[44] Nick Diakopoulos and Stephen Cass, IEEE Spectrum. *Interactive: The Top Programming Languages 2017*. July 18, 2017. URL: https://spectrum.ieee.org/static/interactive-the-top-programming-languages-2017 (Accessed: Apr. 16, 2018).

[45] Python 3. *unittest — Unit testing framework*. Apr. 26, 2010. URL: https://docs.python.org/3/library/unittest.html (Accessed: Apr. 16, 2018).

[46] Gareth Dwyer, Codementor. *Flask vs. Django: Why Flask Might Be Better*. Feb. 13, 2017. URL: https://www.codementor.io/garethdwyer/flask-vs-django-why-flask-might-be-better-4xs7mdf8v (Accessed: Apr. 16, 2018).

[47] Scott Robinson, Stack Abuse. *Flask vs Django*. Sept. 21, 2017. URL: http://stackabuse.com/flask-vs-django/ (Accessed: Apr. 16, 2018).

[48] AWWWARDS. *What are Frameworks? 22 Best Responsive CSS Frameworks for Web Design*. Feb. 20, 2013. URL: https://www.awwwards.com/what-are-frameworks-22-best-responsive-css-frameworks-for-web-design.html (Accessed: Apr. 16, 2018).

[49] *Bootstrap*. Aug. 11, 2017. URL: http://getbootstrap.com/docs/3.3/ (Accessed: Apr. 16, 2018).

[50] Slant. *Bootstrap vs Semantic UI detailed comparison*. July 27, 2016. URL: https://www.slant.co/versus/504/520/~bootstrap_vs_semantic-ui (Accessed: Apr. 16, 2018).

[51] *Semantic UI*. URL: https://semantic-ui.com/ (Accessed: Apr. 16, 2018).

[52] *Semantic-UI-Calendar*. URL: https://www.npmjs.com/package/semantic-ui-calendar (Accessed: Apr. 16, 2018).

[53] *Moment.js*. URL: https://momentjs.com/ (Accessed: Apr. 16, 2018).

[54] Amos Ndegwa, MaxCDN. *What is a Web Application?* May 31, 2016. URL: https://www.maxcdn.com/one/visual-glossary/web-application/ (Accessed: Apr. 16, 2018).

[55] Segue Technologies. *What is Ajax and Where is it Used in Technology?* Mar. 12, 2013. URL: https://www.seguetech.com/ajax-technology/ (Accessed: Apr. 16, 2018).

[56] *jQuery*. URL: http://jquery.com/ (Accessed: Apr. 16, 2018).

[57] Google Places API. *Radar Search Requests (deprecated)*. URL: https://developers.google.com/places/web-service/search#RadarSearchRequests (Accessed: Apr. 16, 2018).

[58] The London Helicopter. *The Exact Centre of London*. Aug. 28, 2015. URL: https://www.thelondonhelicopter.com/interactive-map-of-london-tourist-attractions/the-exact-centre-of-london/ (Accessed: Apr. 16, 2018).

[59] LaTonya Pearson, Segue Technologies. *The Four Levels of Software Testing*. Sept. 11, 2015. URL: https://www.seguetech.com/the-four-levels-of-software-testing/ (Accessed: Apr. 16, 2018).

[60] Drew Wells, SmartBear. *Unit Testing vs UI Testing*. Oct. 9, 2005. URL: https://blog.smartbear.com/testing/unit-testing-vs-ui-testing/ (Accessed: Apr. 16, 2018).

[61] Doogal. *Random Postcode Generator*. Apr. 24, 2016. URL: https://www.doogal.co.uk/PostcodeGenerator.php (Accessed: Apr. 16, 2018).

[62] Google Maps APIs Terms of Service. *Intellectual Property Restrictions*. Feb. 7, 2018. URL: https://developers.google.com/maps/terms#section_10_5 (Accessed: Apr. 16, 2018).

[63] Google Places API. *Places API Policies*. URL: https://developers.google.com/places/web-service/policies (Accessed: Apr. 16, 2018).

[64] Directions API. *Policies*. URL: https://developers.google.com/maps/documentation/directions/policies (Accessed: Apr. 16, 2018).

[65] Google Maps APIs Terms of Service. *End User Terms and Privacy Policy*. Feb. 7, 2018. URL: https://developers.google.com/maps/terms#section_9_3 (Accessed: Apr. 16, 2018).

[66] Google. *Google Terms of Service*. Oct. 25, 2017. URL: https://www.google.com/intl/en/policies/terms (Accessed: Apr. 16, 2018).

[67] Google. *Welcome to the Google Privacy Policy*. Dec. 18, 2017. URL: https://www.google.com/policies/privacy (Accessed: Apr. 16, 2018).

[68] Massachusetts Institute of Technology. *The MIT License (MIT)*. URL: https://semantic-ui.mit-license.org/ (Accessed: Apr. 16, 2018).

[69] Massachusetts Institute of Technology, Open Source Initiative. *The MIT License*. Aug. 15, 2015. URL: https://opensource.org/licenses/MIT (Accessed: Apr. 16, 2018).

[70] *Trello*. URL: https://trello.com/.

[71] *Toby for Chrome*. URL: https://chrome.google.com/webstore/detail/toby-for-chrome/hddnkoipeenegfoeaoibdmnaalmgkpip?hl=en (Accessed: Apr. 16, 2018).

[72] *Sweet Spot on Github*. URL: https://github.com/the-pigeonhole-principal/sweetspot.

[73] *Google Docs*. URL: https://docs.google.com/.

[74] *Overleaf*. URL: https://www.overleaf.com/.

[75] whatshalfway.com. *Whatshalfway API*. July 2, 2014. URL: http://www.whatshalfway.com/our-api.aspx (Accessed: Dec. 04, 2017).

[76] OpenStreetMap Wiki. *About OpenStreetMap*. Mar. 22, 2017. URL: https://wiki.openstreetmap.org/wiki/About_OpenStreetMap (Accessed: Apr. 16, 2018).

[77] Transport for London. *Our open data*. URL: https://tfl.gov.uk/info-for/open-data-users/our-open-data?intcmp=3671 (Accessed: Apr. 16, 2018).

[78] Leo Kellon, BBC. *Google Maps adds Great Britain public transport data*. May 14, 2014. URL: http://www.bbc.co.uk/news/technology-27394691 (Accessed: Apr. 16, 2018).

[79] Pino Bonetti, HERE Maps. *Comparing Google Maps and HERE Maps offline*. Nov. 17, 2015. URL: https://360.here.com/2015/11/17/comparing-google-maps-and-here-maps-offline/ (Accessed: Dec. 04, 2017).

[80] University of Central Florida. *Python Lists vs. Numpy Arrays - What is the difference?* Dec. 21, 2017. URL: https://webcourses.ucf.edu/courses/1249560/pages/python-lists-vs-numpy-arrays-what-is-the-difference (Accessed: Apr. 16, 2018).

[81] Transport for London. *Stations, stops & piers*. Oct. 29, 2014. URL: https://tfl.gov.uk/travel-information/stations-stops-and-piers/ (Accessed: Dec. 04, 2017).

[82] Google Maps JavaScript API. *Autocomplete for Addresses and Search Terms//*. URL: https://developers.google.com/maps/documentation/javascript/places-autocomplete (Accessed: Apr. 16, 2018).

[83] Aastha Madaan, Shinji Kikuchi, and Subhash Bhalla, eds. *Databases in Networked Information Systems - 8th International Workshop, DNIS 2013, Aizu-Wakamatsu, Japan, March 25-27, 2013. Proceedings*. Vol. 7813. Lecture Notes in Computer Science. Springer, 2013. ISBN: 978-3-642-37133-2. URL: https://doi.org/10.1007/978-3-642-37134-9.

[84] I. Bilogrevic et al. "Privacy-Preserving Optimal Meeting Location Determination on Mobile Devices". In: *IEEE Transactions on Information Forensics and Security* 9.7 (July 2014), pp. 1141–1156. ISSN: 1556-6013. DOI: 10.1109/TIFS.2014.2318435. URL: https://ieeexplore.ieee.org/document/6802367/.

[85] Samta M. Jain (Rajiv Gandhi College Of Engineering Research & Technology). "Privacy-Preserving Optimal Meeting Location Determination on Mobile Devices". In: *International Journal on Recent and Innovation Trends in Computing and Communication* 4.5 (May 2016), pp. 48–51. ISSN: 2321-8169. URL: http://www.ijritcc.org/download/conferences/ICSTSD_2016/ICSTSD_2016_Track/1464421509_28-05-2016.pdf.

[86] Paulo Santos and Heather Vaughn. "Where Shall We Meet? Proposing Optimal Locations for Meetings". In: (). URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.619.1341&rep=rep1&type=pdf.

[87] Nemmara Chithambaram and Craig Allen Miller. "Meeting location determination using spatio-semantic modeling". In: (). URL: https://patents.google.com/patent/US6865538B2/en.