Faculty of Engineering and Physical Sciences

Department of Computing

COM3001: Professional Project

Developing an application for the Visitor Attraction industry to help maintain customer engagement

by Jack Burt

URN: 6418557

Supervisor: Professor Paul Krause

Declaration of Originality

“I confirm that the submitted work is my own work. No element has been previously submitted for assessment, or where it has, it has been correctly referenced. I have also clearly identified and fully acknowledged all material that is entitled to be attributed to others (whether published or unpublished) using the referencing system set out in the programme handbook.

I understand that the University may submit my work as a means of checking this, to the plagiarism detection service Turnitin® UK. I confirm that I understand that assessed work that has been shown to have been plagiarised will be penalised.

If in completing this work I have been assisted with its presentation by another person, I will state their name and contact details of the assistant in the ‘Comments’ text box below. In addition, if requested, I agree to submit the draft material that was completed solely by me prior to its presentational improvement.”

Acknowledgements

I would like to express my gratitude to Professor Paul Krause, for sharing with me his ideas and experiences along with showing me continued support and guidance throughout this project.

I would also like to thank my family, friends and partner for their continued support and encouragement throughout my studies.

Abstract

The Visitor Attraction industry within the United Kingdom has shown steady growth since 2013, with a reported +2% increase in total visits from 2017 to 2018 [1], however the use of mobile technology is still limited, with just 17% of attractions offering mobile applications, with the exception of historical attractions where this figure rises to 47% [1]. A 2018 survey [2] conducted at several art museums showed that 9 out of 10 respondents agreed that mobile applications made information easier to access with the vast majority believing they enhanced their overall experience. However in certain attractions or with certain demographics of customers visiting these attractions, the intrusiveness of a mobile application can often seem off-putting or distracting.

This exploratory project aims to develop an intelligent application that caters to as many demographics as possible, improving customer engagement while promoting repeat custom and avoiding intrusive design. It will tackle the opportunity of encouraging customers to turn a single visit into a hobby they can develop and nurture. The application will use the Royal Horticultural Society's Gardens at Wisley as a specific subject on which to explore these motivations, allowing users to create their own garden which harnesses the expertise of the RHS.

Table of Contents

Table of Figures

1 - Introduction

1.1 - Background

According to an annual consumer survey by Deloitte, the number of people within the UK who own or have access to a smartphone is now 87%, with this number continuing to grow each year [3]. Over 95% of those respondents between the ages of 16-75 used their device within the last day [4]. Clearly, smartphones are a huge part of nearly everyone's daily lives, which gives all sorts of businesses and services an enormous platform from which to interact with their consumers. One area that can benefit from this, are organisations that offer tours or walks for visitors to enjoy, such as gardens, museums, historic monuments, and other visitor attractions.

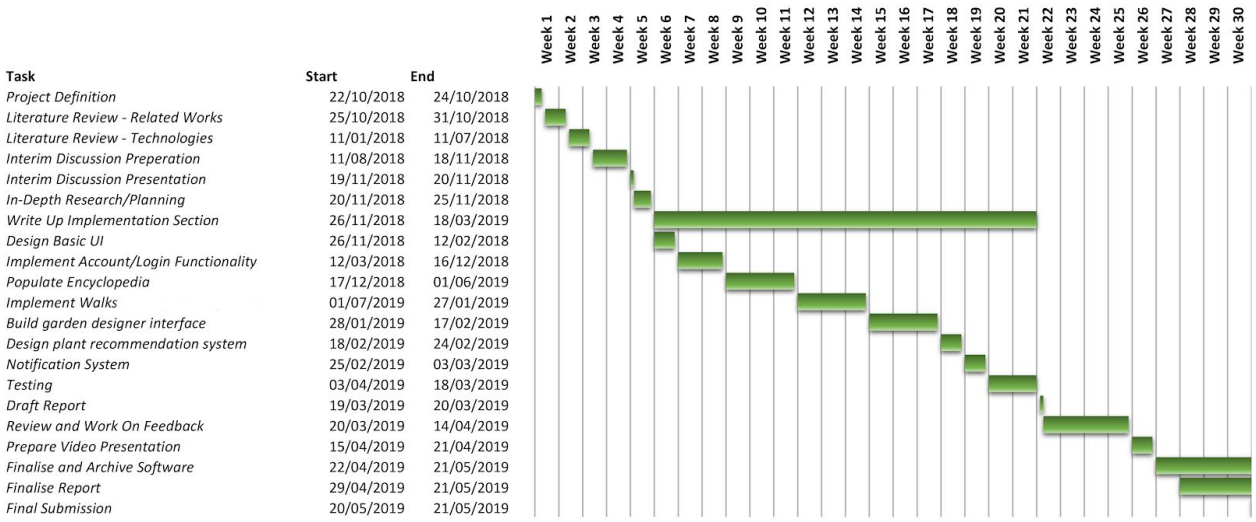
There are currently many different methods a business can use to provide an improved visitor experience. Tour guides, maps/guidebooks, audio handsets, and most recently, mobile applications. Tour guides are an expensive and difficult to organise solution for most businesses, and are very limited in the amount of people they can reach, and it only gets more difficult for attractions at a larger scale, such as national parks. Audio handsets offer few personal touches, with just a pre-recorded message the user can decide to listen to, or not. Maps and guidebooks, while useful, are very basic and limited in what they can offer a user. A mobile application can solve all of these problems. Mobile applications suffer from their own issues, with intrusiveness, privacy, ease-of-use and accessibility all things to be considered.

To avoid suffering from some of these problems, mobile visitor experience applications should be intelligent. They should tailor themselves to different users, with different experiences, knowledge and goals. This project will explore the various methods that will accomplish this goal.

1.2 - Aims and Objectives

- Create an application that is sophisticated but simple to use by all audiences, regardless of knowledge, experience or age
- Tailor the application to the user, making it as context-aware as is possible, leaving them feeling personally catered to, while not feeling overwhelmed with information they don't want or need
- Extend user engagement past leaving the attraction, promoting development into a hobby and encouraging repeat custom.
- Consider security within features of the application, especially surrounding the storage of data gathered about the user

Time management is a very important aspect of any project. However, the scope of this project is ambitious and it is vital that sufficient planning is carried out before development progresses. This will help to ensure that deadlines are met and functional sacrifices are minimised. A Gantt chart was used for this purpose and is shown below in figure 1.



Task	Start	End
Project Definition	22/10/2018	24/10/2018
Literature Review - Related Works	25/10/2018	31/10/2018
Literature Review - Technologies	11/01/2018	11/07/2018
Interim Discussion Preperation	11/08/2018	18/11/2018
Interim Discussion Presentation	19/11/2018	20/11/2018
In-Depth Research/Planning	20/11/2018	25/11/2018
Write Up Implementation Section	26/11/2018	18/03/2019
Design Basic UI	26/11/2018	12/02/2018
Implement Account/Login Functionality	12/03/2018	16/12/2018
Populate Encyclopedia	17/12/2018	01/06/2019
Implement Walks	01/07/2019	27/01/2019
Build garden designer interface	28/01/2019	17/02/2019
Design plant recommendation system	18/02/2019	24/02/2019
Notification System	25/02/2019	03/03/2019
Testing	03/04/2019	18/03/2019
Draft Report	19/03/2019	20/03/2019
Review and Work On Feedback	20/03/2019	14/04/2019
Prepare Video Presentation	15/04/2019	21/04/2019
Finalise and Archive Software	22/04/2019	21/05/2019
Finalise Report	29/04/2019	21/05/2019
Final Submission	20/05/2019	21/05/2019

Figure 1 - *Gantt chart to plan and track the project*

1.3 - RHS

While this project is of an exploratory nature that focuses on a general market of visitor experience applications, my application is designed specifically around the Royal Horticultural Society (RHS) gardens in Wisley. Wisley Gardens was in the Top 15 most visited paid attractions in 2015, with an estimated number of visitors exceeding 1 million [1]. Using one particular site or exhibit as a motivation allows for a more targeted approach that will produce more interesting results than trying to accomodate for an entire industry. The developments made in this project can be applied to a wider range of problems.

The RHS has worked with several students from the University and has been very cooperative and helpful. They have a current goal to try and get more customers involved with gardening at home and have a huge and diverse pool of knowledge that customers can utilise. I aim to make this knowledge more accessible to them through this project.

1.4 - Project Structure

I outline the aims of the various sections of this report below:

Section 1 - Introduction
Introduce the reader to the project, including the motivation behind it and what it hopes to achieve.

Section 2 - Literature Review
Analyse and evaluate related technologies and applications to gain insight on how certain aspects of this project could develop, to help avoid mistakes and issues that have arisen for others, whilst taking inspiration from their successes.

Section 3 - Methodologies and Architectures
Evaluate and make decisions on how to approach product development, including design methodologies, platforms and software choices.

Section 4 - System Requirements and Specifications
Identify and consolidate all functional and nonfunctional requirements to be considered when developing the product.

Section 5 - Technical Research and Analysis
Indepth research into any requirements or areas of implementation that are challenging or complex, including certain decisions regarding technologies.

Section 6 - Design
Propose a structure for how the product requirements are to be implemented, and how the different parts of the system will interact with each other.

Section 7 - Implementation
Give an in-depth and analytical explanation of the processes and tools used during the development of the product, explaining how and why certain decisions were made, and highlight the challenges that were faced.

Section 8 - Evaluation
Evaluate the product through testing of functionality, as well as discussing the extent to which this project fulfilled its goals and objectives.

Section 9 - Conclusion
Draw conclusions on the success of this project based on a number of different criteria. Discuss improvements that could be made, and things that I would have done differently.

Section 10 - Statement of Ethics
Address any questions about the ethics of the research conducted throughout the project.

2 - Literature Review

2.1 - Related Applications

2.1.1 - R.H.S Grow Your Own

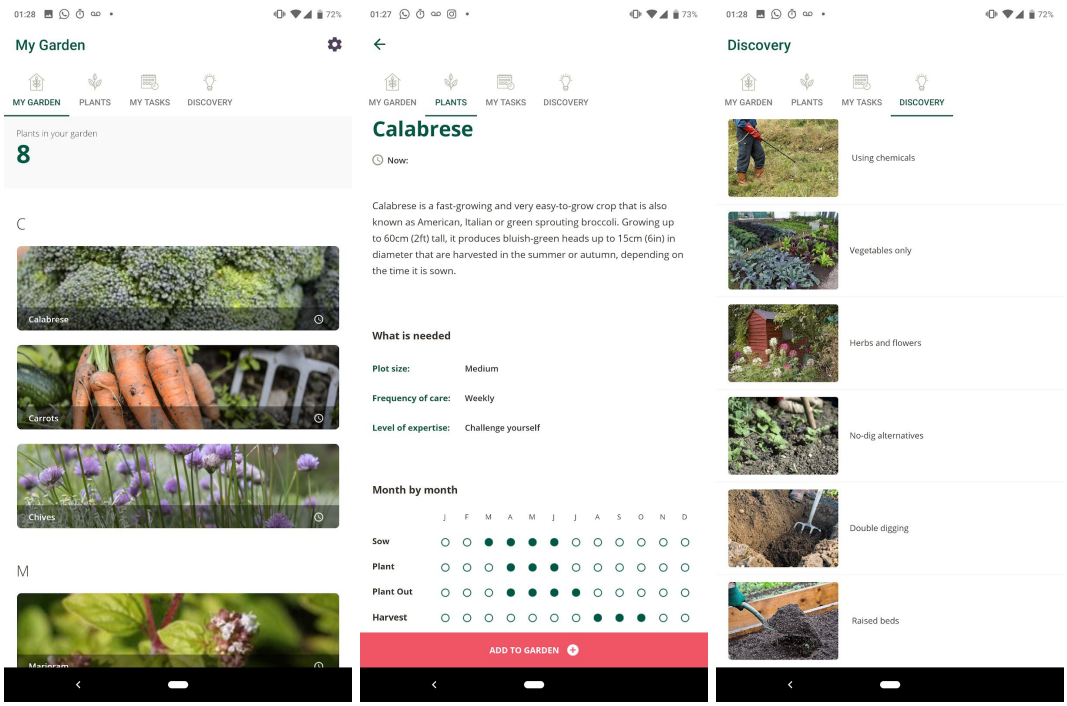


Figure 2 - *Screenshots of RHS' Grow Your Own application*

The RHS has an existing application, 'Grow Your Own', that accomplishes a similar goal to one part of this project. It has 3.3 out of 5 stars on the Google Play store, but from only 10 reviews, and fewer than 5,000 downloads. It's simplistic, mainly supplying a user with a list of plants and information about them, including best times of year to plant/harvest, common problems, etc. It allows you to add these plants to a 'Garden', which is a separate tab enabling quicker access.

You can add notes to a plant and it will create a list of tasks to remind you when it is time to harvest, prune, etc. It is a strong basis for a helpful application, but further development is needed. It currently does not offer much in the way of personalised advice. There is a need for a more sophisticated and context aware application, one that can tailor advice more carefully to the user's needs.

2.1.2 - Home Outside

Figure 3 - *Home Outside application screenshots*

Home outside is a landscape design application with over 100,000 downloads and 3.8 out of 5 stars on the Google Play store. It allows you to design a landscape, which you can store in a gallery and edit at any time. You can place objects, for example a tree or a building, by pressing the ‘Add’ button and choosing from a list of components. The application feels fairly out-dated and doesn’t have the most responsive user interface. The app window doesn’t fit the screen on my device. However, it is an interesting application that could inspire certain design decisions within this project.

As one of the goals for this project is to give users an option to extend their interests beyond visiting an attraction, a portion of the application that allows users to build their own garden from scratch and visualise certain aspects of their horticultural ideas could be very useful.

However, I found trying to build a garden with any detail in this application tiresome. It’s a long-winded process only made more difficult by a sluggish user interface and the inaccuracy of attempting to use your fingers to make precise movements on a small touch-screen. I believe it may be off-putting for a new user who just wishes to get some advice on how they should lay out the plants in their garden.

Building an application such as this is a difficult and time-consuming venture, especially when it is only going to be one portion of my product. Having used Home Outside I researched the feasibility of doing something similar. I found a number of libraries for building a system that enables the translating, rotating and scaling options this application offers. A library called MotionViews [5] seemed the most sophisticated. I opened up their example project on Android Studio and began experimenting with little regard for planning or structure. This reinforced my view that this would be too difficult a feature to implement without sacrificing more important areas of the project.

2.1.3 - Gardena

Gardena [6] is a web application that serves a similar purpose to Home Outside, but in-browser. The benefit of Gardena over Home Outside is that it allows extra precision through the use of a mouse. It also allows you to make custom shapes for your plots, showing you your distances and angles while you draw it, and then giving you the ability to 'plant' certain plants within the plot.

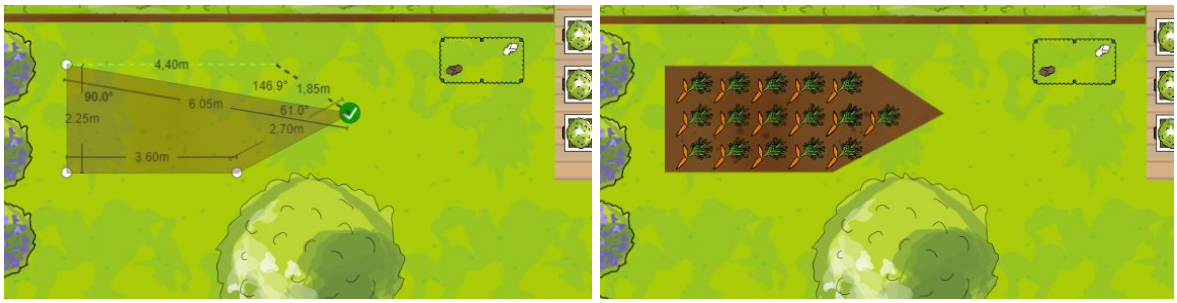


Figure 4 - *Screenshots from Gardena web application*

This is a step up from Home Outside in many ways. Home Outside only allows the dragging/dropping and simple transformations of predefined shapes. Their system works well in a web or desktop application but would likely not translate well to a mobile application due to the size of the screen limiting precision and the use of a touchscreen rather than a mouse.

2.1.4 - Thorpe Park/Alton Towers Applications

Both Thorpe Park and Alton towers have practically identical applications which feature an interesting way of showing users a map of the location.

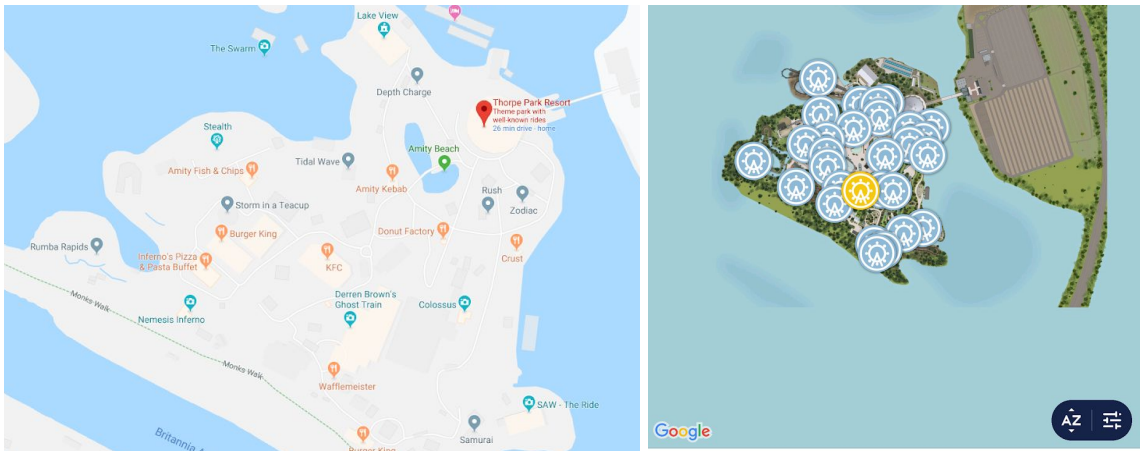


Figure 5 - *Thorpe Park application map (left) compared to google maps.*

They appear to overlay a custom drawn image for their resort over an existing and intractable Google MapView. This allows them to still make use of features built into Google Maps such as markers and routing, while showing the user a much more detailed yet visually simple representation of the area.

The application allows you to filter points of interest depending on your needs. For example, you can view rides and attractions, or facilities such as toilets, cash machines, restaurants, shops, etc. This prevents the need to search through a mass of markers to find what you are looking for.

2.2 - Technologies

2.2.1 - Low-energy Bluetooth beacons

Works by Brandon Hardy for the University of Surrey [7] established a solution to location awareness in areas of little-to-no Wi-Fi or Cellular connectivity. This could be of great benefit to any visitor experience application that involves walking round a building or outdoor area.

Hardy had several unfulfilled objectives due to time-constraints, including a self-guided tour, saving a custom tour to the server, and storage of beacons triggered by the user. I will attempt to explore and expand upon these objectives while making use of his successful implementation of bluetooth beacons. I hope to progress in making a more comprehensive and engaging visitor experience application.

2.2.2 - 3D Mapping from Satellite Imaging

Google Earth's desktop application has a feature which enables you to explore a 3D render of a location based on satellite imagery.

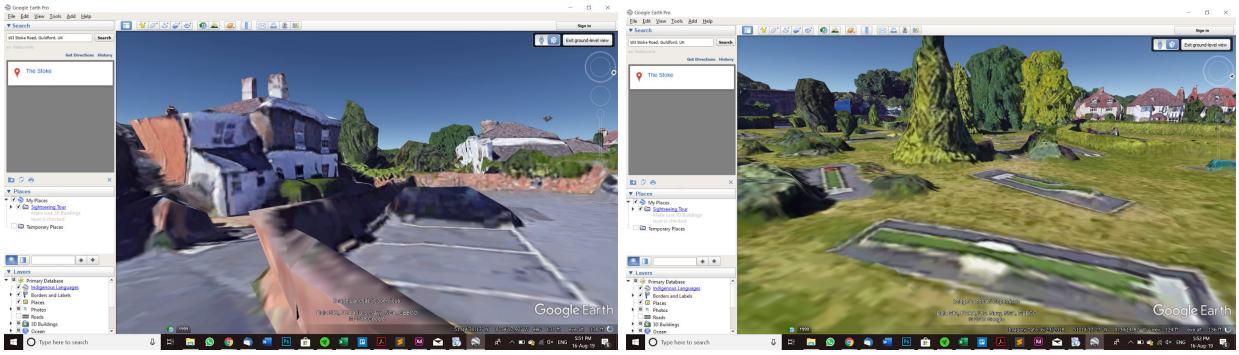


Figure 6 - *Google Earth Desktop's Ground-Level view.*

The accuracy of this technology is impressive. It enables you to travel anywhere you want, unlike Google Street View which relies on accessibility for their camera vehicles. The second screenshot in figure 6, for example, is a mini-golf course local to me, and the height of the trees and surrounding buildings is captured extremely well. Up close, everything is poorly textured and extremely low-poly, but that is to be expected.

I briefly considered the feasibility of using this technology to automatically import 3D models of attractions to be used as a virtual tour, or attempting to build a user's garden. However, after some brief research, it seemed that the only methods for extracting a full 3D model of an area are unofficial workarounds that require a lot of effort for mediocre results. Many of these workarounds would breach Google's terms of service.

2.3 - Summary

To summarise my discoveries from my literature review, there exists some applications for the building and creation of gardens, however they are often cumbersome to use, and do not focus on the details that users would likely find most useful, such as sun exposure figures and recommendations on which plants would be best suited for particular plots.
Also, while many businesses in the visitor attraction industry do have mobile applications, often they are feature poor, and basically act as an electronic guide book, rather than attempting to enrich a visitor's experience during their stay.

3 - Methodologies and Architectures

3.1 - Development Methodologies

Most software development methodologies involve the same set of steps, but have different methods of tackling them. These steps are often referred to as the software development life-cycle, and include:

- Specification
- Analysis
- Design
- Implementation
- Testing & Integration
- Maintenance

Most software development methodologies focus on how to effectively develop in a team. As this is a solo project, I have to take into account how effectively these processes can be applied to a solo developer.

3.1.1 - Waterfall Model

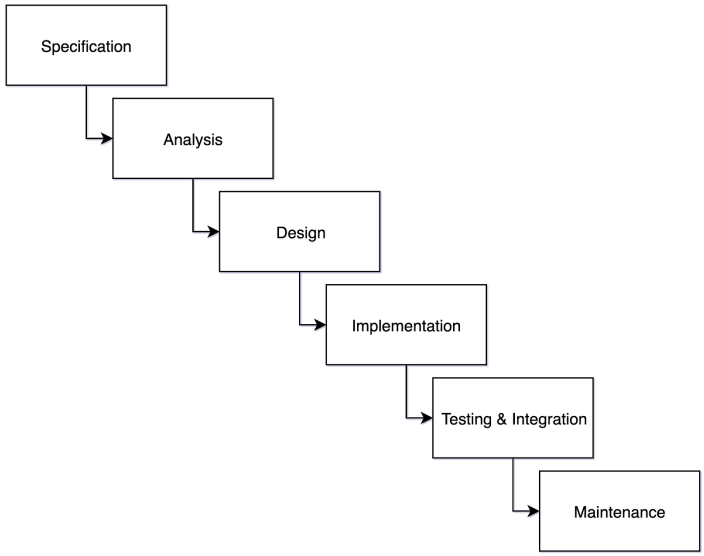


Figure 7 - *Waterfall Model*

The waterfall model is a linear approach to development introduced nearly 50 years ago. Each phase is completed before moving on to the next. Each phase depends on the deliverables of the previous, and as such the model is inflexible. It is most suited to projects that have fixed requirements, significant time available, and experience with all technologies being used. If an issue is found at any stage, often the entire process needs to start over. For example, if during

the Testing and Integration stage it is discovered that there is a major flaw in the design or implementation of the system, the model does not allow for returning to these stages and adapting. This becomes a bigger issue if the client demands changes from the original specification. For this project, I am both the client and developer, so this is not a cause for concern. The benefit of this model on the other hand is that it does allow you to clearly define milestones and deadlines, and encourages disciplined organisation.

3.1.2 - Agile Model

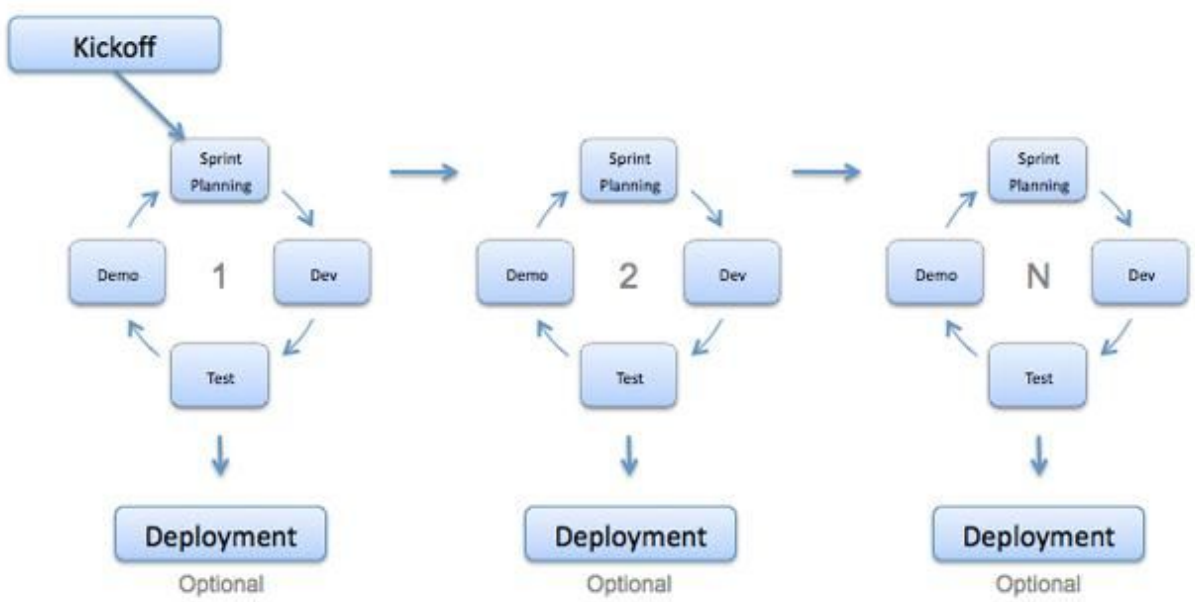


Figure 8 - *Agile development model [8]*

The agile model is based on the principles of rapid, continuous delivery of usable software, by splitting development into sprints. Each deployment builds on functionality developed in the previous sprint, and is thoroughly tested. Each sprint has a specific time allowance. The methodology is best suited to developments that must stick to tighter time schedules. It is much more flexible to changing requirements, and although I mentioned this should not be necessary due to this project being a solo development choosing my own specifications, it's difficult to know what unexpected obstacles could arise at any time. The downside of adopting an agile model is that it is much easier for the application to get derailed if a clear vision is not apparent from the start. This can be the case due to the lack of reliance on extensive planning and documentation compared with the waterfall model.

3.1.2 - Incremental Model

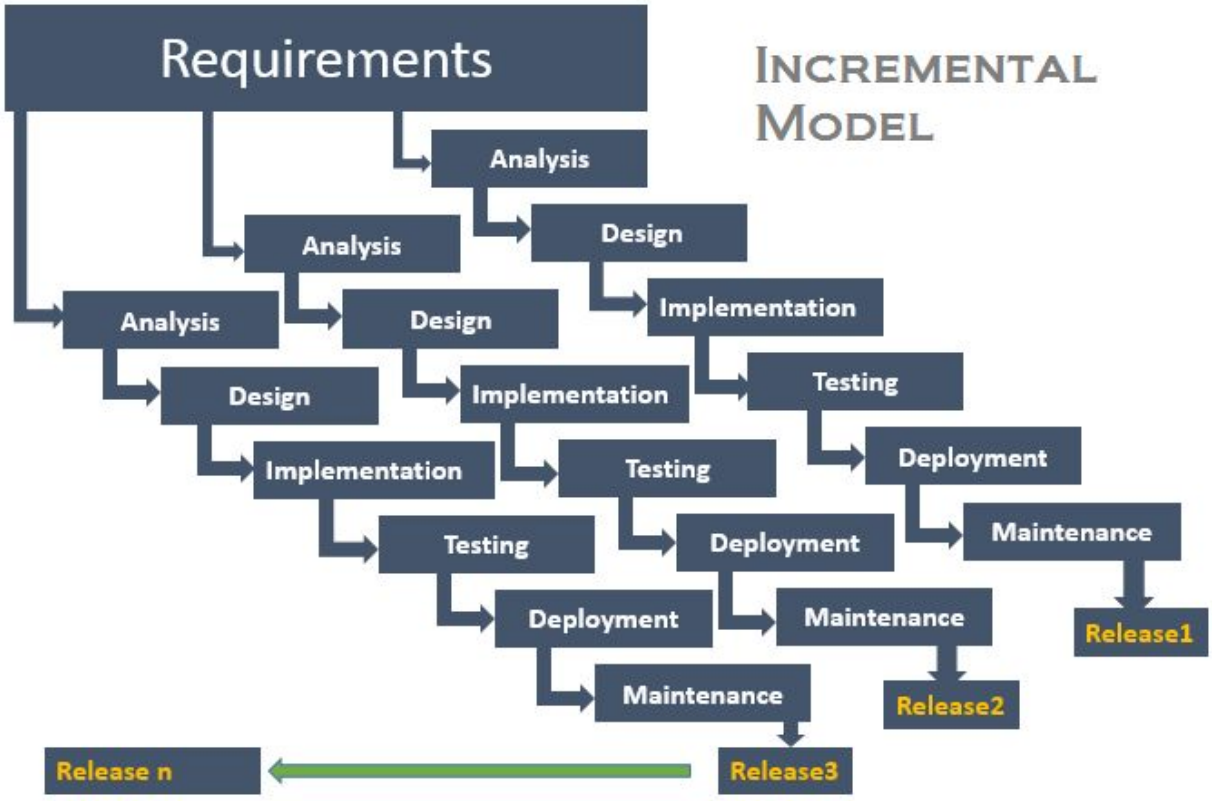


Figure 9 - *Incremental development model [9]*

The incremental model is built on the foundations of the waterfall model, with the key difference that development is done incrementally. Development can be split up into different functionalities that each undergo a strict process like the waterfall method. It is therefore much more flexible, allowing for changes in scope or requirements much more easily than the waterfall method. It also allows working software to be built quickly, and improved upon through each iteration - similar to the agile method. However, there is a need for a clear vision for the entire system from the start, with detailed requirements before it can be broken down into increments, meaning there should still be a detailed planning and design section before development can begin. I think this methodology has many clear benefits for this project, as my application can easily be split into three major increments. For this reason, this is the model that I have selected to manage product development for the project.

3.2 - Hardware Evaluation

Delivering an improved visitor experience could be done on a variety of platforms.

3.2.1 - Web-Based Application

It would be possible to develop a web-based application. One benefit of being web-based is that the application is universal; it can be used on both mobile and desktop devices. It is mostly software independent in that it can run on multiple operating systems, such as Windows/Linux/MacOS and Android or iOS. Web-based applications are accessible through the user's preferred browser.

This universality would be ideal, but it would likely cause problems during development. While in theory a web-based application should be capable of being accessible through any appropriate software/hardware, the reality is that conflicts will arise and behaviours will differ between environments. For instance, where two different users access the application from their Windows 10 desktop computers, it may run perfectly for one but fail to work for the other running an older browser version. Screen dimensions for both mobile and desktop resolutions must be considered, alongside other unusual resolutions such as an Ultrawide 21:9 aspect ratio.

3.2.2 - Mobile Application

Mobile applications have the benefit of portability - key for a visitor experience application as the app will be designed for use onsite at the attraction. While a web-based application can accomplish this goal, a mobile application can be used, dependent on design and at least to some degree, without an internet connection. There are even more considerations to take into account here, however, such as which phone users may be using. With Apple and Android phones securing roughly a 99% market share, both in the UK and globally [10], these were the only two environments I considered.

In an ideal scenario the application would run on both. However, due to time and resource constraints this was not possible. Ultimately, I decided that development for Android would be the best option for this project. Having had prior experience with Android development, I did not have to learn a new language and skill-set for this project, which would likely have resulted in a less ambitious scope. In addition, I did not have access to any iOS devices for testing, and would be forced to use emulators. This would not give a sufficiently accurate representation of the user experience.

The current market share for Android devices is around 4-10% lower than for Apple, but it remains a major platform for an application like this. If development were to continue outside the scope of this project, developing for both platforms would be a primary goal.

3.3 - Development Environments

Android Studio is the official integrated development environment (IDE) for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development. It is a replacement for the Eclipse Android Development Tools (ADT) as the primary IDE for native Android application development [11] .

Once it was decided that Android would be the target platform, Android Studio was the obvious choice for my development environment. I am familiar with the environment from previous projects and alternatives involve workarounds and the use of out of date methods such as the ADT plugin for eclipse.

4 - System Requirements and Specifications

4.1 - Introduction

4.1.1 - Purpose

The purpose of this section is to provide an overview of the product to the reader. It is explained why the product is being developed and who it is being developed for, as well as the software, hardware and other requirements that must be met to ensure the product is successful.

4.1.2 - Document Conventions

The System Requirements and Specifications is based on the IEEE standard, adapted to suit the project.

4.1.3 - Product Scope

Please refer to section 1.2 - Aims and Benefits to view the objectives and goals of the product.

4.2 - Product Overview

4.2.1 - Product Perspective

The product is a self-contained product unrelated to any larger systems.

4.2.2 - Product Functions

The product will provide the following functions:

- Allow the user to plan, follow and store walks around an exhibit/attraction through the “Walks” section.
- Allow the user to view information relevant to the exhibit/attraction to improve their experience through the “Encyclopedia” section.
- Allow the user to further their interests beyond visiting the exhibit/attraction by cultivating their own products and receiving tailored advice on how to do so in the “Garden” section.

A more detailed list of functions can be found in 4.4 - Functional Requirements

4.2.3 - User Classes and Characteristics

Users of this product will likely differ greatly in age, education and technical expertise. It is important that the application is easy to use by those with less education and technical expertise

and it must remain useful to those with more knowledge. Technically literate users must not feel put off by “hand-holding” getting in the way of their experience.

Users can be classified in the following ways:

- Those who wish to use the application primarily while at the attraction/exhibit, in order to improve their experience, mainly making use out of the “Walks” and “Encyclopedia” sections of the application.
- Those who wish to develop their visit into a hobby, mainly making use of the “Garden” and “Encyclopedia” sections of the application.

4.2.4 - Operating Environment

The product will operate exclusively on Android devices and will be designed to be used on Android 9.0 (SDK version 28).

4.2.5 - Design and Implementation Constraints

Time is the major constraint on the design and implementation. The product will be designed and developed by a sole developer in the space of one academic year. The scope of the product is fairly large and complex and therefore it is possible that some features will need to be sacrificed in order to ensure the deadline is met.

There is no monetary budget for development, which means that no paid APIs or libraries can be used unless these offer a sufficient ‘free quota’ or trial period.

4.3 - External Interface Requirements

4.3.1 - User Interfaces

The user interface will aim to be intuitive and simple to use for all classes of users. This means elements must be self-explanatory as to what they do, or have a description to inform the user.

Colours should be contrasting in order to make viewing simple, even for those with any degree of colour-blindness.

Buttons or interactive elements should be large enough to easily be pressed by those with minor motor skills issues.

Text should be large enough to be easily legible by those with mild vision impairments.

Please refer to section 6.3 - User Interface Design where detailed mockups have been prepared.

4.3.2 - Hardware Interfaces

Devices on which the product can operate must have a fully responsive touch-screen that will allow the user to interface with the various parts of the application.

Devices must have the ability to interface with the internet, either through a mobile connection, such as 3G/4G, or via Wi-Fi.

All modern Android smartphones which have not sustained significant damage should meet these requirements.

4.3.3 - Software Interfaces

There are some major APIs vital to the functionality of this product. These include:

- Dark Sky API [12]
    - This API will help calculate the weather conditions and forecasts within the “Garden” section
- Firebase APIs [13]
    - This will include Firebase Auth and Cloud Firestore, and enables users to read and write their data to a cloud database.
- Google APIs [14]
    - This will include Maps API for displaying and interacting with a map in the “Walks” section, Directions API for plotting a route on the map, and Places API for the user entering their garden location on sign up.

There will be other libraries used within the product, but most are minor and the product doesn’t rely on them staying updated for basic functionality. Examples include:

- StateProgressBar [15]
- MaterialScrollbar [16]
- ArcGIS [17]

4.4 - Functional Requirements

Due to the exploratory nature of this project, the functional requirements are somewhat flexible. I have considered only features I believe could be realistically implemented within my time constraints, but accept that there may be some changes due to unconsidered technical constraints that may come to light. I have attempted to draw up these requirements considering the three main functionalities of this application; Encyclopedia, Garden and Walks.

1. Create an account
    - 1.1 Input basic information through text fields, i.e name
    - 1.2 Input address through an address finder, or manually if preferred
    - 1.3 Have user sign privacy agreement
    - 1.4 Create account through unique email and strong password
    - 1.5 Validate at every step, disallow progress while field invalid
2. Sign into an account
    - 2.1 Through email and password
    - 2.2 Through social media login
3. Sign out of account
    - 3.1 Sign out of account from anywhere within application
4. Navigate Application
    - 4.1 Navigate to ‘Garden’ through tab at bottom of screen
    - 4.2 Navigate to ‘Encyclopedia’ through tab at bottom of screen
    - 4.3 Navigate to ‘Walks’ through tab at bottom of screen
5. Encyclopedia Functionality
    - 5.1 Select a plant to view all relevant information about that plant including:
        - 5.1.1 Name
        - 5.1.2 Description
        - 5.1.3 Recommended plot size
        - 5.1.4 Recommended frequency of care
        - 5.1.5 Recommended level of expertise
        - 5.1.6 A month-by-month view showing when to plant/harvest
    - 5.2 Live filter plants through search bar
    - 5.3 Simultaneously filter plants by certain characteristics including:
        - 5.3.1 Plot Size
        - 5.3.2 Level of expertise
        - 5.3.3 Category
        - 5.3.4 Frequency of Care
    - 5.4 Scroll bar shows letter indicator for fast and accurate scrolling
6. Garden Functionality
    - 6.1 Create a new plot, specifying:
        - 6.1.1 Name
        - 6.1.2 Width
        - 6.1.3 Height
        - 6.1.4 Nearby Objects

6.1.4.1 Add a nearby object, specifying:
6.1.4.1.1 Height
6.1.4.1.2 Distance from plot
6.1.4.1.3 Angle from plot
6.1.4.2 Delete a nearby object
6.2 Edit an existing plot, being able to change all the information entered on creation
6.3 Delete an existing plot
6.4 Select a plot and display all relevant information about that plot, including:
6.4.1 Shade Rating
6.4.2 Rainfall (7-day forecast)
6.4.3 Temperature (7-day forecast)
6.4.4 Recommendations regarding which plant is best suited to plot

7. Walks Functionality
7.1 Create a new walk, specifying:
7.1.1 Points of interest chosen through encyclopedia fragment
7.1.2 Distance parameters chosen through a ‘Slider’
7.1.2.1 Manually changing Edit Text changes slider value to match
7.1.2.2 Moving slider changes Edit Text to match
7.2 Favourite a walk
7.3 Delete a walk from favourites
7.4 Calculate best route dependant on certain parameters:
7.4.1 Maximum length of walk
7.4.2 Minimum length of walk
7.4.3 Specified points of interest
7.5 Display walk detailing/displaying the following information:
7.5.1 List of points of interest
7.5.2 Line to show route through points
7.5.3 Total time and distance for walk
7.5.4 Time and distance to each point of interest
7.5.5 Percentage completion of walk
7.5.6 Above details should update dynamically based on user’s movement

4.5 - Nonfunctional Requirements

4.5.1 - Performance Requirements

Functions should be optimised in order to reduce loading times throughout the product. Should a function routinely take longer than three seconds, implement a progress bar, or at a minimum a loading screen to signify to the user that the product is still working.

4.5.2 - Security Requirements

Data should be stored securely so that personally identifiable information is not accessible to anyone except the user to whom the data belongs and the database administrator.

5 - Technical Research and Analysis

Based on my requirements, there were certain technologies and methodologies I ascertained to require in-depth research. These include deciding which database service my application should use, as well as the theory behind the more mathematical aspects of my project.

5.1 - Server-side Research

5.1.1 - Firebase - Cloud Firestore/Auth

Firebase is a mobile and web application development platform owned by Google, containing a variety of products. The most relevant to this project include Firebase Auth and Firebase Cloud Firestore.

Firebase Cloud Firestore

Firestore is a flexible, scalable NoSQL cloud database used to store and sync data for client- and server-side development [18].

Firestore allows a free quota of 50,000 reads, 20,000 writes and 20,000 deletes per day - far beyond what can be expected for the scope of this project. Even if this application were to be launched commercially, this limit is still very generous, and is unlikely to be reached without huge success in terms of number of users and regular usage among those users. Should these quotas be met, Firebase offers two paid plans, either monthly or pay-as-you-go. The monthly plan quintuples the number of reads/writes/deletes per day and costs just $25 per month, or the pay as you go plan costs just $0.18 per 100,000 document writes, or $0.06 per 100,000 document reads [19].

Firebase Auth

Firebase Auth is an authentication service that allows for easy account creation and login, supporting various social login providers such as Facebook, Twitter and Google, as well as email and password. It integrates directly with Firestore, allowing users' to access data specific to them using their unique User ID generated upon creation.

Firebase Auth allows for 10,000 authentications per month, and costs $0.06 per verification beyond this on the pay as you go plan [19]. For the scope of this project this is far beyond the necessary requirements.

5.1.2 - Amazon Web Services - Databases/Cognito

Amazon Web Services offer a variety of database services, including relational databases, key-value databases, document databases and many more. They also offer an authentication service they call Cognito.

AWS Databases [20]

AWS purpose-built databases support diverse data models and allow users to build highly scalable and diverse applications. They support relational databases and technologies such as MySQL, PostgreSQL, MariaDB and Oracle, as well as six other services including key-value databases and document databases with MongoDB compatibility.

The pricing varies depending on the type of database you chose, but there is a free 12-month trial, sufficient for this project.

AWS Cognito [21]

AWS Cognito is an authentication service that provides a secure user directory that scales to hundreds of millions of users. It supports multi-factor authentication as well as social media logins.

Cognito allows 50,000 active users per month, with additional users costing around $0.005, this cost decreasing as the number of users increases. The free 50,000 active users is far beyond the necessary requirements for the scope of this project.

5.1.3 - Summary

From my research, both services are good options. AWS is extremely flexible and scalable. However, Firebase seems simpler to setup and use. I have experience with both Firebase Auth and Firestore, and was impressed with the versatility and ease-of-use.

The simple integration of Firebase Auth into the Firestore database ensures quick development. While Firestore only offers a NoSQL database structure, which has both its benefits and drawbacks, I will be storing so little data that it will not be an issue.

Firebase is the service I will be using within this project.

5.2 - Client-side Research

This section will explore two features of the application, calculating sun exposure statistics in the Garden section of the application, and calculating an optimal route for the Walks section. I foresee both of these problems being more complex than they appear at first glance, and need to ensure I am not caught out by the complexity, which could cause delays or potentially unfulfilled requirements.

5.2.1 - Sun and Shadow Calculations

One of the biggest challenges faced was the complex calculations necessary to compute sun exposure and shadow length figures within the gardening portion of my application. Initial research prompted results such as Suncalc.org, among other web applications that perform a similar but more basic approach to the calculation. The main take-away from these sites was the computation of a ‘shadow length’ variable, given a location, time and object height.

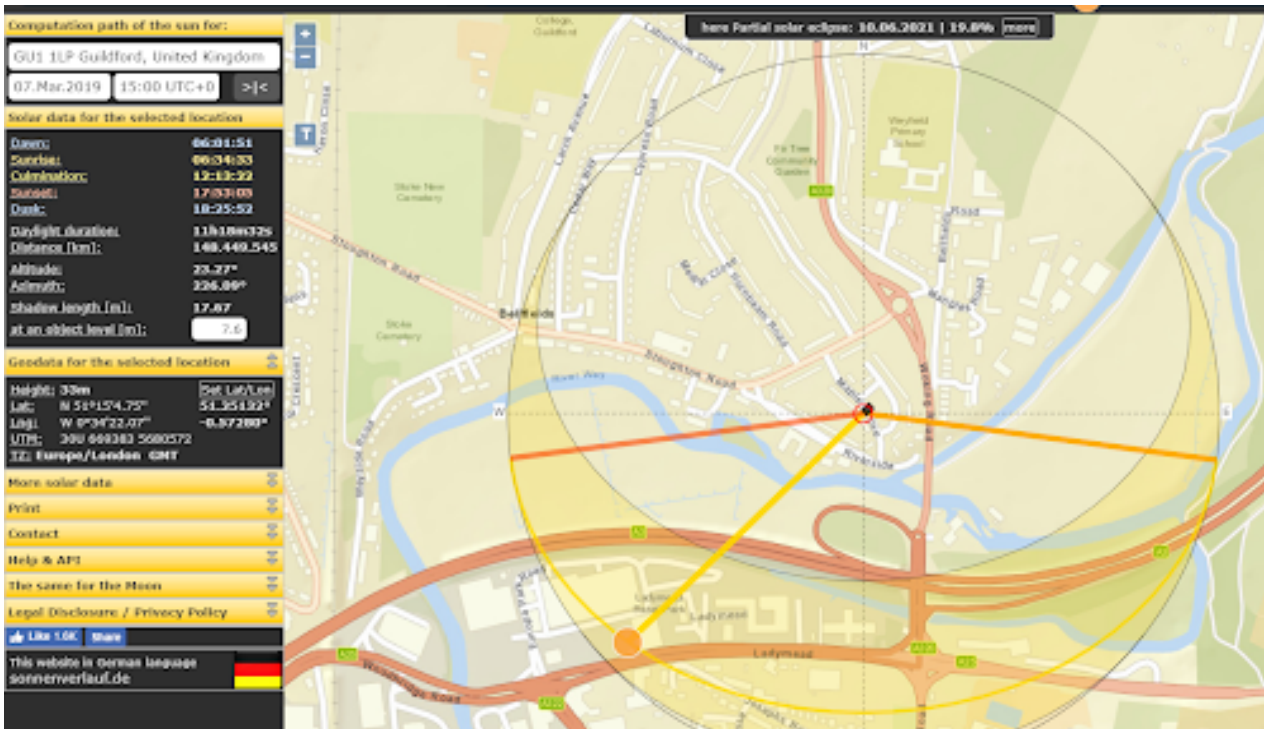


Figure 10 - *Suncalc web interface for calculating various sun related data*

I was unable to find an easy to use library and decided to calculate this value myself. This proved challenging.

A short but basic article by David Maslanka [22] served as an introduction, familiarising me with solar geometry and the various values I’d need to calculate to find the shadow length of an object. Once more comfortable, I obtained my formulae from a resource by Paul Schlyter [23].

During implementation, as well as following Schlyter's examples, I regularly compared the values obtained through my implementation with the values calculated by a website called satellite-calculations.com[24], to ensure accuracy.

Only two inputs are required for calculating the length of a shadow: the height of the object we are finding the shadow length of (from user input), and the altitude of the Sun. The calculations required to compute the altitude are very involved.
The formula for calculating shadow length is below:

$$L = \frac{h}{tan(\alpha)}$$

Where: L = Length of Shadow, h = Height of Object, α = Sun Altitude

The following section will explore in more detail the calculations required to calculate the altitude. As stated before, I obtained the formulae and explanations using Paul Schlyter's resource, and this can be used for an even deeper explanation.

Firstly the current date and time is taken from the device. The date is converted into a 'day number' d which is essentially the number of days from 00:00 on January 1st 2001. The longitude and latitude is taken from the address input by the user for their garden.

From the day number we can calculate a few values that we will need for future calculations. These values are known as orbital elements. That is, elements needed to specify the orbit of an object around a primary body, such as a planet around the Sun or a satellite around the Earth [25]. For our calculations, we will be assuming the Sun is orbiting around the earth, despite the reality being opposite.

The longitude of perihelion is an angle measured from the vernal equinox eastward along the ecliptic to the ascending node of a planet's orbit, and then continued eastward along the orbital plane to the perihelion [26]. It is calculated using our 'day number' d in the following equation:

$$w = 282.9404 + 4.70935E-5 \cdot d$$

The eccentricity is another orbital element that describes the shape of an orbit or any curve that is a conic section. It is the ratio of the distance between the foci and the major axis. The orbital eccentricity for most planets in the Solar System is less than 0.1

To calculate eccentricity: $e = 0.016709 - 1.151E-9 \cdot d$

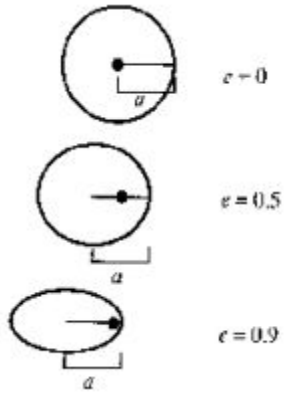


Figure 11 - *Eccentricity Diagram [27]*

The mean anomaly is the fraction of an elliptical orbit's period that has elapsed since the orbiting body passed periapsis (the extreme points in orbit where the orbiting body is at its furthest or nearest point from its host) [28]. If this value were negative, for example in the case of a date before our day zero (January 1st 2000), it has to be normalised to a value between 0 and 360 degrees by adding 360 until it lies within this range.

However, in this scenario the date cannot be manually entered and the year should therefore always be 2019 or later, thus this should not be an issue. It is still something I should take into consideration with my implementation. A user's clock could be manually set to an earlier date which could cause issues if not handled correctly. The mean anomaly can be calculated as below:

$$M = 356.0470 + 0.9856002585 \cdot d$$

The next value to be calculated is the obliquity of the ecliptic. This is the inclination of Earth's equator with respect to the ecliptic (the mean plane of the apparent path in the earth's sky that the Sun follows over the course of one year [29]).

This value - abbreviated to $oblecl$ is calculated by:

$$oblecl = 23.4393 - 3.563E-7 \cdot d$$

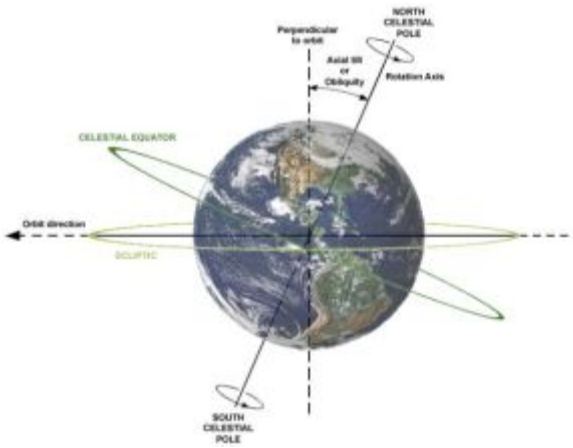


figure 12 - *Obliquity of the Ecliptic [30]*

The Sun's mean longitude (L) can also be calculated now that we have the longitude of the perihelion (w) and the mean anomaly (M). This is computed through a simple addition:

$$L = w + M$$

Alongside the mean anomaly, there are also two other "anomalies" that define a position along an orbit, the others being the eccentric anomaly, and the true anomaly. Both of these angular parameters define the position of a body moving along a Keplerian orbit [31]. The relationship between these two anomalies can be seen in figure 13. The angle 'E' is the eccentric anomaly, and the angle 'f' is the true anomaly.

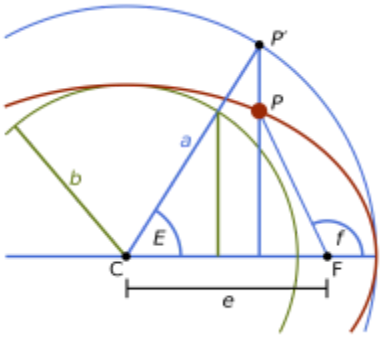


figure 13 - *Eccentric Anomaly [32]*

As the Sun's apparent orbit is so small, an approximation of the eccentric anomaly (E) will be sufficiently accurate. It can be calculated as below:

$$E = M + e\,sin(M) \cdot \frac{180}{\pi} \cdot (1 + e\,cos(M))$$

We can calculate the true anomaly through computation of the Sun's rectangular coordinates in the plane of the ecliptic, where the X axis points towards the perihelion:

$$x = cos(E) - e$$
$$y = sin(E) \cdot \sqrt{1 - e^2}$$

The true anomaly (v) can be calculated from these coordinates:

$$v = atan2(y, x)$$

We also need the distance between these two coordinates (r) for later calculations (simply from $r = \sqrt{x^2 + y^2}$).

Now we can compute the longitude of the Sun from the addition of the longitude of the perihelion and the true anomaly calculated above: $lon = v + w$

Finally, the Sun's ecliptic rectangular coordinates need to be calculated and rotated to equatorial coordinates in order to compute the Sun's Right Ascension and Declination Angle. The Right ascension is the angle measured eastward along the celestial equator from the Sun at the March equinox to the (hour circle of the) point above the earth in question. When paired with declination, these astronomical coordinates specify the direction of a point on the celestial sphere in the equatorial coordinate system [33]. We calculate these as below:

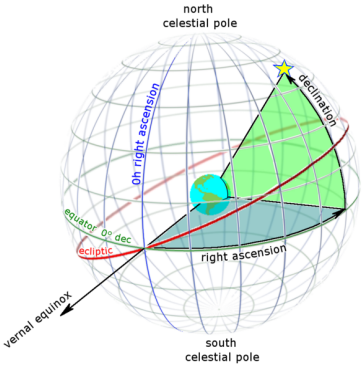


Figure 14 - *Right ascension [33]*

$$x = r\,cos(lon)$$
$$y = r\,sin(lon)$$
$$z = 0$$

Using our $oblecl$ we can rotate these coordinates as below:

$$xequat = x$$
$$yequat = y\,cos(oblecl) - z\,sin(oblecl)$$

$$zequat = y\ sin(oblecl) + z\ cos(oblecl)$$

Which we can convert to our Right Ascension and Declination angle by:

$$RA = atan2(yequat, xequat)$$
$$Decl = atan2(zequat, \sqrt{xequat^2 + yequat^2})$$

There's now only a few steps left for finding the altitude. First we need to find the local Sidereal Time, which is the time scale that is based on Earth's rate of rotation measured relative to the fixed stars [34]. To find this we need the GMST (the Sidereal time at the Greenwich meridian), the UT (Greenwich Time) and the longitude. These all need to be in the same format, either degrees or hours. 'Converting' degrees to hours can be done by dividing by 15 and vice versa. Either of the following formulae are valid depending on the format desired for the local Sidereal Time:

$$SIDTIME\ (hours) = GMST + UT + LON/15$$
$$SIDTIME\ (degrees) = 15(GMST + UT) + LON$$

I chose to use the second in my implementation, as keeping everything in degrees was more useful for the rest of the calculations required to find the altitude. GMST is simply calculated by adding or subtracting 180 degrees (or 12 hours) to the Sun's mean longitude (L) we calculated earlier. UT is the current time in the UTC time standard.

The Hour Angle is the angular distance between the meridian of the observer and the meridian whose plane contains the sun. It is zero at local noon, meaning the Sun is at its highest point above the horizon. It can be computed through simple subtraction of the local Sidereal time and Right Ascension: $HA = SIDTIME - RA$

There's one final calculation we need to do to find the altitude. We convert the Sun's Hour Angle and Declination to a rectangular coordinate system where the X axis points to the celestial equator in the South, the Y axis to the horizon in the West and the Z axis to the North celestial pole. These also need to be rotated along an East-West axis so that the Z axis will point to the zenith.

$$x = cos(HA) \cdot cos(Decl)$$
$$y = sin(HA) \cdot cos(Decl)$$
$$z = sin(Decl)$$

And to rotate we use the following where lat is the observer's latitude.

$$xhor = x\ sin(lat) - z\ cos(lat)$$

$$yhor = y$$
$$zhor = x\,cos(lat) + z\,sin(lat)$$

The final calculation for altitude is:

$$altitude = atan2(zhor, \sqrt{xhor^2 + yhor^2})$$

With this altitude value, all the necessary values have been gathered to be able to calculate shadow length. The only other value needed is the height of the object which the user will enter manually.

The azimuth of the sun, which is needed to calculate the directions that the shadow is being projected, can be calculated using the $yhor$ and $xhor$ figures found earlier when calculating altitude, as follows:

$$azimuth = atan2(yhor, xhor) + 180$$

5.2.2 - Shortest Route Calculations

For the walks section of the application, I quickly realised that I needed to implement a solution to the Travelling Salesman problem. The travelling salesman problem is as follows; "Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city and returns to the origin city?" [35], the only difference being thinking in plants rather than cities.

There has been much research into the solutions of this problem, although due to its being an NP-hard problem, the perfect solution currently does not, and may never exist. The problem was formulated in 1930, but an exact solution is yet to be discovered, as with all NP-hard problems. It is an optimisation problem, looking for the cheapest and quickest algorithm that can be found to reach a solution, possibly not even the optimal solution, but close enough.

Google's Directions API, which likely will be used in the Walks section for plotting distances and routes anyway, has a built in solution to this problem. It is very simple to use and ideal for a predefined list of points the user wishes to visit. However, there is a complication; should there be more than one option for a plant, maybe as many as five different locations the user could choose to visit to see this plant, which point would be the best in terms of overall distance, taking into account all other points. What if, as an example, the user chose ten different plants they wished to visit, and each plant has five different locations across the garden. How should the specific plant be chosen from the group? If I was to carry on using Google's algorithm, I would have to brute force the solution, testing every possible route until the best one is found. For a very small list of points of interest, with each point having very few options, this is fine. For

example, if the user chose two points of interest, and each of those points has three different locations/options that the user could visit, there are only eight possible routes the user could take to visit those two points of interest using any of the location options for each. However if they chose ten points of interest, and each of those points had five different locations they could visit, there are 9,765,625 different routes that could be taken, as routes can be calculated by:

$n = q^p$, where n = number of routes, p = number of points chosen, q = options per point.

This number grows exponentially larger, for perspective, a small increase to 15 points with 8 options each, the number grows in excess of 35 trillion possibilities. Clearly, if we're working with a big site with lots of locations of each plant, and we're letting the user pick any more than 5 points, brute forcing this calculation is simply not possible.

I began formulating a solution. The idea was that instead of calculating every possible route, to instead calculate the distance between each plant, and every other plant, and then using a Shortest Path First (SPF) algorithm (possibly Dijkstra's algorithm) using the distances as edges, and the plants as vertices.

I sketched out a diagram as seen in figure 15 and attempted to work out a formula for the total number of edges (distances from every point to every other point) for a certain number of vertices (point of interest), bearing in mind that a distance would not have to be calculated between plants of the same type (i.e. when there are multiple options for each plant, only work out distances to each other plant, not other options for their own plant).

I worked out the sequence could be written as:

$$q(pq - q) + q(pq - 2q) + q(pq - 3q) + ... + q(pq - (p-1)q)$$

After calculating the sum of the first n terms (S_n), I found:

$$S_n = \frac{(pq)^2 - p(q^2)}{2}$$

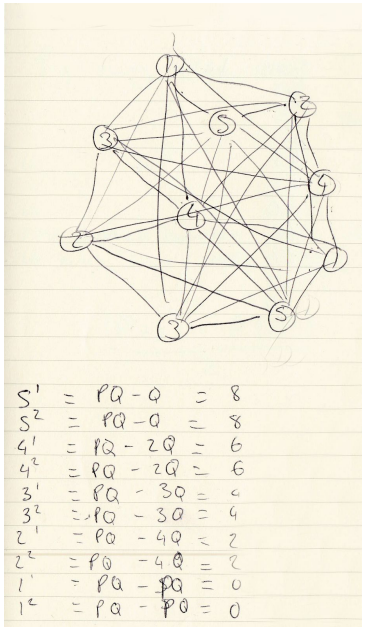


Figure 15 - *Sketch*

Having worked out the equation to calculate the number of calculations required to find the optimal route, I plotted these graphs to compare how much better this solution is to the brute force solution suggested earlier, as can be seen in figure 16.

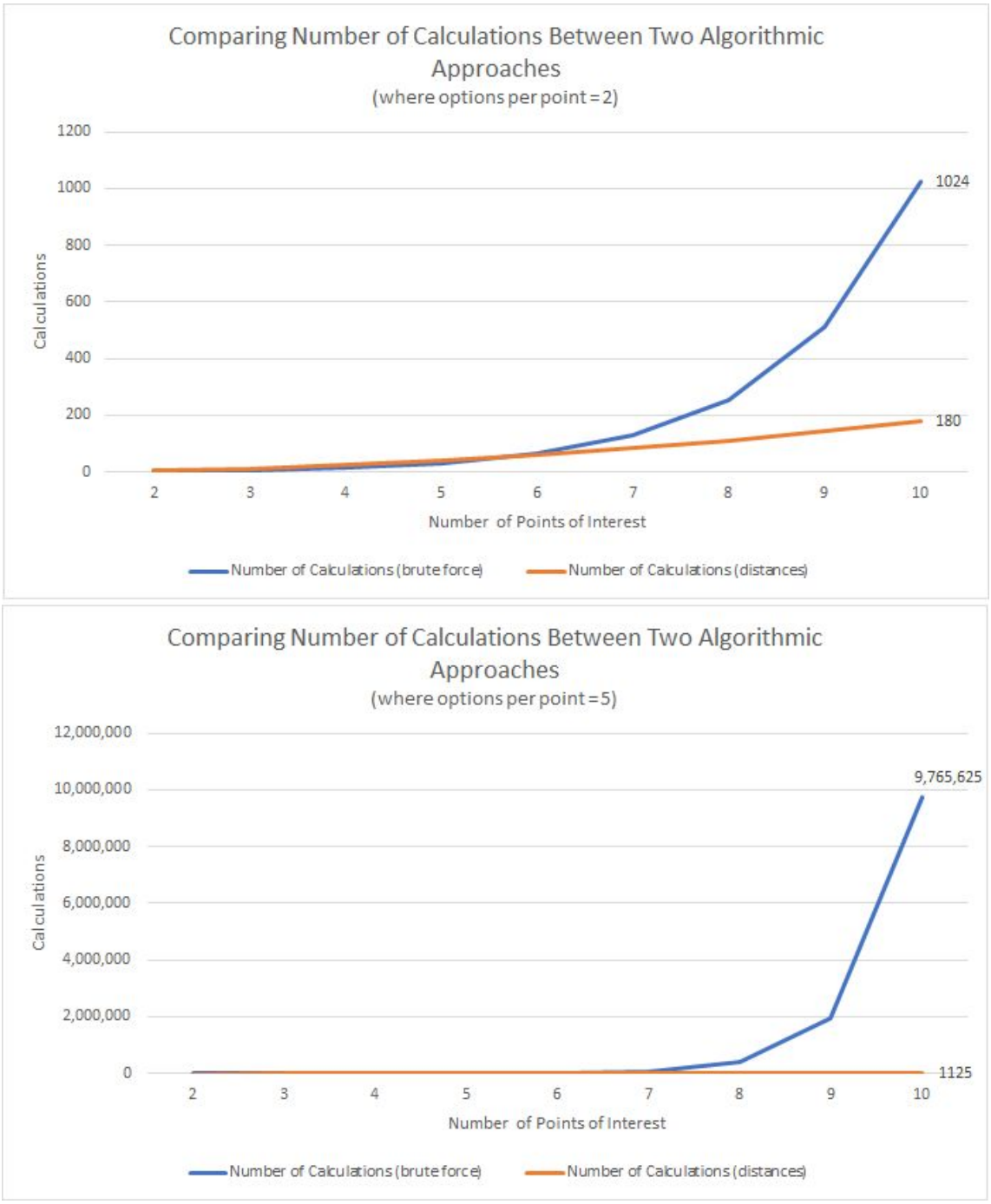


Figure 16 - *Graphs to show the difference in calculations between two algorithmic approaches*

The number of calculations for the 'distances approach' is not a final figure as the SPF algorithm still must be run using these distances as edges. However, running this algorithm, which in the case of Dijkstra's algorithm has a worst-case time complexity of $O(V^2)$ should not be the biggest concern in these calculations. Both calculating a full route, and finding the distances between all points, will need API calls, which take much longer and have a limit enforced on them. Therefore, I believe this second approach is much better suited for this application.

6 - Design

6.1 - Introduction

This section will look at the techniques and methods that could be used to meet the aims and goals outlined at the start of this project.

6.2 - Architecture Design

There will be three different sections to this app, which include the following:

An encyclopedia - users will be able to access all of the RHS' knowledge on plants and flowers at face value, being able to filter plants by certain attributes or by their name.

A gardening section - users will be able to create, store and view plots which can be customised to include dimensions and surrounding foliage/buildings that may provide shade. Using the geographic location of the garden, users should be able to get information on how many hours of sunlight, short-term forecasts and estimated rainfall to not only help them choose plants best suited to their environment, but how to maintain them.

A walks section - users will be able to select plants they are most interested in, and have walks created for them specifically designed to see as many of their interests as possible, while staying within the distance range they can provide. These walks can be favourited and re-visited whenever the user wishes.

6.3 - User Interface Design

All mockups were created in Adobe XD utilising the Google Material components sticker sheet [36] for many of the UI elements.

Figure 17 - *Splash Screen*
The splash screen that can be shown when the user first opens the application. This is to only be shown briefly while the application loads its necessary views/components.

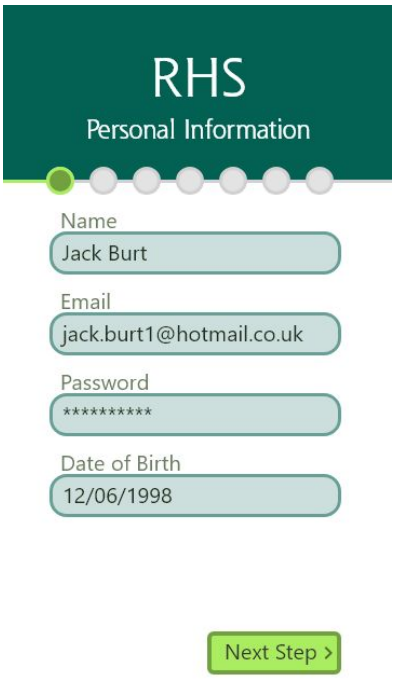


Figure 18 - *Setup Wizard 1*
The first page of the setup wizard, with a progress bar at the top and relevant fields for data entry.

Figure 19 - *Setup Wizard 2*
Another page of the setup wizard - this time asking the user for their address.

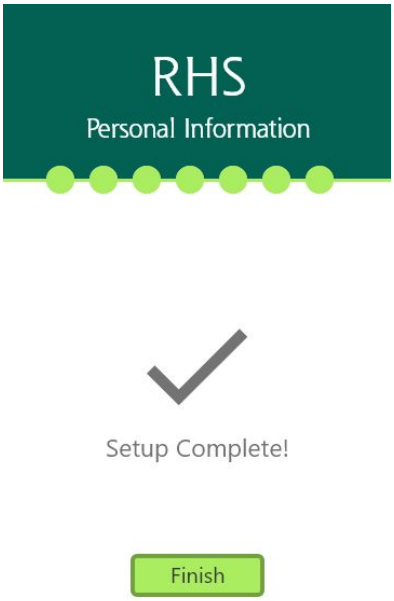


Figure 20 - *Setup Wizard End*
Final page of the setup wizard

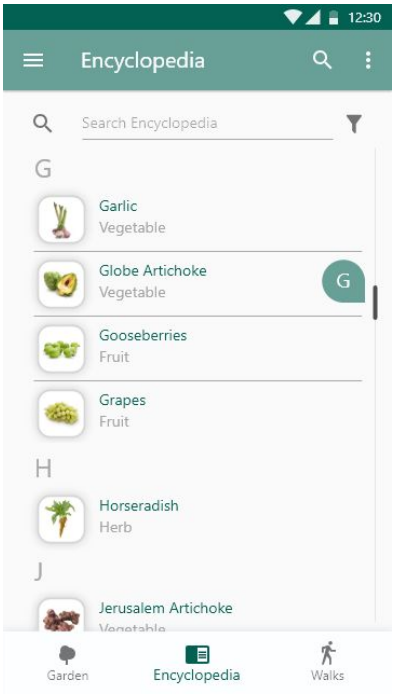


Figure 21 - *Encyclopedia*
The Encyclopedia page showing a scrollable list of plants, with a search query bar at the top along with a button to open filters.

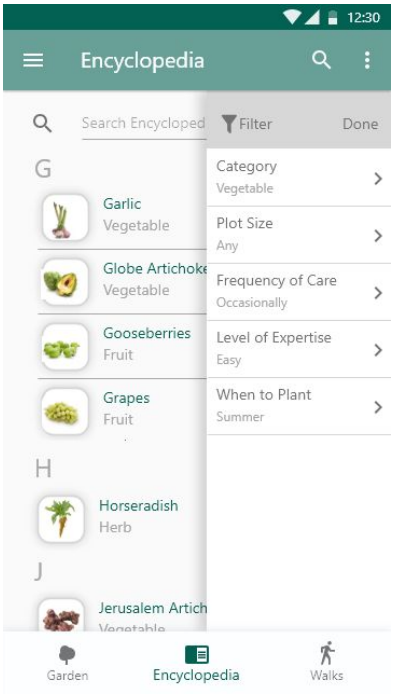


Figure 22 - *Filter Drawer*
Encyclopedia page with open filter drawer, where users can select filters to apply/disable

Figure 23 - *Plant View*

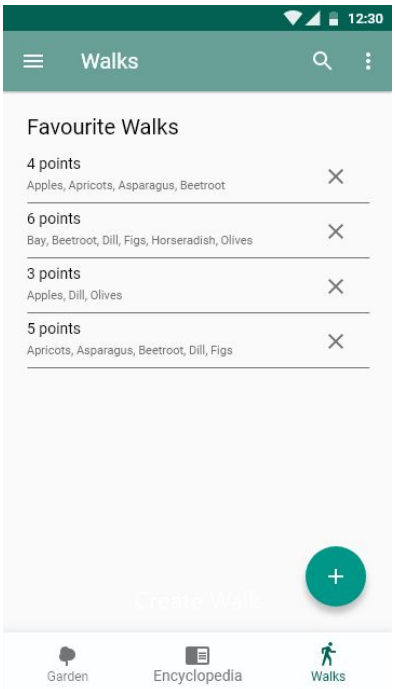


Figure 24 - *Walks*

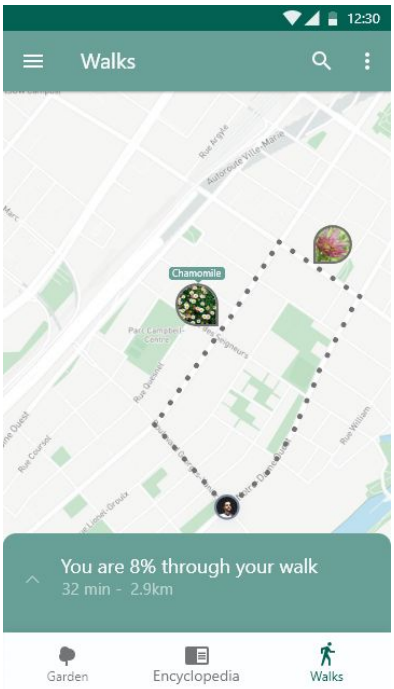


Figure 25 - *Walk View*

The plant view of the Encyclopedia section which shows users information about the plant such as a description, recommended plot size, and other attributes.	A list of favourited walks the user can re-visit along with a Floating Action Button the user can use to add a new walk. Walks can be deleted from this list by pressing the ‘x’ symbol.	A route with points of interest is plotted on a map, with a toggleable card at the bottom (see figure 26)

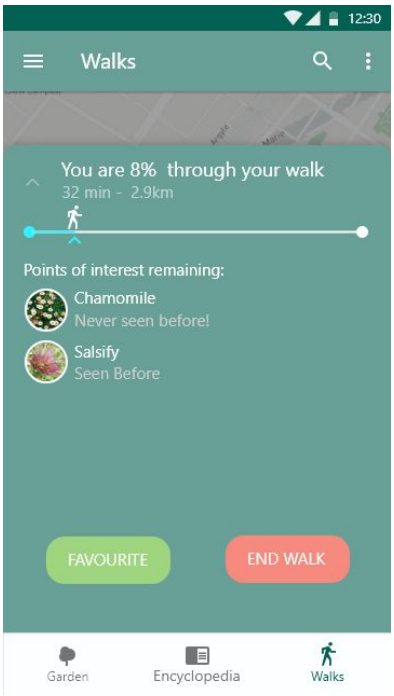
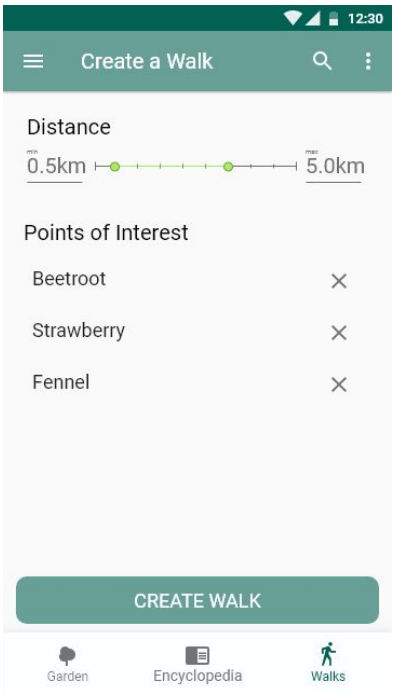
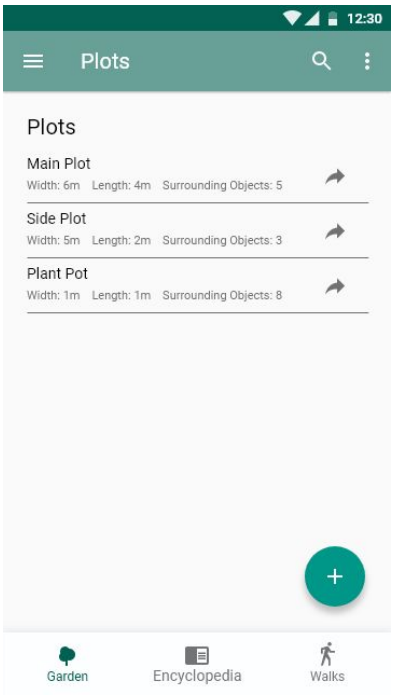

		

Figure 26 - *Walk Details* A card slides up over the map when toggled, displaying the details of the walk such as progress, times and distances, as well as two buttons for favouriting and ending a walk.	Figure 27 - *Create Walk* A view which allows the user to choose a maximum and minimum and add points to the walk - chosen through reuse of the encyclopedia view (see figure 21)	Figure 28 - *Garden Plot List* A list of plots that have been added to the garden, with brief information about each one and a button to edit the details. A Floating Action Button features in the bottom right to add a new plot.

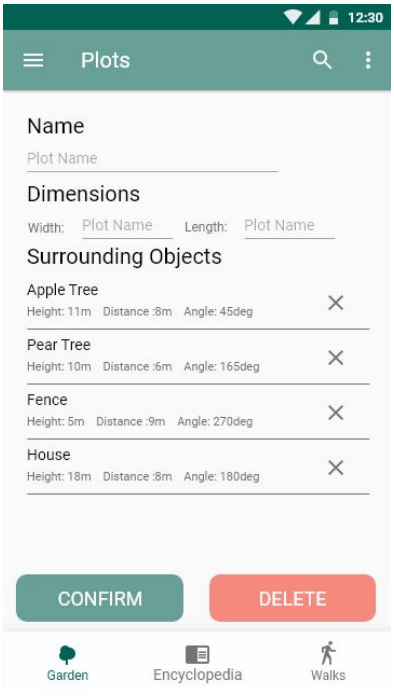


Figure 29 - *Create Plot*
The screen for creating a new plot, featuring input options for name, width, length and surrounding objects. This screen is also re-used for editing a plot, hence the delete button, which would not appear when creating a walk for the first time.

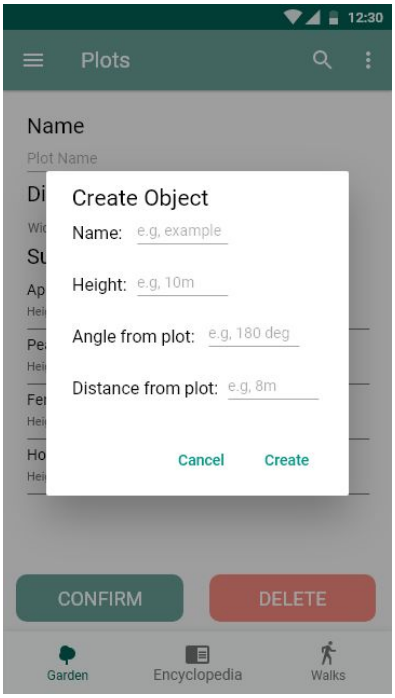


Figure 30 - *Create Object*
A dialog will appear when the user chooses to add a new object to a plot, which will prompt them to enter details such as name, height, angle from plot and distance to plot, before adding to the list in figure 28.

Figure 31 - *Plot View*
This screen will be shown when the user selects an existing plot, and will show them detailed information such as forecasted rainfall and temperatures, as well as information about sun exposure. The colour of information will change depending on the positivity/negativity of each figure.

6.4 - Database Design

A Firestore Cloud database will be used, which is a NoSQL database, so there is no relational structure. However, there will only be on top-level collection in the database - ‘users’. Users will then have two sub-collections; ‘plots’ and ‘walks’. Firestore will automatically convert custom objects to and from types it recognises, therefore as far as database implementation goes, we can think of it as storing those objects as they are, instead of breaking them down into Firestore’s types such as ‘string’, ‘number’, ‘geopoint’, etc. Figure 32 shows an entity relationship diagram for the database, showing how the various objects relate to each other. Even though the database is NoSQL which means there are no ‘relations’ (tables), the data still relates to each other, such as a user being able to have multiple ‘Walks’ while each of those walks can only have one user. The fields that each object contains have been based off of the functional requirements outlined in section 4.4 as well as the technical research carried in section 5.

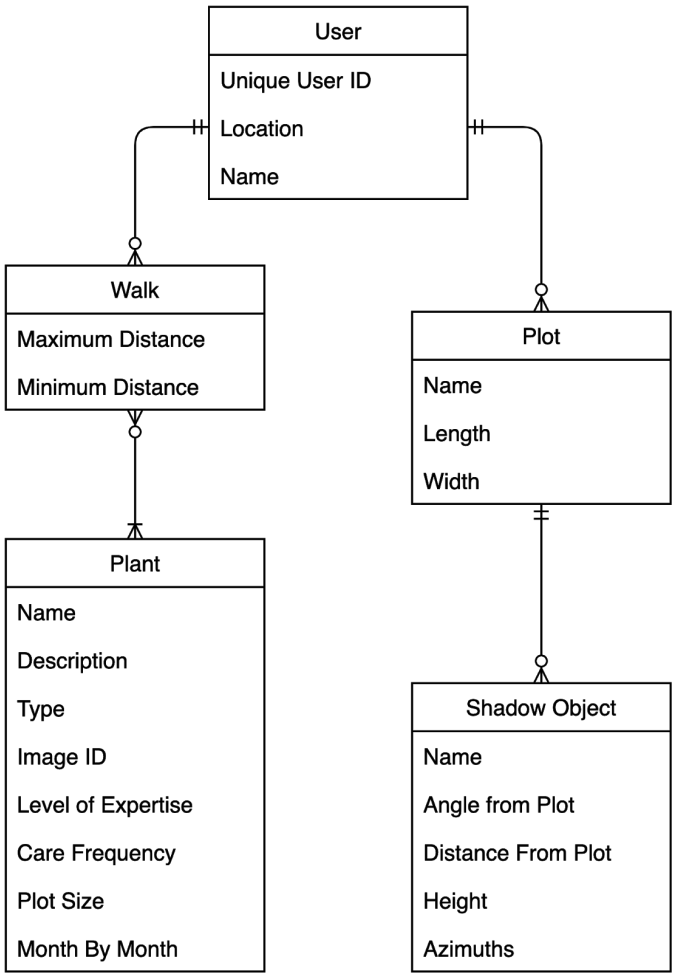


Figure 32 - *Firestore db design*

7 - Implementation

7.1.1 - Introduction

This section will look at the implementation of the application. While the source code will be available to view in its entirety outside of this report, I take sections of this code where relevant to further explain the delivery of the project.

The project has been developed entirely in android studio. I have not explored developing the application to be cross-platform and run on Apple devices. It is unrealistic to accomplish this under the scope of this project. A similar application on an Apple device would require knowledge of a different language, such as Swift or Objective C. Should development extend past the scope of this project, such as if the application were to be brought to market, migrating to Apple would definitely be a goal.

To give an overview of the entire system, I have created a full application flow diagram, as can be seen in figure 33*:*

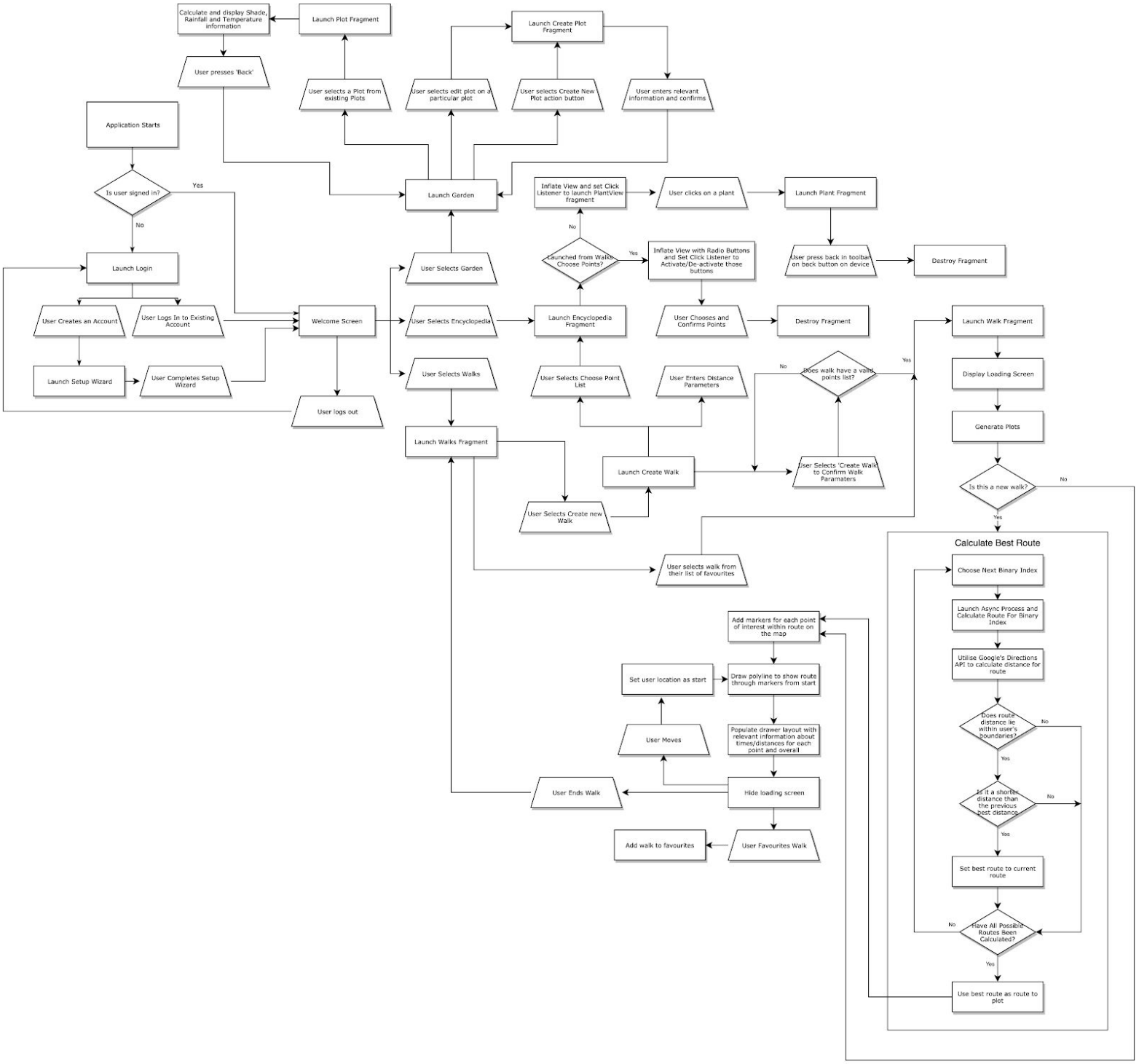


Figure 33 - *Fully implemented application flow chart*

7.1.2 - User Account Creation/Login

Implementation of account creation and login was made simple by Firebase Auth. To launch the login activity, the application simply checks if the user is logged in using Firebase Auth's built in *getCurrentUser()* function. If this function returns *null*, it means there is no user logged in, and the Login activity is launched.

The Login activity was simple, making use of Firebase Auth's *signInWithEmailAndPassword()* method, which would handle all authentication, and simply return a positive or negative response depending on if the authentication was successful or not. If successful, the user information is passed to the Main Activity and the user is allowed to continue to use the application, otherwise they are prompted to try again or can create a new account.

For account creation, a ViewPager was utilised to allow smooth and intuitive progress through the fragments needed for the wizard. I implemented a StateProgressBar [15] - a library which allows the user to view a visual representation of how far through the process they are. This was implemented by simply setting the StateNumber on the progress bar to the relevant setting whenever a new tab on the ViewPager was selected.

The fragments for the setup wizard are configured in a custom PagerAdapter extending from FragmentPagerAdapter, and all information is passed from each Fragment straight back into the SetupWizard activity, meaning when the user has worked their way to the end, all data is already prepared and held in the SetupWizard activity and passed straight into FirebaseAuth's *createUserWithEmailAndPassword* method. A map is passed into this method which allows data to be stored in a database unique to each user as the account is created. The data being stored in this instance is the user's Location and Name.

On a successful account creation, we can see the user is created by visiting the Firebase console and checking on the Authentication tab. Each user has a randomly generated unique User ID.

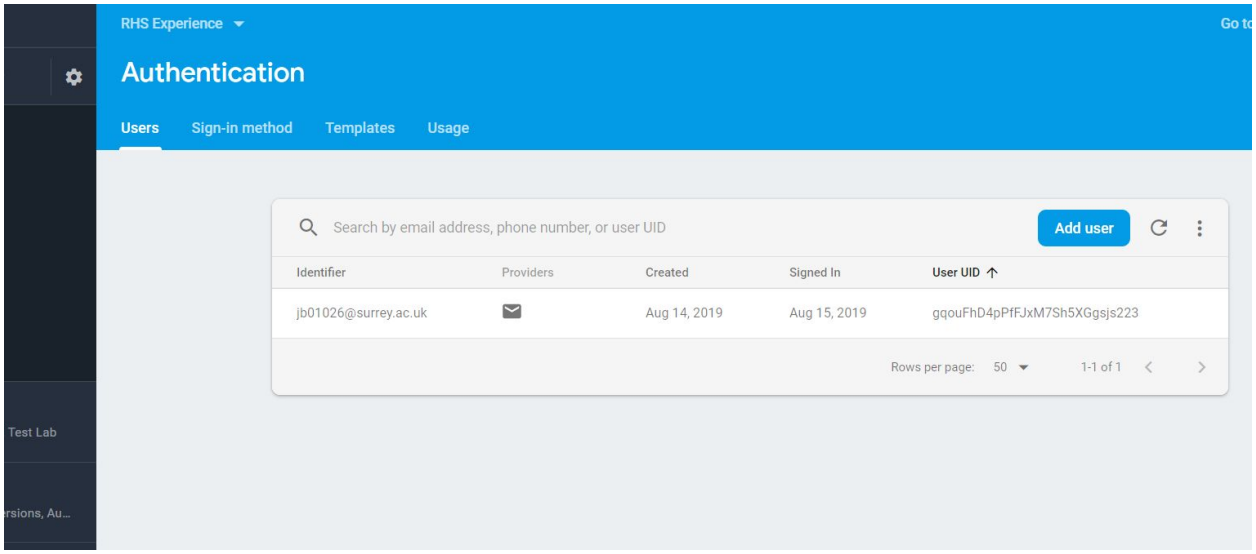


Figure 34 - *Firebase Console Authentication - List of Users*

This User ID is used to create a document in the Firestore database under a collection called ‘users’. As we can see in figure 35 below, the document contains a field for both ‘location’ and ‘name’, and throughout the rest of the application nested collections can be created to store further data for each individual user.

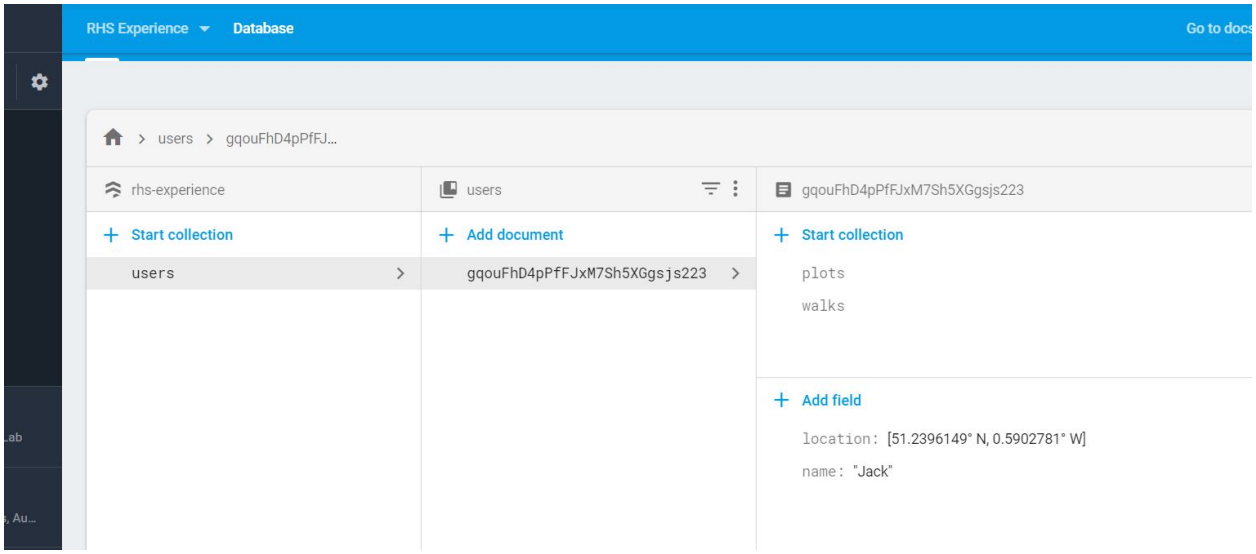


Figure 35 - *Firebase Console - Firestore Database*

7.1.3 - Encyclopedia

The encyclopedia is a small part of the project and closely mimics the ‘Grow Your Own’ application already released by the RHS. I have attempted to make it simpler to use and also added a way to filter plants through a search function and toggleable filters.

Initially this section was implemented through a ListView. However, with further research I discovered that a RecyclerView would be a better option. It is more efficient, meaning it would better cope with much larger lists as the encyclopedia dataset grows. This is due to the fact that cells are reused while scrolling. Rather than having hundreds or more cells, there are only 8 or 9, dependent on how many fit on the screen, which are repopulated with the relevant information on scrolling.

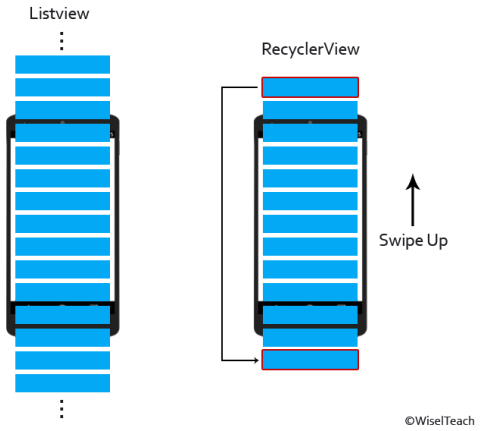


Figure 36 - *RecyclerView Diagram [37]*

RecyclerViews are much more complex than ListViews, and were difficult to understand and implement. Once understood, it was simple to customise more than I was able to with a ListView. For instance, I could add alphabetical dividers within the list by returning a different *CustomViewHolder* if the ‘Plant’ within the list wasn’t actually a plant but was a placeholder object added at a change of letter.

To populate the RecyclerView, I used an SQLiteDatabase and some dummy data stored in a CSV file. I wrote the CSV file myself based on the information accessible within the Grow Your Own application. I assume that the RHS would have a pre-existing database containing all the information used for their website and application. I did not focus on how the dummy data within this application was to be stored/accessed, as it is only a workaround for not having access to the RHS database. This could be arranged with the RHS should the application be launched commercially.

The benefit of storing the data in a CSV file, rather than utilising Firebase’s storage options, is that there is no Internet connection necessary to access the data, as it is all stored on the device. There is a tradeoff however, as this will increase load times and increase the file size on the user’s phone.

After trying and failing to add filters within the ListView, the RecyclerView made it simple to filter results by a string supplied by a SearchView, and update the list in real-time. Data is filtered within the RecyclerView adapter, and the method to filter the data and refresh the list is called

each time the search query is changed, or the ‘Apply’ button is pressed on the filter drawer. It is not yet clear to me whether updating this list in real time will cause issues with much larger list sizes. However this is something I have considered, and should this be the case, I would instead choose to have a ‘Submit’ button used to update the list, instead of doing so as the user types the query.

There is a filter drawer built into the encyclopedia which allows users to filter by plant type, plot size, or any other information specific to certain plants. For this, a submit/apply button was used, as I felt the checks per plant were more complex, and by re-filtering with every check/uncheck within the menu, optimisation issues may have arisen.

The filtering method contains two steps that run every time it’s called. The first is to filter by the string query, which means checking every plant in the list to see if the name contains the query entered by the user, and adding it to a new filtered list if it does. The second part is for the checkbox filters check/not checked within the filter drawer. Every plant in the list produced after filtering by the string query gets tested, where a switch statement is used for every attribute the plant has. For example, a switch statement is defined for the plant’s *getPlotSize()* method, where for the relevant case, e.g, MEDIUM’, the filter method checks if the ‘Medium’ box is still checked. If it is, the next attribute is tested, until all attributes have been tested. If it is found that none of the necessary boxes have been unchecked by the user, the plant is added to the list, and the final list is returned and used for the Recycler adapter.

There is one final method to modify the list before it is displayed to the user in the EncyclopediaFragment. Every time the first letter of a plant changes in the list, for example between ‘Asparagus’ and ‘Basil’ in my dummy data, a new plant is created and added to the list, but instead of ‘Fruit’, ‘Herb’ or ‘Vegetable’, its type is set to ‘Divider’ and name set to the new letter. The RecylerViewAdapter then displays a different view for these ‘Divider’ plants, simply displaying the new letter. This makes navigating, what could be a long list of plants simpler visually, especially with the addition of a MaterialScrollBar [16] .

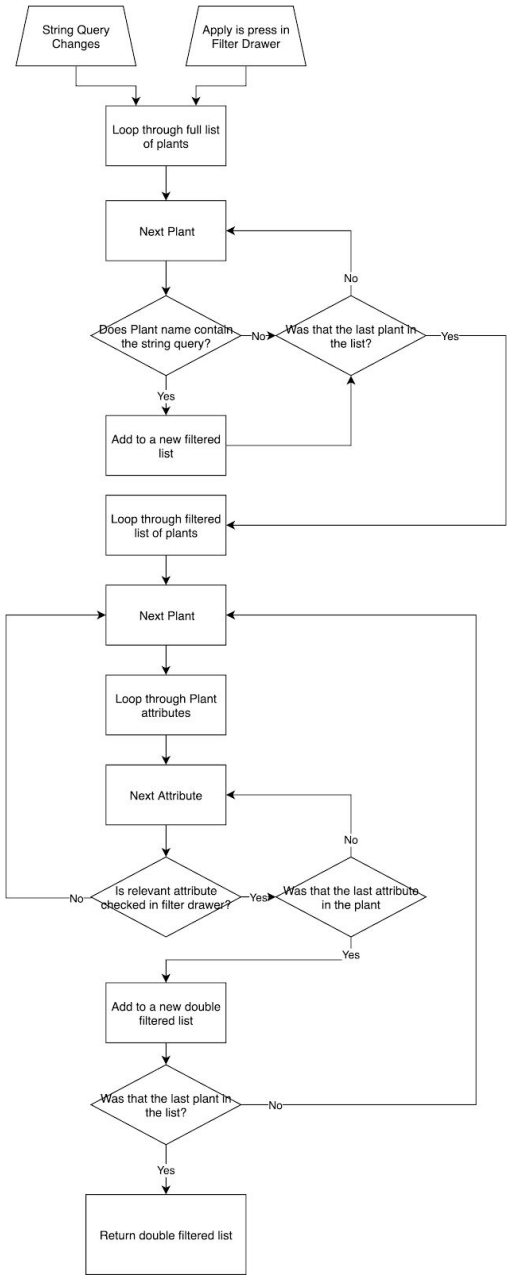


Figure 37 - *Filter Flow Chart*

MaterialScrollBar is a library which lets the user scroll up/down a RecyclerView, and is able to show them a label to let them know where they are in the list. Implementing this was simple through the built in AlphabetIndicator class.

7.1.4 - Walks

For the purposes of this application, the actual physical location of each potential point of interest was unimportant, and as such, were randomly generated within certain LatLng bounds from a given starting point. For testing purposes, this location was set to my home, and the LatLng bounds allowed the points to be spread over Guildford town center. This was useful for testing distances/times for walking as there are many streets and footways on Google Maps for pathing.

Forty plants (two of each of the sample data set) are generated in total, completely randomly using Java's Math library to generate a random number. Because of the way the Walk fragment is implemented, these points are regenerated each time a walk is selected or created. This leads to some confusion for favourite walks which will change routes and distances each time they are selected. This will even occasionally lead to a walk that may have had a valid route through the points within the distance parameters, not having a valid route when the points regenerate. For the purposes of this project, this is acceptable, but not ideal.

The user is able to select a maximum and minimum distance for their walk, as well as the list of plants they wish to visit (up to a limit of seven as discussed later on). The route generated will be the shortest possible route adhering to these guidelines, and if there is not an acceptable route available, the user will be returned to the creation screen where they are prompted to alter their parameters.

I needed to use Google's Directions API for the purpose of visually plotting my route, and getting data such as time and distance. This API already has a Travelling Salesman solution built in through the use of their 'optimize waypoints' parameter, so implementing my own solution was unnecessary. Implementing the Directions API was not simple.

I used a separate DirectionHelper class to interface with the API. It extends AsyncTask so that it can run asynchronously to the other methods within the Walk fragment. As the Directions API does not have an Android library I could use to access it, I had to build a url in the correct format, and then read and parse the JSON returned from the API. Having had little experience with parsing JSON, it was a challenging problem that required substantial research. Much of the code I ended up adapting from various answers on StackOverflow.

There was one main question and answer that ended up being more useful [35]. The *decodePoly()* method was lifted directly from an answer in this thread, and much of the rest of the DirectionsHelper class was written by taking fragments of various answers from this thread, and merging and adapting them to create a working helper class. Google's developer guide [39]

was vital in helping me understand the code, and the changes I needed to make to any pre-existing code in order to make it work for my purposes.

To summarise how the class works, a URL in the form of "https://maps.googleapis.com/maps/api/directions/json?origin=place_id:PLACE_ID&destination=place_id:PLACE_ID&key=YOUR_API_KEY" is built with any additional parameters, such as 'waypoints' in my case. Each point of interest has its LatLng values converted to display as *'latitude,longitude*|*'* and added to the URL. Importantly the travel mode is set to walking with *'&mode=walking"* and waypoints are optimised with *'waypoints=optimize:true*|*'*. By optimising waypoints, as discussed in my earlier technical research, Google's own Travelling Salesman solution is used to find the best route through the given waypoints.

A HTTP request is then sent with the URL that has just been built, and a JSON file is returned, similar to the one seen in figure 38.

This JSON is parsed into a list of LatLngs, each decoded using the *'decodedPoly()'* method so that a Polyline could be drawn on the map. The overall distance and time, as well as the distance and time between each leg was also parsed to be used in the BottomSheet information, along with the order of the waypoints so they could be correctly displayed in this view.

Custom Map Markers were created from a custom made marker shape containing the same images used in the Encyclopedia section. Creating these markers was challenging as I do not fully understand the manipulation of Bitmaps in Android development. I experimented with and adapted code from dozens of StackOverflow question on the topic to use my Plant object and drawable resources.

Despite the help received from online resources such as StackOverflow, this whole section proved to be one of the most challenging and time consuming problems I faced within the project.


```
{
   "geocoded_waypoints" : [
      {
         "geocoder_status" : "OK",
         "place_id" : "ChIJPRlxxKLQdUgR4rR7j-4E8DI",
         "types" : [ "premise" ]
      },
      {
         "geocoder_status" : "OK",
         "place_id" : "ChIJObdKDpjQdUgRlRR7qo_aRjs",
         "types" : [ "street_address" ]
      },
      {
         "geocoder_status" : "OK",
         "place_id" : "ChIJcyGT05DQdUgR2InLzdIR4vw",
         "types" : [ "street_address" ]
      },
      {
         "geocoder_status" : "OK",
         "place_id" : "ChIJpye4FJrQdUgR9k2VHw36ZiE",
         "types" : [ "street_address" ]
      },
      {
         "geocoder_status" : "OK",
         "place_id" : "ChIJpY2plp7QdUgRE6ulxvDVdYs",
         "types" : [ "establishment", "point_of_interest" ]
      },
      {
         "geocoder_status" : "OK",
         "place_id" : "ChIJPRlxxKLQdUgR4rR7j-4E8DI",
         "types" : [ "premise" ]
      }
   ],
   "routes" : [
      {
         "bounds" : {
            "northeast" : {
               "lat" : 51.2424857,
               "lng" : -0.5651651999999999
            },
            "southwest" : {
               "lat" : 51.2330994,
               "lng" : -0.5744787
            }
         },
         "copyrights" : "Map data ©2019",
         "legs" : [
            {
               "distance" : {
                  "text" : "0.6 km",
                  "value" : 561
               },
               "duration" : {
                  "text" : "7 mins",
                  "value" : 435
               },
               "end_address" : "17 Stoke Mews, Stoke Rd, Guildford GU1 4DZ, UK",
               "end_location" : {
                  "lat" : 51.2389327,
                  "lng" : -0.5720898
               },
               "start_address" : "The Stoke, 103 Stoke Rd, Guildford GU1 4JN, UK",
               "start_location" : {
                  "lat" : 51.2424857,
                  "lng" : -0.5713910999999999
               },
               "steps" : [
                  {
                     "distance" : {
                        "text" : "0.4 km",
                        "value" : 432
                     },
                     "duration" : {
                        "text" : "6 mins",
                        "value" : 343
                     },
                     "end_location" : {
                        "lat" : 51.2386523,
```


Figure 38 - *Directions JSON*

However, there was still the issue of each plant having multiple locations. At this point, I was successfully drawing a polyline through Google's best guess at the fastest route between the given points. However, in the early stages, these points were predetermined for testing purposes, so I had to give the users an option to choose these points for themselves. Technically this wasn't difficult to do, give the user a choice of plants with a limit, and use these points and let Google do all the work once again. The encyclopedia section of the application

could be re-used for selection, with the simple difference of radio-buttons being used for selection.

Following my research explained in section 5.2.2, I decided due to time and resource constraints, that brute forcing a best route would be best suited for the scope of this project. This left me with two options in order to keep loading times and API calls as low as possible, either limit the user to a small number of points, with only two possibilities for each point, and brute force it, or alternatively limit each point to having only one possible location, and not limit the user on points. Neither option is ideal in a commercial implementation of the application. A better solution would have to be implemented in order to avoid either of these limitations. This will be discussed further in the evaluation section.

I decided brute forcing it on a small scale was a better solution as it better demonstrated the proof of concept by having multiple options per point of interest. This is more representative of a real world visitor experience application. The limitations set were seven points of interest with two possible locations for each point, meaning the total number of possible routes is $n = 2^7 = 128$. Few enough that brute forcing would be trivial, even when doing so through Google's api calls. At the maximum amount of calls, this task takes roughly 30 seconds. This figure will vary depending on the quality of internet connection.

I would be running my DirectionsHelper method for every possible combination of points. At first I struggled to grasp where to start when it came to realising this problem in code, until I began visualising each plant as a binary digit. In my implementation all plants have two possible locations, each location could be represented as a 0 or a 1. To further visualise it I drew out a list of all possible combinations for a route through four plants, and realised each combination could be simply displayed as a binary representation of a number between 0 and the maximum possible routes - 1, in this case $2^4 - 1 = 15$.

0000	0001	0010	0011
0100	0101	0110	0111
1000	1001	1010	1011
1100	1101	1110	1111

In my implementation I tried to keep in mind that, while I wouldn't be using it for this project, I should aim to have my method work for n locations per plant, rather than two, if I could. As this is what would be needed for a commercial realisation of this project.

My first attempt was a crude nested for-loop, the pseudocode for which can be seen in figure 39.

Although it worked, apart from not being very pretty or sophisticated, it had a few issues:

- I needed to be able to change how many nested levels there were dynamically depending on how many points the user chose to put in the list.
- It does not take into account the possibility of a point of interest having more or less locations than two, which was one of my aims.

```
for (int i = 0; i < 2; i++) {
    for (int j = 0; j < 2; j++) {
        for (int k = 0; k < 2; k++) {
            ...
                ...
                    ...
                        for (int n = 0; n < 2; n++) {
                            List<LatLng> plotList;
                            plotList.add(pointOfInterest1.get(i))
                            plotList.add(pointOfInterest1.get(j))
                            plotList.add(pointOfInterest1.get(k))
                            ...
                            plotList.add(pointOfInterest1.get(n))

                            Route r = new Route(entrance, plotList)

                            newDirectionsHelper(asyncResponse).execute(r)''
                        }
        }
    }
}
```

Figure 39 - *Pseudocode for a nested 'for' loop that was later scrapped*

I decided to research recursion as this seemed to be the answer to my issues. A particular StackOverflow answer [40] lead me to a more elegant solution that solved the issues I had with my earlier implementation.

For each call of my DirectionsHelper, I compared the result with the previous best, and if the new result was better, set that as the new best result. Once all possible routes had been checked and compared, the best route was plotted on the map and all relating information was entered where needed.

Adding a walk to favourites was made very simple thanks to Firestore. The Firestore database is initialised as firedb with *getInstance()* at the beginning of the class, and the user is initialised as fireUser with *getCurrentUser()* then also. It's then as simple as calling:

firedb.collection("users").document(fireUser.getUid()).collection("walks").add(walk)

For any collection or document that doesn't exist when called with '*.collection*' or '*.document*', firebase will create it before it references it, so I don't have to worry about any collections or documents being null or not existing. As outlined in section 7.1.2, all users have a document corresponding to their unique User ID, so it's then just a matter of referencing that and adding a walk to a 'walks' collection inside their document. Firestore lets you store any custom object as long as it adheres to its guidelines of each custom class having a public constructor that takes no arguments, in addition to a public getter for each property [41]. This means I don't even have to worry about converting the object into a certain type or list of types, and converting it back when reading the object from the database. All you have to do is call '*document.toObject(Object.class)*' and it will automatically convert back to the correct object type.

I implemented a working *getLocation()* method in the WalkFragment, but unfortunately due to increasing time pressure sacrificed this feature in order to meet my deadline. However, I will give some insight on how I believe I would have implemented the ability to track the walk as the user moves:

Every time *getLocation()* is called, the *Route* object found while calculating the best route would be used in a new DirectionsHelper call, except the 'entrance' variable will change to the current user location. A new polyline will be drawn from that new entrance, but still end at the original entrance, and the times and distances will be updated based on the object that DirectionsHelper will return.

For the percentage completion value for the walk, in the walk fragment there would be a 'walkLength' variable set at the beginning of the walk, and the difference between that distance and the one calculated from the new DirectionsHelper call will be used to work out the percentage.

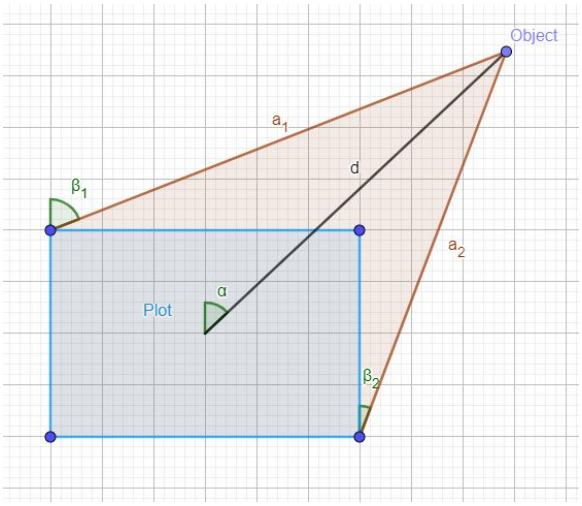
I believe this would be a simple implementation, but it is not a feature that is important to the goals of this application, nor is it particularly innovative or interesting, It's not a crucial loss to the product as a whole.

7.1.5 - Garden

Following my technical research into Sun calculations in section 5.2.1, I had all the information I needed to calculate the shade of a plot in the 'Garden' section of the application. I thought extensively about the best way to display the calculations to users in a way that made sense and wasn't too overwhelming. Visually showing the shadows on a graphical representation of the garden, plot and objects would be the most effective solution. However, I did not have the time nor resources to effectively implement a solution like this.

I decided that informing the users of this information with figures was my best option. I came up with the idea of producing a ‘Shade Rating’ - a figure ranging from 0-100 that indicated how much sunlight could reach the plot throughout the day, with 0 meaning that in the hours between sunrise and sunset, there was always an object shading the plot, and 100 meaning there was nothing shading the plot during these hours. A figure like this strikes a good balance between being helpful for users, and not overwhelming them with lots of different numbers that are hard to visualise, such as the length of shadows and azimuths over the course of a day.

Calculating this in theory was simple. It is effectively a percentage of how often the plot is exposed to sun out of total hours of sunlight in the day. However I had to decide what defined being shaded. For example if something tall and narrow such as a washing line was shading the plot for most of the day, should that mean it is to be counted as ‘shaded’ for the entire time, or as 90+% of the plot will be exposed should it count as exposed. That would mean taking into account the dimensions of the object in all three directions; X, Y and Z, not just its height. An object such as a tree will vary dramatically in its X and Y dimensions dependent on its Z, with very small measurements for X and Y at its base, and very large at its top or mid-section depending on the tree. The application cannot ask for input this detailed on an object, as a standard user is not going to be able to, or likely have a desire to accurately measure these dimensions. I decided to stick to having the relative location of the object to the plot and the height of the object, and define the plot as ‘in shade’ if any objects shadow overlaps with the boundaries of the plot.



Referring to figure 40, we can see that it’s not just when the azimuth of the sun is equal to the angle of the object from the center of the plot (α) that the shadow is cast in the direction of the plot. It is when the azimuth of the sun lies between two angles, (β_1 and β_2) that it will be casting a shadow in the direction of the plot. I attempted to work out on paper a formula to calculate β_1 and β_2, however with much struggle and little progress decided to look into a library that enabled programmatically plotting points and lengths and getting the azimuths from there.

Figure 40 - *Object Shadow Diagram*

I found the ArcGIS Geometry library [42] had everything I need to calculate these additional angles for my Shade Rating. I will outline the method I used to calculate the angles below, with code having been modified for readability.

I began by plotting a point at 0,0 for the center as a reference point. Then to calculate the angles I began by plotting the four points from my plot based on the width and height given to

me by the user. To give an example of plotting one of these points, the upper left point of the plot can be plotted as such:

```
Point upperLeft = new Point( - plot.getWidth() / 2, plot.getLength() / 2 );
```

As dividing the width and length by two gives me the coordinates assuming the plot is centered on (0,0), they just need to be made negative or positive depending on which corner is being plotted.

The point for the object casting the shadow could be plotted as such:

```
Point shadowObject = GeometryEngine.moveGeodetic(
    center, object.getDistanceFromPlot(),
    meters, object.getAngleFromPlot(), deg
)
```

The *moveGeodetic()* method returns a new Point, moved by the given distance at the given azimuth (angle) from the given input Point [43]. The center point was passed as the input point, both the distance from the plot and angle from the plot has been entered by the user thus can be obtained from the *object* object passed to the method, the method also takes in the units.

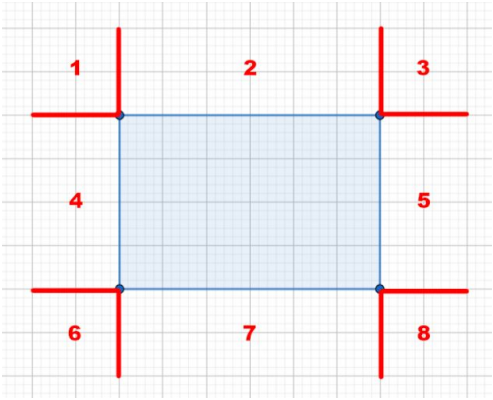


Figure 41 - *Object Segments*

Referring back to figure 40, the next step is to calculate the lengths between the object casting the shadow, and the 'correct' two corners of the plot. The angle of the shadow object from the plot will determine which to corners these will be. As can be seen in figure 41, if the object lies in segment 1, the corners relevant to the calculation would be bottom left and top right, whereas in segment 7, it would be bottom left and bottom right. A simple series of nested if statements was used comparing the (x,y) coordinates of the object to the (x,y) of each corner to determine which segment the object lied in.

The connecting lines between the corners and the object casting the shadow can then be calculated as such:

```
distance1 = distanceGeodetic(shadowObjectPoint, corner1, meters, deg);
```

```
distance2 = distanceGeodetic(shadowObjectPoint, corner2, meters, deg);
```

The azimuths we are looking for can then simply be retrieved as such:

```
azimuth1 = distance1.getAzimuth2()
azimuth2 = distance2.getAzimuth2()
```

The reason *getAzimuth2()* is used and not *getAzimuth1()* is that the former returns azimuth value at point 2 towards point 1, which is what we're looking for in this scenario, while the other does the opposite.

The two values the azimuth of the sun must lie between in order for the shadow to be cast on the plot has been calculated, now the azimuth of the sun itself must be calculated. Referring to the technical research carried out in section 5.2.1, calculating both the altitude and azimuth of the sun was as simple as implementing all of the formula noted in this section in Java, making extensive use of the Java math library. None of the code implementing these calculations itself is worth thoroughly explaining, as the theory behind it has been detailed in great depth in my research.

Once the altitude and azimuth of the sun has been calculated, we can finally begin calculating the sun exposure for my 'Shade Rating'.

There are two sets of methods used to do this. As we are effectively looking for a percentage figure of 'time plot is not being shaded' out of 'total sunlight in a day', there is a set of methods for calculating the 'max sun exposure' and the other to calculate 'actual sun exposure' taking into account objects.

```
public double calcSunExposureDay ( GardenPlot plot, Date date ) {

        double totalHours = 0;

        foreach (0.25hours in 24hours) {
                boolean shaded = false;
                double sunAltitude = calculateAltitude();
                double sunAzimuth = calculateAzimuth();

                if ( altitude > 0 ) {
                        foreach ( object in plot.shadowObjects() ) {

                                if ( object.getAzimuth1 < sunAzimuth < object.getAzimuth2 ) {
                                        if ( object.shadowLength > object.distanceFromPlot ) {
                                                shaded = true;
                                        }
                                }
                        }

                        if ( !shaded ) {
                                totalHours += 0.25;
```

```
                        }
                }
        }
        return totalHours;
}
```

Figure 42 - *Pseudocode for calculating total hours of sun exposure for a given plot*

Referring to figure 42, we can see the general idea of how the method works. For a given day, the method checks every 15 minutes to see if the object would be shaded. It begins by checking if it is between sunrise and sunset (altitude > 0), and if so, if the azimuth of the sun lies between the two azimuth values calculated earlier. If so, if the length of the shadow cast by the object (calculated using the altitude and height - discussed further in my earlier technical research) exceeds that between the object and the plot, it is being shaded. If it is not being shaded, 15 minutes will be added to the total hours, and the loop repeats.

The methods for calculating maximum sun exposure are carried out in the exact same way, except the only check necessary is if it is between sunrise and sunset (altitude > 0), and if so, the 15 minutes are added.

For each set of methods there are variants that calculate actual/maximum sun exposure for a given day, month, or year. These methods simply call each other, for example, the method for calculating the monthly sun exposure calls the method for calculating the daily sun exposure 28-31 times, depending on how many days are in that month, and the same can be said for calculating the sun exposure over a year calling the monthly method 12 times. All three of these values are shown to the user in the plot view, however the shade rating only takes into account the sun exposure for the year, to give the best overview.

That is all of the sun-related information now calculated to be displayed in the relevant fields for a plot. The other values I wanted to calculate for the user which I felt would be most relevant to them was rainfall and temperature statistics, as sunlight, moisture and temperature are three of the most important aspects to consider for the successful cultivation of plants.

There are many weather forecast libraries to consider, however the one I chose was the Dark Sky Api, [12] which offers a wealth of past data and a forecast for up to a week. The API itself works in a similar way to the Directions API, in that I would have to generate a URL, and parse the returned JSON to retrieve the information I was looking for. Fortunately however, an Android Client Wrapper [44] did most of this work for me. Getting this library up and running was simple, and forecasts can be retrieved with a weather request, sent asynchronously, which was perfect as it meant my Sun Calculations could be processed at the same time.

Eight requests are sent in total, the first to get the current data for the day, and then a simple loop to retrieve the requests for the next 7 days. The data returned is then formatted and the colour is adjusted before being injected into the view.

```
private void setColour ( TextView view, int upperBound, int value) {

      colourValue = (value / upperBound) * 100

      colour = HSVToColor ( colourValue , 0.7, 0.95 );

      view.setTextColour( colour);

}
```

Figure 43 - *Pseudocode for a method to set text colour dependent on positivity/negativity of data*

As specified in the UI Designs in section 6.4, I had planned to adjust the colour of each Text View containing data to represent the positivity/negativity of that data. For example a shade rating of 90+ should be green, while a rating of 10 would be red. A high chance of rainfall would be green, whereas a low chance would be red, and a high temperature such as 30 would be green, while 0 and below would be red. I found an efficient way to calculate the colour was to create a method which takes in the view, an 'upperBound' value - which is the value the data would have to be in order to show as a perfect green, and the actual value of the data. The pseudocode for the method can be seen in figure 43.

By calculating a 'colourValue' which is a percentage, it means that I can pass this value directly as the Hue value for a HSV color, as 0 (a negative datum in this case) equates to red, while 100 (a positive datum in this case) equates to green. The saturation and brightness were lowered somewhat to accommodate to a white background and offer more contrast, especially when the colour was close to a yellow.

For the creation/storage/editing/deletion of Plots, much of the methodology is nearly identical to that addressed in section 7.1.4 - Walks. The same Firebase methods are used, with the only difference being the collection in which the plots are stored, and a GardenPlot object being stored instead of a Walk object.

One key difference between the storage of these objects however is I added the ability to edit a Plot after creation. This was also made simple through Firebase, and is almost identical to creating a new Plot, the only difference being;

collections("plots").document("EXISTING_PLOT_ID")

is called instead of;

collections("plots").add.

The same Create Plot view/fragment is used to edit a Plot as the one to create a plot. The only differences are that the data for the plot is automatically entered into the fields when editing a plot, and a Delete button is made visible also.

8 - Evaluation

In this section I will evaluate the product against the Aims and Objectives outlined in section 1.2. I will explain my testing methodology and objectively reflect on my successes and failings, posturing why particular sections may have fallen short of expectations, and considering how things could have been improved.

8.1 - Methodology

Each functional requirement will be manually tested, with the result of the test being evaluated against the expected outcome. These tests will be carried out on my personal mobile device - a Pixel 2 XL running on Android 9.0, Pie: released August 6, 2018. Successful test results will be coupled with screenshots as evidence.

Nonfunctional requirements will also be tested, however the success of these requirements is more subjective.

8.2 - Testing

See appendix for testing table.

8.3 - Results

8.3.1 - Functional Requirements

The majority of functional requirements tested passed, however there were three functions that failed:

Test Number	Requirement Number	Requirement Description	Cause of failure
4	2.2	Sign into account through social media login	Not implemented
17	6.4.4	Recommendations regarding which plant is best suited to plot	Not implemented
25	7.5.6	Details should dynamically update based on user's movement	Not implemented

Requirement 2.2 was a low priority requirement that adds a small improvement to the user experience. Signing in through social media accounts such as Google, Facebook or Twitter is a great way to enable users to start using the application quickly, however not including this feature does not detract from the overall product significantly.

Requirement 6.4.4 is what I believe to be the biggest loss for this project. One of the primary goals of this application was to calculate various data that could be used to recommend planting options to users. However, as this requirement was never implemented, users have to draw their own conclusions based on the figures instead. All of the necessary information to do so was implemented, such as sunlight exposure, rainfall and temperature. The main obstacles holding development of this feature back was my dummy data on plants, and time management issues. Should I have had more time to complete this project, I believe these are the methods I would have attempted to implement to get a functional recommendation system working:

- The width and length data for each plot could have been used to suggest plants that would have adequate space to grow, the data was readily available in the form of the 'Plot Size' in my dummy data set.
- The rainfall over the course of the week could have been utilised to inform the user when they should water their plants, or suggest plants that need less water to grow. The dummy data did not contain information on how often certain plants needed watering.
- Historic temperatures could have been used to determine overall climates, and plants best suited to grow in these climates. Dark Sky API does feature a time machine feature which allows for historic weather retrieval. The dummy data did not contain information about any plant's preferred temperature to build recommendations from.
- The calculated Shade Rating could have been used to recommend users plants that need little sun for plots that aren't exposed to as much sunlight. However dummy data did not contain information about how much sun was needed for particular plants to grow.

Lack of information in dummy data would have to be resolved regardless in a commercial release, but as mentioned previously in this report the RHS would likely have a database with most of this data readily available to access, so it seems as if this aspect would not be a huge issue.

Requirement 7.5.6 would need to be resolved in a commercial release of this product. For the exploratory nature of this project it is not a huge loss to the application, as it was more important to show that the theory behind calculating routes was considered. The ability to track users movement was successfully implemented, but not used, and as discussed in the implementation section of this report, updating the route to show the changing of route as users move would likely be as simple as using another API call to redraw the route, and removing markers from the map and list as they were reached.

8.3.2 - Nonfunctional Requirements

There were two main nonfunctional requirements mentioned in section 4.5; performance requirements and security requirements.

4.5.1 Performance Requirements

The performance requirements stipulated that functions should be optimised in order to reduce loading times throughout the product, and that functions that take longer than three seconds implement a progress bar or loading screen. There are two main areas where computation can take a significant amount of time, and that is in calculating the best route in the Walks section, and calculating plot data in the Garden section. Both of these have successfully implemented loading screens, and so meet these requirements. Within the Encyclopedia section a loading screen was considered for the initial loading of the plant list, however instead I was able to implement a RecylerView instead of a ListView, which allowed for much better performance and thus loaded much quicker. A RecyclerView also allows for more scalability, and loading times shouldn't be heavily affected as the list grows. However as mentioned within the implementation section, it was not clear to me if live filtering with the search query would cause performance issues at larger list sizes. If this were the case, I would likely implement an 'Apply' button similar to the filter drawer, so this filtering only needs to happen when the user finishes typing their query.

As discussed in my technical research, I did consider different approaches to calculating the best route in the Walks section, and how inefficient the method implemented is for large list sizes. Implementing this alternative method would be a primary goal for further development of this project, and performance requirements would be adapted to set a minimum loading time, even for walks with large numbers of plants and many options per plant.

4.5.2 Security Requirements

All data being collected within the application is stored in Firebase services. Email and password information follows the data security procedures set by Firebase Auth, which leverages industry standards such as OAuth 2.0 and OpenID Connect [45]. Passwords are encrypted and not viewable by even the database administrator.

The only other mildly sensitive information being collected from the user is their name and a location entered upon sign-up. These data are not encrypted, and are visible to the database administrator. However they cannot be accessed by other users without access to their account. Signing in to the Firebase console itself requires access to my Google account which is setup with two factor-authentication requiring direct access to my mobile device or email account.

Prior to any data whatsoever being stored off-device, users must also agree to the privacy policy - stating all personal data that will be collected, and its purpose within the application.

8.3.3 - Evaluation Against Project Objectives

This section will refer back to the aims and objectives outlined in section 1.2.

Objective	Description	Achievement
1	Create an application that is sophisticated but simple to use by all audiences, regardless of knowledge, experience or age	I believe this objective has been achieved. The user interface has been designed to feel natural and intuitive, so that progression around different sections feels simple and natural, often staying true to Google's material design recommendations which attempt to mimic natural motions. The creation of walks and plots was not over-complicated when asking for various information, and rough estimates as to the height and angle of things will often suffice. Where the meaning of data was not immediately obvious, such as the 'Shade Rating', a description was provided, that was only visible upon request, meaning users who understand the datum don't have to read the explanation each time.
2	Tailor the application to the user, making it as context-aware as is possible, leaving them feeling personally catered to, while not feeling overwhelmed with information they don't want or need	I believe this objective has been achieved. Users are able to enter some initial information about their location and garden layouts, which once entered does not have to be thought about. Complex calculations which are able to suggest sun exposure based on your location, surrounding objects and plot dimensions are then calculated specifically for your garden to access whenever you need them. None of this information is intruding upon you or even needs to be configured should the user simply wish to just use the application for a walk guide while at the site.
3	Extend user engagement past leaving the attraction, promoting development into a hobby and encouraging repeat custom.	I believe this objective has been mostly achieved. The Garden section has been designed specifically for this purpose, and is able to present the user with data which will help them succeed in managing their own garden at home. However should I have been able to implement requirement 6.4.4 as discussed earlier in this evaluation section, it could have linked the RHS' expertise on specific plants to the garden and made it simpler for customers to draw conclusions, and a recommendations system could have helped further with this. Perhaps also

		referring customers to RHS' shop where they can purchase the supplies would also have promoted repeat custom and furthered achieving this goal.
4	Consider security within features of the application, especially surrounding the storage of data gathered about the user	I have discussed and considered the security of this application in many different areas of this report, including the evaluation of my nonfunctional requirements. I do believe my consideration has been sufficient to fulfil this requirement, but perhaps implementing my own end-to-end encryption for data such as user's address could have furthered achieving this goal

8.3.4 - User Acceptance Testing

My application was shared among a small group of my peers for some brief feedback on the application as a whole. Due to both time constraints, as well as an unforeseen scheduling difficulties due to my having to go through Extenuating Circumstances, and most of my peers moving on, it was difficult to establish a large group of users and obtain very much feedback. However from those who were willing and able, there were both positive and negative comments, as well as some improvements suggested that were acted upon:

- One user found an issue while creating Plots and Objects, whereby if an object has its 'Distance from Plot' field set low enough that it was found to be within the boundaries of the plot on calculation, the application would crash. I adjusted my code to ignore any such objects and avoid the crash, and added some information the user could read explaining how to input the distance accurately, and the outcome of entering an incorrect distance.
- General comments regarding the user interface were positive, with terms such as 'Clean' and 'Easy to use' being used. However there was some negative feedback about the dynamic colouring of data in the Plot View, where values which were scored in the middle of positive and negative, and were therefore given a 'yellow-ish' colouring were difficult to see. This is something I agree with and would tweak before a commercial release.
- There were a few reports of scaling issues to do with text not being full visible on smaller screens or for larger font sizes. Functional testing was only carried out on one device, and before a commercial release would have to be tested on a much larger range of devices.

9 - Conclusion

Overall I believe this project has been an interesting and successful exploration into the use of mobile applications within the Visitor Attraction industry with an aim to improve customer engagement. This is a market that has been around for a very long time and is unlikely to disappear any time soon. There are many exhibits with applications aiming to fulfil a similar purpose, as discovered in my literature review, however I believe the additional key aspect of giving users a way to extend their engagement past visitation and encouraging repeat custom is something that holds a lot of promise and could definitely be expanded upon in the future. Some additional thoughts on my feelings about improvements that could be made and certain issues that arose throughout this project can be found below.

Further Improvements

The ‘Garden’ section falls short of my original vision in the conception of this project. Prior to any technical research I was hoping to implement not only a recommendation system, but a much more visual experience for the user. Utilising technologies similar to that in web and mobile applications such as Gardena or HomeOutside, combined with the sun calculations and plot system for this application could make for a really interesting feature. Users could feel much more engaged in their garden if they could watch their garden progress towards their virtualisation designed within the application. I did spend a significant amount of time experimenting with the MotionViews [3] project alongside similar libraries and technologies, and was able to envisage their usefulness in the context of the Garden, however knew it would be too large of a commitment considering the other features of this project that needed to be developed within the timescale.

I am also disappointed I could not delve further into the research regarding a shortest-path first solution to calculating a best route where points had a variety of options. Again due to time constraints I was unable to invest my time far beyond just considering the effects this algorithm would have to my loading times on a large scale.

I foresee both of these features being fairly large-scale projects themselves, and ones I would be very interested in exploring should I find the opportunity in the future.

Another problem I had considered in my initial research is internet connectivity issues for outdoor areas. While mobile connectivity is constantly improving, with 3G and 4G coverage being very widespread, there are sure to be areas of certain gardens where this is lacking. This is a problem that has been tackled in works by Brandon Hardy as mentioned in my research, however I decided implementation of this solution was outside the scope of this project. It is definitely a feature that could be considered should further development of this project occur.

Time Management

Time management is something I have had difficulty with throughout my education, so I decided to make it a major consideration within this project, as can be seen from section 1.2, where I drew up a Gantt chart and decided to attempt to refine the scope of this project in certain areas. However, this aspect became an even greater challenge, as unexpected health issues arose during the most intense point of implementation, which rendered me unable to type for over six weeks, even having to undergo physio-therapy to regain a full range of motion and work to my full efficiency once again. It was a difficult task getting back into the same mentality and flow I was in prior to the break, however it can be viewed as a positive learning experience, as there are almost always unforeseen difficulties to any project, and learning how to adapt and overcome these obstacles is a valuable skill that is applicable to all areas of life, both professionally and personally. So while I was unable to stick to my rigid schedule and meet the original project delivery date, I am pleased that I was able to adapt to the change of circumstances and adjust my plans accordingly.

10 - Statement of Ethics

This Statement of Ethics will offer my reflections on the legal, ethical, social and professional considerations of the research that went into, as well as the application of this project. I will outline and analyse four basic principles; ‘Do no harm’, ‘Informed Consent’, ‘Confidentiality of Data’ and ‘Social Responsibility’. Furthermore, this section will explain how I believe the project has met the requirements of the British Computing Society’s Code of Conduct.

Do no harm

The subject of research and discovery within this project is not very sensitive. This project is focused on making improvements with a certain industry, as well as minor quality of life improvements which may help a customer enjoy their visitation experience a little more. The only opportunity for harm caused by the application is a miscalculation or misjudgement of a walk taking users away from their objective and into somewhere potentially restricted or dangerous. While I do not feel as if there is a significant possibility of this, people should exercise caution when following any kind of navigational guide. I do not believe anything within this project is at risk of breaking any laws or legislation.

Informed Consent

This project does not involve the active participation of human subjects for the purpose of data collection or otherwise. The only data collected is used solely to improve the user experience and allow users to enjoy the full functionality of the application. I therefore do not believe there is an ethical or legal obligation to obtain informed consent from the user. However the user is presented with a privacy policy regarding their data on creation of the user’s account, which will inform them of this information.

Confidentiality of Data

Three pieces of personal information are collected from users of the application, this includes their name, address, and email address. None of this information is made public to anyone but the user themself. My project is aligned with the 8 key principles of the Data Protection Act (DPA). I believe the only issue regarding the storage of data, is that there is no option for the user to delete their account within the application, meaning they cannot remove their data from the database. The only reason this was not implemented was due to time constraints, and a commercial implementation of this project would include this.

Social responsibility

I believe this application promotes social good in a variety of ways. Firstly, it is in aide of an industry that often works hard to protect and display items or locations of cultural or historical significance. The application was designed with a specific interest around supporting the Royal Horticultural Society - the UK's leading gardening charity, which promotes horticulture through a variety of methods, not limited to their gardens. The application also encourages users to talk walks and pursue what can often be a very rewarding hobby in gardening. I do not foresee any possibility of any of the results from this project leading to social harms.

BCS Code of Conduct

The British Computer Society (BCS) Code of Conduct can be broken down into four key principles;

- You make IT for everyone
- Show a you know, learning what you don't
- Respect the organisation or individual you work for
- Keep IT real. Keep IT professional. Pass IT on.

I believe after reviewing the full BCS Code of Conduct that this project not only adheres to these guidelines, but in some cases specifically caters towards them. For example, a point within the first subsection 'promote equal access to the benefits of IT and seek to promote the inclusion of all sectors in society wherever opportunity arise', closely resembles one of the key goals of this project, which was to create an application that can be used 'by all audiences, regardless of knowledge, experience or age'. This is reinforced by my user interface requirements outlined in 4.3.1, which outlined my aim to design the application to be useful to those with minor disabilities such as motor skill issues and visual impairments.

References

[1] “Annual Survey of Visits to Visitor Attractions”, *Visit Britain*, 2018. [Online] Available: https://www.visitbritain.org/annual-survey-visits-visitor-attractions-latest-results [Accessed: 16th August 2019]
[2] “Results from Augmented Reality Museum Visitor Impact Study”, *Cuseum,* 2018. [Online] Available: https://www.culturehive.co.uk/resources/cuseum-results-from-augmented-reality-museum-visitor-impact-study/ [Accessed: 16th August 2019]
[3] “Global Mobile Consumer Survey”, *Deloitte*, 2018. [Online] Available: https://www.deloitte.co.uk/mobileuk/#uk-device-penetration-2 [Accessed: 3rd April 2019]
[4] “Global Mobile Consumer Survey: Frequency by device usage”, *Deloitte*, 2018. [Online] Available: https://www.deloitte.co.uk/mobileuk/#uk-frequency-by-usage-of-device [Accessed: 3rd April 2019]
[5] Uptech, “MotionViews-Android”, GitHub, 2017. [Online] Available: https://github.com/uptechteam/MotionViews-Android [Accessed: 8th April 2019]
[6] Gardena. [Online] Available: https://my-garden.gardena.com/ [Accessed: 8th April 2019]
[7] B Hardy, *Using Mobile and Sensor Technologies to improve Visitor Experience*, University of Surrey, 2018.
[8] Try QA, “What is Agile model – advantages, disadvantages and when to use it?”, 2017. [Online] Available: http://tryqa.com/what-is-agile-model-advantages-disadvantages-and-when-to-use-it/ [Accessed 12th April 2019]
[9] Naveen, “What is Incremental Model in software testing and what are advantages and disadvantages of Incremental Model”, *Testingfreak*, 2015. [Online] Available: http://testingfreak.com/incremental-model-software-testing-advantages-disadvantages-incremental-model/ [Accessed 12th April 2019]
[10] “Mobile Operating System Market Share United Kingdom”, *StatCounter,* 2019. [Online] Accessed: http://gs.statcounter.com/os-market-share/mobile/united-kingdom [Accessed: 5th April 2019]
[11] X Ducrohet, et al. “Android Studio: An IDE built for Android”, *Android Developers Blog*, 2013. [Online] Available: https://android-developers.googleblog.com/2013/05/android-studio-ide-built-for-android.html [Accessed 3rd April 2019]
[12] “Dark Sky API”, *Dark Sky*, 2019. [Online] Available: https://darksky.net/dev [Accessed 1st May]
[13] “Firebase”, *Google Firebase,* 2019. [Online] Available: https://firebase.google.com/ [Accessed 1st May]
[14] “Api Library”, *Google API*, 2019. [Online] Available: https://console.developers.google.com/apis/library [Accessed 1st May]
[15] K Gyan, “StateProgressBar”, *GitHub*, 2018. [Online] Available: https://github.com/kofigyan/StateProgressBar [Accessed 11th May 2019]
[16] W Plaga, “MaterialScrollBar”, *GitHub*, 2018. [Online] Available: https://github.com/turing-tech/MaterialScrollBar [Accessed 15th May 2019]
[17] “ArcGIS Runtime SDK for Android”, *ArcGIS for Developers*, 2019. [Online] Available: https://developers.arcgis.com/android/ [Accessed 23rd May 2019]
[18] “Firestore”, *Google Firebase*, 2019. [Online] Available: https://firebase.google.com/docs/firestore [Accessed 25th May 2019]
[19] “Pricing Plans”, *Google Firebase*, 2019. [Online] Available: https://firebase.google.com/pricing [Accessed 25th May 2019]
[20] “AWS Databases”, *Amazon Web Services*, 2019. [Online] Available: https://aws.amazon.com/products/databases/ [Accessed 25th May 2019]
[21] “AWS Cognito”, *Amazon Web Services*, 2019. [Online] Available: https://aws.amazon.com/cognito/ [Accessed 25th May 2019]
[22] David Maslanka, “Solar Geometry”, 2012. [Online] Available: http://mypages.iit.edu/~maslanka/SolarGeo.pdf [Accessed 27th May 2019]
[23] P Schlyter, “Computing planetary positions - a tutorial with worked examples”, *Home Page of Paul Schlyter,* 2012 (est.) [Online] Available: http://www.stjarnhimlen.se/comp/tutorial.html [Accessed 27th May 2019]

[24] J Satre, “Sun & Moon Position Calculator”, *Satellite Calculations*, 2012. [Online] Available: https://www.satellite-calculations.com/Satellite/suncalc.htm [Accessed 27th May 2019]
[25] D Darling, “orbital element”, *David Darling*, 2016. [Online] Available: http://www.daviddarling.info/encyclopedia/O/orbital_element.html [Accessed 27th May 2019]
[26] D Darling, “longitude of perihelion”, *David Darling*, 2016. [Online] Available: http://www.daviddarling.info/encyclopedia/L/longitude_of_perihelion.html [Accessed 27th May 2019]
[27] D Darling, “conic sections”, *David Darling*, 2016. [Online] Available: http://www.daviddarling.info/encyclopedia/C/conic.html#eccentricity [Accessed 27th May 2019]
[28] “Aspis”, *Wikipedia.* [Online] Available: https://en.wikipedia.org/wiki/Apsis [Accessed 27th May 2019]
[29] “Ecliptic”, *Wikipedia.* [Online] Available: https://en.wikipedia.org/wiki/Ecliptic [Accessed 27th May 2019]
[30] “Axial Tilt”, *Wikipedia.* [Online] Available: https://en.wikipedia.org/wiki/Axial_tilt [Accessed 27th May 2019]
[31] “True Anomaly”, *Wikipedia.* [Online] Available: https://en.wikipedia.org/wiki/True_anomaly [Accessed 27th May 2019]
[32] “Eccentric Anomaly”, *Wikipedia.* [Online] Available: https://en.wikipedia.org/wiki/Eccentric_anomaly [Accessed 27th May 2019]
[33] “Right Ascension”, *Wikipedia.* [Online] Available: https://en.wikipedia.org/wiki/Right_ascension [Accessed 27th May 2019]
[34] “Sidereal Time”, *Wikipedia.* [Online] Available: https://en.wikipedia.org/wiki/Sidereal_time [Accessed 27th May 2019]
[35] “Travelling Salesman Problem”, *Wikipedia.* [Online] Available: https://en.wikipedia.org/wiki/Travelling_salesman_problem [Accessed 3rd June 2019]
[36] “Sticker sheets & icons”, *Material*. [Online] Available: https://material.io/archive/guidelines/resources/sticker-sheets-icons.html [Accessed 11th April 2019]
[37] R Suvariya, “DiffUtils : Improving performance of RecyclerView”, 2017. [Online] Available: https://medium.com/mindorks/diffutils-improving-performance-of-recyclerview-102b254a9e4a [Accessed 4th May 2019]
[38] K Ramani, “How do I draw a route, along an existing road, between two points?”, 2018. [Online] Available: https://stackoverflow.com/questions/47492459/how-do-i-draw-a-route-along-an-existing-road-between-two-points [Accessed 6th June 2019]
[39] Directions API Developer Guide, *Google Developer*, 2019. [Online] Available: https://developers.google.com/maps/documentation/directions/intro [Accessed 7th June 2019]
[40] J Neely, “Is there any way to do n-level nested loops in Java?”, *Github*, 2009. [Online] Available: https://stackoverflow.com/questions/426878/is-there-any-way-to-do-n-level-nested-loops-in-java [Accessed 7th June 2019]
[41] “Add data to Cloud Firestore”, *Google Firestore,* 2019. [Online] Available: https://firebase.google.com/docs/firestore/manage-data/add-data#custom_objects [Accessed 25th May 2019]
[42] “Package com.esri.arcgisruntime.geometry”, *ArcGIS for Developers*, 2019. [Online] Available: https://developers.arcgis.com/android/latest/api-reference/reference/com/esri/arcgisruntime/geometry/package-summary.html [Accessed 6th June 2019]
[43] “Class GeometryEngine”, *ArcGIS for Developers,* 2019. [Online] Available: https://developers.arcgis.com/android/latest/api-reference/reference/com/esri/arcgisruntime/geometry/GeometryEngine.html [Accessed 7th June 2019]
[44] J Hiott, “DarkSkyApi”, *GitHub*, 2019. [Online] Available: https://github.com/johnhiott/DarkSkyApi [Accessed 11th July 2019]
[45] “Firebase Authentication”, *Google Firebase*, 2019. [Online] Available: https://firebase.google.com/docs/auth [Accessed August 14th]

Appendix

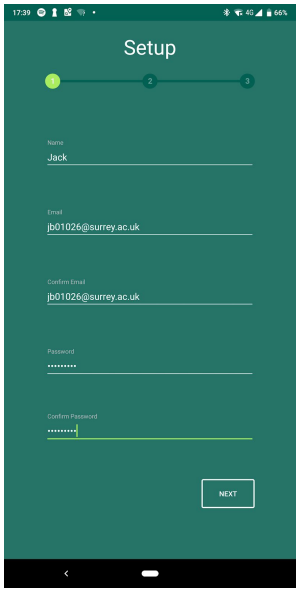
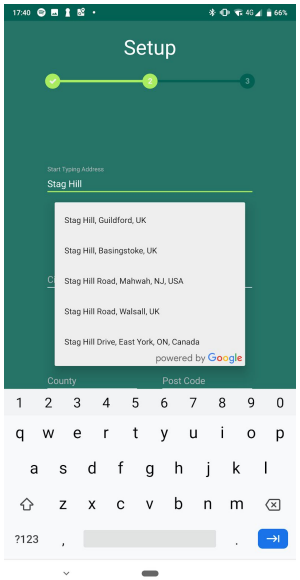
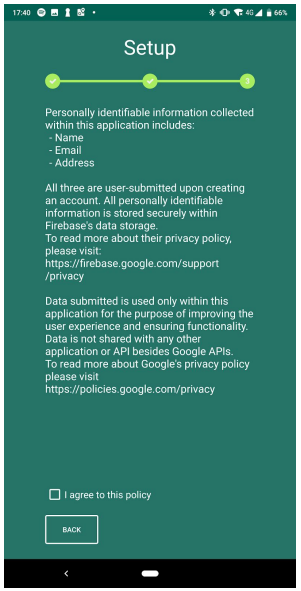
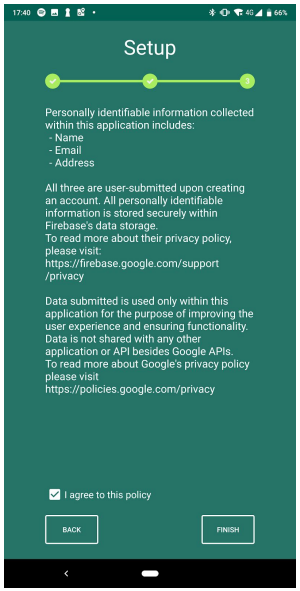
Code Instructions

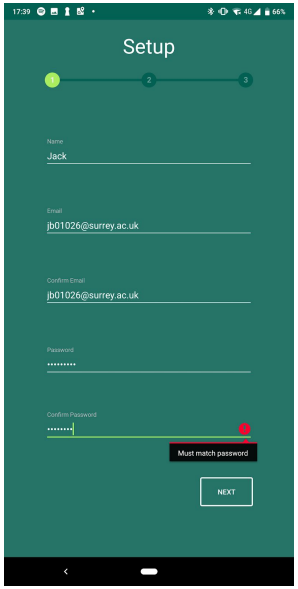
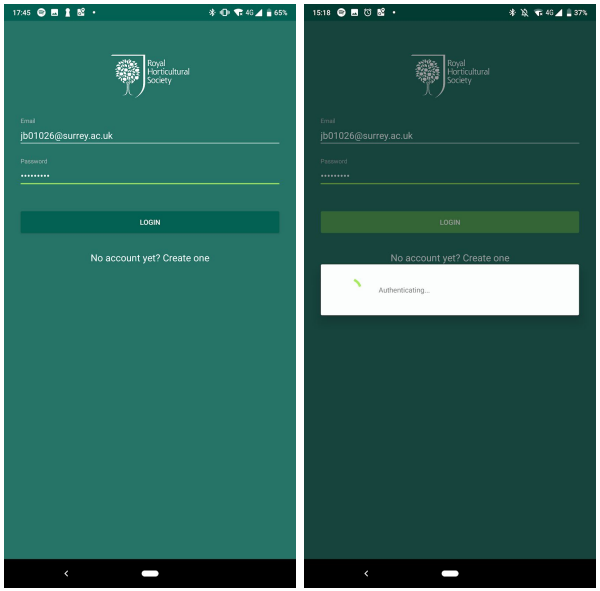
My software was developed in Android Studio 3.2.1 on a Windows 10 Machine.
JRE: 1.8.0_152-release-1136-bo6 amd64
JVM: OpenJDK 64-Bit Server VM by JetBrains s.r.o

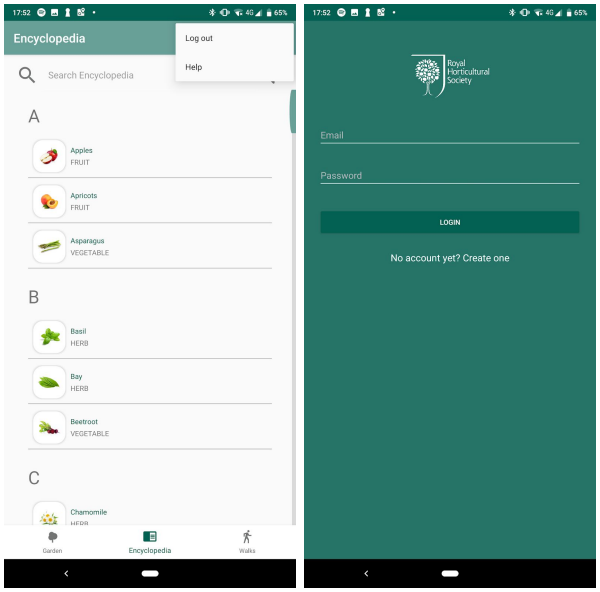
To run the code, import the project into Android Studio, and build. Code has been tested on a Pixel 2XL running Android 9.0. Application should be tested on Android 9.0.

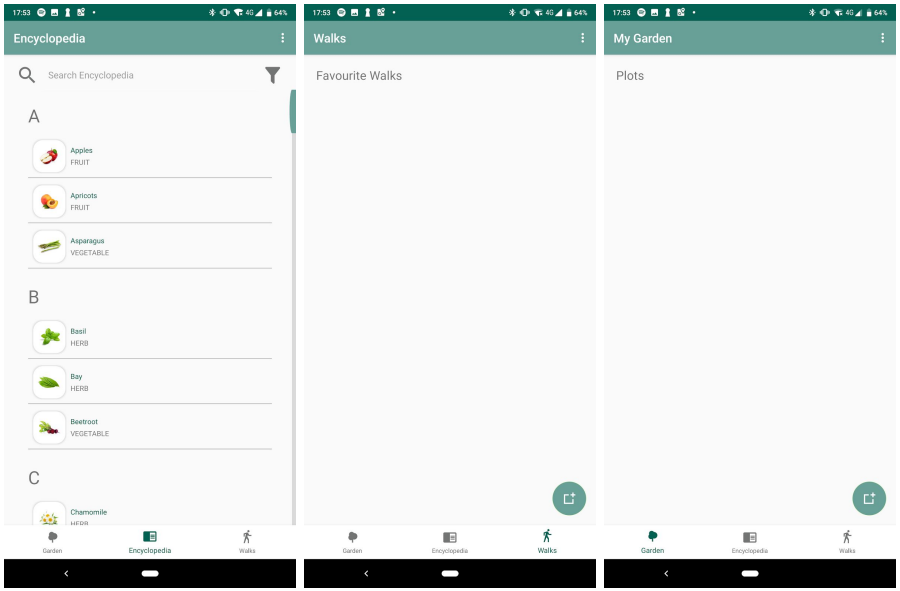
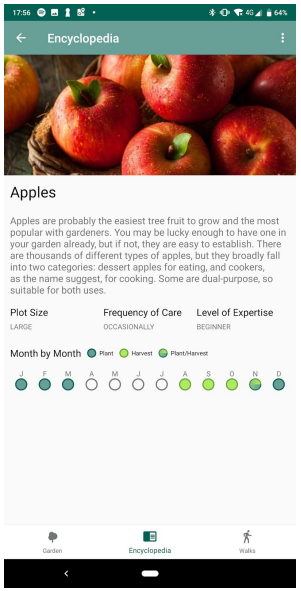
If there are any issues building on a different system, the APK can be manually installed on an Android device. The APK can be located in the same directory at: root\app\build\outputs\apk\debug\app-debug.apk.

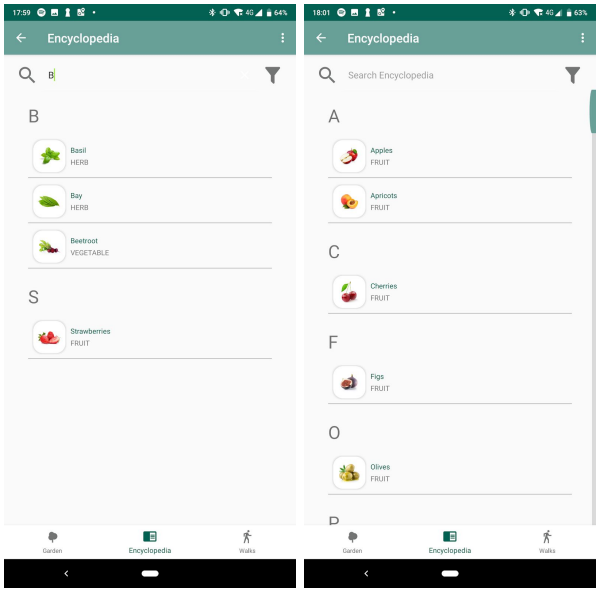
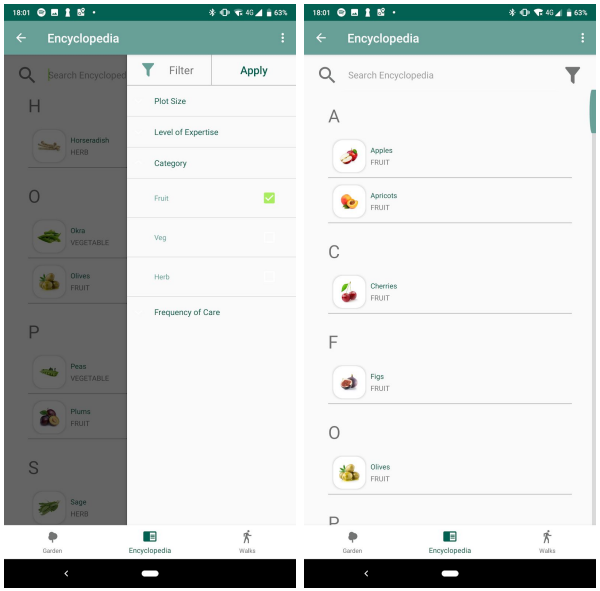
Testing Table

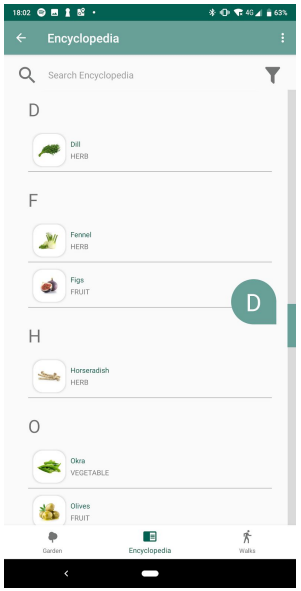
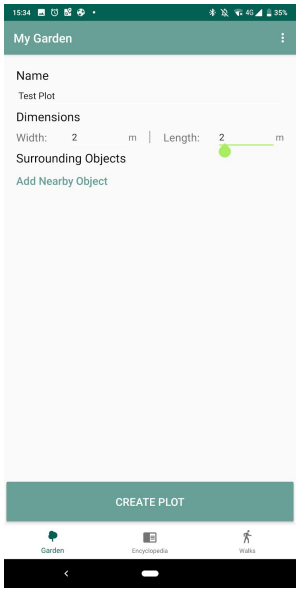
Test no.	Requirement no.	Test Method	Expected Result	Actual Result	Test passed?
1	1.1 - 1.3	Open application, click ‘Create an Account’ and proceed through the steps, checking every field takes input correctly.	All fields allow data entry, with specific data type stipulations where applicable, for example the address finder being successful and filling in fields automatically	All fields allowed data entry and location entries automatically filled it from initial location choice.     	Yes

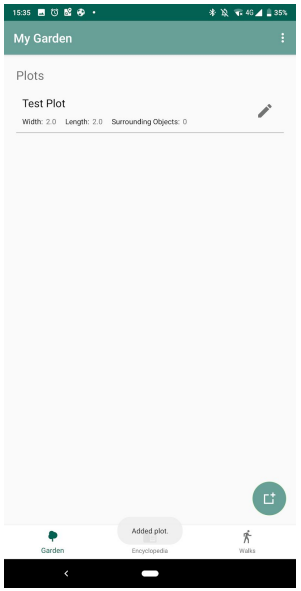
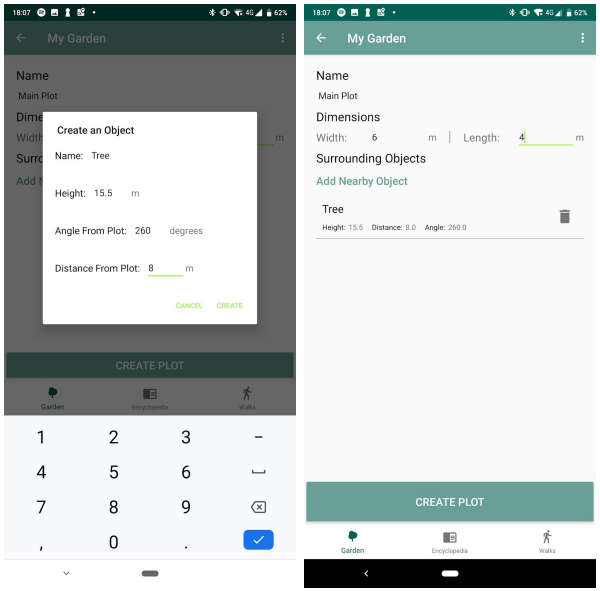
2	1.5	Open application, click ‘Create an Account’ and mistype a field, for example ‘Password Confirm’ does not match ‘Password’. Press next with incorrect data entered.	The application should flag up the error, and not allow the user to proceed until it has been fixed.	The application flagged up the error and would not let the user proceed until it was fixed. 	Yes
3	2.1	Open application, enter valid email and password to existing account and press ‘Login’.	Application should successfully authenticate the user and proceed to the main activity.	Application successfully authenticated user and proceeded to main activity . 	Yes

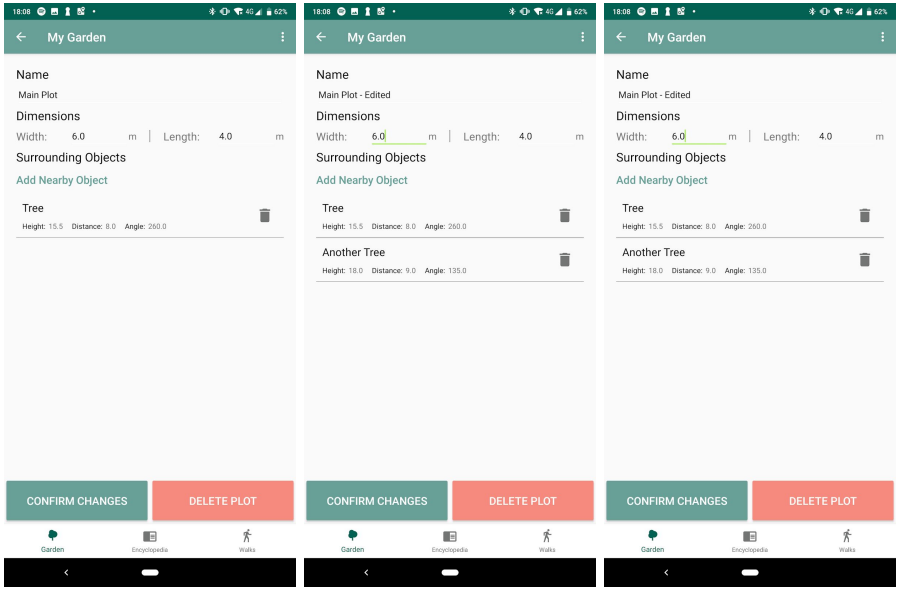
4	2.2	Open application, select ‘Login by’ *preferred social media account*, sign into this account and proceed.	Application should successfully authenticate the user and proceed to the main activity.	No login by social media options available - not implemented.	No
5	3.1	Open application and login to an account, if not already logged in. From anywhere within the app, select the menu button in the toolbar and press ‘Logout’	Application should successfully log out the user and return them to the login screen.	Application successfully logged out the user and returned them to the login screen. 	Yes
6	4.1 - 4.3	Open application and login to an account, if not already logged in. Press any of the tabs on the bottom navigation view.	User should navigated to the tab they pressed - the relevant fragment launches.	User was navigated to the tab they pressed - the relevant fragment launched.	Yes

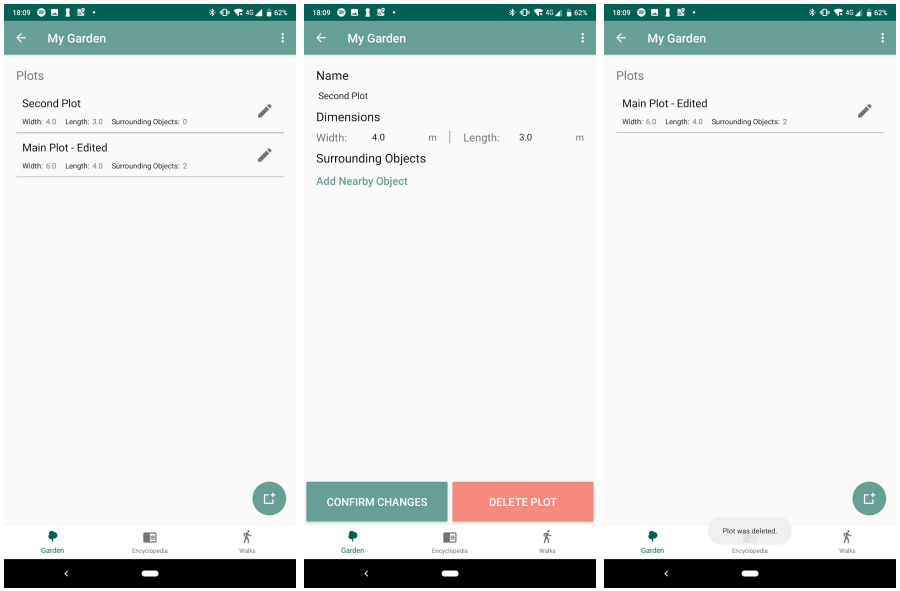
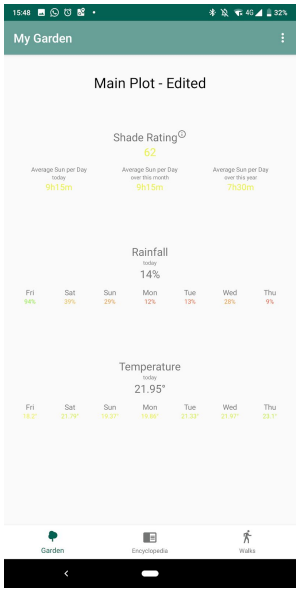
					
7	5.1	Open application and login to an account, if not already logged in. Navigate to the Encyclopedia section and select a plant.	PlantFragment should launch showing the user all relevant information about the plant.	PlantFragment was launched with all the relevant information correct to the plant they chose. 	Yes
8	5.2	Open application and login to an account, if not already logged in. Navigate to the Encyclopedia section	The list of plants in Encyclopedia should change to match the search query input by the user.	The list of plants in Encyclopedia changed to match the search query input by the user.	Yes

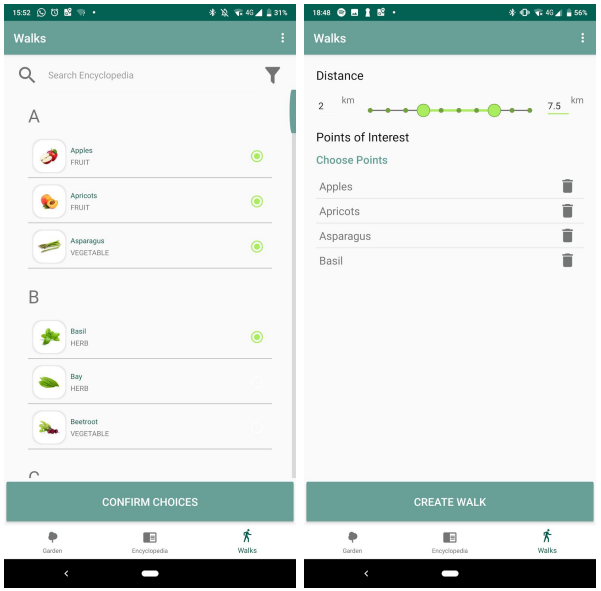
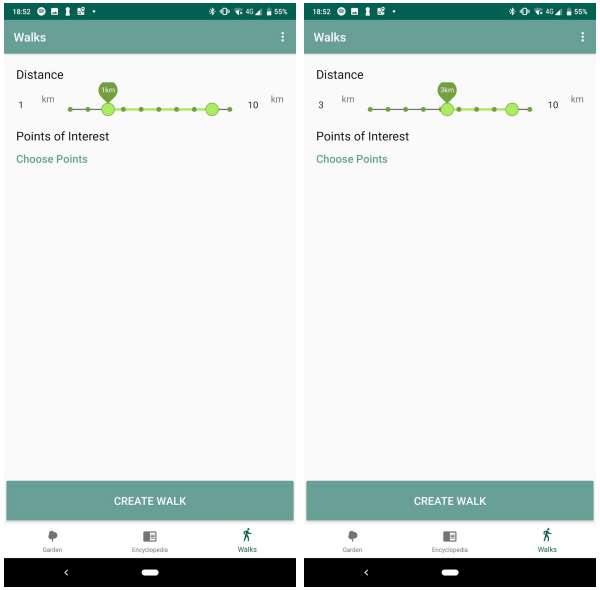
		and begin typing in the SearchView at the top.			
9	5.3	Open application and login to an account, if not already logged in. Navigate to the Encyclopedia section and open the filter drawer. Deselect one or more of the filters and click apply.	The list of plants in Encyclopedia should change to match the filter query input by the user.	The list of plants in Encyclopedia changed to match the filter query input by the user. 	Yes
10	5.4	Open application and login to an account, if not already logged in. Navigate to the Encyclopedia section	The scrollbar should animate outwards and a letter indicator should appear.	The scrollbar animates outwards and a letter indicator appears.	Yes

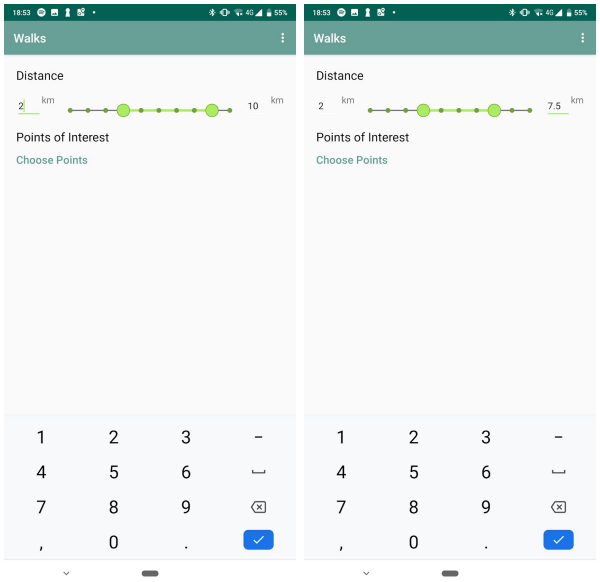
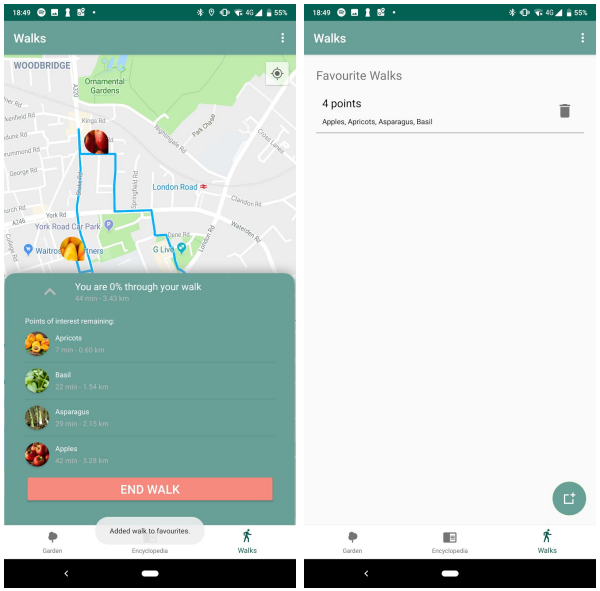
		and scroll up/down the list.			
11	6.1	Open application and login to an account, if not already logged in. Navigate to the Garden section and press the Create Plot action button. Enter the relevant details and press Create Plot	The new Plot should be added to the list and the user should be returned to the list of plots where they can now select or edit it.	The new Plot is added to the list and the user is returned to the list of plots where they can now select or edit it. 	Yes

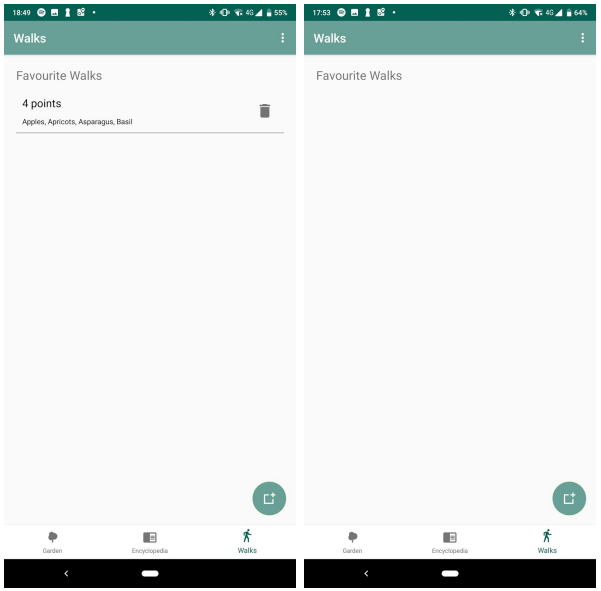
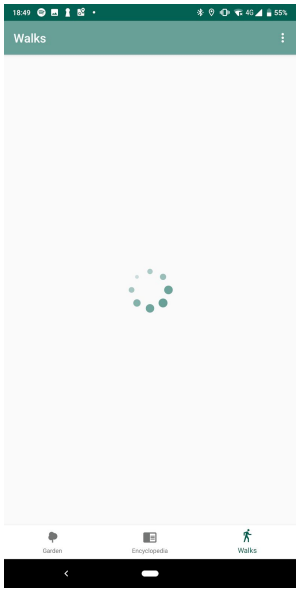
					
12	6.1.4.1	Open application and login to an account, if not already logged in. Navigate to the Garden section and either press the Create Plot action button or edit an existing Plot. Press ‘Add Nearby Object’ , enter the relevant data and confirm.	The dialog should close and the object should be added to the list of Surrounding Objects for the Plot.	The dialog closes and the object is added to the list of Surrounding Objects for the Plot. 	Yes
13	6.1.4.2	Open application and login to an account, if not already logged in.	The object should be removed from the list.	The object is removed from the list.	Yes

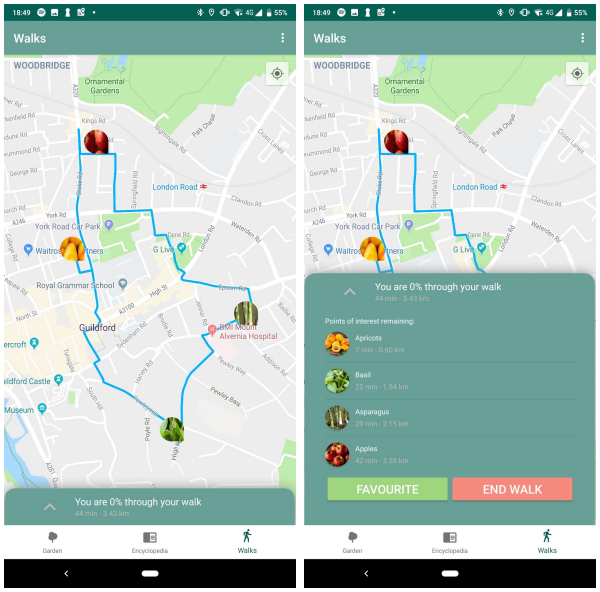
		Navigate to the Garden section and either press the Create Plot action button or edit an existing Plot. Select the ‘Trash Can’ on any object in the Surrounding Objects List.			
14	6.2	Open application and login to an account, if not already logged in. Navigate to the Garden section and press the pencil icon on an existing Plot. Change some data and press confirm changes.	The data in stored in the Plot should change, and the list/PlotView/EditPlot should all reflect this change.	The data in stored in the Plot changes, this change is reflected everywhere where i is relevant. 	Yes
15	6.3	Open application and login to an account, if not already logged in. Navigate to the Garden section and press the pencil icon on an existing Plot. Press the Delete Plot button.	The Plot should be removed from the database and this should be reflected in the list.	The plot has been removed from the database and the list reflects this.	Yes

					
16	6.4	Open application and login to an account, if not already logged in. Navigate to the Garden Section and select a plot.	The Plot View fragment should be launched with all of the relevant information such as Shade, Rainfall and Temperature being displayed.	The Plot View fragment is launched with all of the relevant information such as Shade, Rainfall and Temperature being displayed. However recommendations are not shown (see test 17) 	Yes
17	6.4.4	Open application and login to an account, if not already logged in. Navigate to the	The Plot View fragment should be launched with recommendations accessible to the user based	There are no recommendations present in Plot View as they were not implemented.	No

		Garden Section and select a plot.	on calculated information.		
18	7.1.1	Open application and login to an account, if not already logged in. Navigate to the Walks section. Press the Create Walk action button, then press Choose Points.	The Encyclopedia fragment should launched, and each plant should have a radio button, which can be activated/deactivated when the plant is pressed (unless the limit is reached). Confirm Choices button should add all points to the list.	The Encyclopedia fragment is launched, and each plant has a radio button, which is activated/deactivated when the plant is pressed (unless the limit is reached). Confirm Choices button adds all points to the list 	Yes
19	7.1.2.1	Open application and login to an account, if not already logged in. Navigate to the Walks section. Press the Create walk action button and change the number in either the Maximum or Minimum distance fields.	The slider should move to the closest point and the Edit Text data should change to the closest point.	The slider moves to the closest point and the Edit Text data changes to match the closest point. 	Yes

20	7.1.2.2	Open application and login to an account, if not already logged in. Navigate to the Walks section. Press the create walk action button and slide the Maximum or Minimum distance points on the slider up/down.	The relevant Edit Text should have its text set to the value of the point on the slider.	The relevant Edit Text has its text set to the value of the point on the slider. 	Yes
21	7.2	Open application and login to an account, if not already logged in. Navigate to the Walks section. Press the Create Walk action button and enter the necessary data. Press ‘Create Walk’. Once the Walk Fragment has loaded, open the Bottom Navigation Drawer and press Favourite Walk.	The walk is favourited and added to the walks list, which the user can see if they return to the walk fragment after ending walk.	The walk should be favourited and added to the walks list, which the user can see if they return to the walk fragment after ending walk. 	Yes
22	7.3	Open application and login to an account, if not already logged in.	The chosen walk should be deleted from the list of favourites.	The chosen walk was deleted from the list of favourites	Yes

		Navigate to the Walks section, and press the ‘Trash Can’ icon on any of the walks.			
23	7.4	Open application and login to an account, if not already logged in. Navigate to the Walks section, and press the Create Walk action button. Enter all the necessary information and press Create Walk. Alternatively select a walk from the list of favourites.	The best route should be calculated and then the Walk Fragment will be revealed. A loading screen should be displayed until the route has finished calculating.	The best route is calculated and the Walk Fragment is revealed. A loading screen is displayed until the route has finished calculating. 	Yes
24	7.5	Open application and login to an account, if	A polyline will be drawn to show the route through the	A polyline is drawn to show the route through the points of interest, as well as the points of interest being shown as markers on the map themselves.	Yes

		not already logged in. Navigate to the Walks section, and press the Create Walk action button. Enter all the necessary information and press Create Walk. Alternatively select a walk from the list of favourites.	points of interest, as well as the points of interest being shown as markers on the map themselves. Opening the bottom drawer should reveal all relevant information including points of interest remaining and the time/distance to each.	Opening the bottom drawer reveals all relevant information including points of interest remaining and the time/distance to each. 	
25	7.5.6	Open application and login to an account, if not already logged in. Navigate to the Walks section, and press the Create Walk action button. Enter all the necessary information and press Create Walk. Alternatively select a walk from the list of favourites. Walk around from starting position to see if data updates.	All times/distances should change to reflect how far it is to certain points, and the percentage of walk completed should update accordingly. The line to show the route should adapt to start from the user and finish at the origin.	None of the times and distances, nor the line for the route change dependent on user location. This is because this feature was not implemented.	No