KING'S College LONDON

University of London

6CCS3PRJ

Indoor Wi-Fi Navigation with built-in Augmented Reality for Bush House

Final Project Report

Author: Ion-Alexandru Clapa

Supervisor: Dr. Andrew Coles

Student ID: 1525290

April 15, 2018

Abstract

The recent developments in smart-phone technologies, such as Wi-Fi, have made it possible to build very accurate indoor positioning systems. Moreover, mobile processors have advance capabilities which allow them to render high definition augmented reality objects.

As such, the propose of this project is to develop an indoor positioning system that uses a fingerprinting approach to Wi-Fi positioning, in order to provide positioning and navigation to students in the Department of Informatics. Additionally, the project aims to provide a better navigation experience for users, by adding augmented reality visual cues, and provide information for nearby reference points, i.e. timetables for computer labs.

Using the proposed method, a localisation accuracy of <5 meters is achieved. The system is also able to find a path in under 2-3 seconds, and show augmented reality objects. To incorporate relevant information for some of these objects, data is supplied by the King's College London timetable service and PC-Free@King's.

The system has an extensible design, separated in a multitude of independent subsystems. In the future, this means that subsystems can be easily replaced in order to enable use on a variety of platforms and devices. Furthermore, with simple adjustments, the system can be adapted for use in any building, given the availability of floor plans. Therefore, there is an opportunity to continue the project further to encompass all King's College London campuses, providing navigation to all students and staff.

Originality Avowal

I verify that I am the sole author of this report, except where explicitly stated to the contrary. I grant the right to King's College London to make paper and electronic copies of the submitted work for purposes of marking, plagiarism detection and archival, and to upload a copy of the work to Turnitin or another trusted plagiarism detection service. I confirm this report does not exceed 25,000 words.

Ion-Alexandru Clapa

April 15, 2018

Acknowledgements

I would first like to thank my supervisor, Dr. Andrew Coles, for allowing and helping me to do this project.

I owe my deepest gratitude to my family that has supported me throughout university and made everything possible. A very special thank you goes to Amy, for her invaluable and immense love and support throughout the year.

Lastly, I would like to thank the people that have built the amazing tools and components that have helped me made this project possible.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Given the opening of of King's College London's newest building, Bush House, the new home of the Informatics department, there has been a demand for it to be mapped out, helping to familiarise the students with their new surroundings. To address this, a mobile application (app) will be developed to provide navigation throughout the Informatics department. Adding to this main navigation feature a number of Augmented Reality (AR) features will be implemented into the app to help solve some of the common issues that students encounter on campus.

For example, students may struggle to locate computer labs or other rooms if it is the first time they are using them, or the rooms are hidden from the line of sight. In this case, AR features will be introduced to help the user to gain a better understanding of their surroundings, such as arrows to point to the direction they need to follow in order find a room.

Another problem students incur is the difficulty to find an available computer. This can be particularly challenging during busy periods in the academic year, such as on the approach to coursework deadlines or exams.To solve this issue the app will link up with King's PC Free service, and so the number of available computers in each room will be clearly displayed as the user looks at the room on their phone.

Similarly, it can be frustrating for students when their work is interrupted due to an unknown teaching session. To avoid this the users will be able to view the scheduled bookings for the rooms to know between which hours they will be occupied.

In recent months we have seen the introduction of AR to a number of services and devices, it therefore seems that this tool would be an excellent way to solve the issues outlined above as the user will be able to see all of the relevant information about their location and rooms they are looking at, using their mobile phone's camera.

1.2 Scope

This project has one clear aim to tackle: providing a robust and informative navigation system for the Department of Informatics in King's College London. The navigation system will be based on indoor positioning technologies, such as Wi-Fi.

Additionally, this projects looks to make use of the rise in popularity of AR in order to

provide more information for the user. The system will be composed of multiple independent parts, including:

- Mobile applications that collect and show data.
- A server that positions the user and assists in navigation.
- A data management server that will handle the data flow responsible for the core functionalities.

1.3 Objectives

As previously stated, the main objective of this project is to provide a navigation system that will only use indoor navigation methods for Bush House. This will be achieved by creating two iOS applications: the first will be used to record positions on the floor plans of the building, and the second one will be used to detect the user's current location and consequently provide navigation guidance for the user. Furthermore, the latter application will also include AR features.

In order to achieve this, further objectives are required to be developed:

- A simple server application connected to a database that will be responsible for storing data and delivering it when requested by the mobile apps, or any other logic entity that needs the data to determine the current position, or calculate a path; the connection to the database will be made through a REST API.
- A server that will act as a middle man and will handle all the big computation processes in order to not overload the mobile devices. Such processes include providing navigation instructions given a start and finish points, calculating a route, managing the current position of the user, and providing relevant information to the user based on their location.

Chapter 2

Background & Literature Review

2.1 Context

There are many localisation methods which could be used to create a navigation system for Bush House. However, many traditional localisation methods, such as GPS or GLONASS, are less accurate when used indoors. To better understand the localisation methods already available and the rationale for using Wi-Fi positioning, GPS & GLONASS will be detailed in the following sections, along with alternative indoor positioning systems, and how they are relevant to the project scope.

2.1.1 Global Positioning System and Global Navigation Satellite System

The Global Positioning System (GPS) [1] is a space based navigation system developed by the United States. The system provides the geolocation information to a GPS receiver anywhere on Earth as long as its line of sight is unobstructed and as long as there are 4 visible satellites. GPS answers 5 questions: "Where am I?", "Where am I going?", "Where are you?", "What's the best way to get there? and "When will I get there?"[1].

The Global Navigation Satellite System (GLONASS) is a similar navigation system, started by the Soviet Union and finished by Russia. GLONASS and GPS work in a similar way, but because of the smaller number of satellites for GLONASS, GPS is more accurate. Data from 2010 recorded by the Russian System of Differentional Correction and Monitoring show that the precision of GLONASS for latitude and longitude is 4.46–7.38 metres [5], whereas GPS-enabled smart-phones are typically accurate to within a 4.9m radius under open sky [6]. However, because of the positioning of the satellites, GLONASS is more accurate than GPS in the northern hemisphere. This is due to the fact that the development and positioning of satellites was started from Russia.

Nowadays, the manufacturers of GPS navigation systems say that adding GLONASS has made more satellites available to them, meaning that positioning works better than each of them individually when, for example, the line of sight for GPS is obstructed by buildings, and the microwaves will be weakened by walls or rooftops. Overall, a higher accuracy is achieved [3, 4] by using both of them combined. However, due to the fact that these two technologies

need a direct line of sight in order to provide information about positioning, they suffer when used to determine the current location in indoors environments.

2.1.2 Indoor positioning

As previously stated in section 2.1.1, GPS and GLONASS are not suitable for use indoors as the microwaves will be weakened by rooftops or walls [7]. To solve the issue, Indoor Positioning Systems (IPS) are used. IPSes are systems that can locate objects or people inside buildings using information collected by mobile devices [8]. These systems can either be independent, or combined with GPS to achieve greater accuracy. IPSes use different technologies, such as Wi-Fi or Bluetooth enabled beacons [8]. These technologies will be detailed in the following sections.

2.1.2.1 Wi-Fi and Bluetooth based positioning

Most of the wireless technologies, such as Bluetooth beacons or Wi-Fia access points, can be used for detecting the location. In recent years, the Bluetooth technologies have been supported by many big companies, including Apple, mostly because of the recent developments in low energy Bluetooth sensors. The downside of Bluetooth beacons compared to Wi-Fi access points is that they don't provide an exact location, but rather a promixity composed of a geo-fence, rather than a pinned location. Moreover, in order to use Bluetooth beacons, dedicated Bluetooth devices need to be installed indoors, whereas most of the buildings already have a Wi-Fi infrastructure that provides Internet access. Considering this, a Wi-Fi navigation system can be built on top of the existing Wi-Fi access point systems available in Bush House.

2.2 Methods used for Wi-Fi positioning

Having concluded that Wi-Fi positioning would be the most appropriate localisation method, in the following sections the different types of Wi-Fi positioning methods are considered for use in the navigation system and will be further explained.

2.2.1 Received Signal Strength Indication Localisation

Received Signal Strength Indication (RSSI) localisation measures the signal strengths from the available Wi-Fi access points around the device. It combines the information gathered, and finds out the distance from the device to the Wi-Fi access point. This method is usually combined with other methods of positioning, such as triliteration or triangulation in order to position the user, where the distance is measured using the signal's strength from the Wi-Fi access point.

2.2.2 Fingerprinting Localisation

Fingerprint localisation involves recording signal strengths from access points in different parts of the rooms, and then storing them into a database. This method is RSSI based, because it records RSSI measurements taken around the building, but involves a more labour intense approach since the database needs to be kept updated in case the environment changes.

Fingerprinting usually involves an offline phase, where positions are recorded in the database, and an online phase, where the data recorded is used to determine the location. In the offline phase, the signal strengths are measured at different fingerprints, or reference points. The location can then be calculated in the online phase either in a deterministic way, by finding the closest match in the database, or by using a probabilistic model to see if the user may be in a certain position, using the data previously measured. One major drawback of this approach is that it requires for the database to be updated in case environmental changes happen, such as changing the position of furniture that will make some locations not accessible anymore.

2.2.3 Triliteration

Another method used in combination with Wi-Fi positioning methods is triliteration. Triliteration is the process of determining locations using geometry shapes, such as circles and triangles. The location is calculated using two-dimensional geometry and three-dimensional geometry. Two-dimensional geometry is used by placing two circles to have a certain point lying on them, then the centres of the circles and the two radii provide sufficient information to reduce the possible locations to two. Furthermore, if a point lies on the surfaces of three spheres, then again, the centres of the spheres and the radii provide sufficient information to reduce the possible locations to two [9].

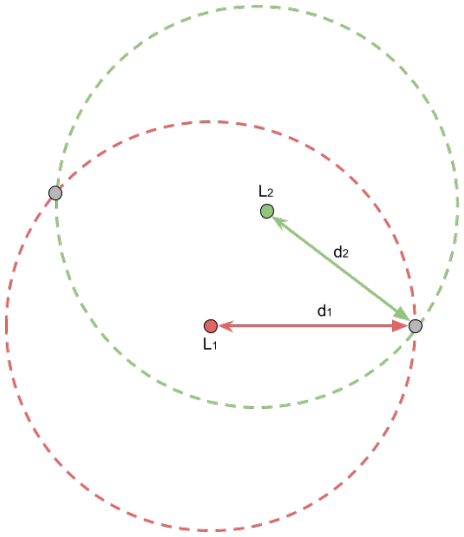

Figure 2.1: Triliteration with two circles [15]. L_1 and L_2 are the interest points, whereas d_1 and d_2 are the distances from those points to position P.

Figure 2.1 shows how triliteration first starts with two circles. The position P can be calculated by creating two circles that have the centres L_1 and L_2, with radii d_1 and d_2. The position can then be calculated using the radii and a set of equations. Using only two circles is not very accurate, since there are two possible results. In order to get the correct result, another circle can be added. By doing this, only one of the two possible results will be on the third circle, as seen in figure 2.2.

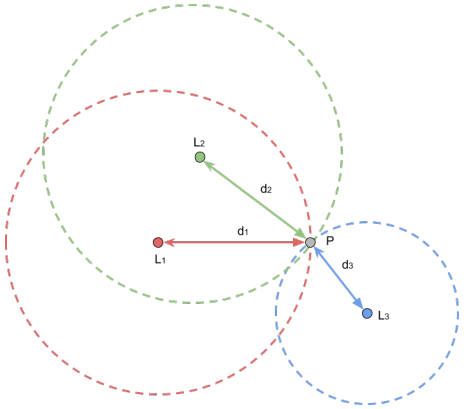

Figure 2.2: Triliteration with three circles [15]. L_1, L_2 and L_3 are the interest points, whereas d_1, d_2 and d_3 are the distances from those points to position P.

To conclude, the project's positioning algorithm will be an IPS that will use fingerprinting as the main method to collect location values, whilst the latitude and longitude will be calculated using triliteration.

2.3 Navigation

After the position is determined, the user needs to find out how to get to other rooms. For this, locations registered on the floor plan, such as the rooms or any other destinations, can be translated into a graph structure. The locations can be seen as nodes, and the paths between them, as vertexes in the graph, where the cost is the distance between two locations, as seen in figure 2.3.

Building the graph gives way to use shortest path algorithms. For this project, Dijkstra's algorithm has been used, because this algorithm performs better when compared to a standard path finding algorithm, when there is little knowledge about the graph.

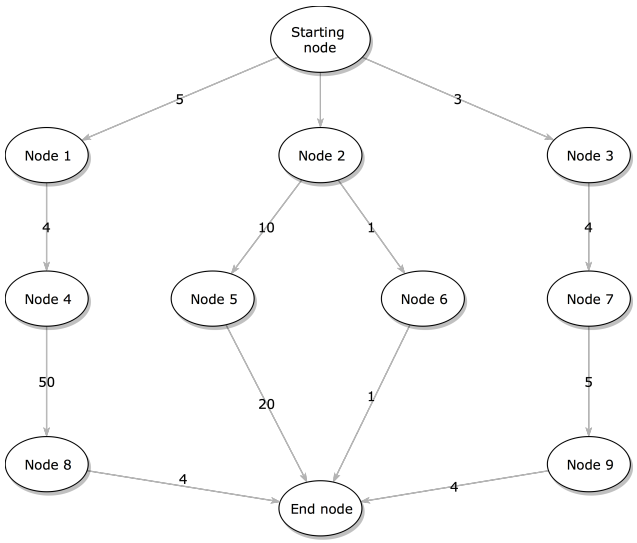

Figure 2.3: A visual representation of a graph. The circles represent the nodes, which are connected by lines, the vertexes. The numbers on each vertex represent the cost to get from a node to another node.

2.3.1 Haversine formula

$$a = sin^2(\frac{\Delta\phi}{2}) + cos(\phi_1) * cos(\phi_2) * sin^2(\frac{\Delta\lambda}{2}) \tag{2.1}$$

$$c = 2 * atan2(\sqrt{a}, \sqrt{1-a}) \tag{2.2}$$

$$d = R * c \tag{2.3}$$

The distance between nodes is calculated using the above formulae. Formula 2.1 is the haversine formula, in which the latitude (ϕ) and longitude (λ) are used. The result is used in 2.2, in which $atan2$ is used. $atan2$ returns the value of the arc tangent of $\frac{y}{x}$ of two values x and

y, expressed in radians. Finally, the distance d is calculated by multiplying the value computed in the 2.3 with R, the Earth's radius.

2.3.2 Djikstra's Algorithm

Djikstra's algorithm is an algorithm that finds the shortest path between the nodes in a graph, either by building a shortest-path tree, that has a root in the source, or by calculating the shortest path between two points. Dijkstra's algorithm picks an the unvisited node from the graph with the lowest distance, calculates the distance through it to each unvisited neighbour, and updates the neighbour's distance if smaller. When the algorithm is done with neighbours, they are mark as visited, and they will be assigned the distance calculated.

Dijkstra's Algorithm

For each edge $(u, v)\ \epsilon\ E$, assume $w(u, v) \geq 0$, maintain a set S of vertices whose final shortest path weights have been determined. Repeatedly select $u\ \epsilon\ V - S$ with minimum shortest path estimate, add u to S, relax all edges out of u.

Pseudo-code

```
Dijkstra (G, W, s)      //uses priority queue Q
        Initialize (G, s)
        S ← φ
        Q ← V[G]        //Insert into Q
        while Q ≠ φ
              do u ← EXTRACT-MIN(Q)      //deletes u from Q
              S = S ∪ {u}
              for each vertex v ϵ Adj[u]
                    do RELAX (u, v, w)   ← this is an implicit DECREASE_KEY operation
```

Figure 2.4: Dijkstra's Algorithm [16]. Describes the main Dijkstra algorithm used to find the shortest path between any two nodes in a graph.

$d[v]$ is the length of the current shortest path from starting vertex s. Through a process of relaxation, $d[v]$ should eventually become $\delta(s, v)$, which is the length of the shortest pathfrom s to v. $\Pi[v]$ is the predecessor of v in the shortest path from s to v.

Basic operation in shortest path computation is the *relaxation operation*

```
RELAX(u, v, w)
    if d[v] > d[u] + w(u, v)
      then d[v] ← d[u] + w(u, v)
           Π[v] ← u
```

Figure 2.5: Relax procedure [16]. This is used in the main Dijkstra algorithm from figure 2.4 to get the shortest path.

2.4 Augmented Reality

AR is defined as an enhanced version of the physical, real-world reality of which elements are added by computer-generated or extracted real-world sensory input such as sound, video or graphics [10]. In other words, AR brings elements of the digital world into the user's perceived world, through either special devices, such as "Smart Glasses", or through a smart-phone's screen. The information shown is usually related to the local environment captured through the smart-phone's screen, and becomes interactive and digitally manipulable [11]. The aim of AR in this project is to create a better experience for user, and a good example of this is to inform them about the timetable for the current week for a room that is near them.

2.4.1 AR in Navigation

Combining AR with a navigation system can enhance the user experience and improve the effectiveness of the information shown. Because of this, many manufacturers have incorporated AR into their navigation systems. For example, in the car industry AR is used to show information such as the final destination, current speed, speed limits or potential hazards on the windshield.

A good example of how AR has improved navigation is NASA X-38 spacecraft. The spacecraft was flown using the LandForm software which overlaid map data on video to provide enhanced navigation for the spacecraft during test flights [14], such as taxiways, tower controls, or runways. This information has proved to be very useful in special conditions, such as low visibility.

By following these examples, a similar solution will be developed in order to provide better navigation around Bush House. An arrow to follow will be shown on the screen when the user is navigating, and around the path, information about the rooms around will be shown. This information will include timetables and computers available in the computer labs positioned around the user.

2.5 Related Work

The subject of indoor navigation systems that use existing infrastructures of Wi-Fi connections has been researched into great detail. Many methods have been approached, including RSSI and fingerprinting.

Active badges [21] was the first indoor positioning system, developed by AT&T Cambridge. It involved an infrared sensor that was worn by a person. The locations in a building were covered with a network of infrared sensors. All the positions around the building of the fixed sensors were stored in an internal database, and the location of the badge could be then determined. In a similar way, AT&T has developed an ultrasonic tracking technology [21], that was supposed to be more accurate than the previous active badges. Users were tagged with ultrasonic tags, which emitted ultrasonic signals to receivers on the ceiling. Although it was accurate, the drawback of this was that it involved a large numbers of receivers to be mounted across the building; additionally, these receivers had to be placed and aligned, in order to provide accurate results [21].

Shin, Cho and Cha have tried to automatically build an indoor map. Their algorithm, SmartSLAM [18] is able to construct the indoor plan and draw the corridor outlines of the building. The approach taken by them uses fingerprinting and simultaneous localisation and mapping (SLAM). SLAM is a localisation technique where a map of an unknown environment is constructed and updated while simultaneously keeping track of the current location within the map [19]. This approach is usually used for self driving cars, unmanned aerial vehicles, or even inside the human body [20]. The system developed is the first one that uses SLAM on a smart-phone. However, Shin et al showed that this takes a toll on the battery, because the algorithm continuously scans the sensors which has a high energy use [18].

On the 29th of March, Apple has launched an application called "Indoor Survey App", that allows users to register buildings along with their floorplans in their system [22]. Given these floorplans, then users are able to record locations by "dropping points" as "you indicate your position within the venue as you walk through". When registering these locations, the application uses a combination of Wi-Fi and radio signals to track positions; the application incorporates the solution used by WiFiSLAM, a company by Apple. This method provides a simpler and easier way to provide a data set to use for indoor positioning, by using already existing floor plans of the building. However, the only drawback of this is that locations need to be first recorded and kept updated by the admin users, in order to provide accurate results.

Another fingerprinting solution is COMPASS [31]. Positioning is achieved by registering reference points using the already existing Wi-Fi positions. Unlike a traditional fingerprinting approach, COMPASS adds the user's orientation received from the mobile phone's digital compass in order to improve the positioning. COMPASS has an accuracy of 1.65 meters, but it is important to mention that this is only an experimental approach that has not been tested on a large scale, but only in an environment of 312 square meters [31]. Therefore, it is impossible to say how the system will behave on a much larger scale, outside the limits where it was tested [31].

In conclusion, these papers and applications show that a very complex navigational system can be built on top of the existing sensors or Wi-Fi networks that are available, without requiring extra costs. Nowadays, most smart-phones are equipped with an accelerometer and a gyroscope. However, if a fingerprinting approach is taken, keeping a database with received signal strengths updated is very important. Future technologies will include methods that will try to reduce the time of building and managing this type of database [32].

Chapter 3

Requirements and Specification

3.1 Domain Concepts

Before discussing the requirements, in order to get a better grasp of the project, based on its decomposition, the following concepts and terms are believed to be important and relevant:

- Admin user – A type of user that has a better understanding of the navigation system and application and will be responsible for managing and measuring data.
- Application Programming Interface (API) – a set of methods used to communicate between software components.
- REpresentational State Transfer (REST) – a style of architecture that defines properties based on HTTP requests. The operations available through HTTP requests are GET, POST, PUT, PATCH and DELETE.
  - GET request to /resources – Will retrieve all the resources.
  - POST request to /resources – Will add a new resource.
  - PUT request to /resources/:resource – Will add or overwrite the resource.
  - PATCH request to /resources/:resource – Will partially update the resource specified.
  - DELETE request to /services/:service – Will delete the specified service. If a service is not included, the operation will delete all the services.
- CRUD – is an acronym for create, read, update and delete, which are the basic operations of a database.
- Raspberry Pi – A small single-board computer.

3.2 System Components

In order to fully understand the functional requirements that are going to be defined in section 3.3, it is important to visualise the organisation of the system. This can be done by using figure 3.1. The components have been separated into the presentation layer, which displays the services available by communicating with the lower tiers, the logic layer, which controls the system's main functionalities, and the data layer, which houses the database servers (the data is kept independent of the other layers). This is based on a three tier architecture [17].

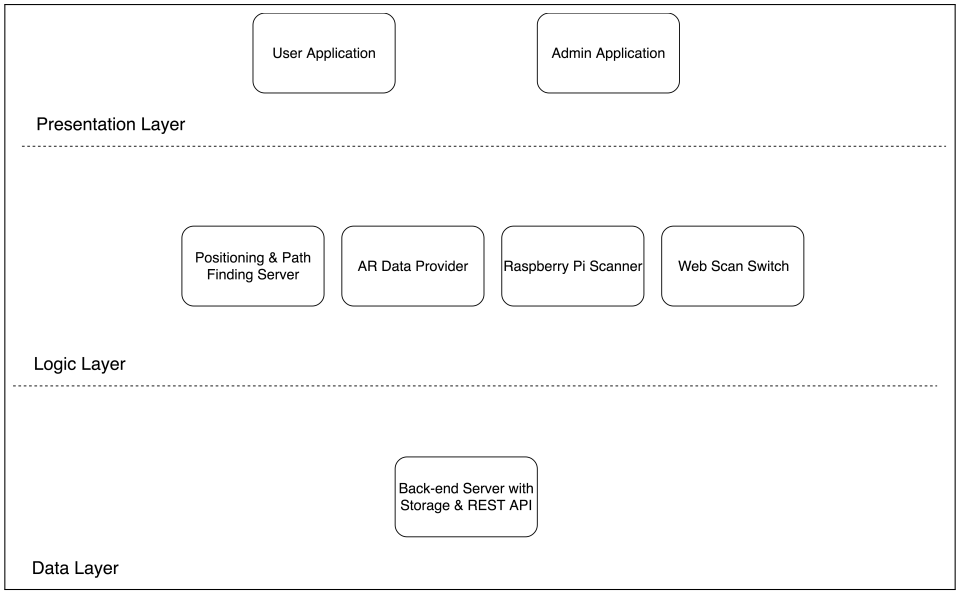

Figure 3.1: High Level Organisation of the System. The round rectangles represent the components for which the requirements are defined in this chapter. These components have been organised in layers, using the tasks that they are performing.

3.3 Functional Requirements

The requirements have been divided for each component that has been developed. These components are: an admin iOS mobile application, a user iOS mobile application, a back-end server that manages the storage through a RESTful API, a back-end server to manage positioning and navigation, and a data provider for AR. Finally, a Raspberry Pi will be used to scan and upload Wi-Fi measurements, due to the limitation imposed on iOS, which will be detailed in the Conclusion chapter. The connection between the the mobile device and Raspberry Pi will be made through another web application which will act as a on-off switch that will let the Raspberry Pi know when it is time to scan.

The specific requirements for each component are detailed:

3.3.1 Admin Mobile Application Requirements

- AM1 – Admin should add a room by providing a name for it.
- AM2 – In the list of rooms, the admin user should be able to delete a room along with the data in it.
- AM3 – Admin should be able to select a room and start measuring Wi-Fi data for that room.
- AM4 – Admin should be able to reset the data measured so far for a specific room.

3.3.2 User Mobile Application Requirements

- UM1 – User should be able to see their current location.
- UM2 – User should be able to see a list of registered rooms.
- UM3 – User should be able to search for a destination (a room).
- UM4 – User should receive instructions on how to get to the selected destination.
- UM5 – User should be able to receive an estimated arrival time from the starting position to the destination.
- UM6 – User should be able to use their mobile phone's camera to see directions towards the selected destination.
- UM7 – User should be able to see information about the timetable of a room.
- UM8 – User should be able to see information about the availability of computers in a room.

3.3.3 Back-end Server with storage and REST API Requirements

- BAPI1 – Database should store, update and delete rooms.
- BAPI2 – Back-end should store relevant data about rooms, such as Wi-Fi strengths positions.

- BAPI3 – The REST API should provide a link to get data from the database.
- BAPI4 – the REST API should provide a link to upload and update data in the database.

3.3.4 Positioning & Path Finding Server Requirements

- PPF1 – The server should be able to determine the current location of the user.
- PPF2 – The server should be able to create a route and calculate the route's distance, given a starting position and destination.

3.3.5 AR Data Provider

- ARDP1 – The provider should supply the mobile application with an image of the timetable for the room that data has been requested for.
- ARDP2 – The provider should supply the mobile application with a status text of the availability of the computers in the room data has been requested for.

3.3.6 Raspberry Pi Scanner Requirements

- RPS1 – Scan Wi-Fi networks around and store the signal strength, the MAC address and the network name locally.
- RPS2 – Upload the data measured in the database when requested, using the API from 3.2.3.
- RPS3 – Check continuously if data is needed every 2 seconds.

3.3.7 Web Scan Switch Requirements

- WSS1 – Have a Boolean flag to know if data needs to be scanned or not.
- WSS2 – The component should be able to temporarily store some measurement data. This data is going to be used to compare to already measured data in the process of positioning a user.

3.4 Non-functional Requirements

- Interoperability – The subsystems should be able to fully integrate with other subsystems in order to provide maximum flexibility.
- Maintainability – The system should provide a simple and easy way to extend feature base. For this, the follow concepts must be followed:
  - Separation of Systems – Separation of Systems – The system must be decomposed in multiple independent subsystems which must be as decoupled as possible. The systems should use the appropriate programming languages and framework to achieve their requirements and tasks.

  - Architectural Patterns – Each subsystem should follow the appropriate architectural pattern and design patterns provided by the framework that is being used, or the programming language that the system is written in.

- Usability – The system should provide an easy to use interface, and as clear as possible. Although it can be simple, the interfaces must provide clarity of interactions, and a satisfactory user experience.

3.5 System Requirements

Both the admin and user mobile applications require Wi-Fi to be enabled and an active Internet connection. The user mobile application will require iOS 11 in order to use the ARKit framework for the AR features. The back-end server with storage storage, along with the server for positioning and path finding, and the web scan switch will require a Linux or macOS machine with Swift and Vapor installed. As well as the mobile apps, they will require an Internet connection. The AR data provider needs a machine that runs Python with Flask, and the Raspberry Pi only needs a Linux based operating system and an Internet connection to communicate to the other components. All the servers and data providers will be deployed externally, on Heroku, for better accessibility.

3.6 Specification

The specification section describes what parts need to be implemented in order to meet the aforementioned requirements.

3.6.1 Admin Application

Table 3.1: Admin Application Specification Details Table.

Requirement Code	Requirement Details	Specification
AM1	Admin should add a room by providing a name for it.	A UIViewController with a UITableView that has a list of rooms should greet the user. From here, by pressing a plus UIButton, a UIActivityController with an UILabel for the room's name will be shown. The user adds a room by pressing add after typing the name.
AM2	In the list of rooms, the admin user should be able to delete a room along with the data in it.	Sliding on the UITableViewCell of a specific room, should show a delete UIButton, that the user can press in order to delete the room.
AM3	Admin should be able to select a room and start measuring Wi-Fi data for that room.	Selecting a room from the UITableView with rooms will open another UIViewController with an UIImageView. From here, the admin can tap on the desired location to register it.
AM4	Admin should be able to reset the data measured so far for a specific room.	Sliding on the cell of the desired room in UITableView will show a "Clear" button.

3.6.2 User Application

Table 3.2: User Application Specification Details Table.

Requirement Code	Requirement Details	Specification
UM1	User should be able to see their current location.	The current floor plan will be shown on an UIImageView. Placed on that, a subclass of UIView of small dimensions (a square) will show the user's current location.
UM2	User should be able to see a list of registered rooms.	Pressing on a UIButton will launch a UITableView with the list of all the rooms.
UM3	User should be able to search for a destination.	On the top of the UITableView with the list of rooms, a search box will be available for the user to make queries.
UM4	User should receive instructions on how to get to the selected destination.	A path will be drawn on the floor plan shown on the screen from the user's current location to the selected destination.
UM5	User should be able to receive an estimated arrival time from the starting position to the destination.	Under the floor plan, two UILabels will show the estimated arrival time.
UM6	User should be able to use their mobile phone's camera to see directions towards the selected destination.	An AR object (an arrow) will show the direction to follow to get to the destination.
UM7	User should be able to see relevant information about the rooms that are around, such as timetables or available computers.	Based on their current location, AR objects will be placed in rooms to show the timetable for the current week.
UM8	User should be able to see information about the availability of computers in a room.	Again, based on their current location, AR objects will be placed in rooms to show information about the availability of computers in rooms.

3.6.3 Back-end Server with storage and REST API Requirements

Table 3.3: Back-end Server and Storage Specification Details Table.

Requirement Code	Requirement Details	Specification
BAPI1	Database should store, update and delete rooms.	Store the data in CRUD database.
BAPI2	Back-end should store relevant data about rooms, such as Wi-Fi strengths positions.	Data about rooms will be broken down in several entities, such as Room, Location, Access Point and Measurement.
BAPI3	The REST API should provide a link to get data from the database.	The REST API will support GET requests for every entity in order to fetch data from the tables in the database.
BAPI4	The REST API should provide a link to upload and update data in the database.	The REST API will support POST and PATCH requests if needed for entities in order to submit or change data in the database.

3.6.4 Positioning & Path Finding System

Table 3.4: Positioning and Navigation System Specification Details Table.

Requirement Code	Requirement Details	Specification
BS1	The server should be able to determine the current location of the user.	Using the measurements from the current scan along with the ones saved in the database, perform a best-match algorithm to determine which location ”fits” the best. The result will be returned in JSON format.
BS2	The server should be able to create a route and calculate the route's distance, given a starting position and destination.	Given the two locations, the server will calculate a path between them using Dijkstra's algorithm.

3.6.5 Scanning System

The scanning system brings together two of the previously mentioned components: the Raspberry Pi scanner (see section 3.3.6) and the the web scan switch (see section 3.3.7). This has been done because they both are components that work together in order to provide measurements by scanning the data from the Wi-Fi networks around.

Table 3.5: Scanning System Specification Details Table.

Requirement Code	Requirement Details	Specification
WSS1	Boolean flag to know if data needs to scanned or not.	The system will incorporate a model that will store a boolean flag. This boolean flag will be modified by a PATCH HTTP request that the server needs support. To check if data needs to be scanned or not, a GET HTTP request needs to be able to be made to retrieve the flag's value.
WSS2	The component should be able to temporarily store some measurement data. This is going to be used to compare to already measured data in the process of positioning a user.	When data doesn't need to be stored into the database, the server should have a Boolean flag that can be set to let the scanner know about this. In this case, data will be only temporarily cached into an in-memory database and deleted after it has been used.
RPS1	Scan Wi-Fi networks and store the signal strength, the MAC address and the network name locally.	Scan using the iwlist on Linux and store the output in a text file.
RPS2	Upload the data measured into the database when requested.	When data is requested from the Web Scan Switch, a Java application will parse the text file and upload the data using the RESTful API.
RPS3	Check continuously if data is needed.	Automate this process using a bash script that runs every 2 seconds.

3.7 Limitations

Whilst developing the project, a few limitations have been encountered, which can be found below:

- Security has been ignored. For the sake of simplicity, everyone can access and submit data. In other words, there is not a user based system with access control that will be able to distinguish between users and what they can access or modify. However, all the network requests are made over HTTPS in order to make them less prone to attacks. Furthermore, the design of the back-end servers has been made extendable, thus adding more features such as security is easy.

- The user interface (UI) was not considered as being a priority, and the main focus has been to make it as simple and usable as possible. There is not a lot of UI feedback and there are not many visual cues that give user feedback, but this can be easily added as the whole architecture relies on the low layered systems, and not on the mobile apps.

- The process of scanning is very long and slow. The mobile device has to request the Raspberry Pi to scan for data through a server, then the Raspberry Pi scans, parses the results and uploads them into the database. Finally, the data is downloaded from the database on the mobile device. This would be unnecessary if the scanning capability would be opened by Apple on iOS. Given the fact that the system has been designed to be extensible, a simpler solution can be implemented easily.

- Error handling on the server is poorly handled. Some of the cases where a request is poorly formatted will return errors, but they are not very detailed and they don't cover all the cases.

- The system works only if the floor plans for the building are available. The design relies on assigning Wi-Fi measurement values to positions on floor plans, which will create locations.

- Right now, PC-Free@King's has data for only one computer lab in Bush House. Although this is a limitation, the project will be developed so that it will be capable to provide data for any given room, as long as the data is available. Therefore, when data will be available for the rest of the computer labs, no changes will required to be made.

Chapter 4

Design

4.1 Technologies Used

4.1.1 iOS

iOS is a mobile operating system, created and developed by Apple [23]. It is the operating system that currently runs on many of the company's devices, such as iPhone, iPad, or iPod Touch. It is considered to be the second most popular mobile operating system, after Android [23].

4.1.2 Swift

Swift is a programming language created by Apple, which was launched as an alternative for Objective-C, in order to create iOS apps [24]. The goal of the creators of Swift is to create a programming language that could be used on a variety of systems, such as mobile, desktop or even cloud services (servers) [24]. Soon after the launch, the programming language has been made open-source, which has opened the way for it to be used in more than just iOS-running devices. Swift is believed to be:

- Safe: it includes features that make the applications crash less. For example, Optionals are wrappers that can either hold a value or not. In this way, by safely extracting the value before using it, a crash is avoided when the value would be non existent.
- Fast: it was made as a replacement for C-based languages, which have always been very fast.
- Expressive: "Swift benefits from decades of advancement in computer science to offer syntax that is a joy to use, with modern features developers expect" [24].

Because this is the most popular way to create iOS applications, Swift has been chosen in order to make the user and admin applications in this project.

4.1.3 Vapor

Vapor is a web framework that provides a way to create server side applications using Swift. It is a secure framework, which offers encryption for all network requests as default, and a

very fast framework. Benchmarks have been showing that it is nearly 100 times faster than other web frameworks, such as Ruby and PHP [25]. Due to its rapid rise in popularity and the supporting and helpful community, this framework has been chosen as an experiment in order to determine how feasible it is to create web applications using Swift.

4.1.4 Python

Python is a interpreted programming languages, meant to be used for general-purpose programming. Python is very easy to learn and use, regardless of the programmer's experience [26]. This programming languages will be used for the AR Data Provider, mainly due to the fact that there are a multitude of libraries that achieve most of the tasks one would like. The vast number of community-contributed modules allow for "endless possibilities" [26]. In this project, the Selenium library will be used in order to effortlessly automate actions in a web browser, when using King's timetable service.

4.1.5 Flask

Flask is a lightweight microframework for Python [27]. This allows to write server applications using Python. The framework makes it possible to create the server that will provide data for the AR features in the user application. Because the design is very decoupled, this framework makes the connection between the AR Data Provider and any system that requires its data.

4.1.6 Linux

Linux is a family of free and open-source operating systems. It is a very popular system that runs on a variety of platforms, from phones to desktop computers, and to servers and mainframe computers [28]. The Raspberry Pi runs a light distribution of Linux, called Raspbian. This operating systems offers basic functionalities for the Raspberry Pi devices, and it is made to be very optimised for low-performance processors.

4.1.7 Java

Java is a general-purpose programming language that was inspired by C++ and is more focused on the object-oriented side of programming. It is one of the most popular programming languages in use. Java apps can run on any Java virtual machine, regardless of the operating system, or architecture. Because it is very popular and very widely used, this programming language has been chosen for the Raspberry Pi application.

4.1.8 ARKit

ARKit is a framework that allows programmers to bring AR experiences in their iOS applications [29]. It was launched together with iOS 11 and makes using AR on iOS much more easier than before. The framework provides a lot of functionalities that the developers can easily use, such as scene understanding and lightning estimations, which detects the vertical and horizontal panes which are around, and makes use of the camera to detect the amount of

light available and later apply it to the virtual objects [29]. This framework has been chosen because of its ease of use on iOS and its high level of performance optimisation.

4.1.9 Heroku

Heroku is a cloud platform that allows web applications to be deployed. It supports a vast number of programming languages and frameworks, mainly because its active community. Heroku brings a very easy and simple solution for deployment and has been chosen as a solution in our project mainly because of this reason. The server that manages the database, the scanner switch, and the positioning & path finding system have been deployed on Heroku, in order to better integrate them with the mobile apps. If they would be run locally, on a laptop, their IP address would change often, and configuring the code would be required before running the applications. Thus, by deploying on Heroku, a static address is assigned to them, and no further configuration is required.

4.1.10 PostgreSQL

PostgreSQL is a relational database system, based on SQL [30]. It currently runs on most of the major operating systems, and is supported by the majority of programming languages [30]. Because the servers have been deployed on Heroku, and because Heroku supports PostgreSQL as its main relational database system, this component has been chosen for the database system in this project.

4.2 Architectural Design

The architecture has been divided into four independent layers. This approach makes the whole system easier to manage, and very simple to replace or change parts of it, if they need to be extended or improved. The System Architecture Diagram shown in figure 4.1, describes the aforementioned four layers. The top layer, the User Interface Layer, is composed of the two applications: the admin application, used to measure and register positions in Bush House, and the user application, which will make use of the data for positioning and navigation. The second layer, the Scanning Layer, deals with scanning data when requested (when recording a position or when positioning is needed) and uploading it in the database. This layer is composed of a Raspberry Pi that scans, parses results and makes HTTP requests in order to upload the data, and a simple web application that will act as a bridge between the Raspberry Pi and the mobile applications, by letting the Raspberry Pi know when data is needed.

The lower two layers, the Logic Layer and the Data Management Layer, work together to receive, process and provide data for the User Interface Layer. Data is saved by the Data Management Layer which has an API that acts as an interface for the database. The Logic Layer processes available data in order to come up with the most optimal route to get from point A to point B, to determine the current position, or to provide data to be used for AR, such as timetables or the number of available PCs.

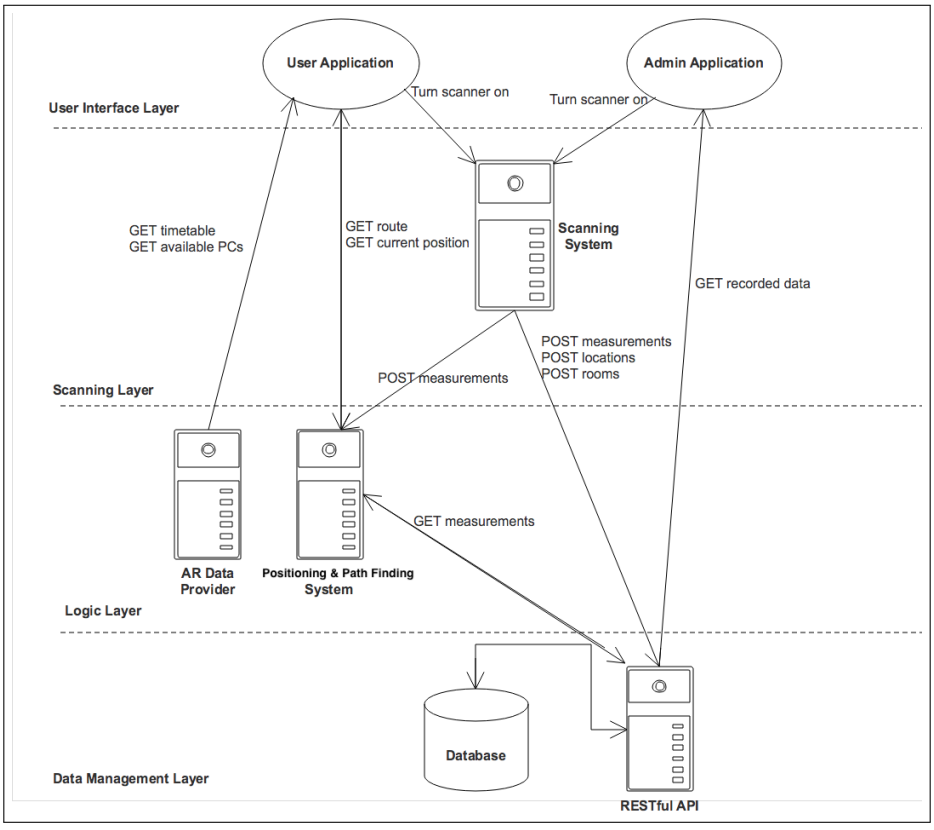

Figure 4.1: System Architecture Design. Depicts the 4 layers of the navigation system, along with the subsystems and how they interact between themselves.

4.2.1 User Interface Layer Overview

The user interface layer will act as a front-end to the system, and will deal with all user specific tasks. Due to the nature of tasks that need to be accomplished, this layer has been divided into two separate applications. One single application could have been developed, but for the sake of simplicity and avoiding a user-based system, they have been separated. The admin application is meant to be used as a measurement tool in order to record locations for the rooms in Bush House, whereas the user application will be used by any user who would want navigation assistance for Bush House, without having write permissions for the database, as the admin application has.

4.2.1.1 Admin Application

The admin application has been designed using the Model-View-Controller (MVC) architectural pattern. MVC is the go-to architectural pattern that is used when developing iOS application, therefore our app is following it as well. The model handles all the data processing and fetching, whereas the controller converts the data from the model and shows it in the view. The application uses components already defined in UIKit, which is the framework for user interface elements in iOS. The whole architecture of the application is described in figure 4.2. The code of the each component of each element of MVC will be detailed in chapter 5, the Implementation chapter, and it will be fully available in the Appendix.

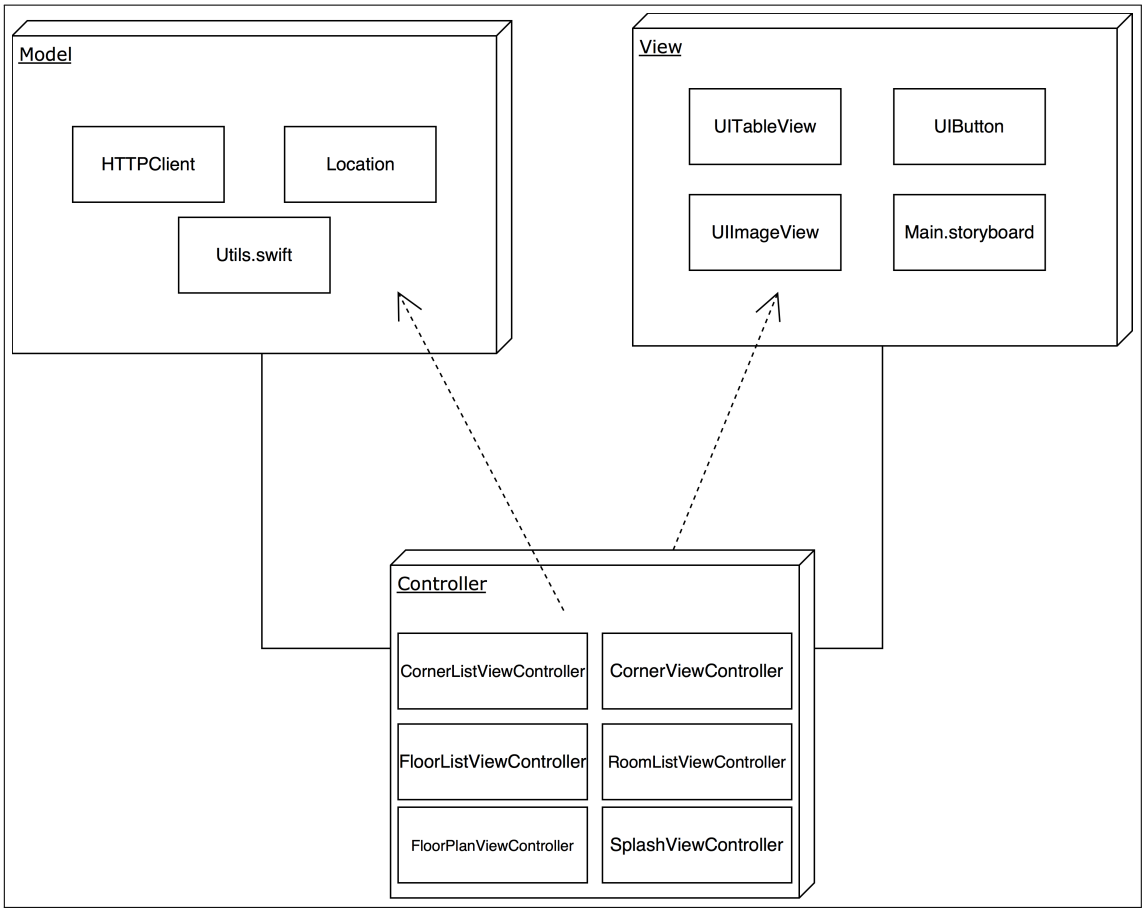

Figure 4.2: Admin Application MVC Design. The design is split into model, view and controller. The rectangles inside each component represent the classes that are part of it.

Upon the first launch of the application, the admin user will be required to calibrate the application. This process involves registering the corners of the mobile device's screen. The values registered will come into play when using triliteration (see section 2.2.3) to calculate the values of the positions measured in latitude and longitude, in the following way:

- The values on the screen of the corners of the floor plan will be measured by the admin, by tapping on the screen.
- Then, the values from the 2D plan (the screen) are assigned to the positions in latitude and longitude of the corners of the building that are already entered in the application.
- Now, any position from the 2D plan can be translated into latitude and longitude using triliteration, following these steps:
  - Three corners are chosen and the distances between them in the 2D plan are calculated. The distance in meters is calculated after using the haversine formula (see section 2.3.1).
  - The distance in the 2D plan from the three corners to the measured location is calculated.

– The distance in meters from the corners to the chosen point is found. This is done by translating the known distances calculated in the first step.
– Using triliteration, the values in latitude and longitude of the chosen point are found.

After the position is calculated both in 2D values, and latitude and longitude, the application must upload them into the database, using the API.

4.2.1.2 User Application

The user application is designed following an MVC pattern as well, shown in figure 4.3. The components of MVC achieve similar goals as the admin application.

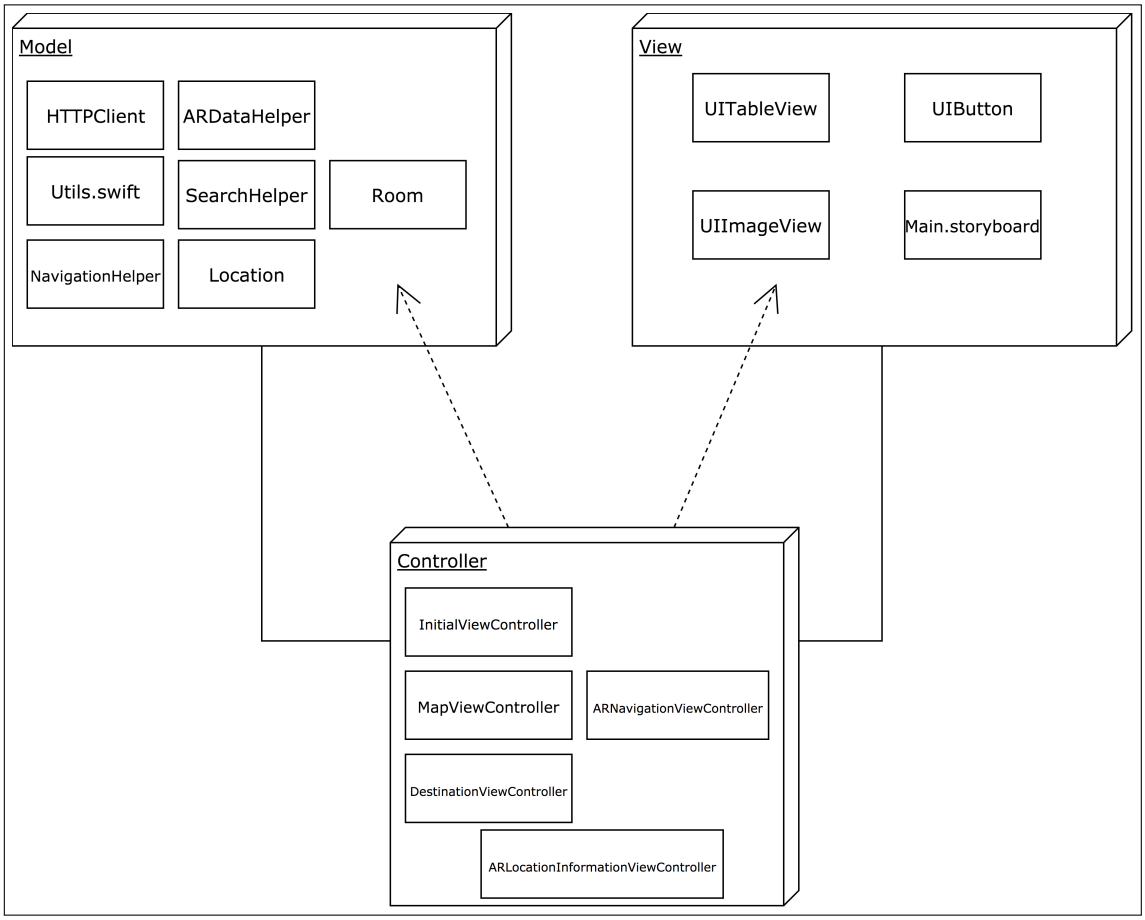

Figure 4.3: User Application MVC Design. The design is split into model, view and controller. The rectangles inside each component represent the classes that are part of it.

The user application will greet the user by calculating their current position by showing a dot on the floor plan that belongs to the floor they are currently on. From there, the user can view a list of possible destinations and by selecting one, a path will be shown on the screen on how to get there. Another action that the user can make is using the AR features. The AR features are:

- Images of timetables will render and placed in the rooms that are around.

- The number of computers will also be shown and placed in the rooms that are around.

One important matter to mention is how the AR objects will be placed. Based on the user's current location, the closest locations to each room that are around are calculated. By using those and the ARKit+CoreLocation library, the AR objects will be placed on the corresponding Location objects, using their latitude and longitude values.

4.2.2 Scanning Layer

The scanner layer will handle measuring for Wi-Fi signal strengths and uploading them in the database when needed. This layer is comprised of two applications: the scanner which will run on a Raspberry Pi, and a small web application that will handle the scan requests between the mobile applications and the Raspberry Pi.

4.2.2.1 Raspberry Pi Scanner Application

The first application is a Java application managed by a Linux script that runs every 2 seconds and implements the following functions:

- Scan and store Wi-Fi networks found around using Linux system commands.
- Check if the data needs to be uploaded.
- If yes, upload data onto the back-end storage, and then set the need to scan flag to off.
- If not, don't do anything.
- Run the above mentioned set of steps every 2-3 seconds.

4.2.2.2 Scanner Switch Application

The second part of the scanning layer is a small web application that acts as a switch. The Scanner Switch Application is going to be developed using Vapor, a Swift framework which follows an MVC pattern as well (see figure 4.4). Although this design pattern has been used traditionally for applications that have a graphical user interface, it has become popular for web applications as well. In this case, the application runs on a server, without a graphical user interface, so the JSON takes the role of the View component in this case.

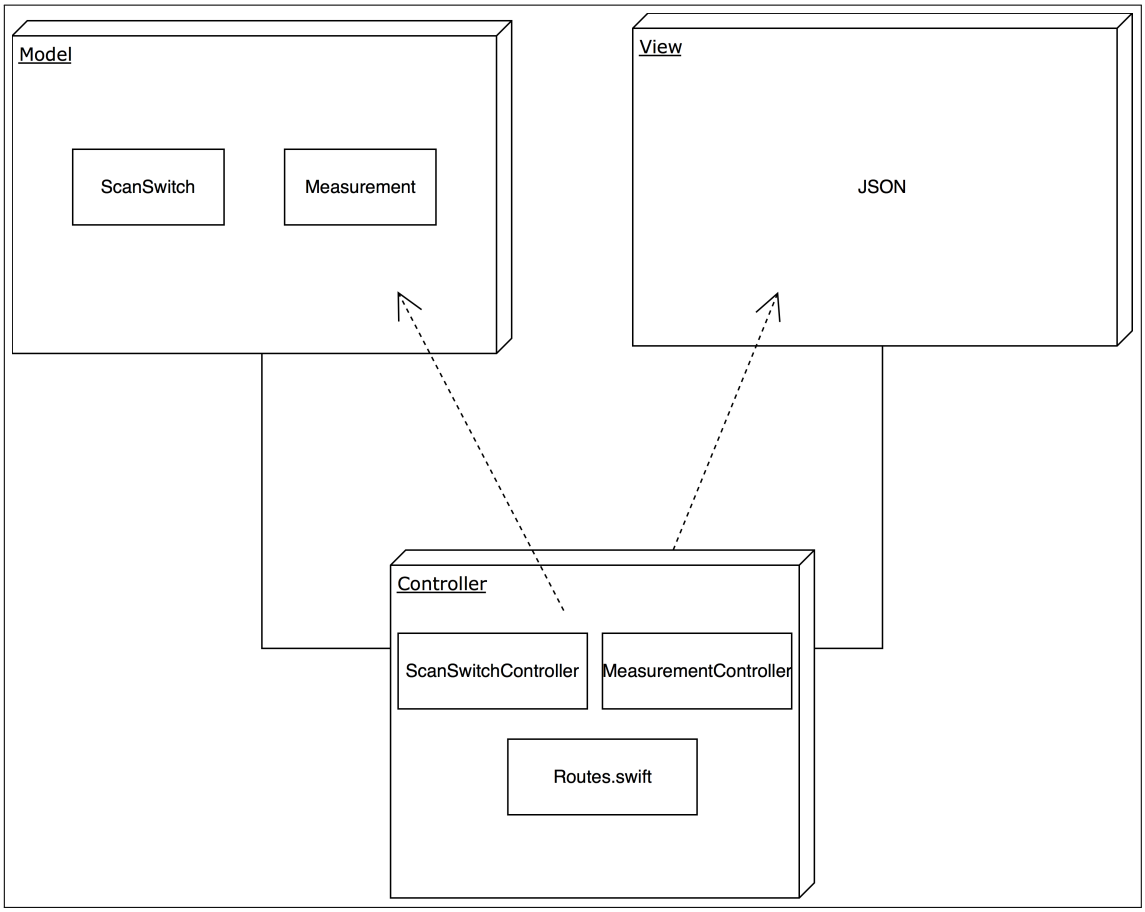

Figure 4.4: Scan Switch MVC Design. The design is split into model, view and controller. The rectangles inside each component represent the classes that are part of it. The arrows show that the Controller manages both the Model and the View.

The goal of this application is to let the scanning application on the Raspberry Pi know if data is needed or not. The data is stored in an in-memory database which will contain the following fields:

- scanSwitch: Can be true or false, depending if data is needed or not.
- locationID: This will be used when uploading data in the database in order to link Measurement objects to a Location object.
- roomID: This value will be used to link the Location object for which the Measurement objects are scanned to a room.
- storeData: Can be true or false, if the data needs to be stored or not. For example, when measuring locations the data needs to be scanned, but when the scanned data is used only for navigation or positioning, the data doesn't need to be saved. If the data doesn't need to be saved, the measurements will be temporarily stored on this server, and then discarded after the positioning process has finished.

This application implements a REST API that handles HTTP requests in order to access

the data. HTTP requests are made to "http://address/resource", where resource is the resource targeted. The following resources are available:

- For resource "/scanSwitch/1":
  - PATCH: Updates the values of the aforementioned fields.
- For resource "/measurements":
  - POST: Creates a temporarily stored measurement.
  - DELETE: Deletes all the measurements.

4.2.3 Logic Layer

The Logic Layer is composed of two systems: the AR data provider and the positioning & path finding system. These two subsystems provide the core algorithms and functions that the system relie on.

4.2.3.1 Positioning & Path Finder System

This part of the project is the most important one, because the positioning and path finder system calculates all the data to position the user and find optimal paths. Because it is developed using Vapor, an MVC based web framework, it follows an MVC design pattern. Similar as before, the View component is here represented by the JSON responses.

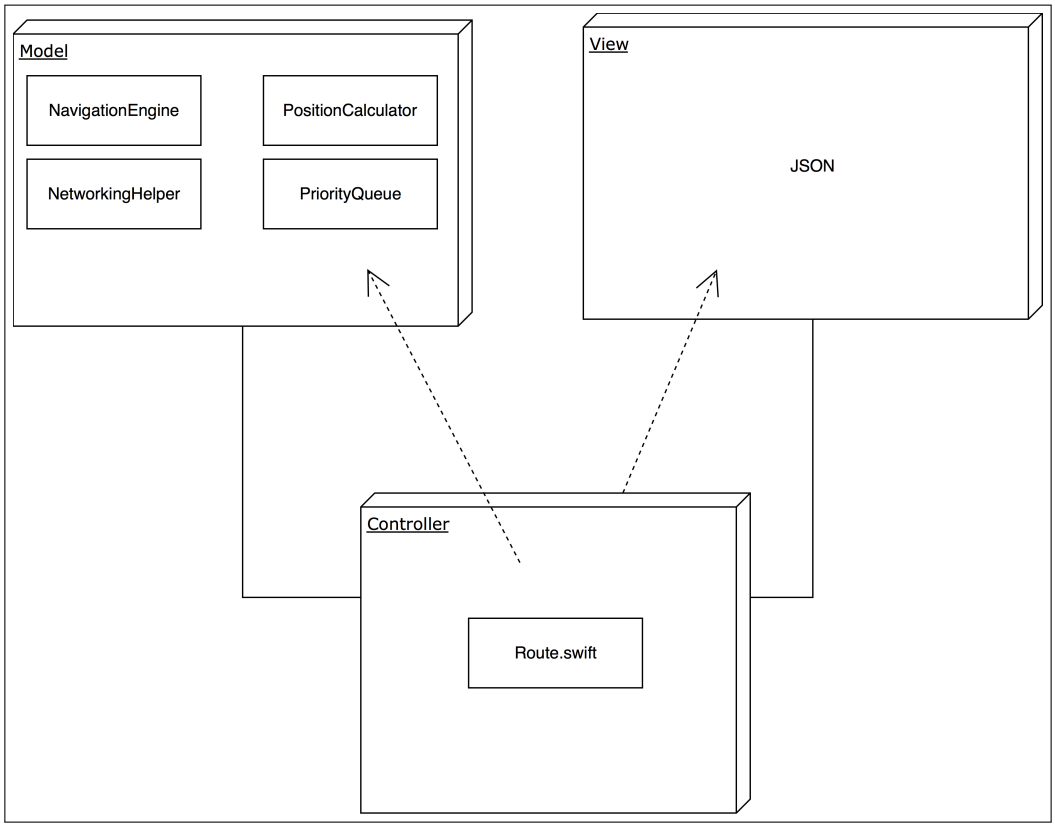

Figure 4.5: Positioning & Path Finder System MVC Design. The design is split into model, view and controller. The rectangles inside each component represent the classes that are part of it. The arrows show that the Controller manages both the Model and the View.

The positioning algorithm will be based on a best match algorithm, working as follows:

- For each measurement received, query the database and find all the past measurements recorded from the corresponding Wi-Fi access point.
- Calculate the difference in signal strengths between the current measurement and the past one.
- Find the location that has the minimum difference from all the differences calculated.
- Keep count of how many times locations have matched.
- At the end, find the location with the most matches.

Finding routes will be done using Dijkstra's algorithm, described in section 2.3.2. A graph will be built using the data from the database, and based on that, the shortest path will be calculated. The end result will be a list of location IDs to follow along with the total distance of the path.

This system implements an API to retrieve the calculated data, such as the current position. The API handles HTTP requests to "http://address/resource", where "resource" is the targeted resource. The resources available to access are the following:

- For resource "/determinePosition":

  - POST: accepts an array of current measurements and returns a JSON object of a location that has the 2D coordinates along with latitude and longitude

- For resource "/calculateRoute":
  - POST: accepts a JSON object that includes a "startLocationID" and a "finishLocationID" which represent the start and finish locations of the desired route. The return result will be a JSON object that is made of the distance of the path, along with an array of the location IDs to follow to get from start to finish.

4.2.3.2 AR Data Provider

The AR Data Provider will consist of an application that will be able to parse data from the university timetable system and from PC-Free@King's which provides how many computers are in the computer labs. Unfortunately, PC-Free has data for only one room located in Bush House, so not all rooms will be supported. This limitation has been detailed in section 3.7. The data will be obtained through a RESTf API, which, as before described with the previous APIs, handles HTTP requests made to certain resources:

- For resource "/timetable/[roomCode]":
  - GET: Returns a JPG image of the timetable for the current week, for the specified room.

- For resource "pcfree/[roomCode]":
  - GET: Returns how many computers are available in the specified room, and if there are not any, returns "No computers are available".

4.2.4 Data Management Layer

The Data Management Layer holds and manages the data which the logic layer uses. This layer is ran using a Vapor server that manages a PostgreSQL database. Similarly to the previous logic layer, the database can be accessed through an API. Due to the fact that the implementation is done using Vapor, the design is done by using an MVC patter, shown in figure 4.6. As before, the View component here is represented by the JSON responses from the API that is going to be detailed below.

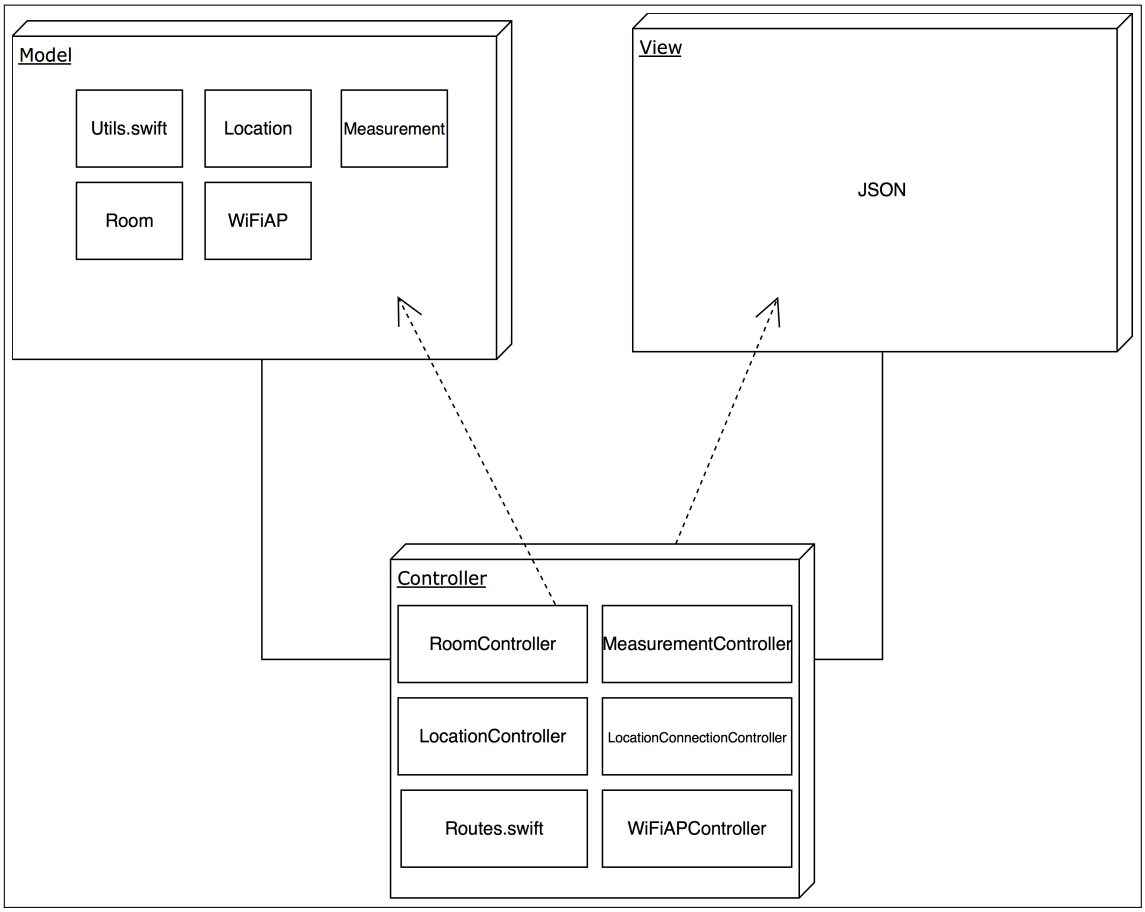

Figure 4.6: Back-end Data Management Server MVC Design. The design is split into model, view and controller. The rectangles inside each component represent the classes that are part of it.

The API handles HTTP methods to "http://address/resource", where the resource is targeted. Additionally, "/resource/id" can be accessed, if available, in order to access a certain resource, identified by the id. The resources that can be accessed are the following:

- For resource "/locations":
  - GET: Returns a JSON representation of all locations in the database.
  - POST: Accepts a JSON representation of the Location object and creates a new instance with the specified values.
- For resource "/locations/[id]":
  - GET: Returns a JSON representation of the location that has the specified id.
- For resource "/locations/floor/[floorNumber]":
  - GET: Returns a JSON of all the locations that are on the specified floor.
- For resource "/rooms":
  - GET: Returns a JSON of all the rooms created in the database

  - POST: Accepts a JSON representation of the Room object and creates a new instance with the specified values.

- For resource "/rooms/[id]":
  - GET: Returns a JSON of the room that has the specified id.
  - PATCH: Replaces the values from the room identified by the specified id with the new values submitted.
  - DELETE: Deletes the room that has the specified id.

- For resource "/rooms/clearData/[roomID]":
  - GET: Clears the data that is associated with the room that has the specified roomID.

- For resource "/rooms/search":
  - POST: Accepts a JSON with a query to search for and will return all the rooms as JSON that match the query.

- For resource "/rooms/connectingLocation/[id]":
  - GET: Returns the location as JSON from the room with the specified id that is connected to an outside connection (i.e. another room).

- For resource "/rooms/floor/[floorNumber]":
  - GET: Returns all the rooms as JSON that are on the specified floor number.

- For resource "/accessPoints":
  - GET: Returns a JSON with all the access points that are in the database.

- For resource "/accessPoints/[id]":
  - GET: Returns a JSON with the access point that has the specified id.

- For resource "/measurements":
  - GET: Returns a JSON of all the measurements in the database.

- For resource "/measurements/[id]":
  - GET: Returns a JSON of the measurement identified by the specified id.

- For resource "/measurements/address/[macAddress]":
  - GET: Returns a JSON of the measurements from the access points that have the specified mac address.

- For resource "/locationConnections":
  - GET: Returns a JSON of all the connections between locations.

- For resource "/locationConnections/id/[locationID]":

- GET: Returns a JSON of all the locations that are connected to the location identified by the specified location id.

- For resource "/linkLocations":
  - GET: Links all the locations in every room between themselves.

4.2.4.1 Data Storage Objects

The database stores data used for positioning and navigation. The data models defined to be used in the database, together with the relations between them are shown in figure 4.7.

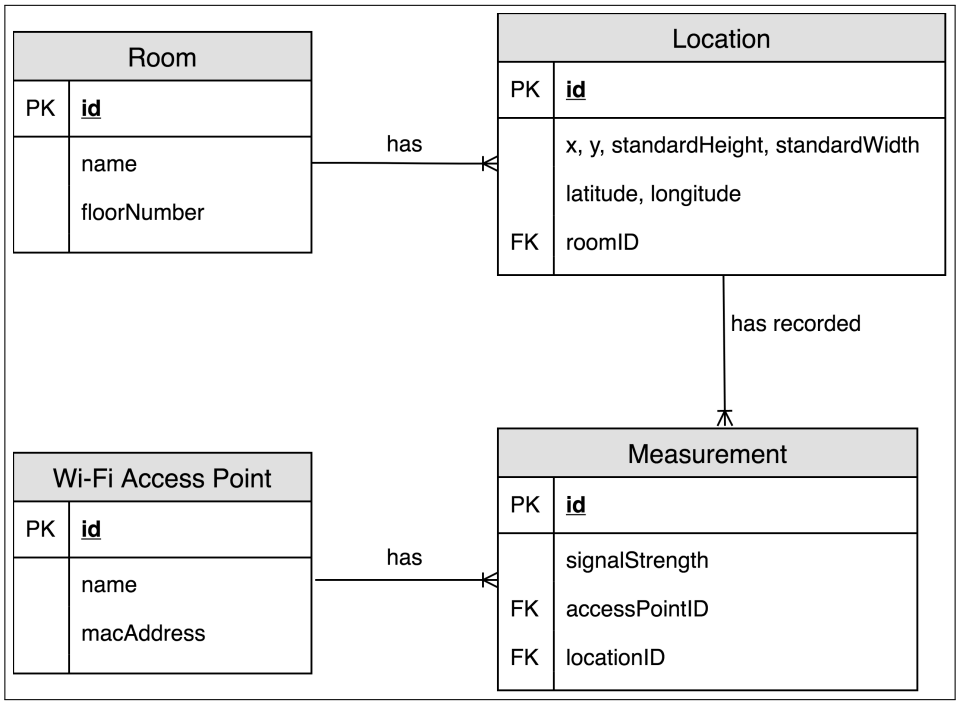

Figure 4.7: ERD diagram of the database objects. The rectangles represent the data models along with their properties; they are connected between themselves by relations. The arrows represent a one to many relation.

For the data models just presented in figure 4.7, the following properties have been defined:
Room:

- Name: the name of the room as a string.
- Floor number: the floor number where this room is located.

Location:

- 2 coordinates (x, y) for the location measured on the image, along with 2 more values: width and height. These values represent the size of the screen the location has been recorded. Based on these values, x and y can be adapted to any screen size.
- Latitude

- Longitude
- Room ID: the link between a location and its corresponding room.

Measurement:

- Signal strength: the value of the signal strength recorded.
- Access Point ID: the link between a measurement and its corresponding Wi-Fi access point.
- Location ID: the link between a measurement and the location where it was registered.

Wi-Fi Access Point:

- Name: the name of the network as a string.
- MAC Address: the unique identifier of the network.

4.3 Data Flow

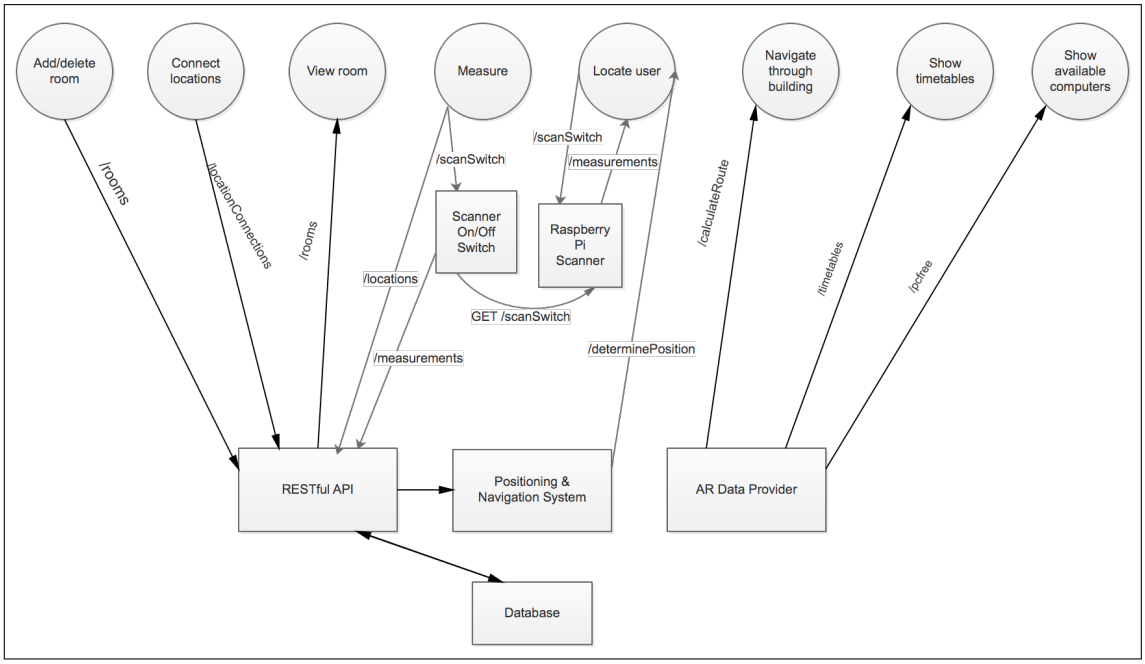

Figure 4.8: Data Flow Diagram. Represented in the figure is how the information is passing throughout the whole system. Circles represent the processes, and the rectangles and squares represent the data processing units.

On the first run of the admin mobile application, the user will be greeted with a list of the available floors to measure. From there, on each floor rooms can be added by inputting their name. The data is then converted into JSON format and sent to the database through the RESTful API that receives a POST request to "/rooms", where data is parsed and saved into a Room object. After rooms are created, the admin application will request to fetch them by making a GET request to "/rooms" which will return a list of the registered rooms. This will facilitate registering for locations in the rooms the user will select.

Next, locations can be measured in the selected room by following the steps:

- A Location object is created by making a POST HTTP request to "/locations".
- The Scan Switch is turned on by being set to true and the roomID and locationID that need to be assigned to the measurements scanned are sent. This is done by making a PATCH HTTP request on "/scannerSwitch/1".
- The Raspberry Pi detects that the Scanner is turned on, saves the roomID and locationID, and then the scanned data is assigned to those ids and the measurements are uploaded by using the RESTful API and making a POST HTTP request on "/measurements".
- Data is being processed from the JSON format and saved into the database.

After the locations are created and Wi-Fi measurements have been scanned, the next step is to connect certain locations in order to provide navigation. The admin user will then open the

corresponding floor plan for the floor that they are on currently. This will send a GET HTTP request to "/locations/floor/[floorNumber]", where the desired floor number is specified. Given the set of locations, the admin user can select two of them that are on the floor plan in order to connect them. This will send a POST HTTP request to "/locationConnections" with the corresponding location ids that need to be connected together.

The next steps are made by the user application, which will request to position the user after the application is launched. The positioning process starts by turning the Scanner Switch on, but this time by setting an additional flag, "storeData", to false. This will let the scanner know that the current measurement data will be only needed to compare it to the already existing data, and not stored in the database. After scanning is done, the measurement data is downloaded on the mobile phone as JSON, and then sent to the Positioning & Path Finding System by making a GET HTTP request to "/determinePosition". The current position will be returned as JSON, parsed by the mobile application, and shown on the floor plan.

In order to facilitate navigation, a POST request will be made to the Positioning & Path Finding System, which will contain a JSON of the current position and the finish position. The server will then return a JSON of the path to follow and the distance of the path. The application will show the route on the floor plan, along with the estimated time of arrival and the total distance. The navigation process in the user application can be put in "Augmented Reality" mode as well, where an arrow will guide the user. Whilst walking to the destination, visual information will be available, which are part of the AR features earlier mentioned. This data will be retrieved by making GET HTTP requests to "/timetable/[roomNumber]", which returns an image of the timetable for the specified "roomNumber", or to "/pcfree/[roomNumber]" which will return the number of available computers for the specified "roomNumber".

4.4 Third Party Content

The system uses some third party code components which allow it to meet part of the requirements and specifications. By using third party content, time is saved and productivity is increased. Below, the third party content used in this project are described.

4.4.1 SwiftSpinner

"SwiftSpinner" is an open source library to show visual cues as loading spinners when a background task needs to be executed. This allows to have a better user experience, because by using a spinner, the user knows that they have to wait for data to show in order to use a specific part of the application.

4.4.2 ARKit + CoreLocation Library

The system uses "ARKit + CoreLocation", which is an open source library that makes it easy to place elements of AR on locations, given the latitude and longitude. By default, the library using CoreLocation for positioning and handling locations. CoreLocation is a system framework implemented in iOS which uses the mobile phone's GPS and GLONASS chips for that, but this comes into conflict with what the project wants to achieve: an indoor navigation system that uses only IPSes. Therefore, the library has been modified in such a way that it

will use the project's indoor location system, instead of CoreLocation. This modification will be explained in the Implementation chapter, where examples of the code will be provided as well.

Chapter 5

Implementation & Testing

This chapter will outline how the design plans previously described have been implemented and put into play. The chapter will start with discussing how the development process has been approached, and then the system features. The chapter will conclude by describing the issues that have been raised during the process, along with how they have been resolved. A priority in this project has always been to make it as extensible and as decoupled as possible.

5.1 Development Approach

The approach to software development was dictated by the fact that the project was conducted by only one individual. As such, a lean approach was adopted, where small iterations of the project and features are incrementally implemented and integrated.

The process has started by prototyping in order to figure out the feasibility of using certain libraries to achieve the proposed goals. Thus, the first experimental application was one that used ARKit in order to place different AR objects around Bush House in convenient location. This led to a better understanding on how ARKit will be later used and how feasible it is to implement the planned features. After this, the capabilities of Swift running on the server through the Vapor framework, and a small server that implements a REST API has been developed.

During the development process, the independence between all the parts of the system has been closely followed, along with increased re-usability and flexibility to change without affecting the whole system in any way. The order of the developed work has been decided based on the dependencies between the sub-systems and layers, considering the following points:

1. Create the system where data can be stored in.
2. Build the way that allows the system to collect data.
3. Create the data processing units (the logic layer).
4. Finish by integrating all the sub-systems into the front-end that is going to use the end result of the logic layer (the user application).

Given the previous points that describe the strategy adopted, the following steps have been taken to in order to build the system. Each step describes what part has been developed, and

what requirements have been satisfied.

1. Create the database schema for the database objects along with their respective classes (BAPI1, BAPI2).
2. Make the database objects accessible through a REST API. The resources necessary for the applications to retrieve data have also been developed here (BAPI3, BAPI4).
3. Build a scanner and parser on the Raspberry Pi and an autonomous script that will run the scanning process every 2 seconds (RPS1, RPS2, RPS3).
4. Create a Scanner Switch on a server that will notify the Raspberry Pi when the data is needed (WSS1, WSS2).
5. Build UIViewController to view available floors that are available to measure from the Admin Application.
6. Build UIViewController on the Admin Application that will manage adding and deleting rooms from the database (AM1, AM4).
7. Build a UIViewController on the Admin Application to view the floor plan corresponding to the selected room. Event handlers for tapping on the screen have been created here too. This will start the scanning process by turning the Raspberry Pi on, creating a Location entity associated to a room, and then the data that has been scanned from the Wi-Fi networks around will be assigned to that Location (AM3).
8. Create the event handler for the Admin Application to connect two Location entities. The link between them will make navigation possible. This concludes data collecting, which gives way to develop the positioning algorithm (AM3).
9. Implement the path finding algorithm by using Dijkstra's algorithm, and the positioning algorithm (BS1, BS2).
10. Create the UIViewController on the User Application that shows the user's location in the correct floor and room (UM1)
11. Create UIViewController that shows all available rooms to select as destinations, along with a way to search for them (UM2, UM3).
12. Show the path calculated from the server on the User Application, with its respective distance and estimated time arrival (UM4, UM5).
13. Create the button to switch to the AR mode; this will show an arrow when the user is currently navigating to a destination (UM6).
14. Implement the parser to get timetables from King's College London Timetable service (ARDP1).
15. Implement the parser to get the number of computers in computer labs from PC-Free@King's (ARDP2).
16. Build the UIViewController on the User Application that will allow the user to see augmented reality objects of timetables of the rooms around (UM7).

17. Build the UIViewController on the User Application to show the number of unused computers of the rooms around, if available (UM8).

5.2 System Functionalities

This section will describe how each part of the system has been implemented and developed. It will start by outlining the lowest layer, the Data Management Layer, and it will conclude with the upper most layer, the User Interface Layer. The core functions for each layer will be portrayed in detail, along with pieces of code where needed.

5.2.1 Data Management Layer

5.2.1.1 Database

Initially, the database has been implemented by using SQLite3, but because of deployment issues on Heroku, it has been changed to PostgreSQL. The first models to be created in the database have been Room and Location. After the design phase, the following models have been created:

Room: This is the entity which lays at the base of all the other models. It follows the model outlined in the Design chapter 4, section 4.2.4.1.

```
roomName: String, floorNumber: Int, id: Identifier
```

Location: This entity belongs to a Room and describes a single location in a room. It has 2 sets of coordinates, because a Location is based into 2 coordinates system: 2D (on the floor plan), and geographic (latitude and longitude). The 2D coordinates is used in order to place a Location object on the mobile phone's screen, and the geographical ones are used for navigation purposes. Between a Room and a Location there is a one-to-many relationship (a Room has many Locations), and a Location must belong to a Room.

```
x: Double, y: Double, standardWidth: Double, standardHeight: Double,
latitude: Double, longitude: Double,
roomID: Int,
id: Identifier
```

Measurement: This entity contains only the signal strength part of a network scan. The initial implementation included the access point from where the signal was collected, but in order to avoid duplication, the two entities have been separated through a one-to-many relationship from Access Point to Measurement. Finally, a Location is connected to Measurements through a one-to-many relationship.

```
signalStrength: Int, accessPointID: Int, locationID: Int
```

Wi-Fi Access Point: This entity describes the Wi-Fi Access Point from where the Measurements have come from.

```
name: String, macAddress: String
```

5.2.1.2 REST API

The REST API has been implemented using Vapor which relies on Swift. HTTP requests have been handled using the HTTP module under Vapor. JSON encoding and decoding have been made using the JSON module under Vapor, as well. The resources and endpoints used to access the database for CRUD operations have been implemented by following the points described in the Design section.

5.2.2 Logic Layer

5.2.2.1 Positioning & Path Finding System

This system has been implemented using Vapor and by using Swift. The system relies on just two classes – a PositionCalculator and a NavigationEngine, which will be described into detail in the following sections. In order to access the functionalities of these classes, several endpoints have been developed.

PositionCalculator.swift

This class is responsible for calculating the user's current position, based on a set of Measurements received in a JSON format. The class is a singleton and inherits from NSObject, which is the class that has base functionalities in Swift. Calculating the position starts by sending a JSON array which has network names, MAC addresses and signal strengths from the scanned networks. The JSON data is included in a POST HTTP request made to "/calculatePosition", from the Java application that runs on the Raspberry Pi. A sample scan result in JSON format is:

```
[
    {"macAddress":"00:62:EC:FD:EC:50","signalStrength":-62,"name":"eduroam"},
    {"macAddress":"00:62:EC:FD:ED:72","signalStrength":-59,"name":"The Cloud"},
    {"macAddress":"00:62:EC:FD:EC:C2","signalStrength":-64,"name":"KINGSWAP"},
    {"macAddress":"00:62:EC:FD:EB:14","signalStrength":-63,"name":"SLaMFT"}
]
```

The following step is to parse the data received, and to call the determinePosition(for measurementsCollection) in the calculator, which returns a Location object.

```
func determinePosition(for measurementsCollection: MeasurementsJSONCollection) ->
    Location?
```

The method follows the methodology outlined in Section 4.1.3.1. It starts by initialising a dictionary object, "locationMarks", which will store the number of matches a locationID has. In other words, the keys will be location ids that have resulted in calculation, and for any location id, the result retrieved will be the number of matches. Next, a loop will iterate through the measurementsCollection, and for each measurement, the MAC address and the network's name will be extracted and used to query for old measurements from the database. The following code is the implementation of this:

```
var locationsMarks = [Int: Int]()
```

```
    for currentScanMeasurement in measurementsCollection.measurements {
        let currentScanMacAddress = currentScanMeasurement.macAddress
        let currentScanSignalStrength = currentScanMeasurement.signalStrength

        guard let searchResults = searchForOldMeasurements(for:
            currentScanMacAddress)?.results else { print("Not found!"); continue }
```

Following this, another loop iterates through the searchResults, and for each result, calculates the following difference:

$$difference = |(currentScanSignalStrength * -1) - (result.signalStrength * -1)| \quad (5.1)$$

Because the signal strengths are received in negative values, in order to get the correct difference between them, they are multiplied by -1 to get their positive value and then the end result is taken in its absolute value. The following step is to find if this difference is smaller than the ones recorded before the current step for this search result, and if it is, store its locationID.

```
for result in searchResults {
    let diff = abs((currentScanSignalStrength * -1) - (result.signalStrength * -1))
    if diff < minDiff {
        minDiff = diff
        minLocationID = result.locationID
    }
}
```

At the end of the for-loop that iterates through the searchResult, for the locationID that had the smallest difference between its measurements and the scanned measurements, its matches value is increased in the dictionary.

```
if locationsMarks[minLocationID] != nil {
    locationsMarks[minLocationID] = locationsMarks[minLocationID]! + 1
} else {
    locationsMarks[minLocationID] = 1
}
```

The last step is to find which locationID has the most matches, and return the corresponding Location object for that locationID.

```
for (key, value) in locationsMarks {
    if value > maxMatches {
        maxMatches = value
        locationID = key
    }
}

guard let location = getLocationDetails(for: locationID) else { return nil }
return location
```

The result is then encoded into a JSON format of the Location object. Additional fields have been added in order to have additional information in the user application, such as:

- A ”success” field which is true if the calculation was successful, or false if it failed.
- ”floorNumber” which will help the user application show the correct floor plan.

A sample result of a positioning calculation will look like this:

```
{
    "roomID": 2,
    "latitude": 51.5124257954173,
    "standardHeight": 1024,
    "id": 3,
    "floorNumber": 7,
    "x": 348.5,
    "longitude": -0.117183612390869,
    "y": 862.5,
    "standardWidth": 768,
    "success": 1
}
```

NavigationEngine.swift

This class is responsible for calculating routes between any two given Locations. Same as before, the class inherits from NSObject and is a singleton. Calculating a route starts by a POST HTTP request to ”/calculateRoute”. The input received is a JSON object that has only two values: the location ID of the starting point and the location ID of the finishing point. This JSON object is parsed by a HTTP requests handler class in the application, data is fetched from the database to receive more information about the two locations, and then the process starts. The below input is a sample input received by the server:

```
{
    "startLocationID": 1,
    "finishLocationID": 24
}
```

After the input has been parsed, the process will create the graph which Dijkstra’s algorithm will use. The first step towards creating the graph was to create the data structures that will act as nodes as edges. The data structure for nodes has already been created, which is the Location entity, but for edges, the following data structure has been built:

```
struct Edge: Decodable {
    let rootLocationID: Int
    let childLocationID: Int
}
```

The structure inherits from ”Decodable” which is an interface that allows a JSON format to be transformed into an instance of the structure. Taking this into account, the next step is

to get the connections between all the Locations; in other words, the edges between them. The edges are stored into a dictionary *edges*, where $edges[locationID]$ is an array of locationIDs that belong to Locations connected. The code below depicts what has been said: first, all the Locations are fetched, and for each Location, the connections are fetched and the graph is assembled.

```
func createGraph() {
    guard let locations = NetworkingHelper.shared.fetchAllLocations() else { return }
    self.locations = locations // the result is saved into a global variable

    for location in locations {
        let id = location.id
        distance[id] = INF // For each locationID, the array of distances which will
            be later used is initialised; in other words, the distance is infinite =>
            it's unreachable
        prev[id] = 0 // Same for the array of parents; the initial value is set to 0,
            in other words, the location does not have a parent for now

        guard let currentEdges = NetworkingHelper.shared.getEdges(for: location.id)
            else { continue }

        for edge in currentEdges {
            // If the array of ids has not been initialised, the initialisation happens
                here
            if edges[edge.rootLocationID] == nil {
                edges[edge.rootLocationID] = [Int]()
            }

            // The child's locationID is appended here to the array of connections in
                the dictionary
            edges[edge.rootLocationID]!.append(edge.childLocationID)
        }
    }
}
```

Next, after the graph has been created, the value in the *distance* corresponding to the starting point is updated to 0, the node is set as visited, and the priority queue is initialised. Before the algorithm starts, the starting node is added into the priority queue. The following code describes the main process of the function:

```
while(!queue.isEmpty) {
    guard let node = queue.remove() else { break } // Removes the first node in the
        queue; to avoid force unwrapping the value which causes a crash, a guard-let
        structure is used
    visited[node] = false // First set its value as not visited
    if let edges = edges[node] { // Again, avoid force unwrapping
        for neighbour in edges { // Iterate through all the node's neighbours
            if let dist = distance[node] { // Unwrapping it's distance
```

```
                let alt = dist + getDistance(from: node, to: neighbour) // Compute the
                    alternative distance
                if let neighbourDistance = distance[neighbour] { // Unwrapping the
                    neighbour's current distance
                    if alt < neighbourDistance { // Checking if the previously
                        calculated distance is better, and if it is, replace it and add
                        the node in the queue
                        distance[neighbour] = alt
                        prev[neighbour] = node // Add the node as the neighbour's parent
                        queue.insert(neighbour)
                    }
                } else {
                    distance[neighbour] = INF // If it's value is null, initialise to
                        INF.
                }
            } else {
                distance[node] = 0 // Set it to 0 if it's null.
            }
        }
    }
}
```

An important point to discuss here is how distance is calculated. Because the Location object already stores the latitude and longitude values, it is therefore easy to calculate the real life distance between them. This uses the haversine formula previously described in section 2.3.1. This function is already implemented in the Foundation framework which Swift uses. The Foundation framework provides access to essential data types, collections and others, that define the base layer of any Swift based application.

The last step of the algorithm is to create the path by backtracking it, using the *prev* array. As stated in the Background section, this array stores an element's parent; in other words, for $prev[v] = p$, where p is the parent of v. The path will be backtracked from the destination to its start.

```
func createPath(parentsList: [Int: Int], distances: [Int: Double], start: Int, finish:
    Int) -> [String: Any]? {
    if distances[finish] == INF { // If the distance for the destination is infinite,
        that means it has never been reached.
        return [
            "sucess": false
        ]
    }

    var path = [Int]()
    var currentNode = finish
    path.append(currentNode)
    while (currentNode != start) {
        if parentsList[currentNode] != nil {
            path.append(parentsList[currentNode]!)
```

```
            currentNode = parentsList[currentNode]!
        }
    }

    return [
        "success": true,
        "distance": distances[finish]!,
        "path": path.reversed() as [Int] // This array has all the locationIDs to
            follow.
    ] as [String: Any]
}
```

The end result after creating the path is then encoded into JSON format and returned to the mobile phone. The JSON object will have the distance of the path, the array of locationIDs and a success field which role is to inform the client if the calculation has been successful or not.

```
{
    "success": true,
    "path": [
        1,
        3,
        13,
        15,
        16,
        18,
        20,
        21,
        24
    ],
    "distance": 78.4722368482206
}
```

5.2.2.2 AR Data Provider

The initial design of this system was to be included in the one system, together with the Positioning & Path Finding system. However, due to the lack of a web automation library that in combination with a parser will retrieve data, these two systems have been separated instead.

Instead of Vapor and Swift, this system has been written in Python, using Selenium, an automation tool for browsers. Next, the software has been deployed to work through a server with a lightweight API to access data. The framework used for deploying it and API management is Flask, which allows to do web development using Python.

Timetable information

Getting timetable information for a specific room starts with a GET HTTP request to "/timetable/:room", where ":room" is the room code (e.g. 6.01 is computer lab 1 on the sixth

floor in Bush House).

The way the provider works is fairly simple. Using Selenium, Chrome is launched on the server, then the web browser navigates to "http://timetables.kcl.ac.uk". From here, the Locations module of the timetable website is accessed. This shows a form that is going to be filled in automatically with the required data in order to obtain the desired timetable.

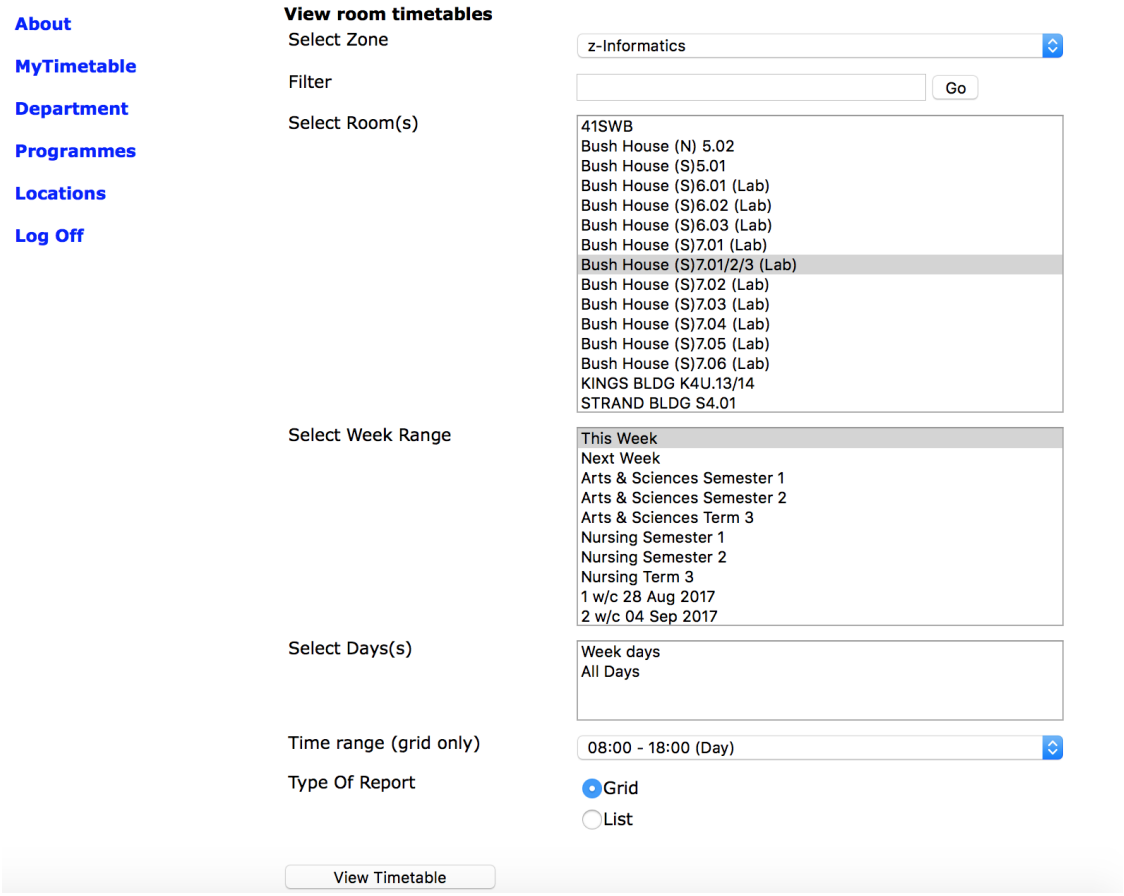

Figure 5.1: KCL timetable with form to choose rooms. This is the standard form where students can put information to see the timetables. This will be filled in automatically by the Python script with the necessary information, and the "View Timetable" button will be pressed at the end.

The end result will be a screen-shot with the timetable for the requested room. The image will be set as background for an augmented reality object in the user application. Next, it will be placed in the closest Location that belongs to the requested room, from the user's current position. This process will be further detailed in the section for the user application.

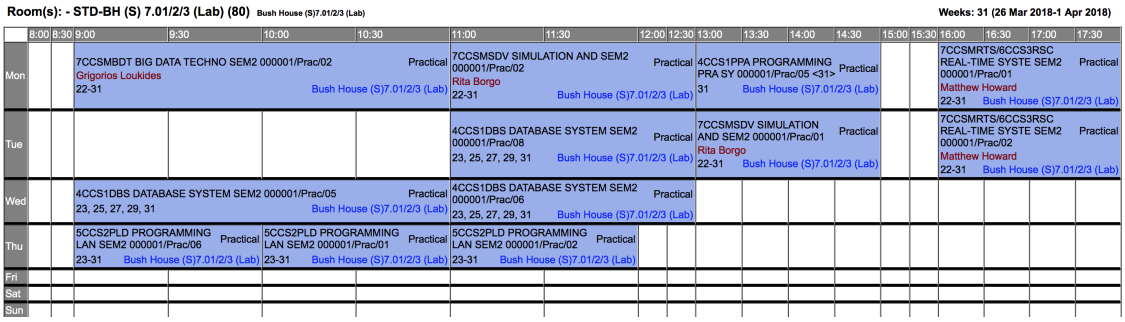

Figure 5.2: Sample timetable for room 7.01/2/3. This is the result of the form shown in figure 5.1.

PC-Free information

The PC-Free provider works in a similar way as the timetable one. To access data for a room, a GET HTTP request is made to "/pcfree/:room", where as before, ":room" is the code of the room. The web driver then navigates to "http://pcfree.kcl.ac.uk", and searches for the element that matches the room for which the data was requested. Once found, a screen shot is taken from the whole screen, and then cropped around the area of interest.

```
def get_available_computers(room):
    for element in driver.find_elements_by_class_name('campus_info'):
        square = element.get_attribute('innerHTML')
        bs = BeautifulSoup(square, "html.parser")
        full_room_name = bs.find('h2').text
        if "BH(S)" in full_room_name:
            if room in full_room_name:
                location = element.location
                size = element.size
                img = driver.get_screenshot_as_png()
                img = Image.open(BytesIO(img))
                left = location['x']
                top = location['y']
                right = location['x'] + size['width']
                bottom = location['y'] + size['height']
                img = img.crop((int(left), int(top), int(right), int(bottom)))
                img.save('pcfree.png')
                return
    return None
```

The result is encoded into JSON format, and then sent to the requesting entity.

5.2.3 Scanning Layer

5.2.3.1 Raspberry Pi Scanner

The scanner on the Raspberry Pi is composed of a bash script that scans continuously for Wi-Fi networks, and then runs a Java application that parses the results and uploads them

if needed. In order to perform a scan, the system uses a Linux package, iwlist, and then the results are manipulated by grep, that matches them to a regular expression.

```
sudo iwlist wlan0 scan | egrep "Cell|ESSID|Signal" >> output.txt
```

Next, the Java application is started using Maven. The application will first check from the Scanner Switch if data is needed. If it is, it will parse the output file, and upload them either to the database, if data is needed for measuring, or temporarily uploads them on the Switch server, if data is used for navigation. The process ends by deleting the output file in order to make room for the next scan.

It is important to mention that the scanning process starts at boot time. After the Raspberry Pi boots, the script described earlier runs every two seconds, until the shut-down of the device.

5.2.3.2 Scanner Switch Web Application

This web application is positioned between the mobile applications and the Raspberry Pi scanner. This connection was initially planned to happen directly between the mobile devices and the Raspberry Pi via Bluetooth, but due to the difficulties encountered and the time constraints, this new design has been adopted. It is important to say however, that given the architectural design which is very loosely coupled, this connection type can be changed at any time. The web application is implemented using Vapor and Swift. It contains only two classes, which are ScanSwitch and Measurement.

ScanSwitch.swift

The Scan Switch class acts as a physical switch that stores boolean flags that need to be checked by the Raspberry Pi in order to trigger a scan.

```
shouldScan: Bool
roomID: Int
locationID: Int
storeData: Bool
```

Other than the shouldScan flag, ids for rooms and locations are temporarily stored too. These are used to make the connection between the measurements collected by the scanner with first the Location and then the Room. Lastly, the storeData flag is used by the Java application that decides where to upload the data: the measurements database, or temporarily on the scanner web application.

Measurement.swift

This class is identical to the one described for the server that manages the database. In order to avoid repetition, its fields and properties will not be included. As previously mentioned, this class is used to temporarily store data that is used for positioning. After the positioning process ends, the data is discarded.

5.2.4 User Interface Layer

The User Interface Layer is an important component, because it acts as the primary way for the user to interact with the system. The graphical user interface provides a clear and simple way to measure floor plans, see recorded locations, manage rooms or navigate through the building. The implementation choices and features for each mobile application will be described in the following sections.

5.2.4.1 Admin Application

The Admin Application is implemented as an iOS application, written in Swift. It follows an Model-View-Controller pattern design, which is standard on iOS. The classes included in the mobile application are mainly used to make HTTP requests to ask for data, and then the View Controllers handle showing data onto the screen. Some simple calculations are being made in order to calculate the latitude and longitude coordinates, given the floor plan coordinate values.

5.2.4.2 Model classes

The model for this application is composed of a Location class, which is identical to the class in the server that manages the database, and is used in order to download or upload user locations. In order to avoid repetition, it will not be detailed further. The rest of the model classes are HTTPClient and Utils.

HTTPClient.swift

This class manages all HTTP requests that are being made in the app. Its design is very simple, it is based on a singleton design pattern, and contains only one method which is able to make any type of HTTP requests. The method uses URLConnection, which runs asynchronously on a background thread in order to not block the user interface thread. The method can submit a JSON object in the HTTP request if needed, and if there is a JSON object as a response, it will return it.

Utils.swift

This class contains utility functions which are used along the app in order to do small calculations. All of the calculations are geometry based, such as the haversine distance, the Manhattan distance, a circle intersection, and point coordinates interpolation. The following paragraphs will highlight how these functions are performed.

Haversine Distance

The Haversine has been mathematically detailed in the Background chapter. The code implements the aforementioned equations, and it is taken from the Swift Algorithm Club. The credit for this code has clearly stated in the class file.

```
let haversin = { (angle: Double) -> Double in
   return (1 - cos(angle))/2
}
```

```
let ahaversin = { (angle: Double) -> Double in
    return 2*asin(sqrt(angle))
}

// Converts from degrees to radians
let dToR = { (angle: Double) -> Double in
    return (angle / 360) * 2 * Double.pi
}

let lat1 = dToR(la1)
let lon1 = dToR(lo1)
let lat2 = dToR(la2)
let lon2 = dToR(lo2)

return radius * ahaversin(haversin(lat2 - lat1) + cos(lat1) * cos(lat2) *
    haversin(lon2 - lon1))
```

Circle Intersection

Circle intersection is being made between the corners of the building and the desired location that the admin wants to register. The values of latitude and longitude for all four corners of the building (top right/left, bottom right/left) are being loaded into the app, and then at launch, the user is put to register the locations of these corners in the 2D plan, by tapping on the screen. Based on this association of values, the distances are then calculated in the following way:

1. Two corners are chosen, and the distance between the desired location and each of those two corners is calculated using the Manhattan distance.
2. The hypotenuse of the triangle formed between these three locations is calculated.
3. The set of equations is therefore solved and the possible two intersections are returned.

It is important to mention that the solution is based on an external source which has been adapted to the current needs of the application. The source of the algorithm has been clearly specified in the source code. Please refer to the Appendix for more details.

After the intersection is performed, the Location object can then be uploaded into the database, with its 2D location and its latitude and longitude. The latitude and longitude positions are used for navigation, in order to find the distance, and to place augmented reality objects.

Interpolating 2D Coordinates

Location objects are being registered on different screen sizes. For example, the iPad is used for measuring, because the larger screen makes it more accurate when tapping on the floor plan image. Because of this, the coordinate set changes for each device the application runs on. In order to show the same position on the screen no matter what the size of the screen is, the 2D coordinates have to be adapted, based on the device they have been registered on. Therefore,

the points are interpolated based on the standard size (the source) and the current size of the screen. The code below illustrates this better.

```
static func interpolatePointToCurrentSize(point: CGPoint, from standardSize: CGSize,
    in view: UIView) -> CGPoint? {
   let currentSize = view.bounds.size // Getting the current size
   return CGPoint(
      x: point.x * currentSize.width / standardSize.width,
      y: point.y * currentSize.height / standardSize.height
   ) // Calculate it based on the registered size and the current one and return the
       corresponding CGPoint.
}
```

5.2.4.3 View Controller classes

This is the iOS application which will be used by administrators to manage rooms. The calibration process is achieved by asking the user to tap on the image on the top left & right and bottom left & right corners of the floor plan's image. This calibration is done in the CornerViewController class, and the values are saved in the UserDefaults persistent storage.

After calibration is done, view controllers that are handling rooms managing and location managing are being launched. These will be further illustrated below.

InitialViewController

This is the entry view controller of the app. It was built in order to decide whether the app needs calibration (in other words it is the very first launch), or not. From here, the CornerListViewController or the FloorListViewController are launched.

CornerListViewController

This class implements a UITableView for the user to choose what corner they wish to register. From here, tapping on one of the options will launch another ViewController, with an image of the floor plan. Tapping on the required corner will save the data locally, as previously mentioned.

FloorListViewController

Similarly to the CornerListViewController class, this class implements a UITableView as well. The list contains the floor that are in the Informatics deparment in Bush House, which are level 5, 6 and 7. Tapping on one of them will launch another view controller of the list of rooms registered on that floor.

RoomListViewController

This class implements a UITableView which is filled in with the rooms on the selected floor. From here, the user can add a room by pressing a "+" button and filling in the data, delete a room or clear the data from it. By pressing on a room, this will launch an image of the floor plan, from where the user can record locations. The methods available in this class use

the HTTPClient class to make HTTP requests to the database, in order to perform CRUD operations.

FloorPlanViewController

This is the last and most important view controller in the app. It implements an UIImageView which shows the floor plan for the selected floor. Upon loading, the already registered locations are placed on the floor plan by fetching them from the database. User can tap on the image on the locations where they need to register a location. Next, a request is made to the Scanner Switch to scan data with the roomID and locationID of the Location just created, and the measurements will be uploaded from the Raspberry Pi to the database.

Additionally, this view controller allows admins to connect to Locations, which helps in the navigation process. This is done by tapping on one Location square from the screen, which is set as the root location. The next step is to tap the location to connect the current one to, which is set as the child. After both have been selected, a HTTP request is made to the database through the REST API, and a LocationConnection entity is created.

5.2.4.4 User Application

This is the iOS application which will help users to find where they are in the building, how to get to certain other rooms and to find information about the rooms around them, such as timetables. In the implementation of this application, some model classes have been reused from the admin application or the database, such as Room or Location, or the HTTPClient.

The implementation of the application of the application consists of a model which mainly consists of helper classes that make HTTP requests to different APIs. To get the full implementation of them, please refer to the Appendix. In the following paragraphs, the most important parts of the application implementation will be highlighted.

5.2.4.5 Model classes

IndoorLocationManager.swift

As previously mentioned in the Design chapter, the application uses an external library (ARKit + CoreLocation) to place augmented reality objects in certain locations, given their latitude and longitude. By default, this library uses CoreLocation, which is a library part of iOS which manages locations. Because this library uses the mobile phone's GPS or GLONASS chips to determine the location. However, this defeats the purpose of an IPS, so a new location manager has been developed, that uses the system built to navigate indoors.

The location manager is composed of an interface that lets classes subscribe to location changes, and a few methods to start or stop updating the location, and a private method to determine the current position. Determining the position is being made by making HTTP requests first to the Scanner Switch to turn it on and to get the measurements recorded, and then to the Navigation System in order to determine the current position. Please refer to the Appendix to get the full implementation of this class.

Utils.swift

This class is similar to the one from the admin application. The methods defined here are for mathematical calculations, such as interpolating two coordinates set, which we have seen in the admin application, and to find the bearing between two Locations. The bearing is used for the AR part of the application, when an arrow will point to the direction that the user needs to follow in order to get to the destination. The implementation is based on a StackOverflow answer, which is stated in the source code.

5.2.4.6 View Controller classes

InitialViewController.swift

This view controller is the main entry of the application. On loading, it calculates the user's current position using the IndoorLocationManager, and loads the corresponding floor plan, based on the user's location.

MapViewController.swift

This view controller is composed of an UIImageView of the floor plan, where the user's location will be shown. The top part contains two UIButtons, from where the user can get to the DestinationViewController, which allows them to choose a destination, or to switch to the AR mode.

Upon selecting a destination, a path will be drawn on the screen from the user's current position, to the destination. This is performed using CoreGraphics, a framework part of iOS which aids with drawing basic elements. First, the application will iterate through all the points which are part of the path, and will draw a line between each two of them, in order. The following code illustrates the drawing function:

```
func addLine(fromPoint start: CGPoint, toPoint end: CGPoint, in view: UIView) {
   let line = CAShapeLayer()
    let linePath = UIBezierPath()
    linePath.move(to: start)
    linePath.addLine(to: end)
    line.path = linePath.cgPath
    line.strokeColor = UIColor.red.cgColor
    line.lineWidth = 1
    line.lineJoin = kCALineJoinRound
    view.layer.addSublayer(line)
}
```

DestinationsViewController.swift

This view controller has been implemented using a UITableView and a UISearchController. The UITableView shows all the rooms available on the current floor, which are fetched using the SearchHelper. The UISearchController is composed of a search bar which sits at the top of the table and is used to query the database for matching results, using again the SearchHelper.

Selecting one room will bring the user back to the MapViewController, from where a path will be drawn on the screen, as previously described.

ARNavigationViewController.swift

This view controller is the first one part of the AR mode, and aims to help users navigate, by showing an AR object made of an arrow, which points to the direction they need to follow. In order to know which direction the arrow needs to point to, the bearing angle between the current position and the destination position is calculated. This is done by using the latitude and longitude and values. Values from the mobile phone's compass are continuously received and used to rotate the arrow based on the user's orientation.

ARLocationInformationViewController.swift

The second part of the AR mode is made of this view controller, which shows timetable information and the availability of computers in the computer labs that are close to the user's current position. First, the mobile app requests the closest locations in each room, based on the current position, and then it requests information for each of them. Images are returned for the timetables, and a JSON with the availability of the computers is returned from PC-Free. This is done using the AR Data Provided. Finally, the AR elements are placed using the ARKit+CoreLocation library, using the latitude and longitude values received from the back-end server.

5.3 Testing

In order to deliver quality solutions and products, the software produced needs to be thoroughly tested. The system developed is composed of a multitude of components and subsystems. Each of them need to tested individually, and then as a whole, to ensure that they run correctly.

5.3.1 Unit Testing

Unit testing is usually used to guarantee that the methods part of a component are producing the intended results. These kinds of tests are important when used to test a system in an isolated way, especially when they interact with other systems, because they can detect where the problem comes from.

All the model classes that are part of the server that manages the database have benefited from unit testing. These tests have made sure that given an input, they create the right entity, with the right values. The same strategy has been applied to the Scan Switch server, which temporarily stores scanning data used for path finding.

The positioning system has initially been tested manually, but to ensure that it runs correctly, unit tests have been created in order to check that the user is positioned correctly. For the path finding algorithm, the HTTP client has been replaced with a mock one, that returns manually created data. This data is composed of real world location from across the globe, with set latitude and longitude values. The connections between these locations have been also created manually, in order to test if the route between two of them is chosen correctly. In the graph created, there are multiple ways to navigate from point A to point B, all with different

distances. In other words, the test had made sure that the route chosen is the one with the shortest distance.

5.3.2 Integration between mobile applications and servers

The mobile applications are very UI heavy, because the models are part of the servers. This is usually harder to test by unit testing. Therefore, testing these apps has been done manually, by making sure that the visual elements are placed in the right positions, and the data that they show is right. Moreover, to ensure the integration between the apps and the servers, which is done through the multitude of REST APIs, all the resources that can be accessed through the REST APIs have been thoroughly tested using Postman (an API development environment, which allows API testing).

Chapter 6

Evaluation

This chapter will analyse how accurate the IPS is and will critically analyse the delivered solution. The requirements from chapter 3 will provide the basis for the functional evaluation of the project. In addition, the project will be evaluated from a non-functional perspective. The section 6.5 will provide a comparison between the existing solutions and the solution developed by our project.

6.1 IPS Performance and Accuracy Analysis

The majority of the data was collected from the South Wing of Bush House in this project, as the North Wing houses the academic offices, with restricted access. The data collection started using only one computer lab, and progressively the data set was extended to the corridor, to surrounding computer labs, and across the other floors of the Informatics Department. It has been assumed that if the algorithm performs well on a small and restricted data set, such as one or two rooms, which are very close to each other, on a larger scale, the performance should stay the same, if not improve. Only one wing of Bush House has been mainly used for testing, because on the other wing, the academic offices are found, and the area is slightly restricted.

To evaluate the IPS, the data was collected using the admin application on an iPad. It is strongly recommended that data collection is done on a tablet due to the higher amount of precision when inputting locations onto the floor plan.

The positioning algorithm has been tested in different scenarios. First, given a data set, the goal of the tests were to position the user as accurately as possible in a room where the measurements are known. Measurements have been recorded having approximately 2m between themselves. However, certain positions in a room have been inaccessible due to the position of the chairs or tables. Taking this into account, the algorithm places the user in the location that matches the data set the best, therefore, the accuracy of the algorithm is ≈ 2 meters. The results given this configuration can be seen in figure 6.1.

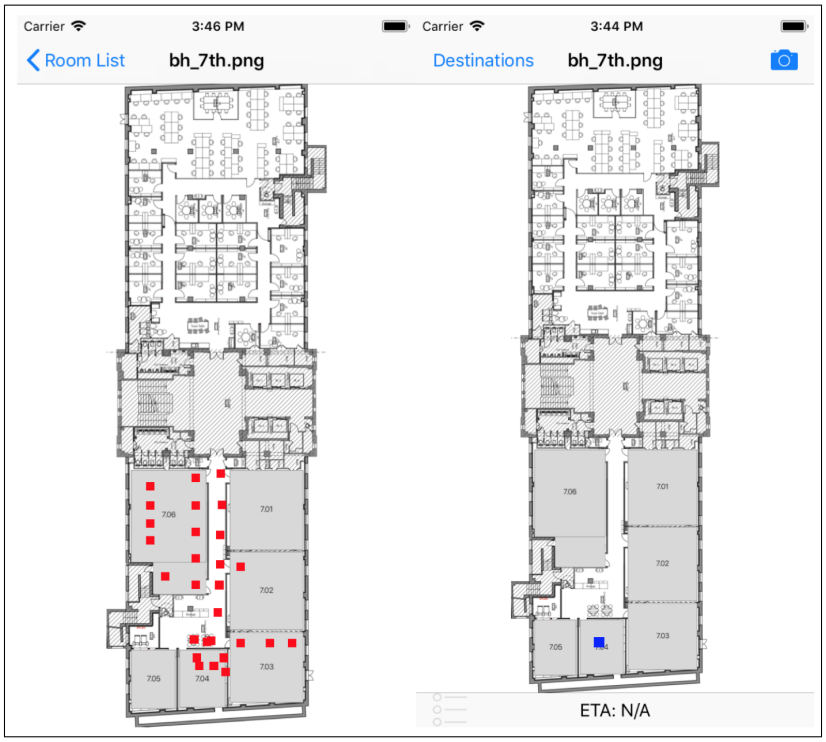

Figure 6.1: Positioning with known locations. The squares in red represent the recorded locations and the square in blue represents the position determined by the algorithm. The screen shot on the left is from the admin application, and the screen shot on the right is from the user application.

The second scenario the algorithm has been tested in was when data has not been measured for a room previously, to see how the algorithm behaves when there is no data available. In this case, based on the Wi-Fi networks that are around, the algorithm find the closest recorded position (see fig 6.2.

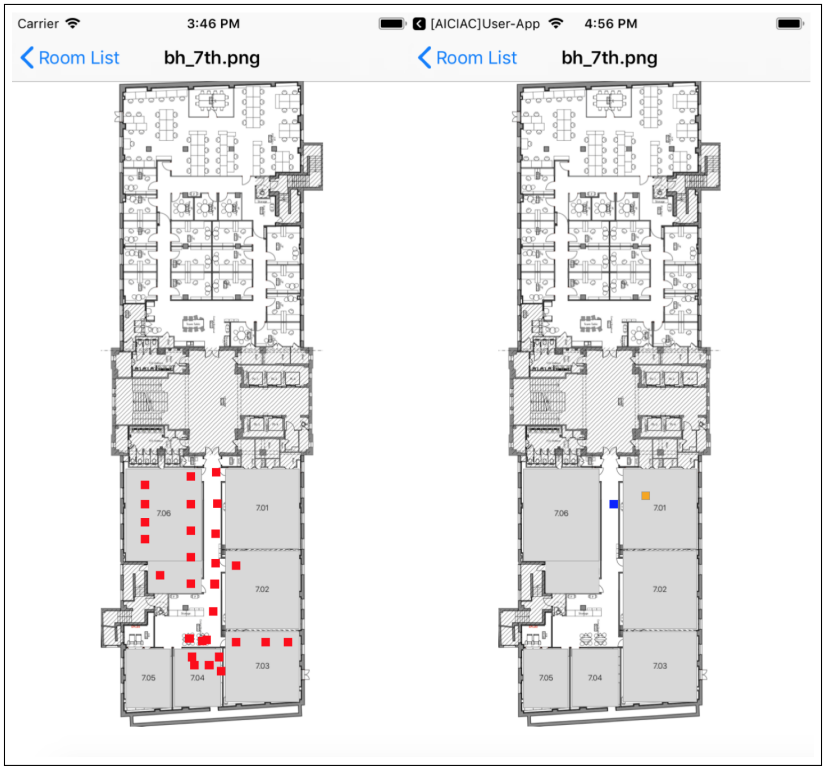

Figure 6.2: Positioning with unknown locations. The squares in red represent the recorded locations. The floor plan on the right shows where the algorithm has found the closest location (blue) compared to the current position (orange).

6.2 Path Finding Algorithm Analysis

The testing approach for the IPS algorithm was also adopted to test the path finding algorithm. The algorithm was tested first with locations in the same room, then between two adjacent rooms, and finally from wing to wing. To test the algorithm, convenient positions have been recorded in order for the path to be shown in a user friendly way on the floor plan (e.g. in front of doors; in this way the path will line up perfectly on the screen and they will not be placed on top of walls). One of the large scale tests was conducted from the end of the South Wing to Dr. Cole's office. Figure 6.3 shows an example where the algorithm has been successful in finding a path between the two locations.

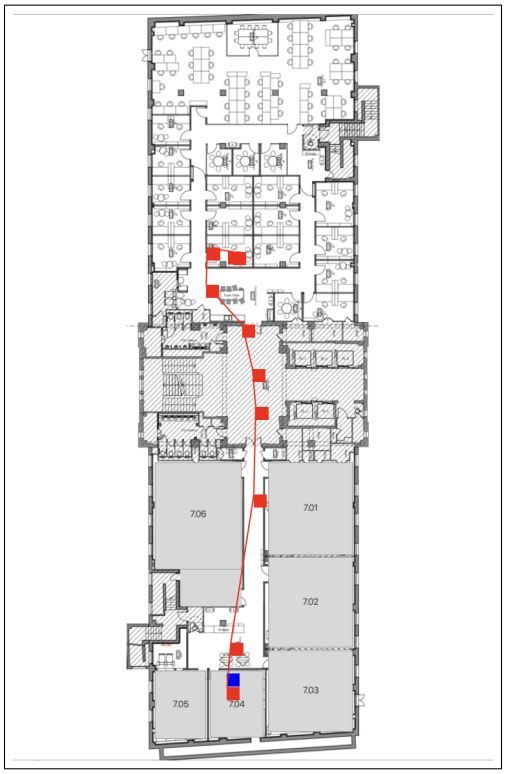

Figure 6.3: Long path between a computer lab in the South Wing and Dr Cole's office on the North Wing of Bush House. The square in blue represents the current position of the user and the squares in red represent the way-points to follow. The last red square represents the destination.

6.3 AR Features Results

Another fundamental aim of this project was to implement AR features. In combination with the path finding algorithm, the first AR feature facilitates the navigation process by showing the direction the user should follow, based on the smart-phone's built-in compass. The user mobile app shows the proposed visual cue, however during multiple tests, the bearing angle of the visual cue proved to be inaccurate. The magnetic field created by the multitude of radio frequencies from the Wi-Fi networks, together with the magnetic field produced by the Raspberry Pi, which needs to be in the proximity of the smart-phone are causing reflections which impact the bearing angle. However, moving around reduces the effect of the reflections, because this allows time for the compass to calibrate. Figure 6.4 shows a screen shot of the user app which includes the AR navigation guidance to direct the user to their final destination.

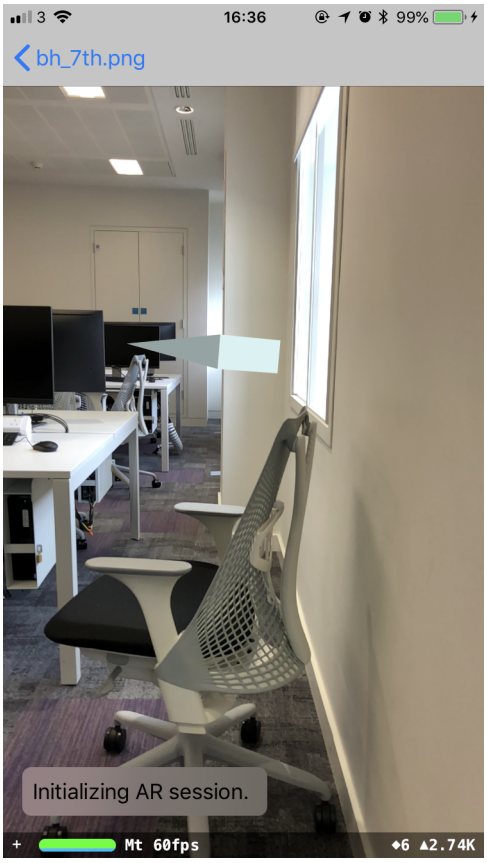

Figure 6.4: Screen shot taken from the user mobile app which shows the guidance arrow for navigation

Additionally, the app is able to pull information from the King's services in order to show timetable information and computer availability. However, at present, the computer availability feature is a proof of concept, showing the same data for each room. This is because PC-Free@King's has information for only one room in Bush House, 4.02.

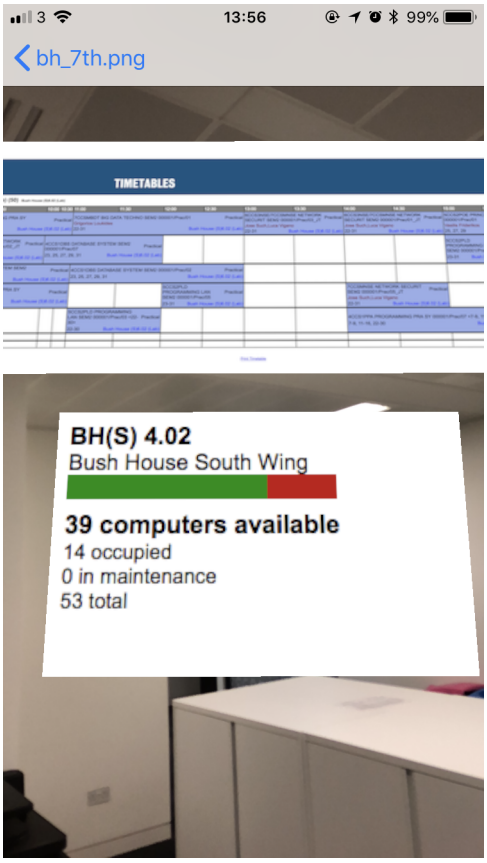

Figure 6.5: Screen shot taken from the user mobile app which shows the information displayed.

6.4 Project Evaluation

6.4.1 Functional Evaluation

The project is able to scan and save Wi-Fi measurements, use them to position a user, and then provide navigation instructions for users to find their way around Bush House. All the requirements set out in chapter 4 were completed and fully tested. An empirical evaluation of the project that looks at how accurate the system is has been provided in sections 6.1, 6.2, and 6.3. In order to evaluate the key requirements necessary to meet the project aims, a non-functional evaluation, focused more on quality attributes, will be further detailed in the next section.

6.4.2 Non-functional Evaluation

- Requirement: *Interoperability – The subsystems should be able to fully integrate with other subsystems in order to provide maximum flexibility.* (section 3.4

  As mentioned in the design and in the implementation chapters, all the servers and data providers have an API to communicate with other components of the system. The data management server has a RESTful API to access the database, the Positioning & Path Finder system has an API in order to calculate data and provide results, and the AR Data Provider has an API that provides required data to show in the user application. All the communications that take place using the APIs use a JSON format, which is a very popular serialisation format. In this way interoperability has been achieved, by integrating all the systems through the REST APIs which communicate using the JSON format.

- Requirement: *Maintainability – The system should provide a simple and easy way to extend feature base. For this, the follow concepts must be followed*:

  - Sub-requirement: *Separation of Systems – The system must be decomposed in multiple independent subsystems which must be as decoupled as possible. The systems should use the appropriate programming languages and framework to achieve their requirements and tasks.*

    The architecture components have been separated first into layers that achieve common goals, and then into separate systems, depending on the task achieved. The user mobile application and admin mobile application have been separated due to the nature of the features, and then based on the data that each handles, the other systems have been decoupled. This is detailed extensively in the design chapter. Because systems are very separated, any of them can be easily replaced without affecting the system as a whole. For example, mobile applications could be developed for other operating systems, such as Android.

  - Sub-requirement: *Architectural Patterns – Each subsystem should follow the appropriate architectural pattern and design patterns provided by the framework that is being used, or the programming language that the system is written in.*

    The best practices and design patterns have been used when appropriate, in each application. Most of the system makes use of an MVC design pattern, mainly due to the frameworks that they use, and the way those have been designed. Evidence of the MVC pattern is presented in the design chapter, along with the code listed in the implementation chapter and the appendix.

  Overall, the separation of systems and the architectural patterns have been considered so that the system is maintainable.

- Requirement: *Usability – The system should provide an easy to use interface, and as clear as possible. Although it can be simple, the interfaces must provide clarity of interactions, and a satisfactory user experience.*

The GUI of the mobile applications has been made clear and simple, by incorporating system visual elements. These elements are used across iOS, therefore, the user is familiar with the way they should interact. For long waiting times in the apps, spinners have been used to keep let the user know that data is loaded, and the application is still running. The floor plan images have been fitted to all screen sizes, and a search bar was used to make it easier to look for rooms.

6.5 Comparison to other existing solutions

When comparing the navigation system built in this project to other existing solutions, the most suitable comparison would be with "Indoor Survey App" [22]. The app built by Apple is the most relevant, as the data is measured using the same technique, but this project's solution provides an arguably better and more extensible system. Below, a full comparison is detailed.

The most important comparison that can be made with Apple's solution is the architecture of the system. A very important aspect to remember is that Indoor Survey App provides positioning and measuring for any building that follows their specific requirements, whereas the app in this project only supports Bush House. However, this is a feature that can be easily added on top of the existing capabilities, since the system has been developed with extensibility in mind.

The way that both of the apps measure relevant positions in the buildings is by using the floor plans. After the user registers their floor plans and has them available in the app, they can record important reference points in the building by tapping on the screen. This will register the position on the image and assign Wi-Fi measurements to it. Even though the ways of working are more or less the same, the system created in this project can support any measuring platform, for an example Android device, whereas Apple's will only run on iOS. However, performance-wise, Apple's application is faster when dealing with scanning for Wi-Fi networks, because all the measurements take place on the phone, and not through a multitude of systems. Nevertheless, if the scanning module that is available on the mobile device would be made open by Apple, this can be easily adopted by our application without any difficulties. This will be detailed in the limitations and future work chapter.

Furthermore, Apple's application is limited to buildings and venues that "attract more than one million visitors annually" [22]. Although the system only supports Bush House, as future work, it can be easily extended to any building that has its floor plans available, with no limit on the number of visitors. It is not clear right now, due to user restriction, if Apple's application supports anything more than just measuring buildings and then positioning users, but our system comes at an advantage because this project's solution uses navigation and also makes use of AR features by using ARKit to improve the user's experience.

Another solution that has a fingerprinting based approach is SLAM [19]. SLAM tries to improve this approach by eliminating the offline measuring process, which brings a lot of limitations by the fact that the area must be surveyed before using the positioning system. This solution brings tracking on unknown grounds and does not have to record data in advance, such as the floor plan; nonetheless, if available, a data set can be connected and used. However, this type of approach comes with a drawback that is very important if implemented on mobile

devices: a very high computational load. This makes it unfeasible to use on mobile devices, because of the limited resources available and because the energy use would be significant. In this case, even though our approach requires the admin user to make prior measurements of the floor plans, the energy use is very low, because all the calculations are being made on an external server.

Chapter 7

Legal, Social, Ethical and Professional Issues

7.1 British Computing Society Code of Conduct

The British Computing Society (BCS) has published a Code of Conduct[1], in order to set the required standards that an individual has to follow, in order to be a member. Throughout developing the project, these standards have been closely followed, and the project is fully compliant with them. There are four standards that the project follows:

- **Public Interest**: The system and the applications built are available for everyone to use, regardless of sex, sexual orientation, marital status, nationality, colour, race, ethnic origin, religion, age or disability, or of any other condition or requirement. Moreover, the intellectual property of all the third parties are fully respected; their work is referenced to in this paper and the code components that belong to them have a clear specification of whom they belong to.
- **Professional Competence and Integrity**: The project has allowed the developer involved to develop new skills and learn new concepts that have been applied here. Learning has been done on a continuous basis throughout the project. The project does no harm to anyone, and the project has not taken part into bribery or unethical inducements.
- **Relevant Authority**: The project has been done by respecting the regulations of King's College London.
- **Duty to the Profession**: The project has tried to improve the professional standards and make a valuable addition to the field it belongs to.

7.2 Code of Good Practice

BCS has also published a Code of Good Practice[2]. However, this code includes many guidelines, compared to the Code of Conduct. Although it is long, many of them are relevant

[1]The code is available online at `http://www.bcs.org/category/6030`
[2]The code is available online at `http://www.bcs.org/upload/pdf/cop.pdf`

to the project, and have been followed through the project. The following points are the most relevant, and have been taken from section 4.2, which relates to Research, and section 5.2 which relates to Software Development:

- From section 4.2: **Investigate the analysis and research by other people and organisations into related topics and acknowledge their contribution to your research**: The project uses methods already detailed and used by other people in research papers. Before coming up with a solutions, these have been analysed in detail in order to explore ideas.
- From section 5.2: **Strive to achieve well-engineered products that demonstrate fitness for purpose, reliability, efficiency, security, safety, maintainability and cost effectiveness**: As previously stated in the Requirements chapter 3 and in the Evaluation chapter 6, the project has set many non-functional requirements, which have been met, which are present in the Code of Good Practice.
- From section 5.2: **Encourage re-usability; consider the broader applications of your designs and, likewise, before designing from new seek out any existing designs that could be re-used**: Re-usability is practised widely along the project, and this can be observed, for example, when the classes used in the database server are used throughout the logic layer components and the mobile applications.
- From section 5.2: **Produce code that other programmers will find easy to maintain; use meaningful naming conventions and avoid overly complex programming techniques, where these are not strictly necessary**: The features have been implemented using clean code, with variable names and function names that portray what data they hold, and respectively what actions they are performing.

7.3 User Sensitive Data

The data collected by the admin application is split into user input data, such as the name of the rooms, and the collection of MAC addresses, network names and signal strengths of the Wi-Fi access points. This data is used for positioning the user. The calculations in order to determine this position are made on the server, and then the position is sent to the application. None of the servers or the applications store this data at any point. Moreover, the path finding process work in the exact same way. In this way, all the data is therefore anonymous and not linked to any user or individual.

7.4 Security of the Data Collected

The system lacks security because no user based system, or access control system has been developed for the database. In other words, all data stored by the database is accessible for everyone as long as one knows how to access it. The network requests that ask for data can be sniffed on a network and in this way, the resources that provide the data can be detected. However, all the content of the network request is being transmitted under HTTPS, therefore it is encrypted and inaccessible for an attacker.

7.5 Wi-Fi Data

The system scans for all the Wi-Fi networks that it can find, and cannot identify whether the network belongs to King's College London or not. Some entities might have regulations that forbid storing data about their Wi-Fi networks, so in order to abide by that, the future implementation of this software should only use the university's access points.

Chapter 8

Limitations & Future Work

Although the system has met the set requirements, there are some weaknesses and areas of improvement before the project can be considered to be ”final”. This chapter will further detail all the limitations and the ways in which the project can be extended and improved in the future.

- **Scanning cannot be done on the iOS mobile device.**
  In order to use Wi-Fi scanning capabilities on iOS and receive the signal strengths used in positioning, one should use a native framework, called NEHotspotHelper. However, Apple has restricted access to this framework to only a few apps that are part of the use cases that they ”support”. To get access, users can apply through a questionnaire, which must pass Apple's checks. For this project, several applications for this framework have been made, but the application does not fall into their supported categories. Because of this, scanning had to be done on an external system, in this case a Raspberry Pi that uploads data directly in the database. This has brought overheads on the system because the whole process is slowed down by the communications between all the subsystems. This will be detailed in the next point.

- **Connectivity between the scanning system and other sub-systems.**
  The scanning process is very slow, because the mobile applications have to wait for the ”turn on” request to be made on the Scanner Switch, then for the Raspberry Pi to detect the changes, and then for the Raspberry Pi to scan, parse & upload the data. The whole process takes about 10 to 12 seconds, measured from when the first request has been made, to when the results have been received. The overheads are brought by the HTTP requests that have to be made, and scanning and parsing processes which take place on the Raspberry Pi. Ideally, this process could be accelerated if the Scanner Switch would be eliminated, and the connectivity would be made through Bluetooth between the Raspberry Pi and the mobile devices that run the mobile applications. The reason this has approach has not been chosen in the end was affected by the time constraints. Several prototypes that have been unstable have been made using Bluetooth, and it the end the intermediary web application has been chosen because it was the simplest and most stable way at that time.

- **Vapor framework is a very early stage framework.**

This project has taken an experimental approach, given the recent rise of Swift running on servers, unlike the traditional way of running Swift, which is on iOS devices. One of the most popular framework that utilises this is Vapor. Although the community is very helpful, and the performance of this framework is quite good compared to existing frameworks, such as Ruby on Rails, it can be said that some functionalities have not been added yet. Therefore, the framework lacks support for certain features that others have. This can be seen when, for example, making HTTP requests, which runs synchronously, bringing overhead on the server. A future solution would have to migrate from using Vapor to using something that is made for production. Another important limitation has been the lack of a way to automate actions in a web browser. Many other technologies support certain libraries that to do this, such as Selenium for Python. Due to this limitation, the Logic Layer has had to be split into two components: the AR Data Provider and the Positioning & Path Finding System.

- **ARKit+CoreLocation framework is slightly unstable.**
  Due to the implementation of ARKit, the approach used by the framework to place AR objects has a few known issues. When the user is walking, ARKit may sometimes change the positions of the placed objects, because the framework might receive some erroneous data, which then outputs wrong values. After a short distance, for example, the framework might ”think” that the user is walking in a different direction that they actually are. Additionally, the location algorithm provides some wrong data as well, which affects the positions of the AR objects as well.

- **Optimising fingerprinting localisation.**
  Right now, the positioning is done only in a deterministic way, based on the available data. This does not always work, since some areas may not have data collected. To overcome this issue, a probabilistic mathematical model could be incorporated in order to determine where the user may be. Typically, a Gaussian distribution is used for this, together with data from recorded reference points.

- **Graphical User Interface.**
  The graphical user interface (GUI) has been designed so that it would be as simple, clear and usable as possible. However, a more attractive GUI should be developed in order to attract more users. These improvements could change the way that the navigation path is shown on the floor plan, or the inclusion of a set of turn-by-turn instructions. Other improvements could be made in the admin application, where more visual cues would greatly improve the experience of the admin user. Overall, the GUI improvements should make users more eager to use the system, and overall improve the quality of the software developed.

- **Extending the solution to support more than one building.**
  The solution developed had in mind to only support Bush House. However, the positioning and navigation algorithms have been developed independent from location. Therefore, any building could be supported, as long as the system is slightly changed. The changes that need to be made are: create a Building entity and the possibility to upload floor plans for it, assigns Rooms to Buildings, and then a way to change the values of the

corners of the floor plans of a building in latitude and longitude (this can be added to the Building entity).

- **Navigation from floor to floor.**
  The navigation algorithm is capable of producing a path as long there are connections between the locations. Therefore, if the locations between floors are connected, a path will be produced between them. However, the current design of the mobile apps makes it makes it impossible to connect two locations that are on different floors, because one floor plan is shown at a time. On the other hand, the app continuously scans for location changes, and the possible destinations that are shown are from the same floor. Thus, if the user changes floors, the corresponding floor plan will be shown, along with the reachable destinations on the same floor. To support navigation between floors in the future, the visual design needs to be changed. One possible solution would incorporate connecting two floors by choosing a room that is the connecting point, e.g. the lift or the stairs.

Chapter 9

Conclusion

The project proves the reliability and feasibility of developing an IPS, given that the floor plans are available. It shows that it is possible to develop a very precise and robust positioning and navigation system to determine indoor locations of Wi-Fi enabled devices, by only using the existing infrastructure available. Therefore, no additional cost is incurred, and no dedicated positioning devices are required, such as GPS chips.

The project aimed to develop an IPS that is different way to previous solutions. Firstly, the IPS was built using a fingerprinting method, combined with existing floor plans. Secondly, the IPS was developed on iOS, which is rare due to the number of limitations imposed by Apple. Although not perfect, the project has been successful in terms of achieving the aims that have been set at the beginning of project.

There is a lot of new research in the field of indoor positioning, which means that in the future many accurate IPSes can be produced. In particular, in this project, an indoor navigation system with built-in AR features has been studied and developed. Therefore, as more research is conducted in this area, improvements can be made to the app, which will hopefully improve accuracy, making it suitable for use by a wider audience and in multiple use cases.

Abbreviations

App	Application
AR	Augmented Reality
BCS	British Computing Society
GLONASS	Global Navigation Satellite System
GPS	Global Positioning System
GUI	Graphical User Interface
IPS	Indoor Positioning System
MVC	Model-View-Controller
RSSI	Received Signal Strength Indication
SLAM	Simultaneous Localisation and Mapping
UI	User Interface

References

[1] What is GPS?,
http://www.loc.gov/rr/scitech/mysteries/global.html

[2] GPS vs GLONASS: Which is Best for Tracking Applications?,
https://www.semiconductorstore.com/blog/2015/GPS-vs-GLONASS
-Which-is-Best-for-Tracking-Applications/810

[3] Benefits of Combined GPS/GLONASS with Low-Cost MEMS IMUs for Vehicular Urban Navigation,
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3355462/

[4] GLONASS significantly benefits GPS,
http://www.oxts.com/glonass-significantly-benefits-gps/

[5] Russian System of Differentional Correction and Monitoring
http://www.sdcm.ru/index_eng.html

[6] GPS: The Global Positioning System
https://gps.gov

[7] Z. Horvath, H. Horvath (2014): *The Measurement Preciseness of the GPS Built in Smartphones and Tablets,* International Journal on Electronics and Communication Technology, issue 1, pp 17-19

[8] Curran, Kevin; Furey, Eoghan; Lunney, Tom; Santos, Jose; Woods, Derek; McCaughey, Aiden (2011). *An Evaluation of Indoor Location Determination Technologies.* Journal of Location Based Services. 5 (2): 61–78.

[9] Triliteration
https://www.britannica.com/science/trilateration

[10] Schuettel, Patrick (2017). *The Concise Fintech Compendium.* Fribourg: School of Management Fribourg/Switzerland.

[11] Rosenberg, L.B. (1992). *The Use of Virtual Fixtures As Perceptual Overlays to Enhance Operator Performance in Remote Environments.* Technical Report AL-TR-0089, USAF Armstrong Laboratory, Wright-Patterson AFB OH, 1992.

[12] Couts, Andrew. *New augmented reality system shows 3D GPS navigation through your windshield.* Digital Trens, 27 October 2011

[13] Griggs, Brandon. *Augmented-reality' windshields and the future of driving CNN Tech.* 13 January 2012.

[14] Delgado, F., Abernathy, M., White J., and Lowrey, B. *Real-Time 3-D Flight Guidance with Terrain for the X-38,SPIE Enhanced and Synthetic Vision.* 1999, Orlando Florida, April 1999, Proceedings of the SPIE Vol. 3691, pages 149–156

[15] Positioning and triliteration,
`https://www.alanzucconi.com/2017/03/13/positioning-and-trilateration/`

[16] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms, Third Edition (3rd ed.).* The MIT Press.

[17] Three Tier Architecture
`https://www.techopedia.com/definition/24649/three-tier-architecture`

[18] Shin, H., Chon, Y., Cha, H. (2012). *Unsupervised construction of an indoor floor plan using a smartphone.* IEEE Transactions on Systems, Man and Cybernetics Part C: Applications and Reviews, 42(6), 889–898.
`https://doi.org/10.1109/TSMCC.2011.2169403`

[19] SLAM,
`https://en.wikipedia.org/wiki/Simultaneous_localization_and_mapping`

[20] Mountney, P.; et al. (Stoyanov, D.; Davison, A.; Yang, G-Z.) (2006). *"Simultaneous Stereoscope Localization and Soft-Tissue Mapping for Minimal Invasive Surgery".* MICCAI. Lecture Notes in Computer Science. 1: 347–354.

[21] Koyuncu, H., & Yang, S. H. (2010). *A Survey of Indoor Positioning and Object Locating Systems.* International Journal of Computer Science and Network Security (IJCSNS '10), 10(5), 121–128.

[22] Apple's survey app helps venues easily create indoor maps,
`https://www.theverge.com/2015/11/2/9657304/apple-indoor-mapping-survey-app`

[23] iOS,
`https://en.wikipedia.org/wiki/IOS`

[24] Swift,
`https://swift.org/about/`

[25] Vapor,
`https://vapor.codes`

[26] Python,
`https://www.python.org/about/`

[27] Flask,
`http://flask.pocoo.org`

[28] Linux,
`https://en.wikipedia.org/wiki/Linux`

[29] ARKit,
https://developer.apple.com/arkit/

[30] PostgreSQL,
https://www.postgresql.org/about/

[31] King, T., Kopf, S., Haenselmann, T., Lubberger, C., & Effelsberg, W. (2006). *Compass.* Proceedings of the 1st International Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization - WiNTECH '06, (January), 34.
https://doi.org/10.1145/1160987.1160995

[32] Evennou, F., Marx, F. (2006). *Advanced integration of WiFi and inertial navigation systems for indoor mobile positioning.* Eurasip Journal on Applied Signal Processing, 2006, 1–11.
https://doi.org/10.1155/ASP/2006/86706