B.Sc.Eng. Thesis
Bachelor of Science in Engineering

DTU Compute
Department of Applied Mathematics and Computer Science

A Platform for Optimizing Indoor Climate driven by Location Based User Feedback

Christian Hjelmslund (s164412)
Sebastian Arcos Specht (s164394)

Supervisors: Ekkart Kindler & Davide Cali

Kongens Lyngby 2020

DTU Compute
Department of Applied Mathematics and Computer Science
Technical University of Denmark

Matematiktorvet
Building 324
2800 Kongens Lyngby, Denmark
Phone +45 4525 3031
compute@compute.dtu.dk
www.compute.dtu.dk

Abstract

The quest for an optimal indoor climate has led to automatic systems that adjusts the indoor climate based on scientific factors. The systems interpret measurable values such as temperature, air-humidity, CO2 level and takes action accordingly. However, it is hard to create a system that fits everyone — different people have different needs and thus their perception of the indoor climate cannot be simplified as such. This leads to the creation of Feedme, a system with the vision of giving the users more control and influence on how the indoor climate is regulated in the room they are located in. The idea is to automatically determine the users' location, by installing Bluetooth Low Energy beacons in buildings and ask them questions about their experience of the conditions of the room they are located in. The system can take action accordingly based on the users' feedback, thus shifting decision-taking towards their needs and comfort rather than plain scientific factors.

The system consists of three main components. A web server written in NodeJS running on a virtual machine communicating with a MongoDB database. An iOS application with the purpose of listening to BLE beacons and giving the opportunity for users to provide feedback accordingly. A ReactJS website that gives building administrators a way to manage the buildings that need to be regulated.

To determine users' location, a modification of the K-Nearest Neighbors algorithm has been implemented, where the BLE beacons' signal strength are used. The success rate of the algorithm varies a lot depending on the setup, with the lowest success rate being picking random rooms and the highest success rate of 100%. The time it takes to determine users' location is roughly two seconds. The whole system has been developed with an agile approach, so the system can be further developed and the experiments are documented, such that they can be reproduced. The actual implementation allows it to be used as a general feedback application thus expanding its use case and potential significantly.

Preface

This Bachelor of Science thesis has been developed in cooperation with the Department of Applied Mathematics and Computer Science at the Technical University of Denmark (DTU). The project's time span was from the 4th of February 2019 to the 19th of May 2019 and is created in the fulfillment of acquiring a Bachelor of Science in Software Technology. The workload has been 15 ECTS points and the supervisors of the project have been Ekkart Kindler and Davide Cali from the Department of Applied Mathematics and Computer Science at DTU.

Kongens Lyngby, March 28, 2020

Sebastian Arcos Specht (s164394)

Kongens Lyngby, March 28, 2020

Christian Hjelmslund (s164412)

Acknowledgements

First and foremost, we would like to acknowledge Ekkart Kindler for his help and advice that he has provided us from day one of the project. He has been very dedicated throughout the project, by attending weekly meetings discussing the direction in which the project was heading and challenging our implementations with critical observations. Secondly, we would like to thank Davide Cali, for giving us the opportunity to be a part of this project and for always being optimistic and enthusiastic during the entire project.

Contents

CHAPTER 1

Introduction

The indoor climate in buildings is an important factor for productivity and efficiency in working environments such as schools, universities, office buildings etc. Skoleklima is a project for optimizing the indoor climate both to give the user the best possible conditions to work in, but also to minimize the cost of maintaining these conditions. In a lot of working environments, people try to control the indoor climate by opening and closing windows or by regulating the thermostats in the room they are located in. These actions do not often provide the optimal climate for everyone in the room and they are certainly not the most efficient economy-wise. People might not have the technical knowledge about what exactly makes for an optimal indoor climate. For this reason, in the Skoleklima project, there has been made some automatic systems to adjust the indoor climate by scientific factors to optimize the indoor climate. However, by shifting to an automatic regulation system, the individual user of the room suddenly does not have any way to influence how the indoor climate is regulated. In the Skoleklima project they wish to include the individual user more in the decision of how to regulate the indoor climate. The purpose of this project is to provide a system, called Feedme, where users can easily give feedback about the conditions of the room they are located in and a provide a way for the administrators of the buildings to read, learn and potentially react based on this feedback.

1.1 Problem Description

As mentioned, the implementation of automated systems introduces a problem which is that the indoor climate is no longer directly dependent on the comfort level of the actual users of the room. The automated system interprets values of the temperature, air-humidity and CO2 level and adjusts accordingly. However, it is difficult to create a generalized model that fits everyone. The actual indoor climate is of course very dependent on the physical environment, whether it is the thickness of the walls or the amount of air flow in the room, but the experience of the indoor climate is very individual and personal as people are different. The optimal indoor climate for seven year old kids in a class room might not be the same as for a twenty-two year-old employee in an office. For that very reason, this project is about giving the users more control and influence over the indoor climate in the room they are located in.

1.2 Project Vision

The overall vision for the project is to create a system where users can easily give feedback about their experience in a particular room. A clear vision is, that the user's location should be estimated correctly and quickly to facilitate the feedback process for the user. When the user open up their smartphone, the user should only be very few steps away from giving feedback. It is therefore also a vision for the project to provide an easily usable user interface for the mobile application. This means, that building administrators should not know too many technical details about the system to set it up. The process of setting up the system to be able to correctly estimate in which room the users are located in, should not be a time consuming process. Ideally, building administrators should not use more than an hour to set up the entire system in a larger building. To manage the building, the building administrator should have access to a website where all the management tasks can be carried out, and where an overview

of the state of the system can be seen. The website should include more technical details about the system, while still be kept simple.

1.3 Design Science

For this project, one of the goals is to provide the user with an easily usable system for giving feedback about the user's experience of the room. Providing this feedback should be easy and fast which is why the system should automatically determine in which room the user is located. Automatically estimating the user's indoor location is a feature that currently seem very unique and unknown to the general public. For this project, an investigation of what currently exists of such technologies is carried out. This gives a better understanding as to what is currently missing in the domain and to what could be improved. Deciding which features should be available to the system is done only after investigating the current state-of-the-art. At the end of the project, the results of the system is properly tested and evaluated. The evaluation of the system includes reviewing how the system performs in comparison to similar systems – it is evaluated both how the system potentially is better at certain areas than the current state-of-art, but also when and how the system might perform worse than similar systems.

As will be discussed in greater detail in Chapter 3, Section 4 (Development Process), the development of the system follows an iterative approach, where at the end of each iteration the system is in a state where it could potentially be released. However, in each of those iterations, the project still includes different phases to ensure the quality and relevance of the product. These phases are general for design science projects[10] and include a requirement phase, where the requirements for certain parts of the system are defined, a design phase where the new part of the system is properly investigated and designed, a construction phase where the new part of the system is actually developed, a testing phase where the system is properly tested and an installation and a maintenance phase, where the system is put into a state that is a close to releasable as possible.

1.4 Report Overview

The report contains 12 Chapters and three appendices. In this Chapter (Introduction), the overall project is introduced along with the description of the problem and the vision for the project. In the next Chapter (Technologies), a brief introduction to some of the technologies relevant for this project is presented. This involves mainly technologies used for location estimation. In Chapter 3 (Related Work), articles and papers with similar purposes as this project are introduced and discussed to give a clear picture of the current state-of-the-art. In Chapter 4 (Analysis), the concepts and overall domain of the project is presented and analyzed. In that chapter, the user stories of the project are also presented concluding with how the project is carried out in terms of project planning and risk assessment. In Chapter 5 (Software Handbook), a guide on how to use the system is presented, which also explains some of the functionality of the system. In Chapter 6 (Location Algorithm), different potential location estimation algorithms are discussed along with a more in depth look at the algorithm we chose and how it is implemented. Chapter 7 (Design), explains the overall architecture of the system including the architecture of each of the 3 main components; the mobile application, the website and the back end server. That chapter also explains the architecture of each component. In Chapter 8 (Implementation), the implementation of the system is presented in more detail. This includes presenting some of the libraries and special functionality that facilitated the development of the system. Chapter 9 (Testing) introduces the overall ideas behind performing automated and manual tests to the system and it analyzes and discusses how each component was tested. In Chapter 10 (Evaluation), an evaluation of of the location algorithm is carried out, by looking at a concrete test example of how the algorithm performs. The results of the evaluation is then compared to similar projects at the end of the chapter. In Chapter 11 (Future Work), it is discussed what steps to take moving forward with the project. Finally, in Chapter 12 (Conclusion), the project is concluded by giving an overview of the results achieved and

how they were achieved. In Appendix A (Glossary) an explanation to the different terms used for describing the system is given. In Appendix B (Detailed Use Cases) all of the detailed use cases, which have been used to implement the system are listed. In Appendix C (Back End Server Testing), the individual unit tests of the back end server are listed.

CHAPTER 2

Existing Technologies

Indoor localization is a field which is in rapid expansion and there are various tools and techniques to determine a person's indoor location. The common denominator is, that there are multiple devices connected to a network communicating with each other and through this communication, localization can be established. Essentially these devices form a Wireless Local Area Network (WLAN) and the messages exchanged with each other are through radio waves. This system of multiple components communicating with each other, is a part of the more known concept Internet of Things (IoT). How the communication takes places between the devices, depends on what technology is used. Common technologies used to create networks for indoor localization are Bluetooth Low Energy (BLE) and Wi-Fi. The Global Navigation Satellite System (GNSS) is an alternative and less precise approach (for indoor localization) which could be used to solve the same issue of indoor localization, but has some significant flaws. First of all, the GNSS signals are almost entirely blocked inside buildings. Secondly, it cannot be used to determine positions on the z-axis, which means that it cannot detect which floor a user is on. By far the most known GNSS system is The Global Positioning System (GPS).

In this section, different technologies that can achieve the desired goal of indoor localization is presented and compared to each other. Existing solutions are shown alongside innovative use cases and potential scalability of such systems. The goal is to give a brief overview of the main differences of the technologies and explain why BLE beacons were chosen for this project. The section will not go too much in detail with the individual technologies.

2.1 Technologies

The three main technologies that this section will introduce are BLE, Wi-Fi and GPS. The presented technologies main utilities are not necessarily within the domain of indoor localization, but in this context they are presented with the focus on indoor localization.

2.1.1 Beacon

Before diving into the concrete differences between BLE and standard Bluetooth, the concept of a beacon has to be introduced.

A beacon is a small radio transmitter, which sends out data at fixed time intervals. In this case, a BLE beacon transmits signals conforming to the BLE protocol. These signals can be detected by devices which can conform to this protocol - for example a smartphone. The signals can also contain data, but in the case of indoor localization, it is actually enough if there is only an unique identifier (UUID in the domain of Bluetooth) distinguishing each deployed beacon. The simplest way of implementing indoor localization is shown in Figure 2.1. There are three rooms, each containing a beacon. The smartphone will receive the BLE signals and look at the signal strength (RSSI value on the figure, the higher the number, the stronger the signal). Each beacon has a UUID and is associated with a room, so the smartphone will determine in which room it (the user/smartphone) is located, based on the strongest signal - since beacon2 has the strongest signal, it determines its location to be room2. Other ways of determining a user's position is presented later in the report.

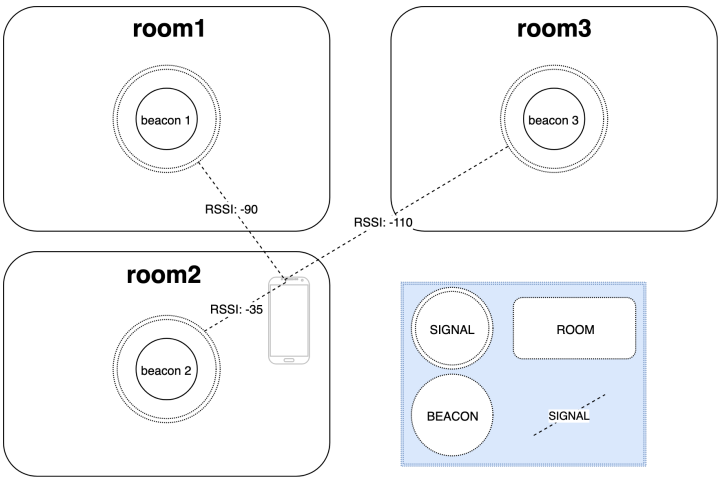


Figure 2.1: Simplest way of implementing indoor localization based on BLE beacons

2.1.2 Bluetooth Low Energy

It is important to make a distinction between Bluetooth and Bluetooth Low Energy. They are both short-range wireless technologies managed by the non-profit organization Bluetooth Special Interests Group (`SIG`). To be able to use this technology officially, one has to register a company and get approval of the Bluetooth SIG organization to use their technology[1]. It is free to register and there are certain benefits to joining, such as specific documentation and exposure of the application which is registered.

Both Bluetooth and BLE operate on the 2.4 GHz operation band `ISM` and are allocated 40 Radio Frequency channels. Data packets are exchanged through these channels. They do however have different protocols, which means that if a device only implements BLE it will not be able to communicate with a device which only implements classic Bluetooth and vice versa [2].

BLE has some very specific features which makes it suitable for this project and `IoT` in general [16]. The first and most important feature is its low power consumption. The low power consumption means, that you can have a network of wireless devices without having to think too much about changing the battery. Since they are battery based, they can easily be changed to other locations, without having to worry about power outlets. This comes with a cost, which is in the form of latency and especially the throughput of data. This is also why classic Bluetooth is still relevant — consumer goods such as wireless headphones, keyboards etc. need more throughput than what BLE beacons are able to provide. BLE's strongest traits are in areas such as precise indoor localization. The beacons chosen for this project have a battery life of up to two years. They also have a fairly long range of up to 70 meters, which means that fewer beacons are required to cover a larger area and thereby reducing the cost of deploying such systems[2].

When communicating through networks and handling user data, one should also take security into consideration. The impact of potential threats varies much depending on the use case of the systems. The main threat in most systems involving wireless technologies such as BLE, Wi-Fi etc. are when malicious actors wants to know the content of the exchanged packets. BLE has various security mechanics. The packets exchanged between two devices are always encrypted using a combination of AES-CCM cryptography. AES is a symmetric key encryption system, meaning it consists of one key shared between the two devices. AES encryption is very secure, so the main security issue in BLE exists before the actual data is being transmitted. There has been found ways of attacking the key-exchange of two devices, leading to passive eavesdropping[17, 3]. Passive eavesdropping is the concept of a malicious

[1] Registration can be found at `https://www.bluetooth.com/develop-with-bluetooth/join`

[2] The beacons used for this project can be bought at the site: `https://ibeacon.dk/produkt/smart-beacon`

actor getting access to the content of packages that are not intended to be read by the actor.

2.1.3 Wi-Fi

Wi-Fi, as opposed to Bluetooth, can be connected to the Internet. It is also a short-range wireless technology. It is managed by the non-profit organization Wi-Fi Alliance [3]. Just as Bluetooth, Wi-Fi forms a WLAN by operating on the ISM band and the exchanging of packets happen through the allocated channels.

For a Wi-Fi to operate, it has two main components; an access point `(AP)` and some clients connecting to the access point. The way it works is that this access point broadcasts messages containing a service set identifier `(SSID)`, saying that there is a nearby Wi-Fi network which can be connected to. The SSID is just a string to distinct the different Wi-Fi networks that a client can connect to. The clients are also called end-user-devices, which can be laptops, phones etc.[5]. In the context of indoor localization, it means that the clients would be phones. The packages, broadcast by the access point, would serve the same purpose as the packages sent by the BLE beacons – they would be fetched by the phone and can be used to determine the location of the user based on the signal strength. Fig. 2.1 could be modified a bit, such that the beacons in the figure would be replaced by access points, and the localization technique is essentially the same. There is a `SSID` instead of `UUID` and the signal strength is the factor that determines in which room the user is located.

When talking about Wi-Fi today, there are different levels of security. Even though Wi-Fi networks using WEP as a security measure are still used, most companies today use WPA2, which is why WPA2 is the baseline of this paragraph. WPA2, just as BLE, uses AES for its encryption. When the client wants to connect to the access point a 4-ways handshake is instantiated and when authenticated, the actual ciphers and the key can be exchanged. With the increase of computing power and software dedicated to crack WPA2 there are several threats and examples of cracking WPA2. The passwords for access points are also user-generated, so Wi-Fi networks with weak keys and passwords are especially at risk. The most common attacks are key-reinstallation attacks and brute force attacks[6].

2.1.4 GNSS

GNSS is the oldest of the three technologies and probably the one which is the least useful in this context. This section will focus on GPS, since it is by far the most used GNSS system. GPS operates on three different frequencies, 1575.42 MHz, 1227.60 MHz and 1176.45 MHz. It was initially developed only for the US army but today the technology is free to use and available for every civilian in every country. Today it is maintained by The U.S. Air Force. In contrast to both BLE and Wi-Fi this technology's only purpose is to provide precise location and positioning information to end users. 24 satellites orbiting the earth forms a network, which communicates with devices on earth such as smartphones[8]. When a smartphone uses GPS it connects to four satellites at once and through trilateration, the position of the user is determined. For trilateration only 3 sensors are needed but for GPS a fourth satellite is used to make the estimated location more precise. Both BLE and Wi-Fi can also use trilateration to precisely determine the location of users.

The exact way of how trilateration is used, can be seen in Figure2.2. The basic idea is that satellites broadcast signals. Once a receiver gets a signal, it knows the distance – which is indicated by the circles around the satellites in the figure. This means the receiver can be located anywhere on the circles periphery. By receiving signals from three satellites at once and if the receiver has to be on the periphery of each circle, it means that the only place it can be located is on the intersection of the three circles. The 3D visualization to the left, shows the practical version of trilateration in use.

Since GPS is controlled by the US government, several countries have released their own candidates of GNSS systems – Russia has a system called GLONASS, China has BeiDu, Japan QZSS and the

[3] You can see more about the organization at their site `:https://www.wi-fi.org/`.

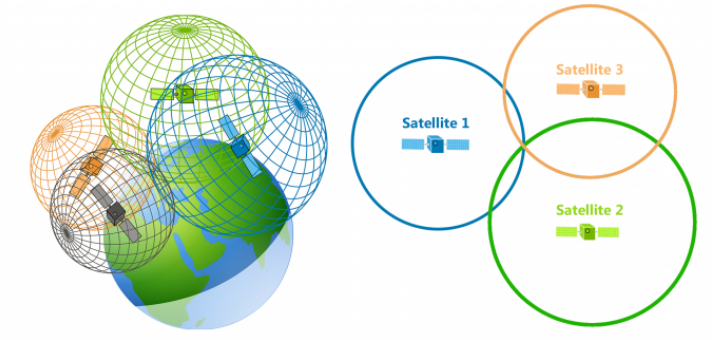


Figure 2.2: Visualization of how trilateration is used. The figure is taken from `https://gisgeography.com/trilateration-triangulation-gps/`

European Union has Galileo[22]. This means that devices today actually have multiple chips dedicated to each systems, which increases the accuracy depending on where you are on the globe. For example high-end Android phones support more than one of the systems[4] (Google Pixel supports all five for example).

The issues with GPS in the context of indoor location are when looking at its accuracy. There are several factors taken into account, so it is worth looking at some specific situations. When using a smartphone, GPS can provide a location within the margin of ≈ 4.9 meters. For most use cases this would not be sufficiently precise. There are also multiple factors which can weaken the precision heavily, such as[7]:

- Satellite signal blockage due to buildings, bridges, trees, etc.
- Indoor or underground use
- Signals reflected off buildings or walls (”multipath”)

Those factors are unfortunately very much present when looking at indoor positioning, which is why GPS is essentially unusable for the use case of indoor localization.

Another issue with GPS is its lack of encryption of signal. The US military uses encrypted signals, but for civilian use the signal is unencrypted. In the early implementation of GPS, encryption was used for civilian use, mainly to ensure that the positioning would not be too precise in case of enemies using the civilian band – it was called Selective Availability `(SA)` [9]. The difference in precision is huge, as seen in Fig. 2.3. The idea was scrapped in May 2000 and since then GPS for civilian use is unencrypted.

2.1.5 Comparison

In this section a comparison of the three technologies is be presented in the form of a table containing the most relevant features of the technologies in this context. The score given to each feature are determined based on the gathered knowledge according to this section, hence it is subjective. The score given is from 1-3, where 1 is the worst score and 3 the best. The scores can be seen in Table 2.1. The parameters chosen are deployment cost, availability, precision and security.

- Deployment Cost: It means the cost of deploying the system in terms of money. This can be the actual cost of additional hardware or the time in which it takes to set up the system.

[4] An overview of which GNSS systems supported by Android phones can be found at `https://developer.android.com/guide/topics/sensors/gnss`

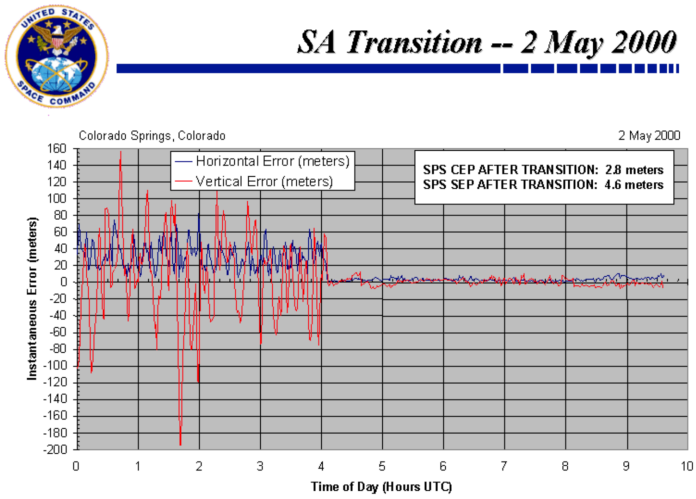


Figure 2.3: Before and after SA was stopped used

- Availability: It means how accessible it is – not in terms of when and where you can use it, but for example to use GNSS, you just need a smartphone and download an application such as Google Maps.
- Precision: It means how precise your location can be determined in terms of distance to your actual location. For example, GNSS cannot be used indoors, therefore it is imprecise.
- Security: It means how easy it is to read, modify and destroy the packages exchanged of the three location services

Scoring	Deployment Cost	Availability	Precision	Security
BLE	1	1	3	3
Wi-Fi	2	2	3	3
GNSS	3	3	1	1

Table 2.1: **Scoring Table** – the scoring is assigned according to the gathered information

A few comments on the reasoning behind the chosen scores:

Deployment cost and availability of BLE got a low score, because to use BLE you need specific beacons only for that purpose – they can be fairly expensive and for the system to work optimally, you also need several beacons. This is under the assumption that Wi-Fi is already installed in most buildings today. Assuming you are in a basement where there is no Wi-Fi, the deployment cost of BLE beacons, are less expensive than installing several access points.

GNSS's scores on deployment cost and availability are based on the fact that to use it, there is no additional hardware needed, assuming you have a smartphone. The reason Wi-Fi's availability and deployment cost are not the same as GNSS's, is because there would still be some cost in developing and setting up of such system – even if you already have access points. For GNSS it could essentially work straight away, by using an application such as Google Maps.

GNSS got the worst score in terms of precision and security, because it does not work well indoor and the signals exchanged for a normal user are unencrypted.

2.2 Choice of Beacon

The choice of a specific BLE beacon depends on a lot of different factors. There are a lot of different manufacturers providing high quality beacons, but when selecting it mostly comes down to the practical usage of the beacons in the current project. For this project the following minimal requirements have been identified:

1. Beacons should be able to access every device in medium size room (e.g. class room or office up to 50 m^2)
2. Beacons should be able to send a minimal amount of data e.g. identification or location data of beacon
3. Beacons should have easy communication/integration with iOS or Android developed apps
4. The casing of the beacons should be able to endure indoor climate for +10 years
5. Beacons should have +24 months of battery life
6. Good quality
7. Danish supplier

The two last requirements were somewhat limiting. Multiple reviews of different manufactures of beacons have suggested Kontakt.io for providing the best quality of beacons [5]. The company has also been ranked as the #1 hardware provider in 2018 proximity report [6]. Furthermore Kontakt.io provides their own SDK for usage in developing both Android and iOS apps, which includes object representation of beacons, gateways etc. and also functionality that for example make device ranging and monitoring easier. The SDK also enables easier location proximity which is of course highly relevant for this project. A beacon from Kontakt.io has been found sold from a danish manufacture called Smart Beacon SB16-2. It has a battery life of 48 months and is also fully compatible with iBeacon and Eddystone (BLE Communication libraries for iOS and Android). It has casing suitable for indoor environment's with easy replaceable batteries.

2.3 Conclusion

It was chosen to use BLE beacons for this project and the main reason is that it is the most precise and robust system. The cost of deployment is also higher with BLE technology, but it was chosen to value the correctness of data over the deployment cost – imagine using GPS to determine the user's room location and the location of the user would be completely wrong. It would result in corrupted data, which is not worth the lower implementation cost. Another strong trait is the easy physical deployment – they are all wireless. If the project has a low budget, a few beacons could be used for some specific rooms and then moved around accordingly, if one's wish is to also gather data from other rooms in the building. Assuming there is not a strong Wi-Fi coverage in every room in a particular building, BLE has an advantage. The Bluetooth SIG and Apple provides very good documentation and frameworks for implementing such systems, which is also an advantage from a developer's perspective.

There are a lot of potential use cases for BLE technology. EasyJet has already developed an iOS app using indoor localization technology with BLE beacons for airports in London and Paris [13]. Navigation of big hypermarkets is another use case and the big Hypermarket Auchan has already implemented a system that combines indoor localization with Augmented Reality. By using the camera

[5] Aislelabs have made a comprehensive guide on choosing the right beacon for different environments: `https://www.aislelabs.com/reports/beacon-guide/`

[6] `https://www.quora.com/What-is-the-best-bluetooth-beacon-manufacturer`

of the smartphone the user can be guided to specific sections of the hypermarket [1]. The collection of data is also made easier with beacons. For instance, a store could count the number of customers per day and track the movement of customers in the store to see which sections are popular etc.

There are other technologies which provide similar solutions that have not been discussed. The combination of Wi-Fi and GPS is a way to provide more precise indoor location. ZigBee is a technology which is very close related to how BLE beacons are used in this project and could be an alternative to the BLE beacons [15].

CHAPTER 3

Related Work

Bluetooth Low Energy is becoming more and more popular for determining indoor location and a lot of different projects utilizing this technology have already been made. This chapter goes through some of the relevant work done on the subject of indoor positioning based on Bluetooth Low Energy beacons.

3.1 Nearest Beacon Algorithm

Xin-Yu Lin et al. have made a very light weight system[11] that builds on iBeacon technology for estimating patients' location at a hospital. In their system, BLE beacons are placed around the hospital which emits a signal received by a mobile application installed on all of the patients' phones. The mobile application can be used to view the patients' medical history and at the same time, in the background, it receives the signal strength (RSSI) values for each beacon. Based on the RSSI values, it estimates which beacon is nearest and sends this information to a server, which can then track the position of all of the patients. The beacons are set up to emit a signal every second but X. Lin et al. use an average filter to get the beacon with the strongest RSSI value from the last five seconds. Of course, this is not always the nearest beacon in terms of actual distance, but it usually is, which is why the algorithm is called the *Nearest Beacon Algorithm*. The information about the nearest beacon is then uploaded to the server which maps the beacon information to an actual location understandable by medical staff. With this system, patients can be located faster in case of an emergency. However, a big weakness of the system is that it does not give any information as to where the patients are in relation to the beacon, only the location of the closest beacon. Assuming that the hospital is equipped with a large amount of beacons this accuracy might be sufficient for the intention of the system i.e. being able to find patients in case of an emergency, but it might not be sufficient in similar contexts. Estimating the user's exact location with a certain error margin might be required in some systems and it might also be cheaper because less beacons would be required.

3.2 Location Estimation using Network of Beacons

You-Wei Lin and Chi-Yi Lin have also made a paper[12] that includes BLE technology. In their project, they use BLE beacons to create a mesh network for receiving location data about the user and for sending personalized messages to the user. The system could be used for example in an airport to improve the overall experience for certain customers such as VIP guests. With the system the airport would be able to track VIP guests, who would then receive personalized message such as ads or navigation, and at the same time the airport staff would know the VIP guests' location at all times. Whereas in most BLE positioning systems, the smartphone's location data is transmitted via the user's application directly to the server through either Wi-Fi or mobile data (such as 4G), Lin et al. use the mesh-network for all communication with the user. This means that the user can be localized and receive personalized messages without internet connection at all. Instead, only Bluetooth has to be enabled on the user's smartphone, which often makes it less costly for the user, a lot more power efficient and usable in more types of environments. The location detection is however rather imprecise as it is only based on the nearest BLE beacon and not calculated mathematically with e.g. trilateration. Instead the server and the application will only be able to know in which room the user might be in or to which beacon

the user is closest. This of course is a disadvantage if the system would be used for precise indoor navigation. Y. Lin et al. had issues with their beacons only being able to send packages for up to 10 meters. This was due to a limited BLE beacon technology and also because their room location did not allow non-line-of-sight at greater distances. They propose that BLE beacons with Bluetooth 5.0 would greatly fix these issues. They also had problems with Wi-Fi interference resulting in a high probability of package-loss and the proposed solution is using Bluetooth 5.0 which has a lot higher data rate (2 Mbit/s). Bluetooth 5.0 also improves the channel algorithm making it less likely to interfere with the Wi-Fi channel.

3.3 Location Estimation using Trilateration

Y. Wang et al. have made a project where BLE technology is used for indoor location estimation. In their paper[21] Y. Wang et al. propose a solution for determining indoor location based on BLE beacons. In this project however, the goal is not to determine the precise location of the user with a certain margin of error, but instead to generate a small region where the user is guaranteed to be found. A visualization of the method can be seen in Fig. 3.1 [21], which is taken from their paper. Each gateway is a beacon sending out a Bluetooth signal with a defined maximum range. If the tag is in range to receive a signal from at least three beacons, then the overlap of the three beacon areas (i.e. the area created by the max range of the beacon signal), creates a new area, where the tag is guaranteed to be located at. In the

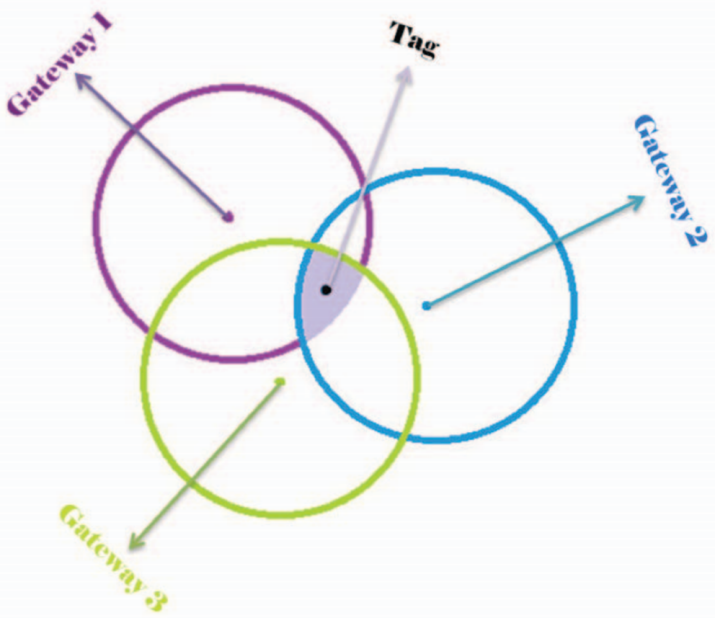


Figure 3.1: Localization method used by Wang et al.

paper, two methods for determining localization namely *Low-precision Indoor localization (LIL)* and *High-level Indoor localization (HIL)* are presented. The approach in *LIL* is to have two power levels for each gateway (a BLE beacon) which maximum range is calculated (e.g. two and five meters). When the tag (e.g. a tablet or a smartphone) is in range of multiple gateways, the region can be found by determining the overlap of signals from multiple gateways and by excluding regions where only high or low-power signals were received. This is done by the use of logical operations and by making the beacons constantly switch between sending a low-power and a high-power signal. *HIL* is based on the same principles with beacons being able to switch between two power-states and send out signals with two different pre-calculated ranges. But in this approach a statistical data-training phase is also

included to provide a more accurate distance estimation based on the signal strength of the BLE beacon. A large amount of measurements are used to determine a relation between distance and signal strength (RSSI) and this relation is used to group each RSSI value of a signal into a corresponding distance band around a Bluetooth beacon. The overlapping of each distance band will then be determined to be the region where the tag is located. The two methods presented are very interesting, because they provide a relatively easy way of accurately determining a region for the location of a user. Very often an exact location with a large error margin is not necessary and do not contribute considerably to the goal of the system. In our project, we are mostly interested in the room-location of the user and thus a location precision of a small region would most likely be sufficient for determining this. However, this approach requires a very time-consuming setup, especially the HIL-approach, as a statistical analysis of the relation between distance and RSSI is necessary to limit the area of the estimated region. This would for most situations probably prove to be too costly, as many measurements would have to be done for each beacon and then redone every time the battery of the beacon is changed since the approach relies on the power-level i.e the battery performance of the beacon.

3.4 LocationEestimation using Path Loss Algorithm

In their paper[19], Adel Thaljaoui et al. present a location system based on BLE technology that introduces a simple but interesting algorithm for relating distance to RSSI. The system is a combination of trilateration as mentioned in the previous section and an algorithm that predicts the distance depending on the environment and the signal strength between the tag (smartphone) and the beacon. The algorithm makes use of a formula called the Log-Distance Path Loss, where a relation between the signal strength of a beacon (RSSI) and the distance to the beacon are formulated:

$$RSSI(d) = RSSI(d_0) - 10 \cdot n \cdot log(\frac{d}{d_0})$$

With the formula it is possible to calculate the RSSI from the beacon received by the tag (RSSI(d)), depending on the distance d. The interesting thing is, that the formula also takes into account the environment of the system by including the variable d_0, which is the RSSI from one meter distance, and the so called path loss index n that merely depends on the propagation environment. The equation can of course also be solved for the distance instead of the RSSI, which makes it very valuable for localization systems. In the project Adel Thaljaousi et al. found that the model together with the trilateration-method could estimate the location of the device down to 0.4 meters. The advantage of the system is, that the algorithm can be relatively easily computed and can therefore run on the user's smartphone without causing notably difference in performance or battery consumption. A disadvantage of the system is, that it does not work well in environments that are not open. In these cases, where obstacles obscure the view from the beacon to the smartphone, the conversion from RSSI to distance will be inaccurate.

3.5 Location Estimation using KNN Algorithm

Finally, a different approach to estimate the user's location is presented by Yu-Chi Pu and Pei Chun You in their paper[14]. In this project, a fingerprinting algorithm with classification is used to map different RSSI values from BLE beacons to an actual location. In their paper they present an accurate indoor positioning algorithm, which can estimate a user's position with an error margin of down to 1.8 meters. Yu-Chi Pu et al. criticize the use of the Log-Distance path loss formula due to so called shadow fading effects. Shadow fading is the effect that the RSSI can fluctuate due to objects obstructing the path from the beacon to the device[1]. You-Chi Pu et al. propose their fingerprinting algorithm as a

[1]Professor in the Signal Processing Systems group, Jean-Paul Linnartz, explains shadow fading effects: `http://www.wirelesscommunication.nl/reference/chaptr03/shadow/shadow.htm`

solution to this problem, where a conversion from an RSSI value to a measurable distance is not needed. The fingerprinting method makes use of a machine learning algorithm called the k-nearest neighbor algorithm (KNN). The algorithm classifies a given point (e.g. a location) based on the classification of the surrounding points. The algorithm requires a learning phase where the classification of a certain amount of points are found. Then, in the testing phase, the classification of a new point is estimated by looking at the classification of the k closest points (nearest neighbors), where k is an arbitrary number, and the distance metric is also arbitrary (usually the euclidean distance metric is chosen). For a given square room, You-Chi Pu et al. set up 6 beacons around the walls of the room. Then a grid of 24 points, each separated by two meters, are measured, and in each of these locations the RSSI values of the 6 beacons are recorded. The exact location of each point inside the room is also measured and stored along with the RSSI value. This is the training data, and it is stored server-side in a database to be a used later as mapping data. When a new device in the room wants to be located, it records the RSSI from each beacon and sends this information to the server. The server then performs the KNN algorithm by comparing the received data to the training data on the server and then finds the k nearest training points. Furthermore, to give the exact location, the server calculates the average of those k nearest points, which is then used for the device's location. Figure 3.2, taken from their paper, gives an overview of how Yu-Chi Pu et al. use the training data to estimate a new device's location [14].

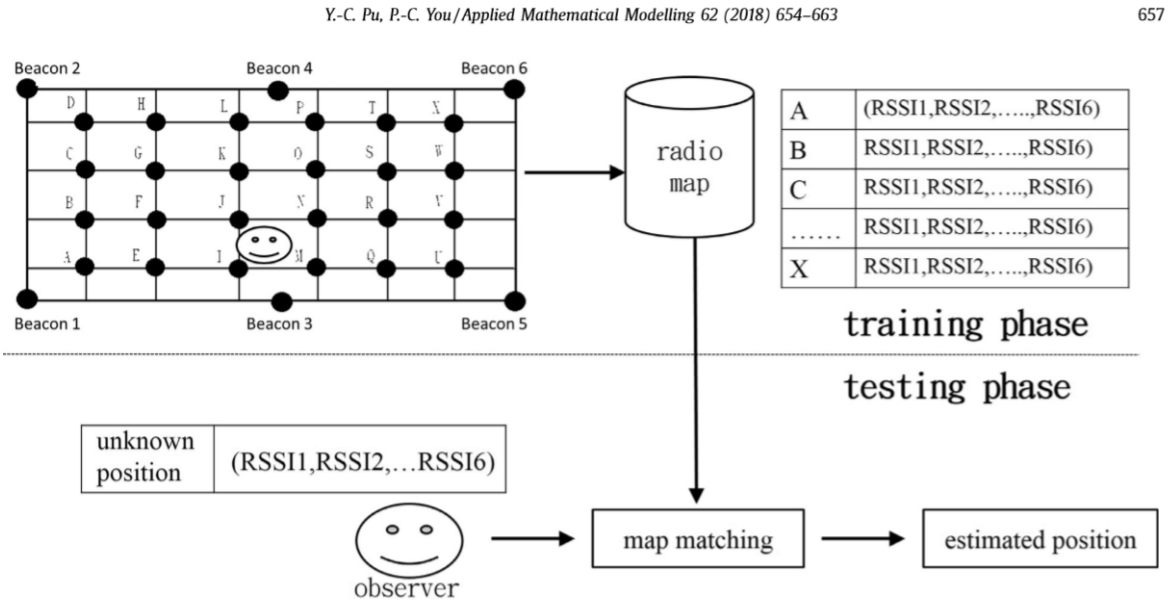


Figure 3.2: Localization method used by You-Chi Pu et al.

CHAPTER 4

Analysis

The intention of this section is to give an overview of the concepts in the project. An overall description of the the main components introduces the chapter followed by the domain, the requirements of the system and the project scope. The chapter concludes with the development process of this project, in terms of which methodologies and tools that were used to develop the system.

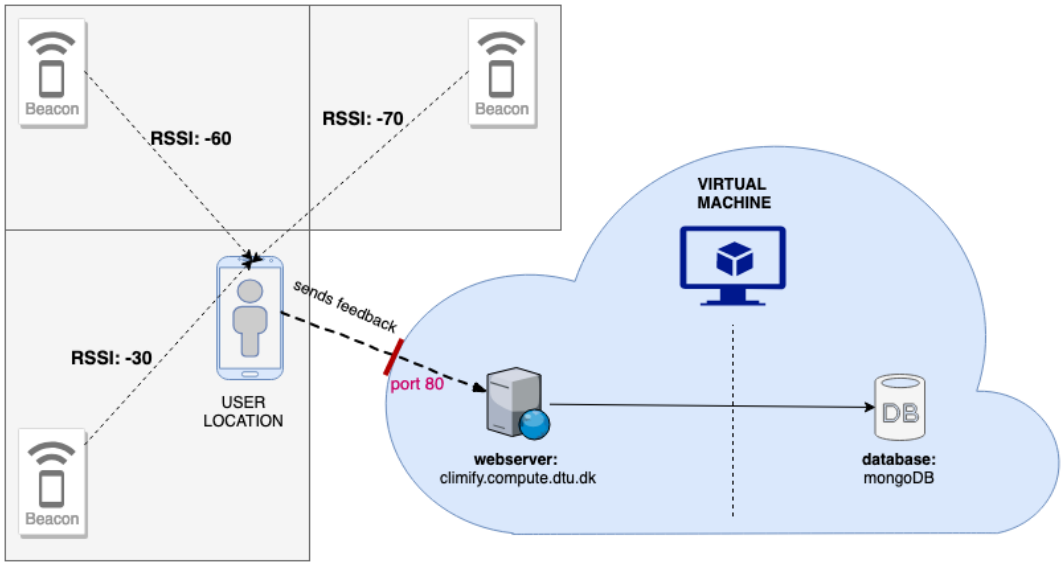


Figure 4.1: The project concept overview

Figure 4.1 gives an overview of how the main components relate to each other. The intention of the figure is to understand the system. This means that the part of the system that involves the location algorithm is not included in the diagram. This also means that the beacons placement on the figure is random.

Starting from the left, there are three gray squares which represent rooms, each containing one beacon. The beacons broadcast signals in a fixed interval, for example every 200ms and with a fixed signal strength (this can be changed through the third party application provided by Kontakt.io). In the bottom room, there is a user with a smartphone. This phone represents the mobile application described in this section. The mobile application has two parts and it is the component that binds the whole system together by getting signals from the beacons and sending this data to the server, while also providing feedback to the sever. The *first* part is its BLE implementation. When using the mobile application, it listens to nearby BLE signals and if the signal received is from a beacon that is a part of the system, it will recognize it and save the signal strength (RSSI value) of that beacon. It recognizes it through the `UUID` which is associated with the signal received. One might ask, how does it know whether a beacon is a part of the system? How does it know in which room a beacon is placed? This leads to the *second* part of the system: the back end server.

The back end server is hosted on a virtual machine provided by DTU where it communicates with a local database. The back end server provides the information of which beacons are a part of the system, the rooms in which the beacons are placed in, the buildings in which the rooms are in, the questions associated with specific rooms etc. When using the mobile application, this information is fetched from the server, so when it detects a BLE signal from a nearby beacon, it matches the `UUID` of the found beacon, with all the registered beacons in the database – if there is a match it knows that the beacon is a part of the system. Then, information such as room number/name, questions about the indoor climate etc. can be send to the mobile application. The location detection algorithm depicted in Fig. 4.1 is the simplest one described earlier in the paper – the beacon with the strongest signal will indicate in which room the user is located in. In this case, the beacon in the bottom room has the strongest signal (`RSSI -30`), which means that the system will register the user to be in that room (just as it happens to be in the figure). The user will be asked questions through the application based on its location, and the feedback will be sent to the back end server and stored in the database. This stored feedback can then be extracted through an API or through a website by users with sufficient authority.

4.1 Domain

To get an overview of the entire Feedme system, a domain model in form of a UML class diagram has been made and can be seen in Fig. 4.2. First, a brief overview of the domain will be given followed by a deeper explanation of each concept and their relation to other concepts in the domain of the Feedme system.

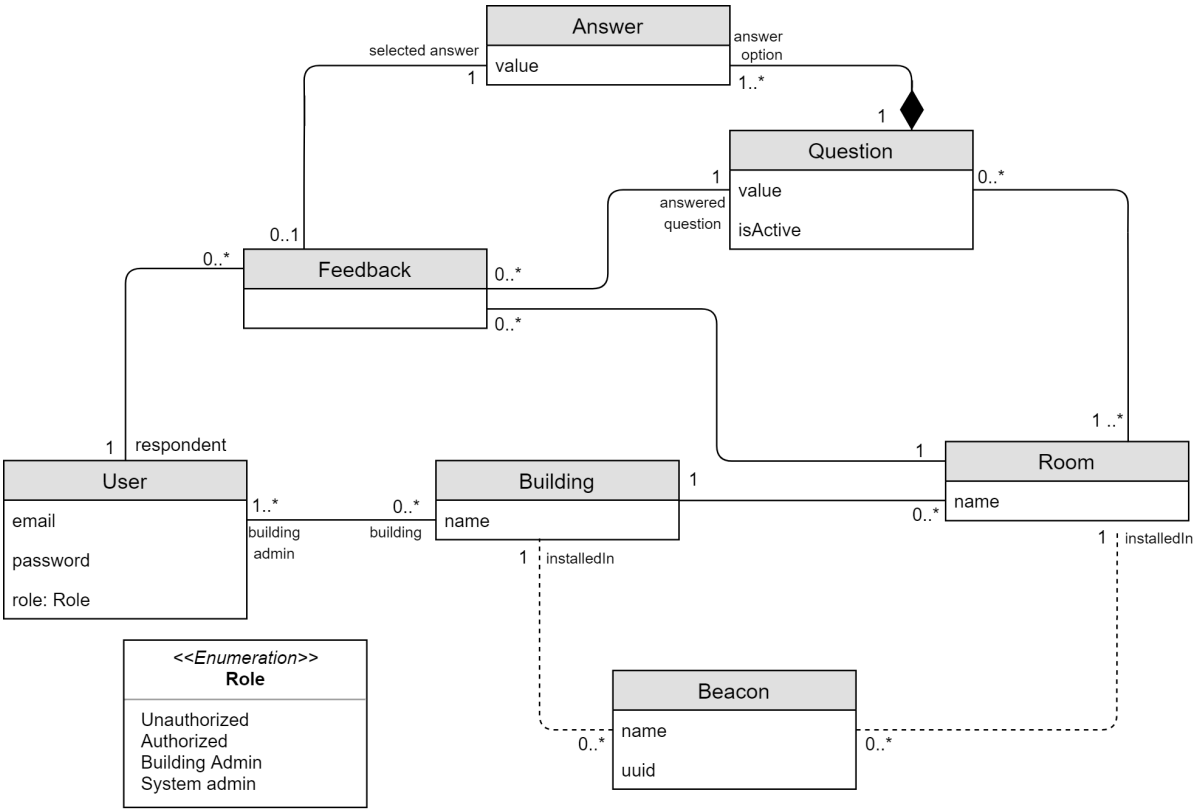


Figure 4.2: UML class diagram of domain model

The Feedme system works in a *building* environment with different *rooms* and it is mainly used by two types of users. Regular users, who use the system to provide *feedback* and later on are able to view statistics about the feedback they have given. The *room*, that the user is located in when providing *feedback* is estimated by the system through *beacons* placed around the building. These users have the role *unauthorized* as they do not need to sign up and log in to use the system. They are by far the largest user base of the system. However, for the entire system to work, it needs to be set up by someone, which would usually be an administrator of the building. This introduces the other type of user, i.e. a user with the role *authorized*. To get this role, users need to register to the system with an email and a password. When a user is authorized by the system, they can begin the setup-process. To set up the system, the building admin needs to register the building and its rooms in the system. Then, the building admin needs to physically place BLE *beacons* around the building and also register these beacons in the system. Depending on which location estimation technique is used, the building admin might also need to scan each room, thereby providing information to the Feedme system about where each room is located in the building. When this process is done, the building admin can begin to post questions for different rooms and modify the building and rooms they are managing and gather feedback from users of the building. Later on, the building admin or other users can learn from the feedback by extracting the feedback from the system to a file. A more detailed definition of each concept can be viewed in the appendices where a glossary of the project concepts can be found.

The domain model is written to generalize concepts as much as possible, such that the system can be used in many different environments. For this project, the system is intended to be used for a school building to provide feedback about the indoor climate, but with the generalization the system could be applied in many similar contexts, such as a working environment with offices or a dormitory with bedrooms etc. The class diagram shows the different concepts that the system is working with and their relationship among them. As the diagram shows, a user can have a direct relation to a building. This is the case when a user is authorized and is an admin on the building. By being an admin, the user can manage the system for a particular building. Each building can have many rooms and for each room there can be an arbitrary number of questions posted. One particular question can also be posted to more than one room making the relationship a many-to-many relation. All users can then respond to these questions creating *feedback* to the system, which means that *feedback* creates an association with a user together with the answer option they chose, the question they answered and the room which the user was located in, when the feedback was given. Each question has an *isActive* property which indicates whether or not the question will appear in the rooms it is assigned to. This is because users might have answered questions which the building admin no longer wishes to display. Rather then deleting the question completely, the building admin can set this property to false. The relationship between a question and an answer is a composition as each question needs to be given one or more answer options. These answer options can take almost any form such as a predefined scale of numbers. The only limitation is, that all of the answer options need to be predefined and therefore not allowing arbitrary options such as an arbitrary number or letter. Depending on what kind of location algorithm chosen for the system, a beacon in the system is either linked to a room or a building indicated in the model by dotted lines between the beacon class and the Room and Building class. Each *beacon* also has an UUID value, which is an ID that is used by the system to uniquely identify the beacon.

A user represents anyone using the system. Depending on what kind of user is using the system, the interaction with the system is different and therefore users are divided into different roles. In the following list the roles for this system can be seen:

1. Unauthorized user
2. Authorized user
3. System admin

How the different roles interact with the system will be explained in greater detail in Section 4.3 (User Stories). The roles are listed in order of the amount of access and rights the user has to the system. The further down the list, the more access the user is given to modify the system. Furthermore, a role always has all of the rights and access of the previously listed roles. I.e. a system admin can perform all of the tasks that an authorized user can perform and an authorized user can perform all of the tasks an unauthorized user can perform.

An unauthorized user is a user that is not registered with an email and a password. It is a user that cannot be securely identified i.e. the system cannot guarantee that only one person has access to that account as no sign up process with a password is needed. This type of user can only use the mobile application's functionality, which includes sending feedback, viewing feedback and other functionality which do not change the experience of the system for other users. These features will be presented in Section 4.3 (User Stories). An authorized user has signed up with an email and a password, and thus is allowed to perform more administrative tasks, that require authentication. Even though, a user seems to have both an email and password, the password itself is not stored in plain text in the database. Instead it is stored encrypted but this is discussed in greater detail in Chapter 7 (Design). To register beacons to the system you do not only need to be an authorized user, you also need to be a building admin for the building where you want to register the beacons. You can achieve this role by creating a building yourself which will automatically make you a building admin on that building. The other option is that a user who is already a building admin, can make other user's building admins as well. The last role is the system admin, which basically has the rights to all functionality. This role would usually only be given to the developers, who need to be able to modify data to potentially fix or improve the system. This could involve tasks such as deleting users or modifying their roles.

In the context of this project the system is intended to enable users to give feedback about the indoor climate in a school building. However, as the diagram in Fig. 4.2 shows, the system includes no references to the term *school*. The system only requires a definition of a building with a custom name where the system is implemented which means that the system can basically be implemented in any context that includes a building with a set of rooms accessed by a set of users. There are of course some requirements to the dimensions of the rooms and buildings, but generally many building types would be accepted. Another important thing to note is, that the model does not include any references to the concept of indoor climate. The system is limited to users given feedback from a particular room but the feedback, meaning the questions answered could be about anything, not only about the indoor climate. In many different scenarios a location detected feedback system could be useful which increases the potential of the system.

4.2 Security

For most users, the system is used in a manner that cannot be harmful to the system as a whole. Unauthorized users are simply allowed to send feedback and afterwards view the feedback they have given themselves or feedback given by other users. For this usage of the system no authentication is required. A user can simply download the application and immediately begin to send and view feedback. The system is still able to identify the communication from a user to the device they are using through the previously mentioned encrypted token generated by the server. But other than that, the user is

completely anonymous. This means that users can view their own feedback as long as they are using the same device every time. However, if the user decides to use a different device the system will see this as a completely different user and, on the new device, they will not be linked to the feedback they've previously sent. Another drawback to having an account-less system is that the user is only identified by a token generated at the server based on the ID of their device. This means, that anyone obtaining that token, which is usually stored non-encrypted on the device, will be able to send feedback on behalf of that user. This might cause a bad experience for the user, as they could potentially see a lot of feedback they have not sent marked as their own feedback by the system. This would also give a false impression to the system and the building administrators analyzing the feedback data, as it would appear that one person has sent more feedback than they actually have. These are some of the threats and disadvantages that the system is exposed to when users do not need to register themselves to use the mobile application. Though you could argue, that there would be little to no gain for a malicious user to send feedback on behalf of other users, and the feedback sent by the users are most likely not sensitive and might not be very valuable for the user to keep if he/she changes device. From the perspective of the user analyzing the feedback data, getting as much feedback data as possible is highly valuable. A forced sign up process might make less people use the application which ultimately might result in less feedback gathered. For this very reason, it was decided not to require a registration process for users sending and viewing feedback.

This means, that the majority of the users of the system, do not need to go through a registration process. However, to administrate the system, including registering buildings and rooms, posting and modifying questions etc. the user is required to register with an email and a password. The functionality that building administrators have access to are exposed to a lot more threats. If a malicious user would have access to this functionality he/she could potentially harm the system to an extend that it would be irreversible. This functionality includes deleting rooms, modifying questions, modifying beacons which could also invalidate the location estimation algorithm etc. For this reason, it was decided to move most of the building administration functionality to a separate application, namely the website. On the website it is required that users register themselves, before being able to access this functionality. When the user has signed up, the user will be given the role *authenticated user* and is allowed to create new buildings. On buildings, which the user has created itself, the user will also be allowed to post new questions, create new rooms, register beacons etc.

Both in the case of unauthorized and authorized users, the user itself is identified by a JSON web token. For unauthorized users, this token is created through a hash function based on a private key on the server and for authorized users the token is also based on the user's password. This strategy of identifying users was chosen for many reasons, one of which being that the user's identity is kept anonymous. An alternative to uniquely identifying users would be to just use their device ID and store this ID in the database. However, in the context of the recent GDPR restrictions, this would generally not be an accepted way of storing data, as the device id could potentially be used to identify the actual user of the application. This, along with the location of the user, would be sensitive information, which is generally to be avoided, unless the system has a good reason for keeping that data. By identifying users with an anonymous, encrypted token this potential problem is avoided all along.

4.3 User Stories

To simplify and explicate the different functions of the system, user stories have been written which specify the actors, the task and the given benefit for each functionality. The Mike Cohn model is used to give a precise and comparable structure to each user story[4]. In a few trivial cases the benefit-part has been left out. Some of the requirements of the user stories have been marked as technical because they require a certain amount of technical knowledge and will therefore not be relevant for all end-users. For more detail on each user story, detailed use cases have been made for the most important functionality. These use cases can be seen in Appendix B (Detailed Use Cases).

These user stories have been presented and discussed with the product owner of the Feedme system. After an agreement of which user stories should be included in a specific sprint these user stories have been made into acceptance tests, which have been used to implement the actual software fulfilling the requirements. In some cases, detailed use cases have been made to further explicate the new functionality. In these cases the detailed use cases have been used to implement the new functionality and to make sure all error cases are covered.

1. As a user, I want to give automatic location-detected feedback to easily contribute to the improvement of the user experience for the room I'm located in.
    - Users should be able to send feedback from a mobile application
    - The time to detect the room should not exceed 1 second
    - The room should be estimated correctly above 95% of the time
2. As a user, I want to see which room I'm in, to make sure I'm in the right room
    - Room location should be displayed through a mobile application
    - The time to estimate the room should not exceed 1 second
    - The room should be estimated correctly above 95% of the time
3. As a building admin, I want to register beacons, to setup the system for automatically estimating the user's room-location.
    - (Technical) Users should be able to register beacons through an API
    - Users should be able to register beacons through a graphical user interface (GUI) on a website
    - Apart from the room only the beacon ID has to be known to register a beacon
4. As a building admin, I want to extract feedback data, to learn about the user's experience in the buildings I'm administrating
    - (Technical) Users should be able to get the feedback data through an API
    - Users should be able to get the feedback data through a GUI on a website
    - Users should only be able to get feedback data from the buildings he/she is an admin on.
    - Users should be able to limit the feedback data they get to feedback given from a specific user
    - Users should be able to limit the feedback data they get to feedback given from a specific room
    - Users should be able to limit the feedback data they get to feedback given in a certain period of time including feedback from the last 1, 7, 31 day(s) or all time.
    - Users should be able to limit the feedback data to a combination of the limitations listed above.
5. As a building admin, I want to add a question linked to a room, to get feedback about the user's experience of that room.
    - (Technical) Users should be able to add the question through an API
    - Users should be able to add the question through a graphical user interface on a website
    - Users should be able to add a custom number of answer options
    - The room, that the question is linked to should be selectable through a list of rooms
6. As a building admin, I want to notify the users in a particular room about an available question, to make them answer that question

7. As a user, I want to see my own given feedback-data, to explore how I've previously experienced different rooms and buildings.
    - Users should be able to see feedback through a mobile application
    - Users should be able to filter feedback based on today, past week, past month and past year
    - Automatically filters feedback to the user's current room location
    - Can choose to see given feedback from other rooms through list
    - All questions answered (in a room) by the user are listed
    - Questions are clickable and a chart with chosen answers are shown
8. As a user, I want to see all feedback for the room I'm currently located at, to explore the general user experience of that room.
    - See feedback through a mobile application
    - Filter feedback based on today, past week, past month and past year
    - Automatically filters feedback to the user's current room location
    - All questions answered (in a room) are listed
    - Questions are clickable and a chart with chosen answers are shown
9. As a building admin, I want a website with a GUI corresponding to the implemented API's, so that I can easily change the system.
    a) Extract feedback
    b) Visualize the feedback through graphs
    c) See the buildings and rooms I am administrating
10. As a building admin, I want to add/register a building, to setup the feedback-system and get feedback from a particular building
11. As a building admin, I want to add/register a room, to setup the feedback-system and get feedback from a particular room
12. As a building admin, I want to delete a building from the system
13. As a building admin, I want to delete a room from the system
14. As a building admin, I want to delete a beacon from the system

4.4 Scope

To ensure that the project is delivered on time with the most important features implemented, it is essential to scope the project. This should also ensure that the delivered system with the restricted amount of features is robust and ready to real-world deployment.

After discussions with the product owner of Feedme and in continuation of the user stories, the project's two core feature is to be able to locate users indoor and to easily setup the system for the building administrator. Therefore two components are needed, a website and a smartphone application, where the website is a system administration tool to extract feedback and a lot of different functionality could be extended to be a part of the website as well. Ideally, the website would be implemented with a nice GUI to please the building administrator and to easy visualize feedback through various graphs – but due to time-based restrictions, the visualization of the feedback was scrapped on the website. It was deemed more valuable to provide quality API's which the administrators can use themselves to

visualize data, rather than spend a lot of time on GUI to also present the data. This does unfortunately mean that the building administrator needs to handle the data themselves and present them the way the want it.

Initially, it was also discussed to try out other ways of locating users, for example through Wi-Fi instead of BLE beacons. However, it was decided to solely focus on BLE to spend the time to develop an algorithm which provides a precise indoor location and instead present other people's work in the domain of whether Wi-Fi is more precise than BLE or not.

An idea of the system providing a navigation system for the users was also briefly discussed, but it was deemed a too time consuming and complex task to be implemented. A navigation system would solely have the purpose to create an incentive for users to use the application more frequent – which is indeed very important, but it does not provide any relevant value in form of a feature, for the building administrator. Instead, a lot of time has been moved from the navigation system to the graphical user interface and user experience of the application, to make sure people actually feel comfortable and want to provide as much feedback as possible.

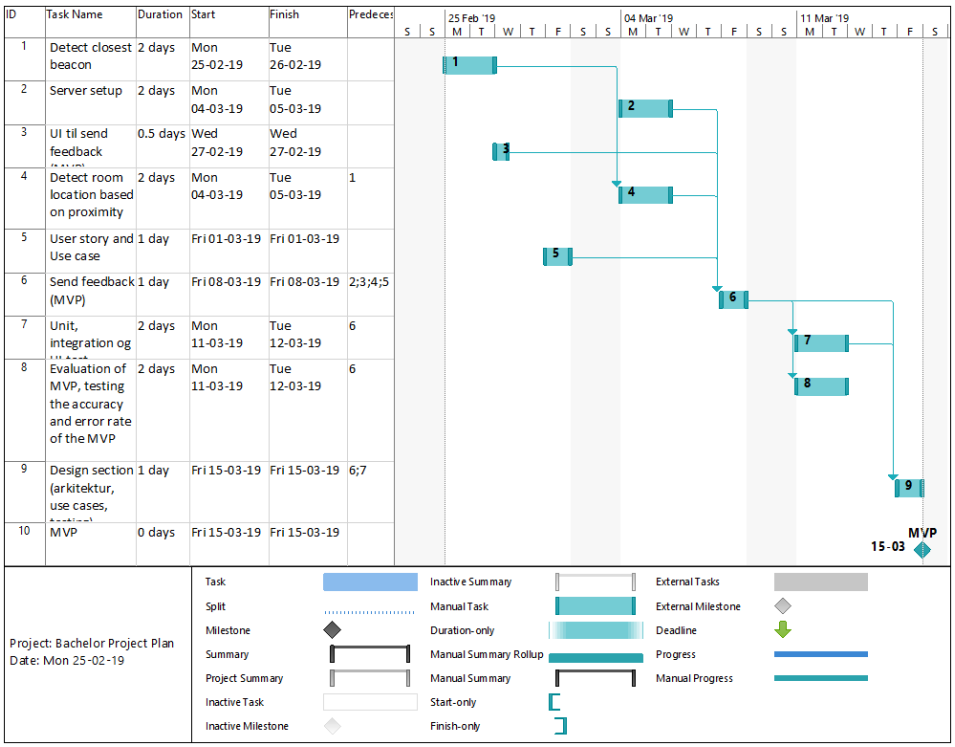
4.5 Development Process

When dealing with a project of this size, it was deemed necessary to use various project management tools to make sure the project is executed according to the planned schedule. In this project, the intention is to use a mix of agile software development and classical project management in terms of a project plan. It might be contradicting to create a project plan that stretches throughout the project period and at the same time claim that the project is following an agile approach. However, the project plan is created more to have an overview and a guideline in which direction the project is aiming, rather than a schedule that has to be followed strictly. It is also used to make sure the sections of the report is written in a timely matter. The project plan is divided into parts, which each represents a sprint. Throughout the development of the system, the project owner has to be presented with the "finished" parts of the system and discuss whether it is satisfactory according to the understandings of each user story. If it is deemed satisfactory, a testing and evaluation of the sprint is carried out. In this section, the different project management tools that are used, are presented in more details.

4.5.1 Sprints and Project Plan

The project has been divided into sprints of 2 to 3 weeks with fixed due dates. Each sprint will have an overall focus and include one or more tasks. The sprints are visualized and managed with Microsoft Project. Here, the sprints and their descriptions are presented with a Gantt Chart and each tasks of the sprint is visualized including their relationship with each other. At the beginning of each iteration, the current sprint is divided into sub-tasks that compliment the overall goal of the sprint and the date for completion, including the duration, is set for each task. Furthermore the resources for all tasks of the sprint are set i.e. the member(s) to work on each task. A milestone is created which depends on one or more tasks from the sprint to determine the completion date of each sprint. When the milestone has been reached, the sprint can be considered complete. In Fig. 4.3, a cutout of the project plan with the Gantt chart is shown. In the cutout only one specific sprint, namely the MVP-sprint is shown.

To manage the progress and completion of the individual sub-tasks of a sprint the Kanban board Trello is used. In the beginning of each sprint all of the tasks are put into the Backlog list, and the associated members can pick the tasks they want to move on with. As the sprint progresses each task will first be moved to the list "In Progress" and later to the list "Done" as shown in Fig. 4.4.



ID	Task Name	Duration	Start	Finish	Predeces
1	Detect closest beacon	2 days	Mon 25-02-19	Tue 26-02-19	
2	Server setup	2 days	Mon 04-03-19	Tue 05-03-19	
3	UI til send feedback	0.5 days	Wed 27-02-19	Wed 27-02-19	
4	Detect room location based on proximity	2 days	Mon 04-03-19	Tue 05-03-19	1
5	User story and Use case	1 day	Fri 01-03-19	Fri 01-03-19	
6	Send feedback (MVP)	1 day	Fri 08-03-19	Fri 08-03-19	2;3;4;5
7	Unit, integration og	2 days	Mon 11-03-19	Tue 12-03-19	6
8	Evaluation of MVP, testing the accuracy and error rate of the MVP	2 days	Mon 11-03-19	Tue 12-03-19	6
9	Design section (arkitektur, use cases,	1 day	Fri 15-03-19	Fri 15-03-19	6;7
10	MVP	0 days	Fri 15-03-19	Fri 15-03-19	

Figure 4.3: Gantt Chart of first sprint

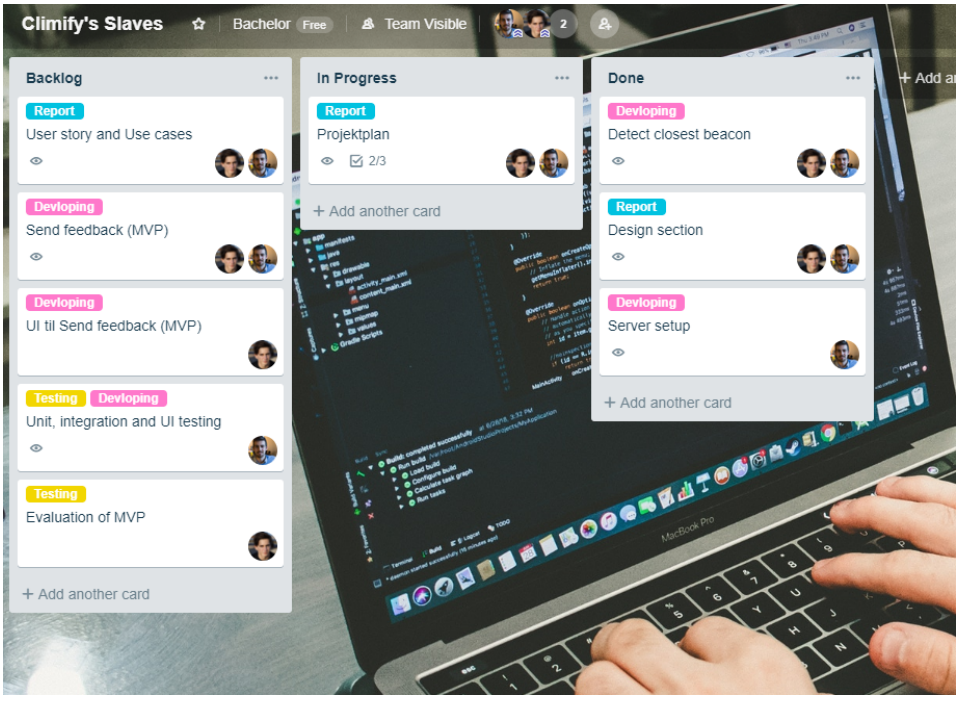


Figure 4.4: Kanban board of MVP sprint

4.5.2 Risk Register

For this project a risk register has been created. The risk register is shown in Fig. 4.5 and the most critical risks can be seen, by looking at the highest number in the score column. The `Score` has been calculated using the classic project management risk formula, $risk = probabilty \times impact$. When the risk register was created, the two most critical risks were the ones involving integrating Feedme's already implemented system into this project **(R04)** and general BLE related implementations **(R05)**. In general it is always hard to integrate an existing and different technology into a new project, which is why this was deemed the task involving the most risk. You are never really sure what to expect and usually there are unforeseen implementation issues which has to be resolved. The reason behind determining score of **(R05)** to be the second most critical risk, is because of the lack of experience working with BLE technologies.

The risk register was developed in the beginning of the project, and since an agile software development was used, some of the found risks was never really relevant. The most critical risk, **(R05)**, was eliminated when it was chosen that the system developed in this project should after all be completely separated from the current state of Feedme.

The register's most important feature is probably not to determine the tasks involving the most risk, but rather what to do when the damage is done and a risk actually takes place. This is why there is a `Solution` column, because it is the column proposing the solution to when the risk takes place. The solutions are alternatives to the original plan, which means that they were not deemed as the best choice for the project initially – they might be unavoidable to use in a situation where assuming for example BLE is simply too hard to implement.

Index	Risk	Description	Propability (1-5)	Impact (1-5)	Score (p*i)	Solutions
1	Inaccessable project planning software	DTU needs to grand access to DreamSpark for access to the project planning software Microsoft Project which is not given by default	3	1	**3**	Use other software e.g. Excel
2	Communication issues with supervisors	Due to sickness or misunderstandings the communication with the supervisors can be limited or completely unavailable which can affect the project especially in the scoping phase	2	4	**8**	Work on other parts of the project which has been fully agreed upon by the supervisors
3	Issues accessing Climify	Admins on Climify needs to grand access to source code which is not publically availble. A delay in this process would heavily affect the project as both the mobile and web app would be dependent on the structure of Climify	2	4	**8**	Work on parts that are not directly relient on Climify such as UI of mobile app, localization algorithm etc.
4	Issues integrating exisiting Climify system into our project	Existing climify system has code that needs to be either remade completely or integrated with our mobile app and server. Climify internal code might not be easy to understand and developed with unfamiliar technologies. Code might also not be implemented in an optimal and easy to integrate way	4	5	**20**	Good communication with supervisor and other admins and developers of Climify. Limit communcation with Climify system as much as possible and replace bad software with own implementation
5	Problems using Bluetooth technology	Using bluetooth to estimate the user location might be very complex and bluetooth is a technology that non of the members has any experience on. Unpredictable issues can emerge	4	4	**16**	If the Bluetooth technology becomes very complex to understand get outside help. Worst case scenario investigate other localization technologies such as WiFi and Zigbee
6	Wrong choice/amount of Bluetooth beacons	Beacons with the right capabilities and specification are crucial for estimating the indoor location and is dependent on the physical environment (size of room, thickness of walls etc.) and on the software of the beacons	2	4	**8**	Buy new beacons preferably from vendor with fast shipping. Change environment by either adjusting the room placement of the beacons or completely changing location.
7	Issues comprehending localization algorithm	Depending on which localization algorithm is chosen the math might be very complex and very time consuming to understand	2	3	**6**	Choose an easier localization algorithm e.g. just to detect nearest Bluetooth beacon or an algorithm that is well-documented
8	Issues developing in Swift	For developing mobile application Swift will be used which the group does not have a lot of experience with	3	2	**6**	Read Swift documenation carefully or use another framework within Swift
9	Issues with automated tests	Inexperience in Swift and JavaScript might result in code that is not easily testable. It might also be difficult to test software that is heavily integrated with Climify.	4	2	**8**	Rely more on structured manual tests
10	Lack of knowledge to make sufficiently secure login system	Having a login-system in a mobile application that interacts with a web application and server involves a lot of security problems, that might be hard to implement as a result of inexperience.	3	2	**6**	Take less security precautions
11	Potential users of the system are inaccessable for testing	It might not be possible to test the system on inteded users of the applications	2	2	**4**	Find other similar groups to test the aplication on
12	Developing map of navigation system turns out to be more time consuming	Making a real-time map for displaying navigation information can turn out to be a very complex and time-consuming task	4	3	**12**	Prioritize the UX of the navigation system less or if the task is too complex implement alternative feature
13	Issues in finding feature that gives incentive to use app	It might be difficult to find a feature that actually motivates people to not only use the application but also to make them send feedback about the indoor climate.	4	3	**12**	Survey to find out or draw inspiration to what feature could motivate people to give feedback

Figure 4.5: Risk register

4.6 Summary

This chapter gives an overview of how the system is perceived and what the main components are. Afterwards the domain of the Feedme system is presented, with the focus on the roles of various users in the system and how the different concepts relate to each other. To make sure the project owner and the developers are on the same page, user stories and use cases were created and presented. This leads to a scope section, which main purpose is to filter out the user stories and features that is deemed too expensive to develop or the ones which are less important than others. The section concludes with the development process of the project, with a focus on tools and techniques used to make sure the project is delivered on time with the necessary features.

CHAPTER 5

Software Handbook

In this chapter, the focus is on how to use the software that is implemented. Feedme is a system, in which users can give feedback about the indoor climate based on which room they are in. The users room are automatically determined by a mobile application and a web server combined. The software in which users can interact with is a web application and a mobile application. Before diving deeper into each piece of software, the roles which a user can have, needs to be clarified. There are two roles: **unauthorized user** and **authorized user**.

An **unauthorized user**, is the user which gives feedback through the mobile application. It does not have an email or password and the user's identity is kept anonymous. The mobile application can also be used to other things besides giving feedback, but this will be covered in greater detail later in this chapter.

An **authorized user**, is a user which can use both the mobile application and the web application. It is the user that sets up the building, the rooms of that building, the beacons and it is the user which receives the feedback given by the unauthorized users. An authorized user has an email and password, because authentication is needed to perform the administrative tasks.

The first subsection is on how the unauthorized user can interact with the mobile application and the second is on how the authorized user can interact with both the mobile application and web application. If the reader is reading the handbook as part of the whole report, there are parentheses in the following section, describing how the user stories corresponds to the implemented features.

5.1 Unauthorized User

The mobile application has 7 screens. The main screen is the screen in which an unauthorized user can give feedback, seen in Fig. 5.1. From this screen, you can either navigate to the data tab or if you are an authorized user, you can navigate to the login page through the side menu that appears when clicking on the settings button in the top right corner.

As an unauthorized user, the parts of the mobile application that are relevant, is seen in Fig. 5.1 and in Fig. 5.2 (covering user stories 1, 2, 7 and 8). Figure 5.1, shows the first screen which is presented to the user upon opening the app. In this case, you have to wait roughly 2-3 seconds before the user's location is determined automatically. The questions shown on the screen are associated with the room you are in, hence when pressing one of the options, you give automatic location-detected feedback (user story 1's requirement of determining the location in less than a second is not fulfilled). After the location has been determined, the room you are in, is presented as seen in Fig. 5.2. The room you are in, is shown in the bottom right corner with the text: "You are in Room 324" (user story 2).

Moving forward to the other features, Fig. 5.2 shows the rest of the screens (user story 7 and 8). Figure 5.2a shows the second tab, the data tab, which gives the user an overview of the feedback given. The feedback can be filtered based on when it was is given, and the filters are from today, past week, past month and all-time, as seen in the top most part of Fig. 5.2a. You can also see the feedback given by all other users, by tapping the *everyone* button down in the right hand corner. The last parameter that can be changed, is from which room you want to see the feedback. When you navigate to the data tab, you are automatically shown feedback in the room you are currently in. To see the feedback from

another room, you can press the room icon in the upper right corner of Fig. 5.2a. In the pop-up shown in Fig. 5.2c, you can change the room and also the building in which you want to see the feedback. The numbers next to the questions which are answered, indicate how many times that particular question has been answered. If you for example click on *How was the air quality?* question in Fig. 5.2a, you will be presented with a pop-up window, shown in Fig. 5.2b, which shows a pie chart with the possible answer options for that particular question and how many times each option has been answered (in percentage). When looking at user story 7 and 8, all the criteria are met.

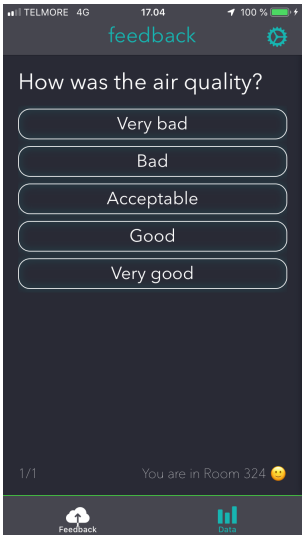
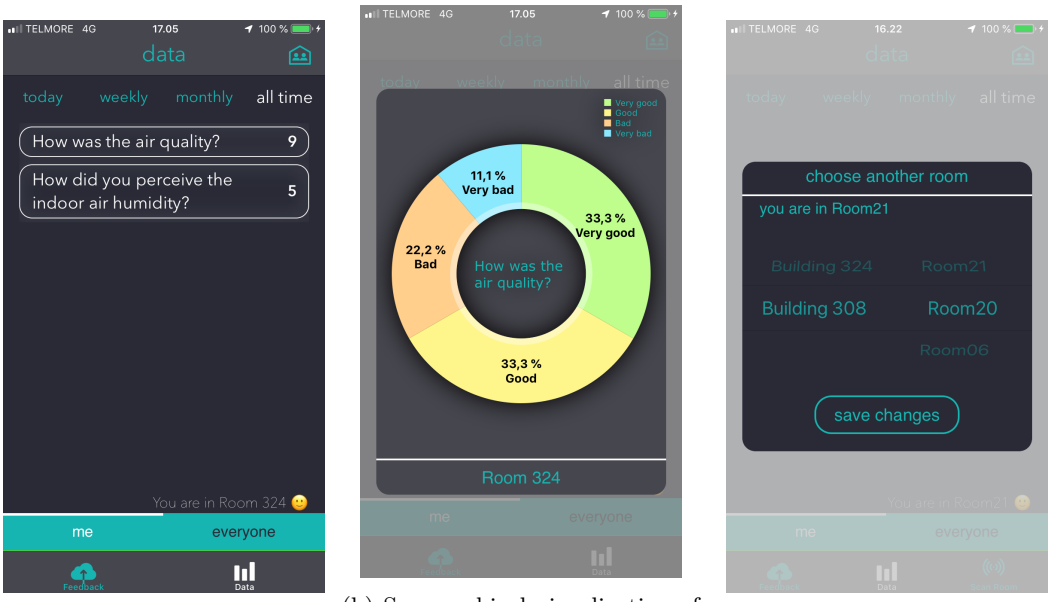


Figure 5.1: The feedback screen (main screen)



(a) See given feedback screen

(b) See graphical visualization of given feedback

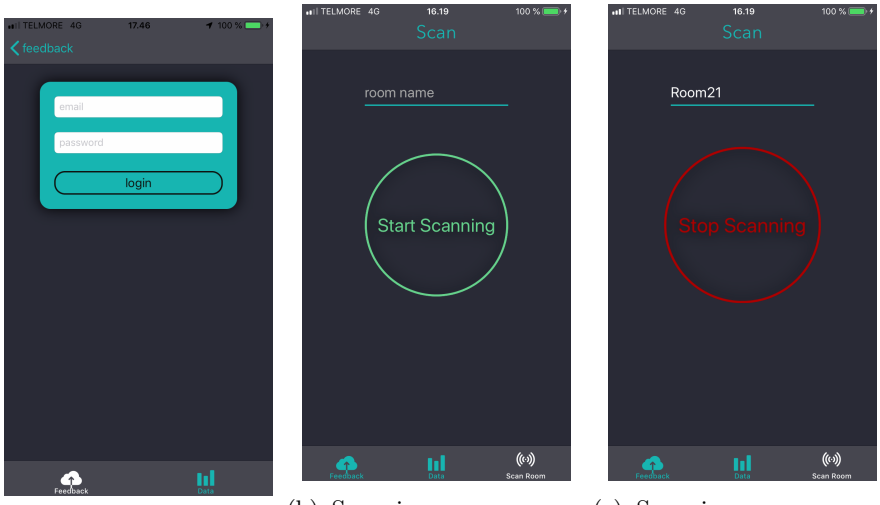
(c) Change room screen

Figure 5.2: The implementation of the user stories 1, 2 and 7

5.2 Authorized User

Most of the interaction with the software as an authorized user, is through the website. However, to register a room (user story 11), you need to use the mobile application. The purpose of this subsection is to demonstrate how the authorized user sets up the system through the mobile application and website.

As the name indicates, there needs to be some sort of authorization in the app, to be an authorized user. As mentioned earlier, you can click on the settings button in the top right corner of Fig. 5.1 to navigate to the login screen, which is shown on Fig. 5.3a. If the user successfully logs in, a third tab will appear, which is called Scan Room (the new tab is seen in the tab bar in Fig. 5.3b and Fig. 5.3c). The tab will be visible as long as the user is logged in as an authorized user. The scanning screen is very simple, and it is where you add a room to your building. The screen is shown in Fig. 5.3b and Fig. 5.3c. The app automatically detects in which building you are in, so the only information which is needed is the room name. Once the room name has been provided, the user can press `Start Scanning` and walk around in the room for around around a minute (the more time that is used in the process of scanning, the better the result) and stop the scanning process by pressing the button `Stop Scanning`. If the scan was successful a message will appear, stating that the room was successfully created. To verify that, the user can navigate to the main feedback screen, and after a few seconds, the text in the bottom right corner should update and show the name of the room which was just scanned (assuming you still are in that room).



(a) Login screen

(b) Scanning screen - user story 11

(c) Scanning screen - user story 11

Figure 5.3: The implementation of the user stories 1, 2 and 7

The website is used for the managing tasks of the system. These tasks include registering beacons (user story 3), extracting feedback data (user story 4), add questions to a room (user story 5), registering buildings (user story 10) and registering rooms (user story 11). These tasks are basically all for the administrators of the building. As mentioned previously, to use this part of the system, a registration process is needed and this is done through the website. When the user navigates to the website for the first time, a sign up screen is presented for the user as shown in Fig. 5.4. Here, the user has to sign up with their email and a sufficiently complex password. Upon signing up, the user is immediately logged in. Currently there is no email verification implemented in the system, requiring the user to verify the email they used to sign up.

When the user has signed up, the user is directed to the main page of the website, which includes a navigation bar, providing the user with a set of buttons to quickly navigate to other parts of the website. Through the navigation bar, the user can register new buildings, and upon doing so, the user

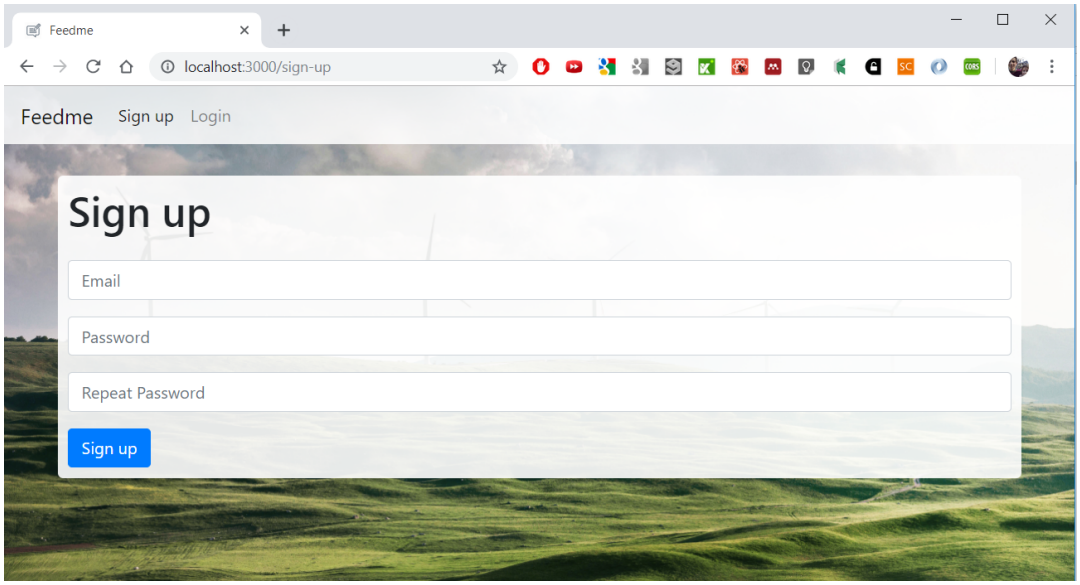


Figure 5.4: Sign up page for website

will automatically become an admin on those buildings. This means that the user can then proceed to do all of the managing tasks mentioned above for that building. In the navigation bar, the user can see all the buildings he/she is an admin for, and navigating to one of these buildings, will give the user an overview of that building. Through the building overview all of the managing tasks can be carried out. The building overview is divided into different components for managing different parts of the system, as shown in Fig. 5.5. The first component is a list of rooms registered in that building, including a button to add new rooms. Here the user can manage all of the rooms registered for that building. Upon clicking on a room, the user is presented with a different component for managing the questions registered to that room. In this view, the user can manage, which questions are active or not, add or delete questions, and modify their answer options. The last part of the building overview is for managing the beacons of the system. To the right of the rooms-component, a list of the registered beacons for that building are shown. In this view the user can add or delete beacons, and optionally modify their UUID.

The last functionality which is done through the website, is extracting the feedback given to the system. This functionality does not have its own view however, instead it is distributed to the different components mentioned previously. To each of the components mentioned above, an icon for extracting feedback is displayed along side the component. This is the blue icon with a number shown in Fig. 5.5. This icon shows how much feedback is given for a specific element. For the icon along with the building name, the number refers to how much feedback is given in that building. For a specific question, the icon refers to how much feedback is given for a particular question etc. When this icon is clicked, a popup will appear with the most important statistics of the feedback including how many users have given the feedback, how many questions were answered etc. The popup includes a button to download the feedback, as shown in Fig. 5.6, which shows a screenshot of the web page when the feedback popup has been opened. When the *Download feedback* button is pressed the user will get a JSON file with all of the details of the feedback data.

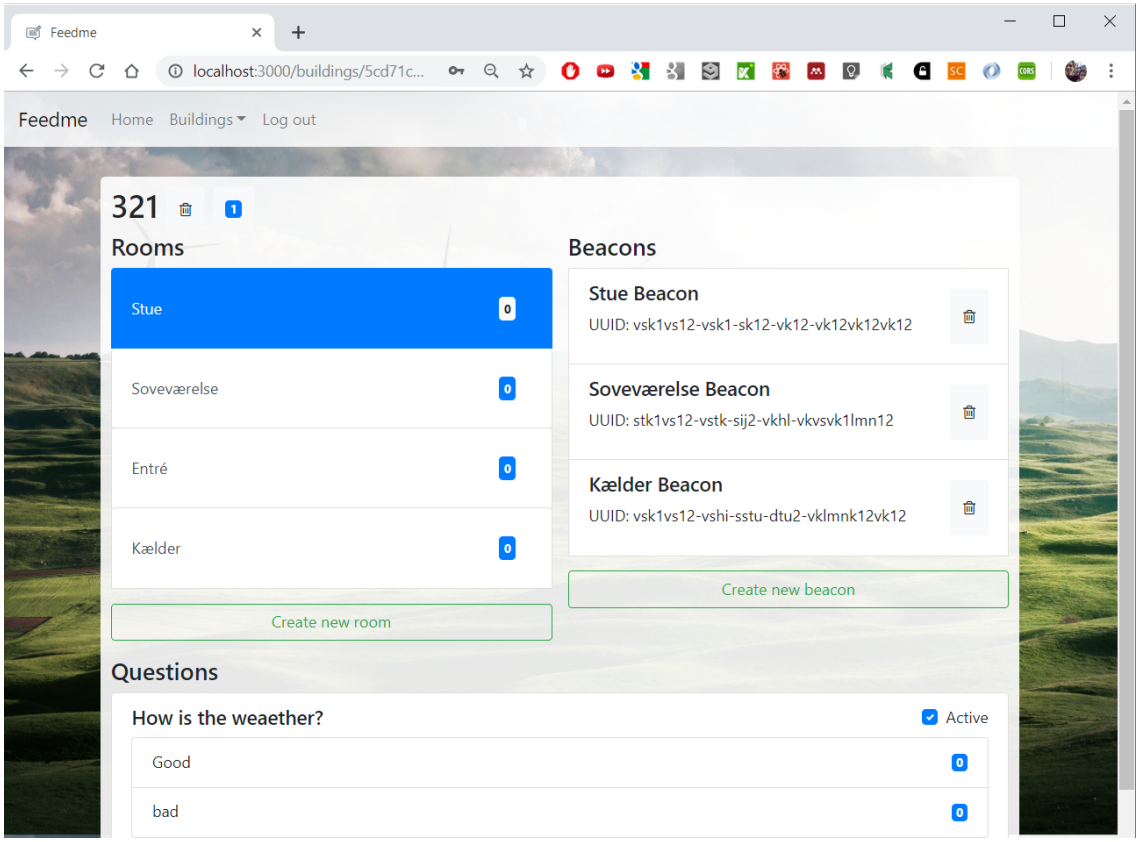


Figure 5.5: Building overview page for website

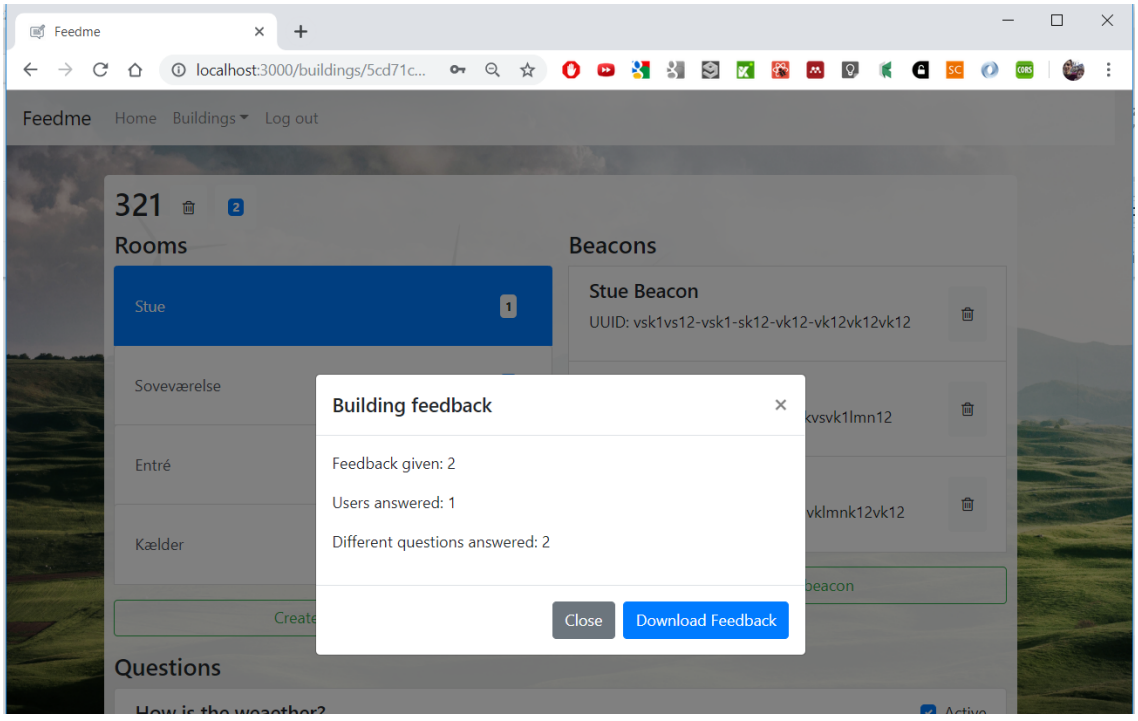


Figure 5.6: Feedback popup page for website

CHAPTER 6

Location Algorithm

This chapter gives a brief overview of the different technologies that can be used to determine a user's indoor location. Afterwards a more in-depth analysis of three specific BLE based location algorithms are presented. Finally, in the last part of the chapter, the reasoning behind picking the algorithm used for this project is presented. The strength and weaknesses of the algorithm are presented alongside the core features of the algorithm.

6.1 Technologies

The three technologies which were introduced in the Chapter 2 (Existing Technologies) were GPS, BLE and Wi-Fi. However, they are not the only technologies which can be used. Other common technologies which could be used are Google Location Services and Zigbee (alternative to BLE Beacons) for example, but they are not discussed in more details in this report.

The different algorithms which can be used to determine indoor positioning, can actually in most cases be used for more than one technology. However, this is not the case for GPS-technology which can only use the trilateration algorithm. When looking at Wi-Fi and BLE, the technologies only depend on the RSSI value. How the different algorithms handle the RSSI value is the only difference. Actually, there are very clear similarities between Wi-Fi and BLE. With BLE the identifier of the signal is called the `UUID` and for Wi-Fi it is called the `SSID`. For BLE the main broadcasting device are beacons and for Wi-Fi it is the access points (`AP`). Some of the differences technology-wise between BLE and Wi-Fi are the accessibility, the signal strength, maintenance of the system. The algorithms used with both Wi-Fi and BLE beacons are often the same, as discussed in Chapter 2 (Existing Technologies). Beacons are also more flexible and can be specifically used for the purpose of indoor localization. Using Wi-Fi only for this purpose is not feasible, because setting up access points is way more costly than setting up beacons. However, using Wi-Fi is a good alternative if you already have Wi-Fi covering the building. This means that the BLE based algorithms presented in the next section, is also applicable to Wi-Fi based systems. Wi-Fi can provide a much higher throughput of data compared to BLE, but this is not needed for this project. As mentioned earlier, in this project BLE technology is used, and it is mainly because of the precision and accessibility that can be obtained. For example in basements it is very normal not to have Wi-Fi installed, or at least the signal would be very weak.

6.2 BLE Based Location Algorithms

For estimating the user's location even just within the domain of BLE there are a lot of different algorithms. The reason for this will be explained further in this section, followed up by a more in detail look at some of the most popular methods for indoor location estimation utilizing BLE technology.

Although BLE technology provides a clear advantage for indoor localization in comparison to other technologies such as GPS, it also has its disadvantages. One of the major disadvantages is the fluctuation of the signal sent out by the BLE beacon. The signal can vary a lot depending on of different factors, which are often uncontrollable. One of these factors are obstacles, such as humans or walls. Another one, is the influence of other signals in similar ranges, such as phones communicating with other Bluetooth

devices or Wi-Fi signals. As discussed in Chapter 3 (Related Work), there are a lot of different methods for estimating the user's location using BLE technology. Furthermore, the subject of indoor localization based on BLE technology is being researched heavily and currently a lot of new papers have been published[1]. One of the reasons for this, is the instability of the signals of BLE beacons. There is simply no single method of localization estimation that works well for all types of environments. In their paper[18] Falzi Subhan et al. present and review different popular methods for indoor localization using BLE technology.

A popular method, and probably the most known localization estimation technique is the trilateration approach which is used for GPS-technology as discussed in Chapter 2 (Existing Technologies). This approach can also be implemented with BLE beacons and the method is very similar to how GPS-technology works. At least three beacons need to be installed in the system. Each beacon sends out a signal that includes a unique identifier for the beacon. When the signal is received by the device (a smartphone for example) the signal strength can be calculated. Instead of calculating the time of arrival, as is the case with GPS-technology, the signal strength value (RSSI value) is used to calculate the distance from the beacon to the device receiving the signal. With three beacons, only one possible location for the device should be available when looking at the intersection of the circles around each beacon, created by their distances to the device. Fig. 3.1 in Chapter 3 (Related Work) illustrates how trilateration can be used to determine a location. In theory the method works well but there are some practical issues which makes the method heavily criticized in a lot of BLE location estimation articles and projects[20, 19]. The main point of the critique is the fluctuation of the signal strength when environments with a lot of obstacles are present. This makes for a very uncertain conversion from signal strength to measurable distance. Another disadvantage is, that when you convert RSSI to distance, a map over the environment needs to be incorporated into the system. This is because when working with real distances (not signal strength, which are relative), you need to know the location of each of the beacons and you would need the exact location of all rooms including where their boundaries (usually walls) are. However, the method has proven to work well in more open, larger environment, which for example is the case for F. Subhan et al. who use trilateration together with a gradient filter to estimate a device's location in a large open room of 120 square meters with no human interference[18].

Another very popular method of location estimation based on BLE technology, also mentioned earlier in the report, is simply determining the nearest beacon and then using the location of that beacon as the approximate location of the device. Here, you compare the RSSI values received from different beacons, and optionally different filters, such as an average filter, can be applied to account for instability of the beacon signal. By applying an average filter, you would not only compare single RSSI values to each other to find the maximum value, but instead you would use the average of a number of measurements for the comparison. This approach makes some of the disadvantages of the trilateration approach less substantial, such as the instability of the beacon signals. With this approach, there is no need to convert the RSSI value to an actual distance, which makes the difference in environment less significant for the precision of the result and it also prevents the need of a map of the environment. An obvious disadvantage however, is the high cost of the system, if a high precision in the location estimation is needed. To achieve high precision a large number of beacons need to be deployed, which can be expensive both because of the actual cost of the beacon, but more importantly because it makes for a longer setup process and requires more maintenance.

The third option which will be discussed is a machine learning algorithm called the *k nearest neighbors* algorithm (KNN), which was also introduces in Chapter 3 (Related Work). As discussed, this algorithm is categorized as a fingerprinting algorithm. This category of algorithms all try to match a new unclassified data set with existing data by comparing the distance between the unknown data set and each of the existing data sets. In the case of the KNN algorithm, the new data set would be the strength of the signal broadcast by one or more beacons, and the existing data would be different

[1] The website Dimensions provide a statistics over how many articles containing the keyword *BLE indoor localization* have been published in recent years: `https://app.dimensions.ai/analytics/publication/viz/overview-publications?search_text=BLE%20indoor%20localization&search_type=kws&search_field=full_search`

locations, where the signal strength of each beacon was recorded. As discussed, the approach requires an offline phase, where the signal strength of different beacons are linked to a certain location. This can be a time consuming task if high precision is needed. With this approach Y. Pu et al. achieved a precision of 1.8 meters in a class room of approximately 78 square meters[14]. This result however, was achieved by having 6 beacons inside the class room, recording a total of 240 samples of signal strength distributed into 12 different locations around the room. Most beacons can be configured to send out signals 5 times each second, but still, this setup can be very time consuming.

6.3 Choice of Location Estimation Algorithm

For this project, two different kinds of methods for estimating the user's location have been implemented; a variation of the nearest beacon algorithm as well as a variation of the KNN algorithm. Users can choose between the two methods depending on the amount of resources they have available and especially on what kind of environment the system is used in. The nearest beacon algorithm was chosen, as it provides an easy option for users who want to implement the feedback system relatively fast, have resources to buy a relatively large amount of beacons compared to the building size, and do not have high requirements to the precision of the location estimation. This environment could be smaller buildings with few rooms. Here, a beacon could simply be placed in each room, and the location estimation would usually work well. Another important reason for choosing this algorithm is that it is a relatively fast to implement, development-wise, as the logic involved for the location estimation is relatively limited – at the same time, implementing the algorithm still gives a lot of knowledge as to how beacons interact with other devices, and it provides a good code structure to build a more advanced location estimation algorithms on.

The second location estimation algorithm chosen is a variation of the KNN-algorithm. The reason for choosing a fingerprint algorithm as opposed to more traditional location estimation algorithms such as trilateration, is because of the criticism, that the trilateral algorithm has received, in similar projects, for being very imprecise in environments with a lot of obstacles. Not only does the environment this system is intended to be used in, namely in schools or work environments, contain a lot of humans, who act as obstacles – the beacons are actually intended to estimate users' location through walls and doors. It was considered very likely, that a conversion of signal strength values through walls would result in a very imprecise distance, which ultimately would make trilateration very difficult. With the KNN algorithm this problem is avoided, as the algorithm actually takes advantage of the walls surrounding each room (which will be discussed in greater detail later in this section). Another advantage of the KNN algorithm is, that it avoids the conversion from signal strength values to measurable distance. Although the conversion does not require much computational power, having to work with distances sets further requirements to the system. One of those is, that a digital map of the building is needed with markings of where each room begins and ends. This either means, that building admins need to have these available for the system to read, or it means that the system should provide a tool to create a digital map over the building. Either option is time consuming both development-wise and for the end-user. Algorithms based on converting signal strength to distance also have the disadvantage of being very reliant on the environment. The popular conversion formula presented in this report takes account for the environment by including a constant, which is the signal strength at 1 meter. This constant needs to be estimated for each beacon in the system. Furthermore, it is probably safe to assume that this value will change, depending on power level, indoor climate changes and other signals etc. This makes for a less precise and less adaptable algorithm. These arguments and considerations are some of the reasons for choosing the two location estimation techniques used in this system.

6.4 Nearest Beacon Algorithm

The first location estimation algorithm implemented in the system is a variation of the *Nearest beacon algorithm*. The logic of estimating the location is distributed to both the back end server and to the mobile application. Each BLE beacon in the system is placed in a room registered in the system. The algorithm basically calculates which beacon has the strongest RSSI value (thus usually being the beacon closest to the device), where the mobile application is running, and estimates the user to be in the same room as the room which the beacon is located in. This is done by having a map on the server where each beacon is linked to a room. When a new beacon is added, building administrators are required to indicate in which room the beacon is located. When the unauthorized user wants to get their location estimated (e.g. when the user gives feedback), the device receives the beacon-room map by the server and looks up which room the beacon belongs to. This room is then presented to the user. To make the location estimation more stable, an average filter is used, to estimate the nearest beacon. When the user wants to know their location, the device listens for every beacon signal it can find. All signals are stored on the device and once every second an average of the last 5 signals received by each beacon is calculated, and then the beacon with the highest average RSSI value, is determined as the closest beacon. Of course, the closest beacon might not always be the beacon with the highest RSSI value but it usually is, which is why the algorithm is called the Nearest Beacon algorithm.

6.5 KNN Algorithm

The second option for location estimation existing in the system, is a variation of the KNN algorithm. First, a brief overview of how the algorithm works will be presented, followed by a more in depth look at how the algorithm is implemented in the context of the Feedme system.

The algorithm is used for classification and in Fig. 6.1 the way KNN algorithm works is illustrated. KNN is a machine learning algorithm which has an offline and an online phase. In the offline phase, training data is collected and classified. In Fig. 6.1, this is illustrated by the blue and red points classified as class A and class B respectively. When a new unclassified point appears, the algorithm calculates the distance to all the surrounding points and determines the k closest, where k is an arbitrary number, and where the distance measure can be customarily chosen. Both k and the distance measure is chosen to whatever best fits the context, and can be estimated through statistical analysis or experiments. In the figure, the points are located in a two-dimensional space, but the algorithm works in any dimensional space. The important thing is, that the distance between the points can be determined. When the algorithm has found the k-closest points, the class of the new point is determined to whichever class is mostly present among the k nearest points. A new point is illustrated in the figure by a square point, and in this case, when $k = 3$, the class of the point would be determined to be of class B, as two out of the three closest points are of this class.

In the context of Feedme, the KNN algorithm is used to determine in which room the user is located. The way the algorithm is implemented however, is substantially different from other projects previously mentioned in this report. Figure 6.2 gives an overview of the specific implementation in the Feedme system. As opposed to previously mentioned systems, the system is only interested in the room of the user, not their exact location. In the offline phase, each room, from where feedback can be given, is scanned with the mobile application by the user. When scanning the room the user indicates in which room the user is located and then proceeds to walk around along the walls of the room with the mobile application. While walking around, the application records the signal strength values of all of the beacons around it. Similar to the nearest beacon algorithm, an average filter is applied, to make the signal values more stable. Every other second the application takes the average of all of the last signal values from each beacon and creates a vector. Each number in the vector represents the (average) signal strength value from each beacon. If the application recorded signals from 5 different beacons, the vector would be 5-dimensional. When the scanning process is done, the application sends all of the recorded average signals, stored in vectors, to the back end server, which saves all of the vectors and links them

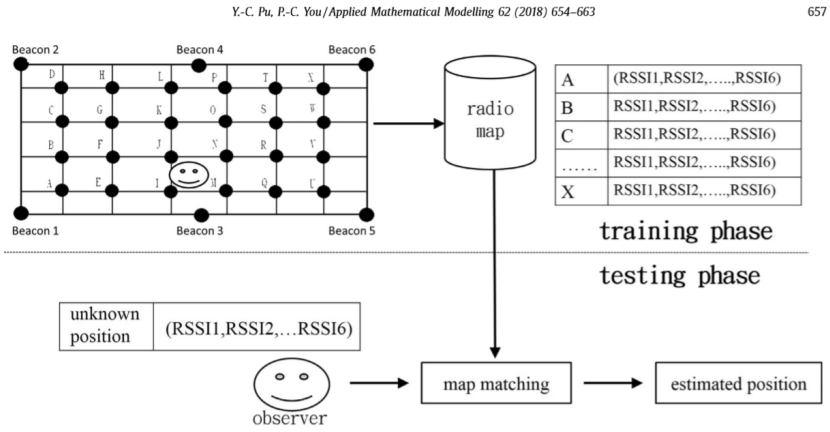


Figure 6.1: KNN Algorithm

to a specific class, i.e. the room, which the user was scanning. In Fig. 6.2, these vectors are illustrated by points, with different colors, each color representing a different room. These points are in this case 3-dimensional as 3 beacons send out signals in this building. When a user wants to get their room location estimated, this process is similar to classifying a new point in the KNN algorithm. The application determines the signal strength of all nearby beacons, and estimates the average signal strength value of each beacon in a period of time. When the estimation is sufficiently stable, the application wraps these values in a vector and sends them to the server, requesting to get their room location estimated. The server then calculates the distance to each other vector stored in the database, and finds the k nearest vectors. The distance measure here, is the euclidean distance between the two vectors. If the dimension does not match, e.g. a vector on the server might have registered signals from more beacons than the vector from the application, then the shorter vector is extended to match the compared vector and the lowest possible signal strength value is put instead of the missing values. When the closest vectors have been found, the new class of the new unknown vector is determined as the most frequently present class among the nearest vectors. If it is a tie, k is decreased by one, and the algorithm is run again for all the previously determined nearest vectors. In Fig. 6.2, the new unclassified point, indicated by the square point, would find only points classified as room C as its nearest neighbors and therefore this user would be estimated to be in room C.

To extend the algorithm further, when the user has been estimated to be located in a specific room, the application could ask the user to confirm the proposed room. If the user does so, the previously unclassified vector could then be saved to the database and used as training data for future location estimation. This makes the algorithm a lot more adaptable for environment changes or beacon signal strength variations due to battery level. The amount of vectors for each room could be regulated depending on different factors, such as the amount of time since the scanning was done, or the amount of vectors currently linked to the room. If the scanning happened a long time ago, the system could adjust, and begin to ask more people, who gave feedback, to confirm the room that they are in, which would update the vectors linked to the room and making the location estimation more precise. It could also automatically delete vectors that are more than 1 month old if a sufficient amount of vectors are already present in that room. This would make the algorithm very flexible to changes to the environment.

Another interesting thing is, that the algorithm actually takes advantage of some of the obstacles between the beacon and the device e.g. the walls surround each room. It can be presumed that the signal strength is substantially different on each side of the wall, which should make the distance between two points located in two different rooms larger. When the KNN algorithm is run, it will much more likely choose vectors in the same room as these vectors appear much closer than vectors located on the other

side of a wall, because the wall is essentially blocking the signals. This would ultimately result in a more consistent and precise room location estimation.

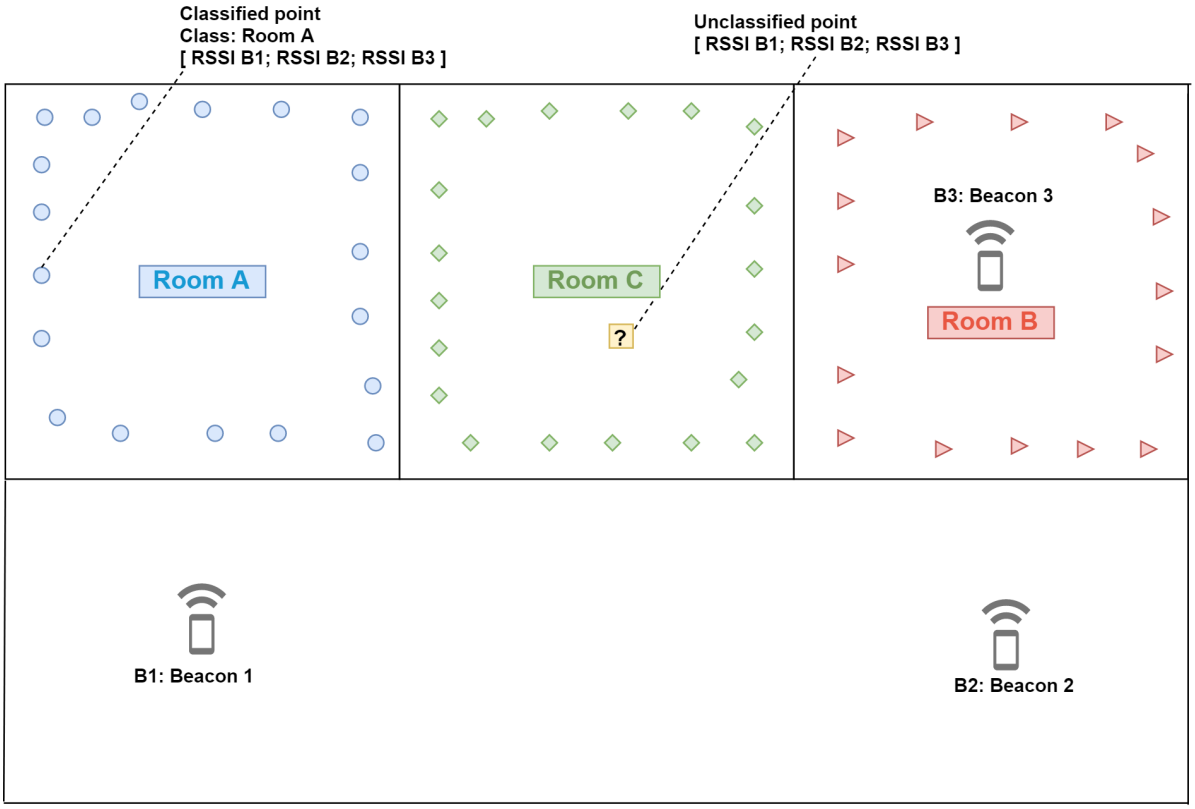


Figure 6.2: KNN Algorithm in the Feedme system

CHAPTER 7

Design

In this chapter, the design and architecture of the system are presented. First, an overview of the different components and how they communicate with each other is given. Afterwards an in-depth explanation of each component is presented.

7.1 System Overview

An overview of the system and its relations is shown in Fig. 7.1. There are three main components: the mobile application, the website and the back end server. Two other components; a beacon and the system database are also a part of the system though less important in this context. The internal structure of each component only shows the most important parts to get an overview of the system. The server provides an API which both the website and mobile application. When the mobile application and the website want to communicate with the server, for example to query information from the database, they communicate with the router component.

In the case of the mobile application, it has a dedicated component called FeedmeNetworkService, which handles all the networking calls. Based on what is needed, it creates GET and POST requests, and if the server gives a successful response it will pass the data to the rest of the system. An important system design choice has been to place the LocationEstimator component in the mobile application. The main task of the component is to fetch the signals sent by the beacons. When it gets the signals, it will do some simple computations to make sure the signals are well-formed. Well-formed means that it will listen for a few seconds and calculate the average of the received signals and filter out odd values, and afterwards pass the data to FeedmeNetworkService. FeedmeNetworkService then makes sure the data is sent to the web server.

Like the mobile application, the website also has a NetworkService component to be responsible for the communication with the server as shown in Fig. 7.1. The other internal components of the website follows a design pattern called *Flux* which will be discussed in greater detail in later in this Chapter. Authorized users interact with the website through its View components which is only responsible for the graphical user interface of the website.

The main internal components of the back end server can also be seen in Fig. 7.1. When either the mobile application or the website want to access or modify data stored in the database, the Router component will pass the identification data of the request to the *Authentication* component. This component will verify that the user should have access to modify or access the requested data and it will also be responsible for handling the user session after the user has logged in. When the request has been validated it will be parsed on to the controller component, which is responsible for getting the necessary data from the database. In case the user wants to post new data to the database the controller will parse this data to the relevant model components which will validate the data before it is sent to the database.

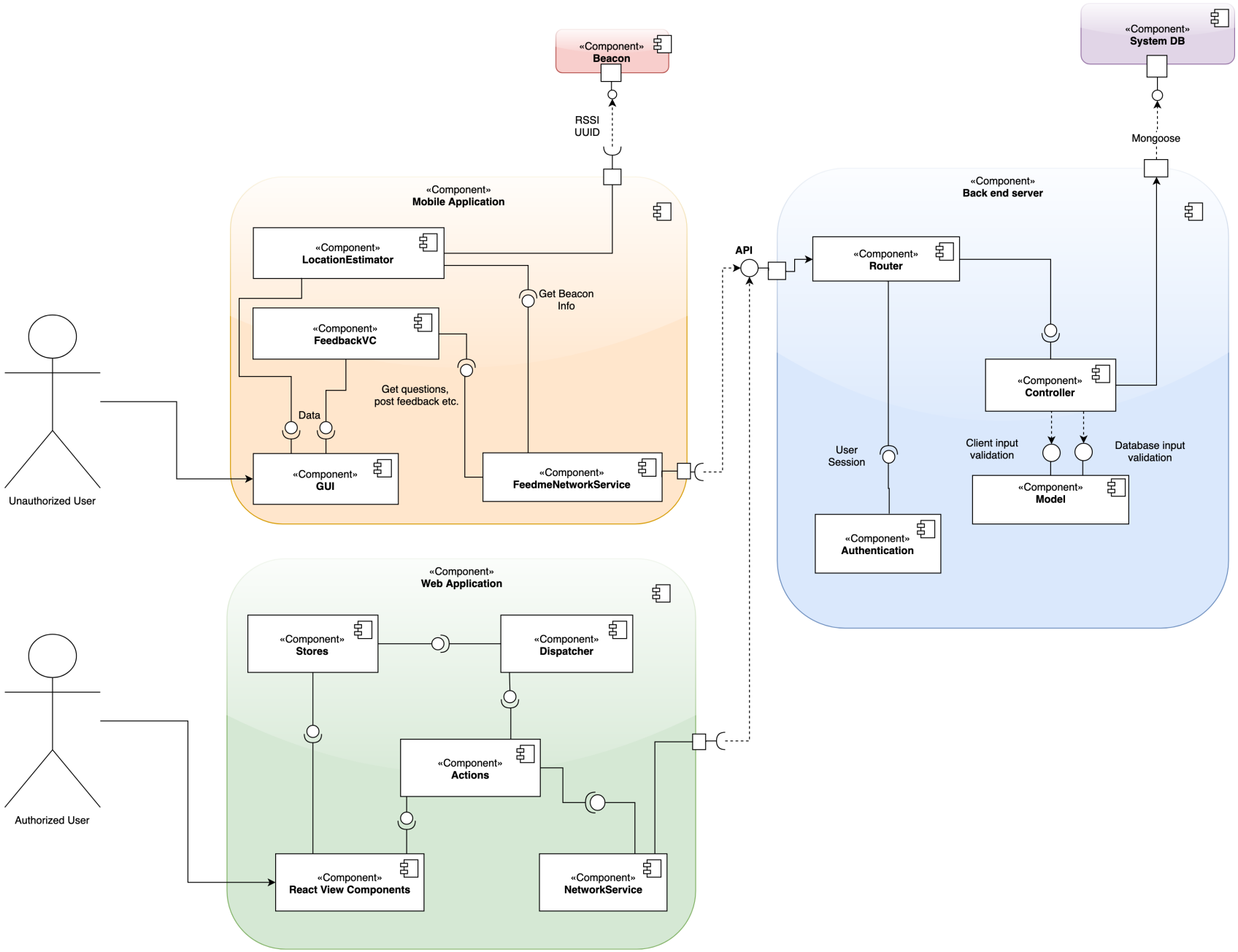


Figure 7.1: Component Diagram

7.2 Back End Architecture

All relevant information to the system is stored in a central place in a database on a virtual machine – the so called back end of the system. Instead of the mobile application and the website communicating directly with the database, web services have been created to facilitate and simplify this communication with the database. The back end server's responsibility is to manage the information stored in the database and make it easily accessible to all clients. This is done by providing an application programming interface (API) that all clients communicate through. The API enables clients to access and modify the information of the system through HTTP requests to the server. To further simplify this communication, the API provided by the server follows the architectural style REST. This means, that clients communicate with the server through stateless, GET, PUT, POST and DELETE requests. The server then handles these requests by either getting, modifying or posting information to/from the database and then sends an HTTP response back to the user. This response has a HTTP status code, informing the client how the handling of the request resulted and optionally a JSON body. This enables the clients utilizing the API to do CRUD operations (create, read, update and delete) on objects stored in the database which includes users, buildings, rooms, beacons, questions, answers and feedback.

The overall architecture of the back end server follows a classical node/express pattern, where the server is divided into different parts depending of their concerns. The server consist of three parts; a router, a model and a controller, each having its own package. In the router-package all the different routes for the API are listed, spread out into different modules/files depending on the kind of object they are concerning. This means all the requests from the clients concerning buildings are put in the buildings module in the router-package and similarly with the other routes. These route-modules are simply responsible for matching the requested URL to a specific controller. E.g. a GET request to the URL *"/api/buildings"* will be matched in the buildings module in the router-package to a controller method named *getBuildings*. Thus, the controller is responsible for handling each request. The way each request is handled depends a lot on the given request, but generally the controller validates the client's information, accesses or modifies the database and then finally sends a response to the client with an appropriate status code informing the client of how to request was handled. Each object stored in the database is represented by a model class on the server. It was chosen to use a a non-relational database for this project and the validation is therefore not incorporated into the the database. Instead, these model objects are responsible for validating both the data received by the client and the data sent to the database. This ensures that data is of good quality and is consistent. The reason for choosing a non-relational database will be discussed in greater detail in the next section.

7.2.1 Node, Express and MongoDB

For this project, a Node server has been implemented for the back end of the system. Node is a runtime environment allowing JavaScript code to be run outside of the browser. This allows for a back end server written in JavaScript. Node also has some built in libraries helping with the development of web services. This includes the HTTP module which is used to easily handle client request. Express is a framework that builds on top of the HTTP module, further simplifying and enabling the creation of a web server through Node. The framework provides a set of features to develop web servers like the *req* and *res* objects, that represents a client's request and the server's response. Node and express were chosen as the tools for building the server as opposed to more traditional tools/languages like Jersey for several reasons. Most importantly, it allows the server to be created in the same language as the website, namely JavaScript. When only JSON (JavaScript Object Notation) needs to be communicated from and to the server, which is the case in this project, being able to write JavaScript code makes for less conversions as JavaScript objects can easily be converted to JSON. This also works well when the database is non-relational, as it was chosen for this project, because the JSON, coming from the clients can be almost directly stored in the database without any conversion or manipulation of data. Node also has a lot of good tools to work with non-relational databases as opposed to PHP which usually works better with relational databases.

As it was mentioned, it was chosen to use a non-relational database for this project. The main reason being the high compatibility with Node and express. For non-relational databases, complex objects can be stored exactly as they are represented in the model of the application. For this project, several complex objects have been identified such as a question holding an array of answer options, or a building having multiple administrators. These complex objects would not be easy to store in an SQL database as multiple schemes would be needed. Another advantage is, that there are no required object schemes in a non-relation databases which means the structure of the data stored in the database can be easily modified. This is an advantage with complex objects which will often change structure during the development process. However, this can also turn out to be a disadvantage as the database itself has no incorporated validation which would ensure consistency and avoid redundancy. Potentially malformed data could be stored in the database and items which are referenced by other items could be deleted without any errors. Furthermore, as complex objects can be stored it is a lot harder to avoid redundancy. The problem with validation and consistency is resolved by using a library called *Mongoose* to validate all data before it is sent to the database. With the library an SQL-like scheme is set up to ensure that each property fulfills certain criteria e.g. a building should always have a name, a user-role is an integer between 0 and 3 etc. This helps to ensure the quality of the data put into the database and with the library, helpful error messages are also generated if the validation of the objects fails. Usage of the library will be discussed in greater detail in Chapter 8 (Implementation). To avoid redundancy, some objects are stored in their own *collection*. This would include objects like *answers* which originally belongs to the *question* object. By having it in its own collection multiple objects can reference it such as the *question* object as well as the *feedback* object. Avoiding redundancy is simply a question of having a good structure of the database even though it is non-relational.

7.2.2 Security

As previously discussed it was decided to require users, that want to perform administrative tasks to the system, to register with an email and a password. A separate module called *Auth* takes care of the authentication of users. For the registration process the user has to create a secure password, meaning a password with a sufficient length and sufficient types of different characters. When the user registers with a secure password, the password is not stored in plain text in the database, but instead hashed with the *bcrypt* hashing function, using a random salt generated at run-time. The *bcrypt* hashing function builds on the symmetric-key block cipher *Blowfish*, and can be run a certain number of rounds depending on the desired security level. With this, whenever a user logs in with an email and a password, the server finds the hashed password in the database for the user with the given email. Through the *Auth* module, the server then uses the salt from the first part of the hash value from the database to hash the sent password and then compares the two, to see if they match. If there is a match, the user is verified and is then allowed to perform administrative tasks in accordance with their role in the system.

As the REST API provided by the server is stateless, the server needs to confirm the identity of the user in each request the user sends. To prevent the user from sending their credentials in each and every request, upon log in or registration, a token is sent back to the user, which is then required to be sent by the user in all future requests to confirm the user's identity. The token is generated with the library *jsonwebtoken* which uses the symmetric-key algorithm HS256 (HMAC with SHA256), to sign a given piece of information. In the case of this project, the piece of information is the user's ID along with their role. This information is signed with a private key stored in a separate file on the server. The signature along with a header indicating the algorithm used to create the signature and the user information are all base64URL encoded to form a string (the JSON web token), that is sent back to the user in the header of the response. Clients that want to authenticate themselves with that user ID and that role must then send a valid token in all future requests. When a new request from a user is received, the server verifies that the token is valid by checking the signature-part of the token with the private key stored on the server. If the token is valid it can be decoded to a user object and the user id and role is immediately given. This enables the server to quickly identify the user without having to access the database.

7.3 Mobile Application Architecture

In this section, the architecture and design of the mobile application are presented. The mobile application is the component of the system which the unauthorized users will interact with to provide feedback to the system. Since the unauthorized users is by far the largest user base, it means that most users will interact with the system through the mobile application. The mobile application is in some situations used by the authorized users as well.

For iOS development there are various design patterns to properly structure the code. Some of the most popular patterns include MVVM (Model View ViewModel), MVC (Model View Controller) and Viper. In general there are a few rules on when to use what, but MVC is the most straightforward and simple design pattern to follow for an application and since the app itself is not too complex, it was chosen to follow the MVC design pattern. It is assumed that the reader knows of MVC and therefore there an explanation is omitted[1].

In Fig. 7.2 a class diagram of the mobile application is shown and it gives an in-depth look on how the system is designed. The class diagram is a UML class diagram. A lot of the elements might look familiar, such as Room, Question, Building etc., and this is due to the convenience of having a model structure that corresponds to the back end model, such that when a networking call is made, the JSON can be parsed and easily be stored in the application. These elements are very simple, which means they don not contain any logic, and therefore implemented as structs which can be parsed around the application, to for example get data and show it to the user, or modify the objects and send them back to the server.

The most important part of the application is found in the `LocationEstimator` class and the `FeedmeNetworkService` class. The goal of the `LocationEstimator` is to get the signals from beacons and send this to the back end server to determine a user's location. It does so by listening to the BLE signals received from nearby beacons and makes sure that the beacons found are also a part of the Feedme system. This is done through the `FeedmeNetworkService` class, which gets the beacons that are a part of system. When it receives the beacons, it matches the beacons from the system with the beacon signals received, by looking at their UUID. If a received signal comes from a beacon which has the same UUID as one of the beacons from the server, it is a match. As shown in the diagram, `LocationEstimator` also calls the `<<FoundNewRoomProtocol>>` protocol. A protocol in Swift is similar to an interface in Java. This means that when `LocationEstimator` receives a user's location from the web server, it notifies the objects that conform to the protocol (`FeedbackVC` and `DataVC`), and they will automatically receive the information about the user's location.

The `FeedbackVC` and `DataVC` are the controllers that controls the view of when the user wants to give feedback and when the user wants to see a graphical representation of the feedback already given. When the user opens the application, it is presented with the feedback screen and it has an instance of the `LocationEstimator` class. This means that when `LocationEstimator` gets the location of the user, the room name is shown on the feedback screen and the user's feedback is associated with that room. This also means that when the location of the user is determined, it can get the questions associated with that room from the server and present them to the user.

The last two controllers that have not been mentioned yet are the `DiagramVC` and `RoomChooserVC`. They represent the pop-up windows shown in Figure 7.3. It is worth noting that the `RoomChooserVC` calls the second protocol in mobile application, the `<<ManuallyChangedRoomDelegate>>` protocol. As seen in the diagram, that means `DataVC` now conforms to two protocols. This is because when the user navigates to the data screen, it is presented by the feedback given in the room in which the user is located at the moment. But if the user wants to see feedback from a different room, it can be done by changing the room as seen in Figure 7.3a. When the user has manually changed the room, the `RoomChooserVC` calls the `userChangedRoom` method in the protocol and since the the `DataVC` conforms

[1]Tutorialspoint has a comprehensive guide on the MVC design pattern: `https://www.tutorialspoint.com/mvc_framework/mvc_framework_introduction.htm`

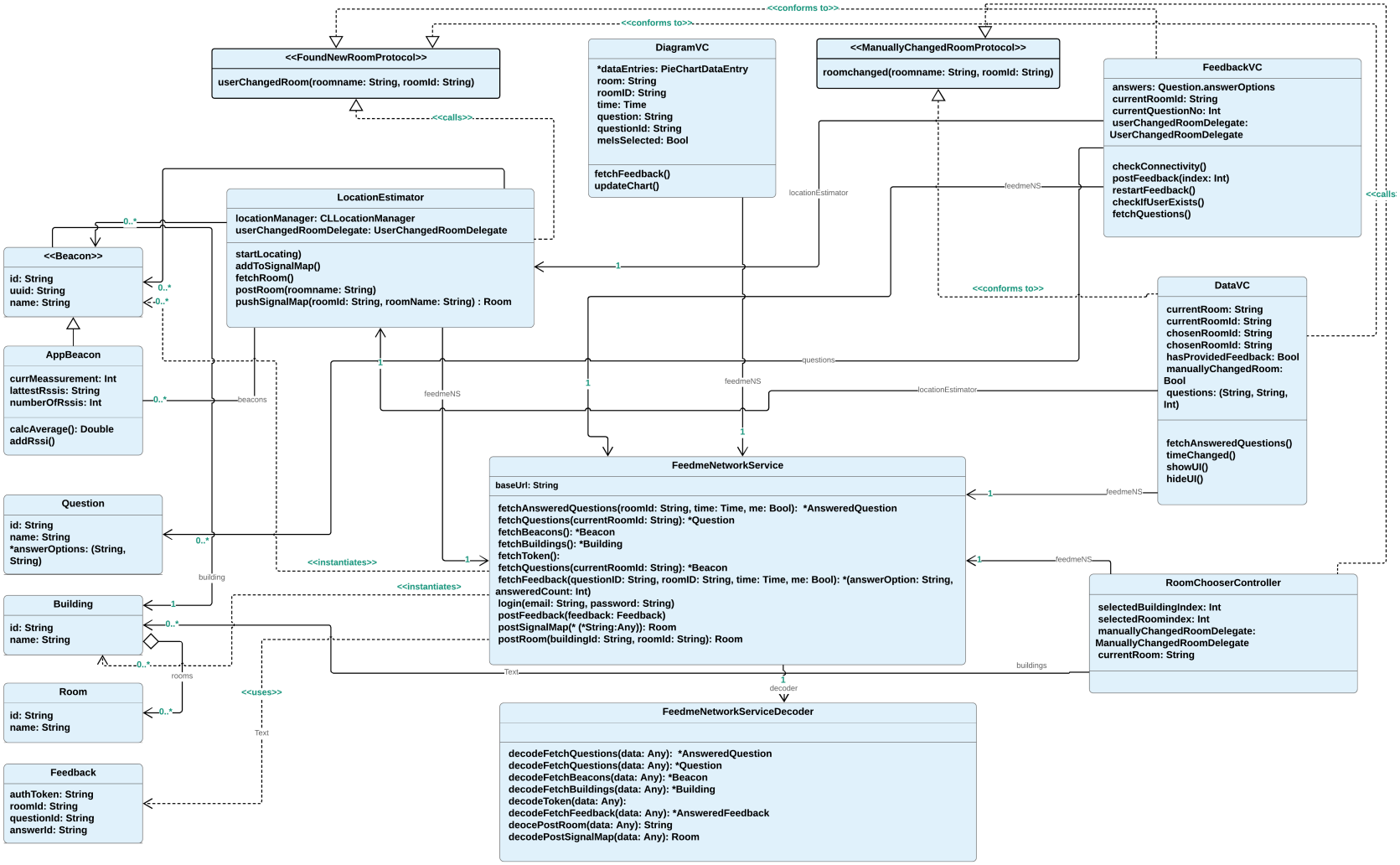
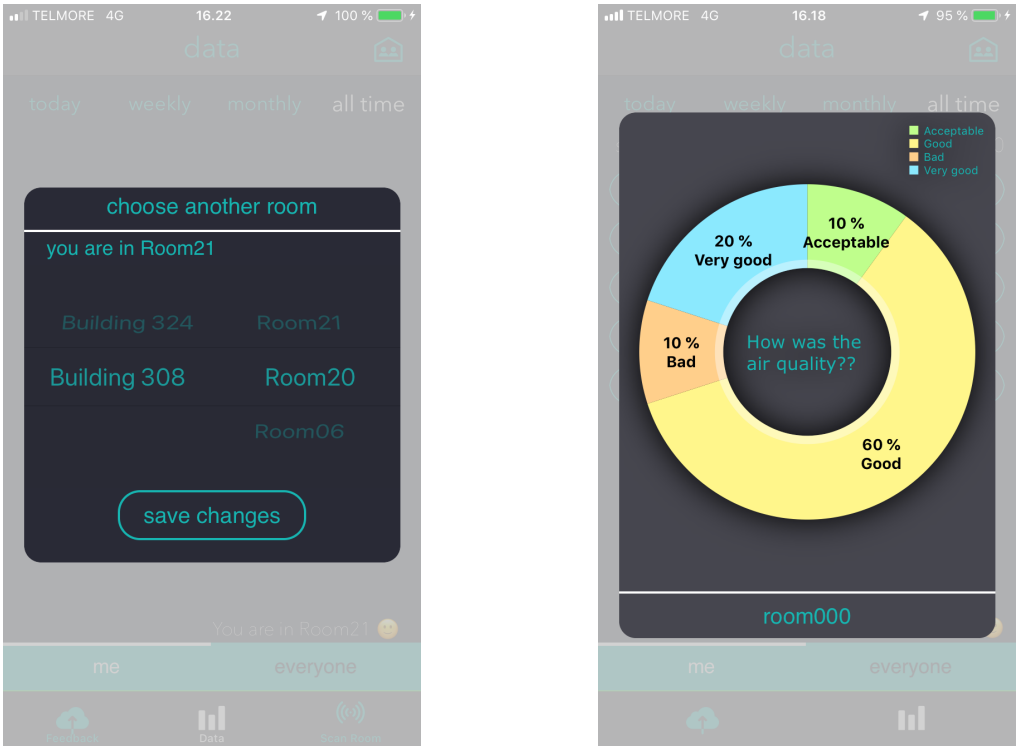


Figure 7.2: UML Class Diagram of Mobile Application

to that protocol, it will show the feedback from the particular chosen room rather than the room in which the user is located.

The last class from the diagram, which has not been discussed yet is the `FeedmeNetworkService` class. `FeedmeNetworkService` is the class which all of the controllers has an instance of. This class is only responsible for the networking requests instead of having the controller classes make the networking requests. When a controller for example wants to get questions from a particular room, it has an instance of `FeedmeNetworkService`, which handles the URL formatting and construction of the HTTP request. However, the data that the `FeedmeNetworkService` receives is in JSON format. Instead of decoding the data itself, it delegates the job to the `FeedmeNetworkServiceDecoder`, which decodes the JSON. The decoder returns the parsed structs to `FeedmeNetworkService`. Another property which the diagram does not show, is that all the methods that `FeedmeNetworkService` has, do actually return two optional values. If there were no errors, the variable which is shown on the diagram is actually parsed on to the view controllers. However, if there was an error, an error is returned instead of the variable shown on the diagram. The networking requests and decoding of JSON is elaborated more in the implementation section.



(a) The view which RoomChooserController controls (b) The view which DataTabController controls

Figure 7.3: The two popup controllers

7.4 Website Architecture

Both the website and the back end server are written in the same language, namely Javascript, and for a lot of Node servers only dedicated to a website, the web server and the website are written in the same system i.e. a web application with both back end and front end. For this project however, it was chosen to completely separate the two concerns, making the website its own standalone application utilizing the API from a separate standalone system, i.e. the back end server built for this project. This design

was chosen for several reasons, the most important being that it makes for a more decoupled and flexible system. The website is not dependent on the web server for the visuals of its layout. If at any point in time it is decided to use another API to facilitate the visuals of the website, this can be easily done by changing the URL of the API requests, given that the new web server provides the same APIs and vice versa. If a different type of front-end is wanted, this can easily be done by creating a different front-end system, utilizing the rest API from the back end server – even in a completely different environment than the browser. The mobile application is a good example of that. The mobile application utilizes some of the same API request as the website, but it displays the data in a completely different manner to the user. Another advantage is, that this separation also truly separates the concerns of the different parts of the system. The website only has one focus namely providing a visual layer for the system, retrieving data from an API and displaying it to the user in browser. This also makes the focus of the back-end server more narrow, as its job is only to provide a simple API usable by different front end clients such as the website as well as the mobile application.

7.4.1 React

For the website a Javascript library called React is used to create the visual design of the website. React is developed by Facebook and is used a lot for single page applications. It is a good way of structuring the visual design into reusable components. As the website of the Feedme system primarily only has one focus, namely managing the feedback system for a building, including managing the beacons setup in the building, it was chosen to use React as the main framework for the website, organizing and structuring the visuals on the website on a single page.

With React, the website can be divided into different reusable blocks called *components*, each being responsible for their own visual design and state. E.g. a component called *building* would be responsible for displaying the buildings of the system and allowing users to create, modify and delete buildings. This building component could be put into a web page on any given page, as it is not dependent on other elements of that page. Components are then put together and into each other, to form the different parts of a web page just like regular HTML. Like HTML, the web page is built by different components or tags such as *div*, *h1*, *a* etc. React extends these simple components with custom made components. Components can receive input from the outside and parse data to its sub-components. With this architecture a website is built by a tree of components, with each component having its own state, parsing data down the tree to sub-components. Components should not be aware of their context which means each component are highly decoupled from each other. Sub-components can communicate up the tree to the components containing them through events. This way, sub components can notify whatever component containing them when different events occur e.g. interaction with the user such as when a button is pressed or text is written in an input field.

A common design pattern when using React, is to divide components into two categories depending on their responsibility. These categories are called *smart* and *dumb* components aka *UI* and *container* components. The purpose of *dumb* components is to display the data that their given, to the user and trigger events when the user interacts with the displayed data. They do not contain any logic or any functions except for a render function which displays the data in the browser. On the other hand, smart components have a state and contain logic to handle that state. They can decide which sub-components to show, get data from external sources, such as an API, parse data to sub-components etc. They generally contain a lot more logic, than dumb components, but do usually not directly show any data. Instead, they will parse the data to *dumb* components, which are responsible for displaying the data.

Figure 7.4 shows the tree of components, which this website is built up upon. Each box is a component, except for the root node, which is the root HTML. As discussed, the components have been divided into *smart* and *dumb* components and these can be seen in the figure, indicated by respectively a blue and a red color. As the figure shows, generally the higher up the tree the more *smart* components there are, compared to *dumb* components and all leafs are *dumb* components. This is a deliberate design

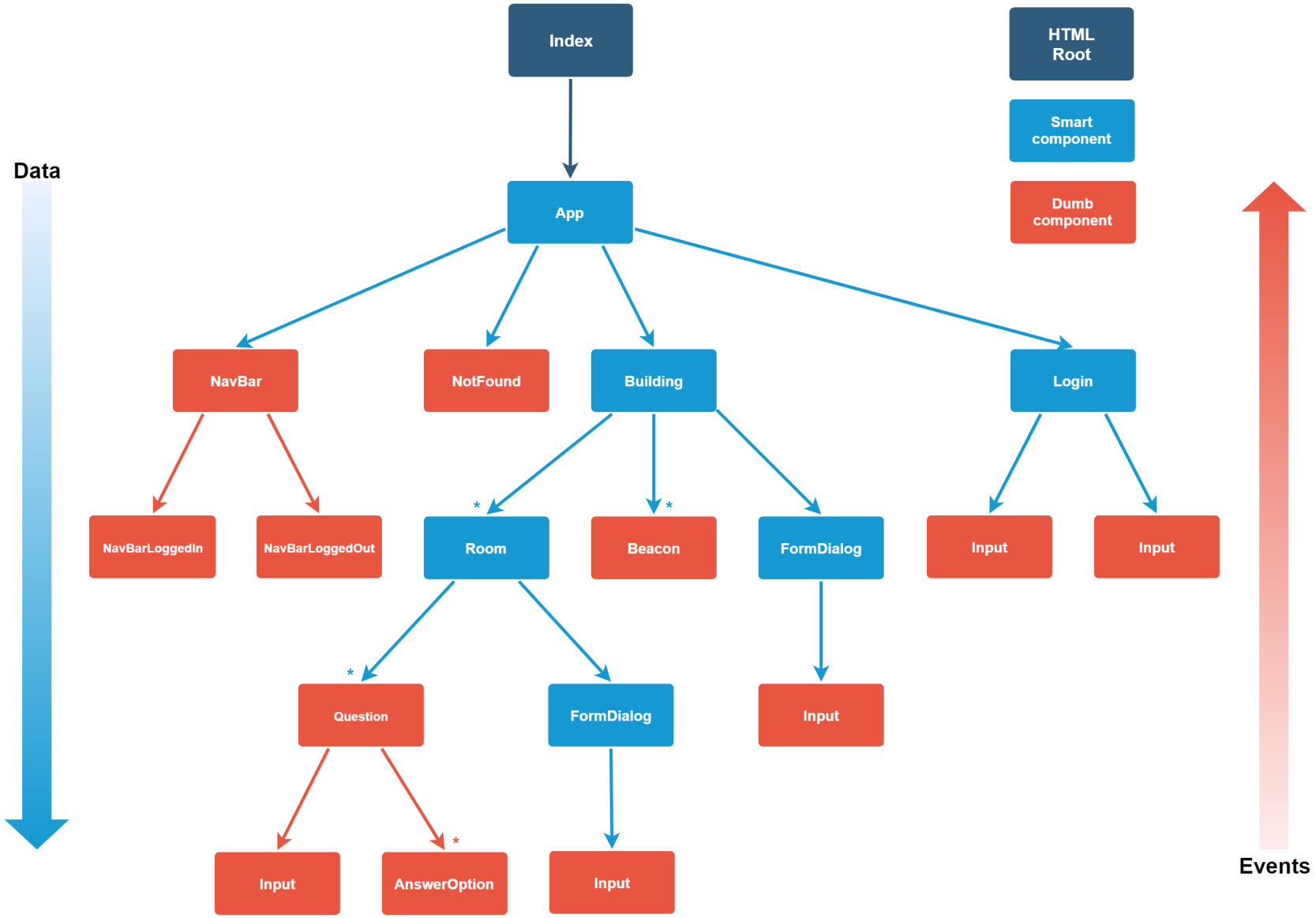


Figure 7.4: Architecture and data flow of react components

which makes the responsibility of each component more clear. All leaf nodes display some data to the user and therefore none of them have any state or logic in them. This makes error handling easier and also makes the code more simple and readable.

Arrows around the tree indicate the data flow in the website. Data is parsed down the tree, eventually reaching a leaf node, which displays the data. When actions occur, e.g. a user types in some text on the web page, events are triggered, which will parse data up the tree to however is listening to these events.

Another thing to notice in the figure, is that some components are used more than one place, e.g. the *Input* component or the *FormDialog*. This is one of the advantages of separating each part of the web page into stand-alone components – it makes code highly reusable in many different environments. The Input component only displays an input field to the user and triggers events when text is changed, and therefore it can be used in a lot in different contexts.

7.4.2 Flux

To properly organize the code and to separate different concerns in the application it was deemed necessary to follow a design pattern. An obvious choice for separating concerns is the MVC pattern, which separates code into models, views and controllers. This pattern however, has been criticised by Facebook developers (who made ReactJS) for not being easily scalable when working with larger applications. The problem is that as the project grows more and more models and views will be dependent of each other because for MVC databinding makes models and views able to talk directly to each other. This, according to Facebook, would result in an application that is very hard to debug. This is illustrated in Fig. 7.5 which shows the architecture of a larger application. In the application a lot of models are dependent on a lot of views, which makes the data flow unclear.

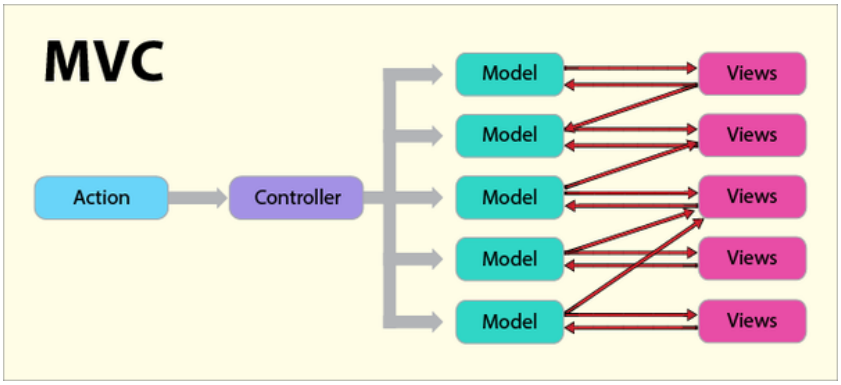


Figure 7.5: MVC architecture in large front-end application

An alternative is a framework/design pattern called Flux which is developed by Facebook specifically for web applications utilizing the React framework. In this design pattern, the different concerns of the application are still separated, e.g views are separated from business logic etc. But in this pattern the data flow is unidirectional and, according to Facebook, this makes it a lot easier to find errors and to debug the program. In Fig. 7.6, which is taken from Facebook's own site, the overall architecture of the Flux design pattern can be seen. Business logic and the state of the application are all put into modules called stores. Views listen to updates in these stores and through event listeners, the UI is updated when the state of the stores is updated. When events occur, that should update the state of the application or perform some business logic, the view components send out actions through action modules. The action modules then retrieves the necessary data (for example by an HTTP request to a server) and through a dispatcher module it sends out actions to all of the stores along with an action type. Stores can then choose to listen to some action types and update the application state accordingly.

Finally, the store modules notify all listeners, that a change to the application state has been made, which makes the views of the application update to reflect the newest changes to the user.

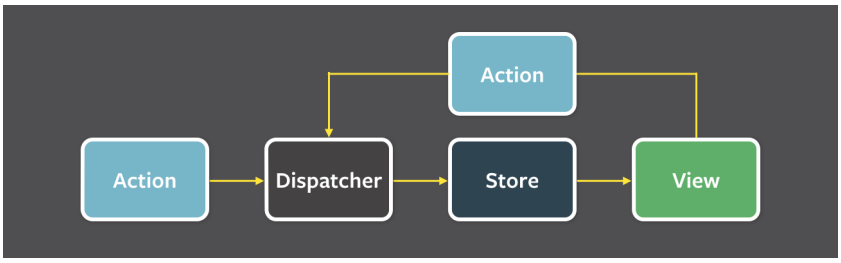


Figure 7.6: Flux architecture

In this project the Flux design pattern has been used to organize the code of the website. Figure 7.7 shows the architecture of the website which follows the discussed Flux design pattern. In green, all of the view components can be seen. Here, only the components that actually contain communication with either actions or stores are shown. In dark-blue all of the stores are shown, which contain the business logic of the application and the application state. At the right-hand side of the figure, all of the action components are shown. In some of the cases these modules communicate with Service-modules, which takes care of the communication with the back end server. The service classes are not listed here for clarity purposes. All action components make changes through the dispatcher component, which parses all of the different actions to all of the stores. The stores then choose to listen to some of the relevant actions for that store. E.g the userActions module could send out a login action, which the userStore would then listen to and properly update the current user's status to being logged in.

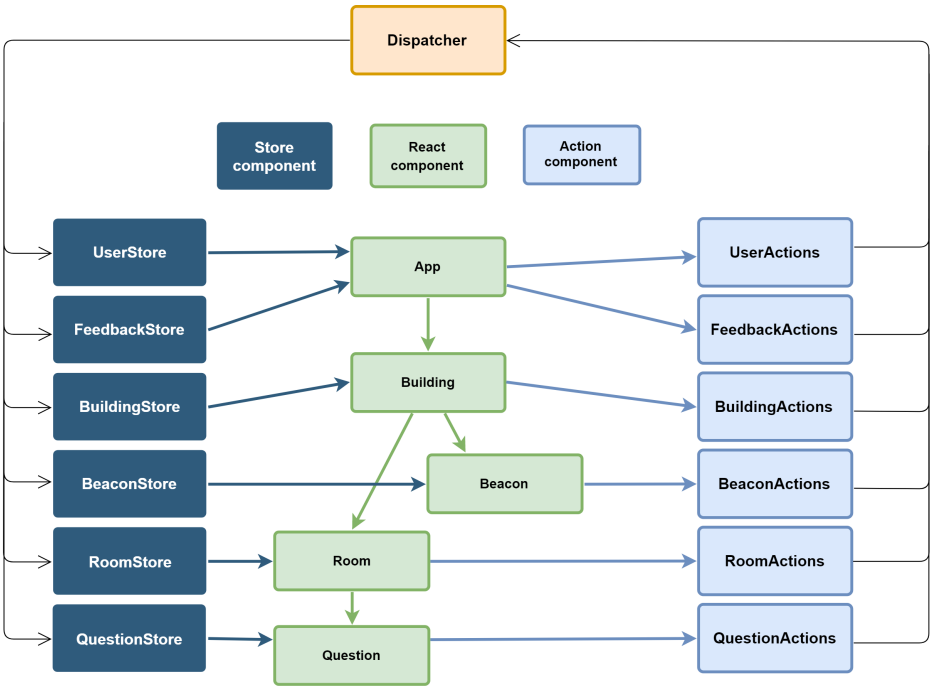


Figure 7.7: website architecture (Flux design pattern)

CHAPTER 8

Implementation

In this chapter, the implementation of the core features are described. This includes the main external libraries that were used, how things can be implemented better in the future and the issues that were encountered throughout the process of developing the software. The chapter contains three sections each containing one of the main components; the mobile application, the website and the back end server. The last section is a small evaluation of the process of implementing the software.

8.1 Mobile Application

The mobile application's main purpose is to fetch the signals from the beacons, communicate with the server and to give the unauthorized user's a pleasant experience giving feedback. In the following two subsections, the focus is on probably the two most important classes of the mobile application; `LocationEstimator` and `FeedmeNetworkService`. In the following sections, it will be discussed in greater detail how the main methods are implemented and what external libraries have been used to develop the system.

8.1.1 Beacon Communication

The class implemented in the application called `LocationEstimator` is actually based on an Apple framework called CoreLocation. The Apple framework implements a protocol, also developed by Apple called iBeacon. The iBeacon protocol provides some nice features, for example if the user comes close to a beacon, the beacon can wake up the phone and send a local notification to that user. However, since the prioritization of the system has been on developing a precise algorithm, there simply has not been enough time to fully utilize these extra features.

When using the CoreLocation framework by Apple, there is one class which controls the whole flow of listening to beacons, which is `CLLocationManager`. To use the `CLLocationManager`, one has to implement its delegate methods. What that mean is, when using the manager, it delegates some methods to the class, that has an instance of the manager (in this case `LocationEstimator`) and those methods should be implemented. The first method which has to be implemented is a method that will ask the user for permission to use CoreLocation. This permission is asked when starting the application for the first time. If that permission is declined, CoreLocation will not work and the application will terminate. If the permission is accepted, the application can start listening to nearby BLE signals. This is done by calling the method `startRangingBeacons(in: region)`, where a region is essentially a tuple containing a beacon `UUID` and an identifier. For all the beacons in the system, the `startRangingBeacons(in: region)` will be called for each of them, and that method starts the process of listening to nearby beacons. If a received BLE signals matches the `UUID` of one of the registered beacons in the system, that particular beacon is stored in a dictionary with its key being the `UUID` of the beacon and the value being the average RSSI value of the latest measurements.

The `LocationEstimator` class also needs to know when to send the averaged signals to the back end server. When you are an authorized user and you are setting up a room, you are sending the signals to the server alongside a room name of choice and a building id. The class will gather as many signals and store it in the dictionary until the authorized user press stop scan (See Fig. 5.2 in Chapter

5 (Software Handbook)) and it will then be sent to the server. However, if you are an unauthorized user, instead of sending a room ID, the mobile application sends a building ID and waits for the server to estimate and return the user's room location. To control how often the data is sent to the server, a timer is instantiated which controls when to send the data to the server. Since a lot of the different view controllers need to use `LocationEstimator` and the ranging (which is equivalent to listen to nearby beacons) of beacons should not be instantiated more than once, the class is instantiated in the file called Appdelegate, which is an initialization class, that is running throughout the lifetime of the app. This means that instead of initializing a different instance of `LocationEstimator` in each view controller, there is a reference to the `LocationEstimator` created upon launching the application. By initializing it in the Appdelegate, you also ensure that the object will never be deallocated, as long as the application is running.

8.1.2 Network Service

Besides establishing connection and listening to beacons, the other main job of the mobile application is to communicate with the server. This job has been delegated to the class called `FeedmeNetworkService`. `FeedmeNetworkService` handles all the networking requests of the application. The network service contains as little logic as possible, because testing the class is difficult, due to the use of an external networking library. This means that there is a dedicated class, `FeedmeNetworkServiceDecoder`, which only purpose is to decode the received data from the web server. By having the decoder separated from the actual networking service class, unit tests can easily be created for the decoder.

```
63      func fetchQuestions(currentRoomID: String, completion:
64          @escaping (_ questions: [Question]?, _ error: ServiceError?) -> Void) {
65
66          guard let token = UserDefaults.standard.string(forKey: "x-auth-token") else {
67              completion(nil, handleError(response: genericErrorMessage))
68              return
69          }
70
71          let headers: HTTPHeaders = [
72              "x-auth-token": token,
73              "roomId": currentRoomID
74          ]
75
76          let url = "\(baseUrl)/questions"
77          AF.request(url, method: .get, headers: headers).responseJSON { response in
78              if response.response?.statusCode == 200 {
79                  let questions = self.decoder.decodeFetchQuestions(data: response.result.value as Any)
80
81                  if questions.isEmpty {
82                      completion(nil, self.handleError(response: response.result.value))
83                  } else {
84                      completion(questions, nil)
85                  }
86              } else {
87                  completion(nil, self.handleError(response: response.result.value))
88              }
89          }
90      }
```

Figure 8.1: The implementation of fetching the questions from a room

Two external libraries have been used, Alamofire[1] and SwiftyJSON[2] which are community-made libraries that handle networking requests (Alamofire) and JSON parsing (SwiftyJSON). Figure 8.1 shows how the app gets the questions from a particular room. The `fetchQuestions` method is a part

[1] https://github.com/Alamofire/Alamofire
[2] https://github.com/SwiftyJSON/SwiftyJSON

of the `FeedmeNetworkService` class. It receives a room id and upon completing the networking requests, it returns either an array of questions or an error. Before calling AlamoFire's request method, a header object of type HTTPHeaders is created at line 71 (Fig. 8.1). Afterwards, AlamoFire's request (line 77) can be sent to the web server. If the response's status code is 200, the decoder will try to decode the received response, as seen on line 79. The process of decoding the request is shown in Fig. 8.2. It receives data of an arbitrary type and tries to pass that into a JSON format which Swift can iterate over. If the passing to JSON fails, it will skip the loop and return an array of empty questions. If it succeeds, it will populate the questions array and return it to the network service's `fetchQuestions` method. This is why there is a check at line 81 in Fig. 8.1, that makes sure that the returned array is not empty.

```
29      func decodeFetchQuestions(data: Any) -> [Question] {
30
31          var questions: [Question] = []
32          let json = JSON(data)
33          for element in json {
34              if let questionName = element.1["value"].string, let id = element.1["_id"].string,
35                  let answers = element.1["answerOptions"].array {
36
37                  var answerOptions: [Question.answerOption] = []
38                  for answer in answers {
39                      if let answerValue = answer["value"].string, let answerID = answer["_id"].string {
40                          let answerOption = Question.answerOption(id: answerID, value: answerValue)
41                          answerOptions.append(answerOption)
42                      }
43                  }
44                  let question = Question(id: id, question: questionName, answerOptions: answerOptions)
45                  questions.append(question)
46              }
47          }
48          return questions
49      }
```

Figure 8.2: Decoding the response

8.2 Back End Server

As mentioned in Chapter 4 (Design), the back end server of the system is made up by a NodeJS server with the ExpressJS framework, communicating with a non-relational database called MongoDB. Node servers are generally known for being very fast to develop and a large part of that is due to the many popular libraries which are available to the platform which is a result of a very active community. One of the libraries is ExpressJS, which is a framework designed to facilitate the making of web applications and REST APIs. The library provides some useful functionality which helps abstract the way the web server communicates with the client to more easily understandable terms and functions. With ExpressJS handling a request becomes relatively simple. The functionality to handle each request is divided into different so-called middleware functions depending on the URL of the given request. Middleware functions are basically a chain of functions, which the server program executes upon receiving a request. When the URL of a certain request matches the URL specified in a middleware function the content of that middleware function is executed before control is parsed to the next middleware function in the chain. Each middleware function is built up by a request object and a response object. When a new request is received by the server, ExpressJS populates the request object with the input received in the HTTP request of the client. The body of the HTTP request is first converted from JSON into JavaScript objects with the use of another library which parse JSON to JavaScript objects. For example *req.body* would contain the contents of the body of the HTTP request – but in a JavaScript object. Similarly, creating responses to the user is done by setting the different content of the response object e.g. setting *res.status* which indicates the status code of the response and setting *res.send* which is the body of the HTTP response.

Another very useful NodeJS library, which has been used extensively in the back end server, is a library which facilitates the validation of client requests. The library is called *Joi* and is used in almost all of the middleware functions handling client requests. The library provides an easily readable syntax for validating any JavaScript object and in this case the library is used to validate the body of the HTTP request sent by the client, after it has been parsed from JSON into a JavaScript object. The body of the HTTP request is required to be JSON, which means it is built up by key value pairs. With Joi, these key value pairs can be easily validated with a simple syntax. In Fig. 8.3 an example of a JavaScript function using the Joi library is shown. The function is used to validate the body of an HTTP request, when the client wants to register a new beacon to the Feedme system. The input could be any JavaScript object, but in this case the input is usually the JavaScript object representing the body of the HTTP request – this should be the new beacon that the user wish to register to the system. When this method is called it checks that the key-value pairs of the object parsed - in this case a *beacon* object - matches a certain scheme defined with the Joi library. As the figure shows, one of these requirements is that the clients need to send the *UUID* of the beacon, which is a unique identifier for the beacon which has a certain format. The format is expressed by a regular expression as shown in the figure. The output of the function does not only determine if the object has the valid shape but the *Joi.validate* function also gives a simple and user friendly error message, if the object did not have a valid shape. As a result, if the client sends a request with a misshaped body, the server will send a response directly parsing the Joi validation error message to the client. This is done by checking if the Joi validation method returns an error, and if so, returning a response with status 400 (Bad Request) along with the error message to the user.

```
function validateBeacon(beacon) {
    const schema = {
        buildingId: Joi.objectId().required(),
        name: Joi.string().min(1).max(255).required(),
        uuid: Joi.string()
          .regex(/^[a-zA-Z\d]{8}-[a-zA-Z\d]{4}-[a-zA-Z\d]{4}-[a-zA-Z\d]{4}-[a-zA-Z\d]{12}$/)
          .required()
    };

    return Joi.validate(beacon, schema);
}
```

Figure 8.3: Example of function utilizing Joi library

A third library used is a library to communicate with the MongoDB database called *Mongoose*. The existence of this library is one of the reasons for choosing the MongoDB database in the first place, as it facilitates and simplifies the communication. The library provides both a layer of validation to an otherwise completely non-validated database structure and it provides some very helpful methods and objects to manage the data stored in the database. Similar to the Joi library mentioned above, with Mongoose you can define schemes for the structure of objects. These schemes can then be used to a validate an object before it is stored in the database. With Mongoose you can define references from one type of an object to another, just like a foreign key would work in the context of an SQL database. With the Mongoose library different *models* are defined based on those schemes, which are basically JavaScript objects that facilitate the communication with the database for a certain type of data in the database. In Fig. 8.4, some JavaScript code exemplifying the Mongoose library is shown. The *Room* object is the previously mentioned Mongoose model. In the code snippet we wish to create a new object of type *Room* in the database and with this syntax the contents of the room object (name, location and building) is validated. Before sending a response to the user the save method is used to save the object in the database. As shown, the syntax is really simple and intuitive and has been used extensively across the different middleware functions in the back end server.

```
let room = new Room({
    name,
    location,
    building: buildingId
});
```

Figure 8.4: Example of the Mongoose library functions

8.3 Website

For the website a lot of different libraries have been used to speed up the creation process of each web page and to facilitate designing the different parts of the user interface of the website. In this section the two most important libraries used for the development of the website will be discussed, namely the ReactJS framework and the Bootstrap library.

ReactJS is a lightweight library which act as a framework to building single page websites. As mentioned in Chapter 7 (Design), React is used to divide each part of the UI on a web page into isolated, reusable components. React does not necessarily make the creation of web pages faster, but it provides an architecture, such that the code is more readable and easier to maintain. Instead of writing web pages in plain HTML, React enables developers to create web pages with a syntax called *JSX*, which is a combination of HTML and JavaScript. In Fig. 8.5 an example of the render function of a React component is shown. The render function is simply the function that decides what UI elements to display to the user. As shown, the syntax is very similar to HTML but instead of only having access to HTML tags (such as h1, a, button etc.) you can create your own set of tags called React components. An example of a custom made React component is the *Input* tag/component shown in the figure. This component simply provides the UI for an input field which in this case is used to obtain the email and the password from the user.

```
render() {
    const {email, password, error} = this.state;
    return (
      <div className="opacity">
          <h1 className="mb-4">Login</h1>
          <form onSubmit={this.handleSubmit}>
              <Input focus={true} name="email" label="Email"
                     onChange={this.handleChange} value={email}/>
              <Input password={true} name="password" label="Password"
                     onChange={this.handleChange} value={password}/>
              {error && <div className="alert alert-danger mt-3">{error}</div>}
              <button type="submit" className="btn btn-primary " >Login</button>
          </form>
      </div>


    )
}
```

Figure 8.5: Example of JSX syntax

The other frequently used library for the website is to facilitate creating a nice and simple user interface of the website. The library is called Bootstrap and is really simple to use – it is basically an extension to the usage of regular CSS for the design. The library gives access to a large number of predefined CSS classes which can be used on regular HTML components to give quick styling to an element. The classes can also include animation or a bit of logic limited to what pure CSS can provide. For the website some of these bootstrap classes were used. In Fig. 8.6 a screenshot of the main view of the website can be seen. The user interface consist of a navigation bar where users can quickly navigate between the different buildings their managing. For this navigation bar, two CSS classes called *navbar* and *navbar-light* are used and for the sub-elements in the building-tab each element uses the bootstrap class *nav-item*. Just applying these classes to the HTML elements gives a minimalistic style to the navigation bar and the building list items including the extra styling for the selected building item. The result of the styling can be seen in Fig. 8.6. Of course, it also required to properly align each HTML element accordingly to get the user interface seen in the figure but the bootstrap library does a lot for you. The library is a very fast way of making the user interface a bit more modern and a bit less boring. Currently Bootstrap is the third most stared Github project with a little over 131.000 stars and thus the library is extremely large. The bootstrap CSS classes are of course to a certain extend very generic and do not give a very unique look to the website. However, if wanted you can easily extend the bootstrap classes with your own CSS classes building on top of the bootstrap design. This is in some cases done with the website to customize some of the generic bootstrap design.

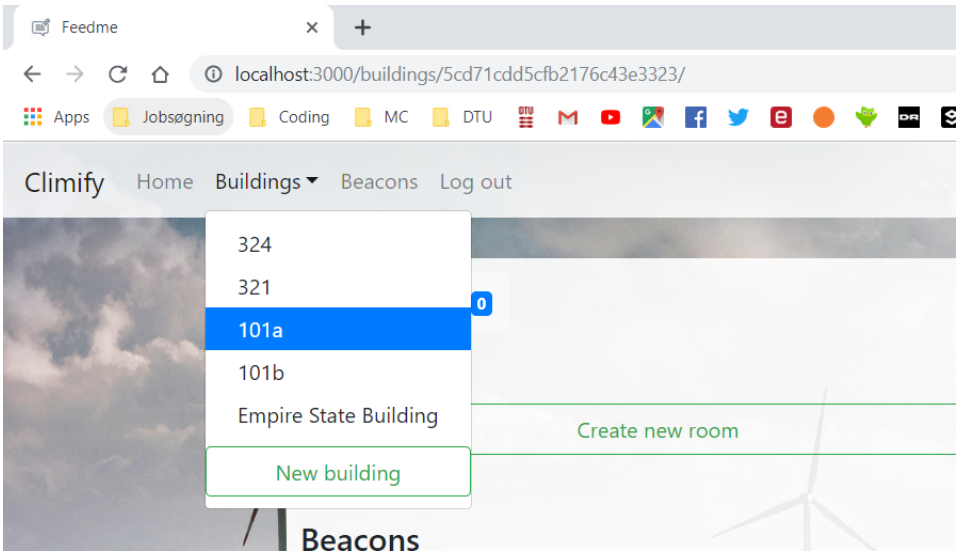


Figure 8.6: Navigation bar in UI of the website

8.4 Development Process

As mentioned earlier, when the project started evolving the main developing process was agile. However, in the beginning of the project the idea was to somewhat create a project plan, that at least covered the very main requirements needed. It might seem contradicting to create a project plan and say that the development process was agile, but the project plan's intention was to gain an overview of the lifetime of the project and make sure that it was moving in the right direction. The project plan did help the planning of when to write the report though. Looking at the software development aspect of the project, the project plan was more or less out of use after the first few weeks and the development process was purely shifted towards an agile approach. It seemed a bit overkill to have a classic project plan and risk registers, when there were only two people in the project and there were daily meetings. This also means, that of the tools mentioned in Chapter 4, Section 5 (Development Process), the most frequently

used tool was definitely the Kanban board Trello. By having made the system with an agile approach, it also allows for further implementations to be easily integrated to the current sate of Feedme.

When looking back at the risk register from the same chapter, a lot of the risks which were flagged as the most critical did not even end up being relevant. For example, the whole idea of merging Feedme with the existing Climify system was scrapped early on and instead it was allowed to develop the system from scratch. The same can be said about the risk of integrating navigation system to the application, because that idea was scrapped as well. The only risk which somewhat have impacted, was the risk of issues with the automated tests. However, it was solved by refactoring some of the classes in the mobile application, such that the architecture did allow for dependency injections.

CHAPTER 9

Testing

The Feedme system is divided into three main components; the back end server, a mobile application and a website. The end user can communicate directly with both the mobile application and the website and only indirectly with the back end server, through the provided API. Each component is responsible for the functionality they provide and that this functionality behaves as expected. When a user presses a button on the website with the title *Create room*, there are a lot of expectations and assumptions to what is going to occur within the next few seconds – there is even an expectation to how fast these things will occur. For the two front end components it is extremely important to match the expectations of the end user. This is a big part of providing a good user experience which is a very important subject – especially in the context of the mobile application where user rating is crucial for the popularity of an application. The back end server has a similar task. When clients use the API provided by the server, they have expectations to what they are going to receive and also how fast. If something is missing in the request they wish to be informed about this quickly and concisely. If an error is present in one of the front end applications, and the application behaves unexpectedly to the end-user, this can have effects on the reputation and the reliability of the applications. If an error is present in the back end server, this can potentially be even more damaging as this error might propagate to multiple clients using the API and thus affecting the experience for even more end users. This is where the concept of testing comes into the picture. With testing we wish to become more confident that our software behaves exactly as expected. It is a tool to assure a certain level of quality for our software. In the following section some of the ideas behind how the Feedme system is tested will be presented. Then, it will be discussed in more detail how testing was carried out in each of the three components of the system concluding with an evaluation of the testing results.

It can be argued that testing might be less crucial in the context of the Feedme system. No money transactions are involved, no highly confidential data are shared and no other systems are currently highly dependent on the Feedme system. However, as mentioned, the popularity of the application highly depends on the user experience. If no users are willing to interact with the mobile application, no feedback is given and users do not have much incentive to use the application in the first place. For this reason it was considered highly important to test the different components of the system thoroughly to minimize errors and to maximize the amount of feedback received. The most important component to test is the back end server, communicating with both the mobile application and the website. The second most important component to test is the mobile application, as it has the majority of the end users. Lastly, there is the website, which has few users and most of these users can be assumed to be more technically skilled as they act as the building administrators in the system. For each of the components, a set of tests have been carried out to assure a level of quality. These tests can be divided into different parts depending on which part of the software they cover. Most of the tests are automated unit tests, with the purpose of testing a small part of the system individually. Unit tests are further divided into black box and white box tests depending on how the tests were designed. Black box tests, which are the large majority of the unit tests, are designed to prove that a unit provides a certain functionality. White box tests, on the other hand, are designed to cover each possible output of a given unit, e.g. when a function has multiple branches that can terminate the function, white box tests would test each of those branches. The second largest category of tests in the system are integration tests. These tests are also automated and they aim to test multiple units together to make sure that these units communicates and interacts correctly with each other. Last category, which is also the category

with the least amount of automated tests are UI tests or end to end tests. These tests aim to assure that whole parts of the system behaves as expected. There are a very minimal amount of these types of automated tests, as the system instead is tested manually, through interacting directly with each component of the system.

9.1 Back End Server

As mentioned, testing the back end server thoroughly has been a very high priority in the project as errors on the server might not only affect the two front end applications in the Feedme system, but also any future client who wish to utilize the functionality that the server provides through its API. In this project the goal has been to divide the project into sprints of 2-3 weeks size and to work according to the principles of agile software development. This means that software should potentially be releasable every 2 or 3 weeks. In order to follow these principles, software should be continuously tested with each sprint instead of performing all of the testing at the end of the duration of the project. To fulfill this goal, testing of the back end server has been done following the software development process *Test driven development (TDD)*. With this approach, automated tests are written before the code that provides the functionality of the system is written. First, the requirements of the functionality are defined, which results in a document (detailed use case) describing both what should occur in the "good" scenarios, when the input to the new functionality is in good shape, but also how the new functionality should respond in case that the system is in a bad shape, or in case the new functionality receives malformed input. This document is then used to write unit or integration tests before the code to the functionality is written. The test will fail at first, but then pass when the code fulfills the requirements. Writing code this way does not only ensure that code is closer to be releasable, but it also ensures that unnecessary code is less likely to be written. Only code sufficient to pass the initially defined tests should be written when following the approach of TDD. Another advantage by following this process of development is, that code is usually also written more separate and isolated to the rest of the system, which makes the code more reusable and it makes the purpose and responsible of the code more clear.

As previously mentioned, most of the business logic of the system is placed server-side. This business logic includes authenticating the user, estimating the user's room location etc. Aside from this logic, the server also has an important task of validating the requests received from clients using the API. As discussed in Chapter 7 (Design), all of this logic is placed separately from the code that interacts with the database or responds to the clients. This makes the logic easily testable, as it is not dependent on any external processes such as a database connection or an internet connection. For this logic, unit tests have been written to ensure that each small unit of functionality behaves as expected. These unit tests also have the advantage of running fast as a result of not relying on any external dependencies.

Tests for the back end server have mostly been written following the TDD approach. In some cases, detailed use cases of the functionality have been made first, to properly investigate how the new functionality should be tested. But in most cases, when new functionality had to be added, the tests were written immediately. With the library used for testing, the testing function requires a description of the test and tests can be grouped into test suites, which also need a description. With these descriptions, you can document the requirements of the new functionality directly in the tests, by describing the overall functionality of the test suite along with descriptions of each requirement tested in each individual test. Because the development of almost all of the functionality of the back end server has been done following the TDD approach, there is now detailed documentation of all of the requirements to each functionality. All of the tests run, along with a description of each test and each test suite can be found in Appendix C (Back End Server Testing).

A large part of the code on the server is written to act on the output of the validation functionality previously mentioned and to give informative responses to the client. If the request of a client is badly shaped, the server should inform the client of this by responding with a proper HTTP status code along with an informative message. This functionality is tested through simulating real requests to the server

through a library called *Supertest*. With this library you can create real HTTP requests with a body and specified headers. These tests are categorized as integration tests as they test multiple units of code in the back end server. For these tests a local database is setup making them faster, but even with this setup they generally run a bit slower than unit tests because they involve database communication. For the back end server no automated end to end tests have been written. Instead these tests are performed manually through the two front end clients i.e. the web application and the mobile application. These tests are carried out simply by interacting with the system through the interface of the two applications.

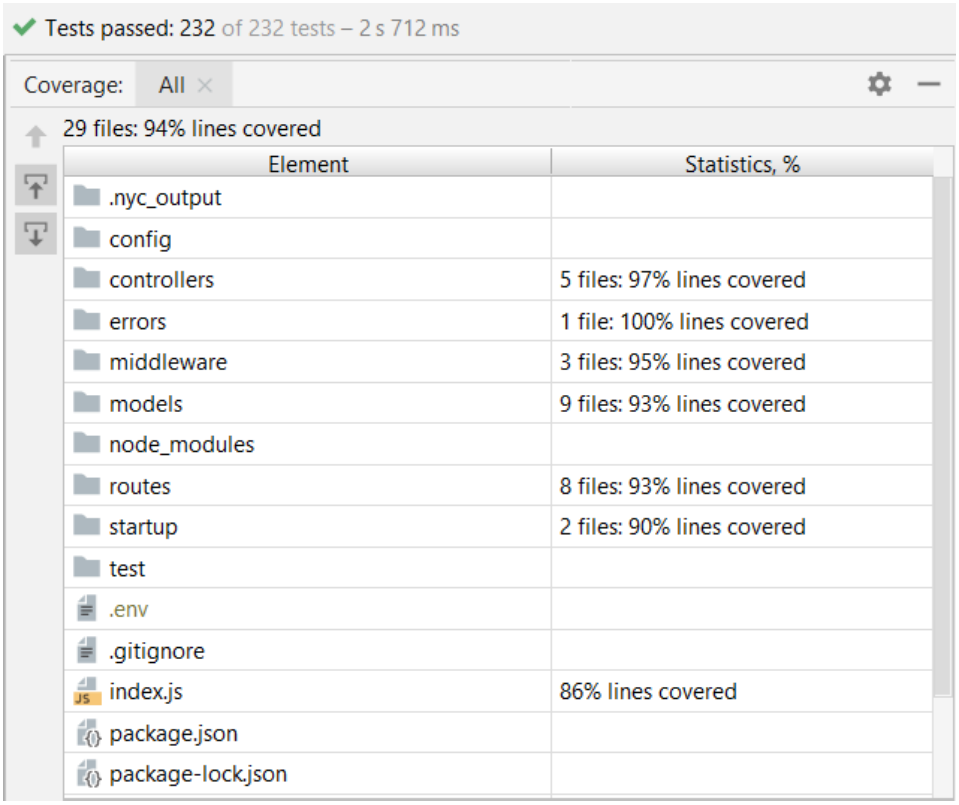


Figure 9.1: Statistics of the code coverage of the back end server tests

Figure 9.1 gives an overview of the total code coverage from the tests written for the back end server of the Feedme system. A full list of all the tests, including their status (pass or fail), the description and the time of execution can be found in Appendix C (Back End Server Testing). In total, 214 automated tests have been written and these tests are distributed into 82 unit tests and 132 integration tests which results in a total code coverage of 94% for the back end server. Code coverage can be a good measurement for how well code is tested in a system, but it can also be very misleading as it simply indicates the percentage of lines of code which have been reached by the tests run. Thus, code coverage does not indicate anything of how well these tests actually verify the functionality of the code they cover, only that they actually reach the code. But assuming that the tests are actually well written, a code coverage of 94% is usually acceptable and sufficient for almost any kind of software[1].

The back end server has also been tested indirectly through manual tests of the website and the mobile application. These tests have been the actual acceptance tests of the user stories made at the beginning of each sprint iteration and the results of some of these tests will be covered in the next section.

[1]Martin Flower has made an article on the subject of Code Coverage https://martinfowler.com/bliki/TestCoverage.html

9.2 Mobile Application

The second most tested component is the mobile application. Since it is the component most users directly interact with, it was given a fairly high priority as well. The testing of the mobile application is a mix of manual tests and automated unit tests. The manual tests cover the front end testing of the application, which means that no automatic tests or unit tests involves the front end. Creating automatic front end tests is possible, but it can be an extensive task and the time was directed towards creating unit tests for the back end of the mobile application instead.

The two core classes of the mobile application, are the networking service, `FeedmeNetworkService` (which relies on `FeedmeNetworkServiceDecoder`, because that class performs the logic of decoding the data received from the server) and the class which listens to beacons, `LocationEstimator`. Unfortunately, there are some issues testing networking services in general and in the case of `LocationEstimator`, it is even a bigger issue with all the BLE communication. The first issue is, that it is not feasible to test based on a real server implementation. That is because of the time it takes to execute all the tests and and the results of the tests will not be deterministic. The second issue is, that for the unit testing part, it is simply not possible to listen to BLE packets through the simulator, which runs the unit tests. However, to solve those issues the concept of mocking an object and dependency injection have to be introduced.

Mocking an object essentially means, that you create an object which has the same behavior as an implemented object in the system. In this particular case the needed behavior is of the class `FeedmeNetworkService` (from now on that will be referred to as "the network service"). The `LocationEstimator` relies on the network service, but as said before, it should not rely on the real server when testing. Therefore a protocol is created. Protocols in Swift are the equivalent to interfaces in programming languages such as Java. This means that the actual implementation of the networking service, is conforming to the protocol `FeedmeNetworkServiceProtocol`, while a new class, `MockFeedmeNetworkService` will also conform to that protocol (see Fig. 9.2) and therefore have the same methods. The key is to make those methods perform the same behavior as the real implementation. The networking service is implemented such that it contains the minimal logic possible. Its only job is to send and receive data from the server. The actual decoding of the JSON into structs in the application, is delegated to the class `FeedmeNetworkServiceDecoder`. Therefore, the mock class also contains test JSON responses (example of some of the test responses is shown on Fig. 9.2), corresponding to the actual JSON responses the web server is sending to the application.

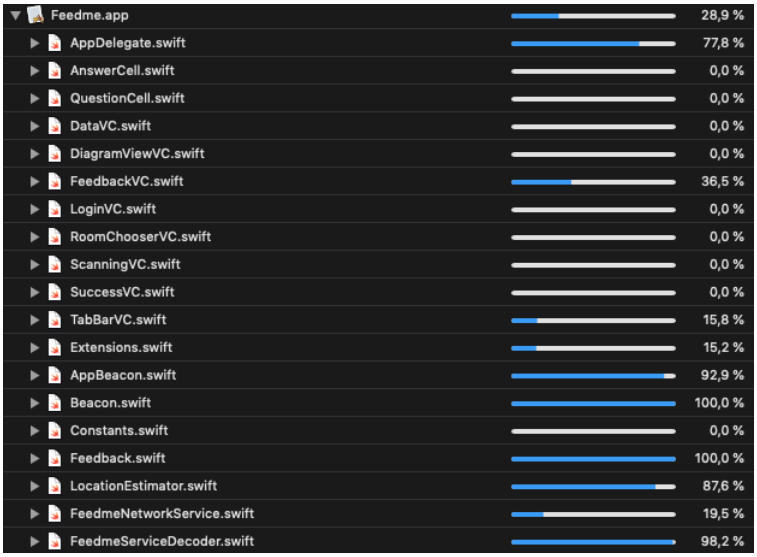
Dependency injection is a technique used to decouple different classes allowing them to be easier tested. With dependency injection, instead of having the class create new instances of the object it depends on, the objects are instead injected into the class either through its constructor or as parameters to the methods where the object is used. In this particular case, `LocationEstimator` depends on the network service, hence making it the dependant. However, when testing the location estimator a mock for the network service is injected instead. The mock just has to conform to the protocol and implement all the methods. As a result, most of the methods of `LocationEstimator` are covered by the unit tests, but the methods which are a part of the `CoreLocation` library (the methods which actually listens for real beacons) are still not covered. To test these methods you would have to mock the `CoreLocation` library provided by Apple which would be very comprehensive. In total, 22 unit tests have been made (with multiple assertions for each test) for the mobile application, covering the core functionalities. The unit tests are shown in Fig. 9.3b. It is worth noting, that mocking an object has some downsides as well. Assume that there was a misunderstanding and the parameter in the response in Fig. 9.3b "name" was actually "buildingname" in the actual request, the tests will pass, because the decoding is done based on what was thought was the right parameter. However, when the app would use the real implementation, the decoding would fail and an error would be returned. An alternative to this could be to setup a test server.

When creating the unit tests, since `LocationEstimator` relies on the model structs that are in the application, when the tests pass they also cover the structs, as seen in Fig. 9.3a (the structs being

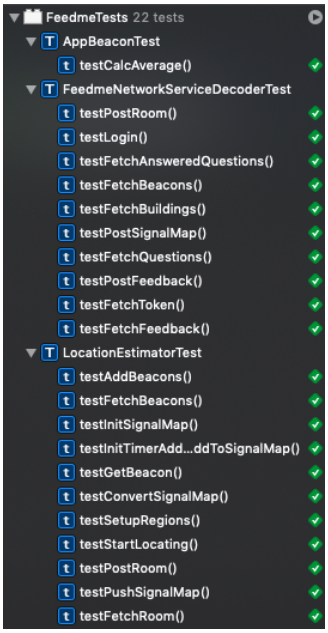
```
131     let mockFetchBuildingsResponse = [
132         [
133             "name": "Building 303",
134             "_id": "5cd491da2fc512294ee17df9",
135             "rooms": [
136                 [
137                     "_id": "5cd710d0cd752136263717eb",
138                     "name": "Rum1",
139                     "building": "5cd491da2fc512294ee17df9",
140                     "__v": 0
141                 ],
142                 [
143                     "_id": "5cd7e911cd7521362637208f",
144                     "name": "Test Room",
145                     "building": "5cd491da2fc512294ee17df9",
146                     "__v": 0
147                 ]
148             ]
149         ],
150         [
151             "name": "Building 308",
152             "_id": "5cd7fa95cd75213626372096",
153             "rooms": []
154         ]
155     ]
156 }
157
158 extension MockFeedmeNetworkService: FeedmeNetworkServiceProtocol {
159
160     func fetchBuildings(completion: @escaping ([Building]?, ServiceError?) -> Void) {
161         if shouldReturnError {
162             completion(nil, ServiceError.error(description: ""))
163         } else {
164             completion(self.decoder.decodeFetchBuildings(data: mockFetchBuildingsResponse), nil)
165         }
166     }
167
168     func postRoom(buildingId: String, name: String, completion: @escaping (String?, ServiceError?) -> Void) {
169         if shouldReturnError {
170             // Giving error response (the data put in the decoder is from the wrong resposne)
171             if let roomid = self.decoder.decodePostRoom(data: mockFetchFeedbackResponse) {
172                 completion(roomid, nil)
173             } else {
174                 completion(nil, ServiceError.error(description: ""))
175             }
176         } else {
177             completion(self.decoder.decodePostRoom(data: mockPostRoomResponse), nil)
178         }
179     }
```

Figure 9.2: Mock of the network service including some test JSON

Beacon.swift, Building.swift etc.). This results in an overall app coverage of 28.9%, which at first impression may seem fairly low. As mentioned earlier, there have been no front end unit testing and the view controllers (the files ending in ”VC” in the figure) mostly handle the front end. This is the primary reason for most view controller classes having 0% coverage, and the same goes for the two files that end in ”Cell”. When looking at Fig. 9.3a, `LocationEstimator` has almost a 100% coverage and the lack of testing the remaining code, is due to the need of listening to BLE signals, which the simulator is not capable of. The `FeedmeNetworkService` has a very low coverage, and that is because it does not really contain any logic, it simply creates URLs and sends/receives data from the server. As mentioned, the received data is passed into `FeedmeNetworkServiceDecoder`, which does contain logic and therefore is tested thoroughly, with a coverage of 98.2%. The tests are run in Xcode, which is an IDE dedicated for developing applications for Apple products (iOS, Mac OS etc.).



(a) Statistics of the code coverage of the mobile application tests



(b) The passing of the unit tests

Figure 9.3: Code coverage and unit tests

The manual tests have simply been carried out through real interaction with an iPhone 7 running Feedme. By observing the actual behavior of the application and comparing it to the expected behavior, the test passes if the actual behavior is the same as the expected behavior. More specifically, the tests are meant to document that the use cases are covered. Fig. 9.1 shows the use cases which the manual tests cover (See Appendix B (Detailed Use Cases), for a more detailed description of the use cases). At the same time, breakpoints have been very useful as well, because it allows you to pause the program during execution and all the internal variables and the states of the object can be investigated thoroughly. As an example, this has been used to make sure that only one instance of the networking service and the `LocationEstimator` are instantiated. The manual tests are especially useful when testing the user experience of the application which includes testing of simple animations or the way the application responds to touches, swipes etc. How the application feels, should not be neglected, because it affects how much time a user wants to spend on that particular app. Using automated tests for UX does not always make much sense. Unfortunately, most of the manual testing of the user experience have been carried by the developers of the system. In practice it would have been better if the features were tested by people who could potentially be end-users of the system. The UX tests have not been carried out systematically and the ”acceptance” of the tests, were mainly based on what felt right, which of course is not completely optimal.

No.	UC	Description	Scenario	Expected Result	Pass
1	B.1	The user sends feedback where their room-location is automatically estimated	Main scenario	A small animation followed by a confirmation is presented	
2	B.1	The user sends feedback where their room-location is automatically estimated	Alternative scenario 1a	A message informs the user that location could not be found	
3	B.1	The user sends feedback where their room-location is automatically estimated	Alternative scenario 1b, 2a, 3a	A message informs the user that internet access is required	
4	B.2	User wants to see in which room it is located in	Main scenario	A message informs the user about the name of the room	
5	B.2	User wants to see in which room it is located in	Alternative scenario 1a	A message informs the user that location could not be found	
6	B.2	User wants to see in which room it is located in	Alternative scenario 1b	A message informs the user that internet access is required	
7	B.5	User wants to see data about its own feedback or the feedback given by others	Main scenario	The answered questions and the pie charts are shown based on the applied filters	
8	B.5	User wants to see data about its own feedback or the feedback given by others	Alternative scenario 2a, 3a, 4a, 5a, 6a, 7a, 8a, 9a	A message informs the user that internet access is required	
9	B.5	User wants to see data about its own feedback or the feedback given by others	Alternative scenario 2b	The questions from all the rooms, which the user has given feedback in is shown	
10	B.5	User wants to see data about its own feedback or the feedback given by others	Alternative scenario 2c, 3b, 4b, 5b, 6b, 7b, 8b, 9b.	A message informs the user that no feedback is given and encourages the user to give feedback	

Table 9.1: Manual testing of the application

9.3 Website

For the web application only manual tests have been carried out to ensure that the component behaves as expected. These manual tests have been performed mostly by interacting with the user interface of the website and manually confirming that the system responds as expected. Logging errors and information to the console have also been a big part of verifying that the internal state fulfills the requirements at different points in time. With the logging statements, the values of variables can be displayed and information about when and how the system behaves can be examined. The statements can be read while interacting with the system through the browser. As mentioned, testing the web application was not a very high priority in the development process which is one of the reasons why no automated tests are currently present in this part of the system. A very minimal amount of business logic is currently present in the web application. At the moment, it is simply a visual layer to API of the back end of the server. This makes automated testing less important, as no complex algorithms are present and as there are simply little code that could perform unexpected. That being said, another

reason for the lack of automated tests is simply insufficient time. It is a clear goal to incorporate more automated tests into the web application in the future. Although there are not much business logic in the system it is still very important to ensure that the user interface acts and looks as expected. To ensure this, it would be very beneficial to have an amount of automated UI-tests. This would not only provide more certainty, that the end user will get exactly the intended experience, but it would also make for a faster way to test the web application as it would be faster to run a suite of UI-tests as opposed to manually testing the user interface manually through the browser.

CHAPTER 10

Evaluation

In this chapter, a small evaluation of the system requirements is presented followed by an evaluation of the location estimation algorithm implemented in the system. Finally, an evaluation of how the algorithm performs in comparison to other similar projects is given.

10.1 System Requirements

The system requirements are mainly described as the user stories from Chapter 4, Section 3 (User Stories). The main requirements have been fulfilled and that is ensured through testing of the application, however some specific sub requirements of the main requirements have not been fulfilled completely. The two most critical user stories, user story 1 and user story 2 have some very specific goals of the automatic location detection.

- The room should be estimated correctly above 95% of the time
- The time to detect the room should not exceed 1 second

These goals could unfortunately not be reached in certain situations, which is elaborated in the next section which gives an in depth evaluation of the algorithm based on a test sample. When developing the user stories, the stories could have been more explicit stating where in the room and how many rooms per beacon is required, to live up to those goals. Because in some set ups, the algorithm achieves a success rate of 100%. This is the case when standing in the very center of each room. The time to detect the room is never under 1 second. When opening the app you will have to wait around 2 seconds for the initial suggestion of which room you are in and to get a even better result, you would have to wait around 3-4 seconds for it to calibrate and give an even better result. Based on Chapter 3 (Related Work) and the experience gained while using the beacons, the requirements were just too unrealistic. When the scanning of the beacons is instantiated, the app receives the first signal after 1 second. This is due to restrictions of the library, which Apple is responsible for, because the beacons advertising interval is actually lower than 1 second. This means that detecting the room in under one second, is simply not possible with the use of the library `CoreLocation`.

The rest of the requirements are fulfilled based on the tests, described in Chapter 9 (Testing). The only requirement which is not implemented, is user story 7: *As a building admin, I want to notify the users in a particular room about an available question, to make them answer that question.* This user story has not yet been completed, but will definitely be taken into consideration for further development.

It was also discussed early on, that the application should have an incentive for the user to use the app. That has mainly been done by ensuring that the the app contains as few bugs as possible and that it gives a good user experience. However, some extra features could have been implemented as well that is not necessarily related to giving feedback, but more to make sure that the users use the app as much as possible.

10.2 KNN Location Estimation Algorithm

To estimate from which room the user is giving feedback, a variation of the KNN algorithm has been implemented. In this section, the results from evaluating the algorithm in a real environment will be presented followed by a discussion and an evaluation of how and why these results were achieved.

To properly test the algorithm, the Feedme system has been set up in an environment where the system could potentially be implemented. The environment chosen was at the Technical University of Denmark in building 319 – a building with multiple rooms of different sizes including a hall connecting the rooms. Figure 10.1 shows a floor-plan of the location where the testing of the algorithm was done. Each room registered in the Feedme system is labeled with a letter from A to E. Some of the labeled rooms however are not actually rooms in the sense that they were surrounded by walls. This is the case for room D, and E, which are part of the hallway of the building. These two spaces are still registered as rooms and their borders are marked by dotted lines in Fig. 10.1. Generally solid lines are boundaries to rooms with actual walls, while dotted lines also marked boundaries to rooms but with no physical border in the environment. Closed doors are marked with two small lines indicating where the door starts and ends.

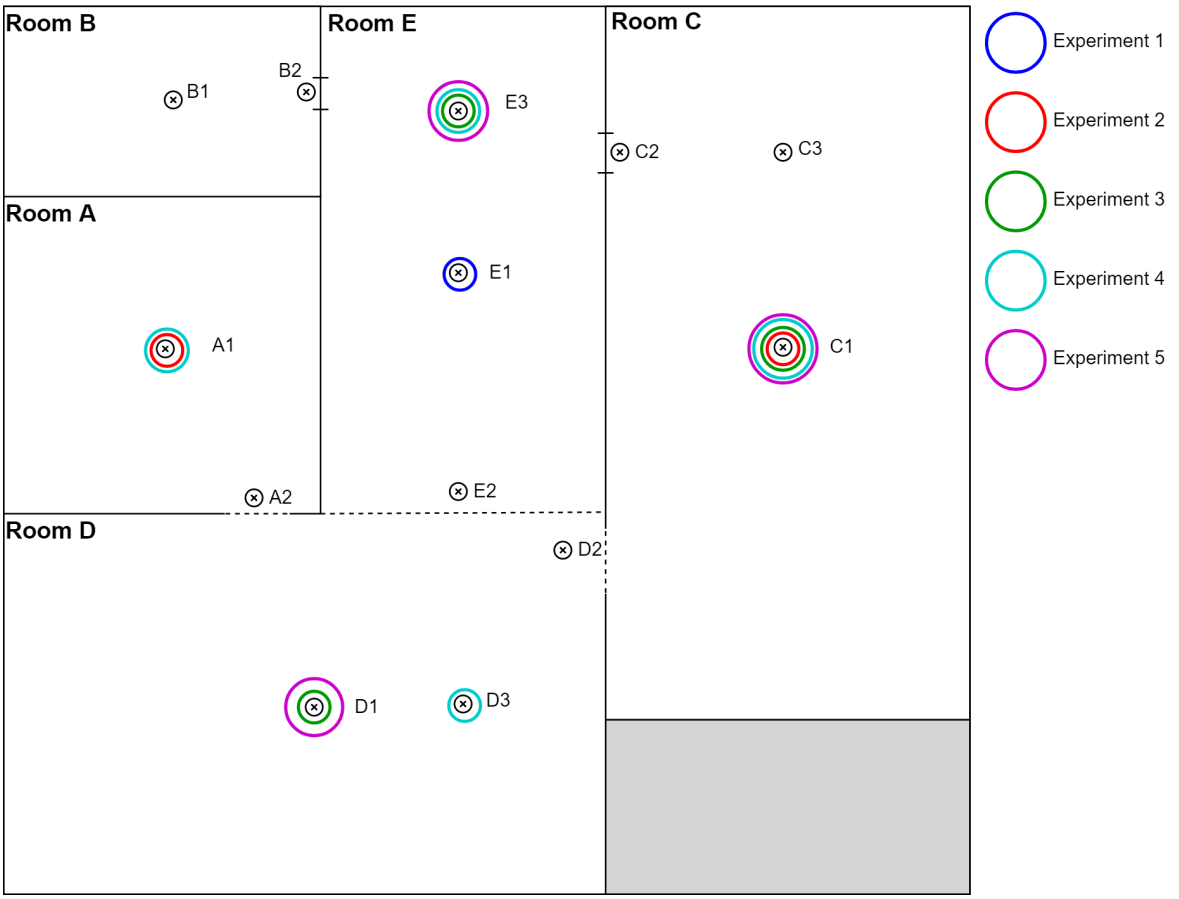


Figure 10.1: Overview of the setup for testing the KNN algorithm

The BLE beacons were set up in different locations for each experiment and the different locations are marked with a cross and a label. The circles around some of these positions mark that beacons were placed there at a specific experiment. On the right of the figure, the legend shows which experiment corresponds to which color. Each experiment includes a 40 second scan of each room independent of the actual size of the room. The amount of time used on scanning each room, can of course be adjusted

in a real world scenario, especially if rooms vary a lot in size, but this parameter was kept constant throughout the experiment to see how the other factors affected the result. After the beacons were set up and each room was scanned with the smartphone, measurements were taken at a total of 10 different locations across the different registered rooms in the building. Two measurements were taken in each room for each experiment, one of which was taken at the center of the room, and the other taken at the boundary of the room, close to the edge of another room. Ideally one measurement should be easy for the algorithm to validate while the other should be substantially harder as it would be very close to another room. The measurements taken at the border of each room were taken to challenge the location estimation algorithm and to provide more knowledge about how the algorithm performs under tough conditions. Using borders that are not restricted by actual walls, challenges the experiment even more. Walls would create a natural boundary and the signal strength on one side of the wall compared to the other, would be significantly different, hence helping the classification of a new point to be in the correct room. The locations of where the different measurements were taken overlap with some of the locations where the beacons were placed - they are also marked with a cross and a label in Fig. 10.1. Each location at the center of the room contain the number 1 (A1, B1 etc.) and each of the edge case locations contain the number 2 (A2, B2 etc.) as shown in Fig. 10.1. Locations that contain the number 3 were locations that in some of the experiments were used for beacon placement.

In total, five different experiments were carried out. For each experiment all of the rooms were included, also the ones not having a beacon placed in it. In the first four experiments the amount of beacons installed varied ranging between one and four beacons. For these four experiments a 40 second scan was performed in each room, by walking around the border of the each room evenly. For the fifth experiment, only three beacons were used but a different type of scan was performed. For each room, in the fifth experiment, 40 seconds were also used on the scan, but when scanning, more time was used along the boundaries of the room that were close to other rooms. This was done to make the algorithm perform more precisely when the device is very close to being in another room and thus might have a harder time deciding in which room the user is located. 12 seconds were spent in each of the ten locations. Every three seconds the room which the location estimation algorithm determined the user to be in was noted down, resulting in a total of 40 measurements for each experiment. If the algorithm computed the correct room, then it was counted as a success. If the algorithm computed any other room then it was counted as a failure.

In Fig. 10.2 the results of each experiment are shown. The first column indicates the experiment number, where the color of each experiment is the same color as the ones indicating the experiment number in Fig. 10.1. The second column shows how many beacons were used for the experiment. The third column indicates the placement of the beacons in each experiment. The labels listed refer to locations indicated by Fig. 10.1. The fourth to sixth column indicate the success rate respectively for measurement taken in the center of the room, measurements taken close to the boundary and for the total number of measurements.

Figure 10.3 shows a graph that demonstrates how the success rate is dependent on the number of beacons installed in the Feedme system. As the graph shows, installing more beacons in the system generally makes for a more precise location estimation algorithm. This is of course due to the fact that the KNN algorithm can have more data to choose from when matching a new unknown data point with the existing data stored in the database. For only one beacon the success rate is extremely low, both for

Experiment #	Number of beacons	Placement of beacon(s)	Center of room	Close to boundary	Total
1	1 beacon	E1	45%	20%	33%
2	2 beacons	A1 and C3	80%	30%	70%
3	3 beacons	C1, D1 and E3	95%	65%	80%
4	4 beacons	A1, C1, D3 and E3	100%	65%	83%
5	3 beacons (border scan)	C1, D1 and E3	100%	70%	85%

Figure 10.2: Testing results for KNN algorithm

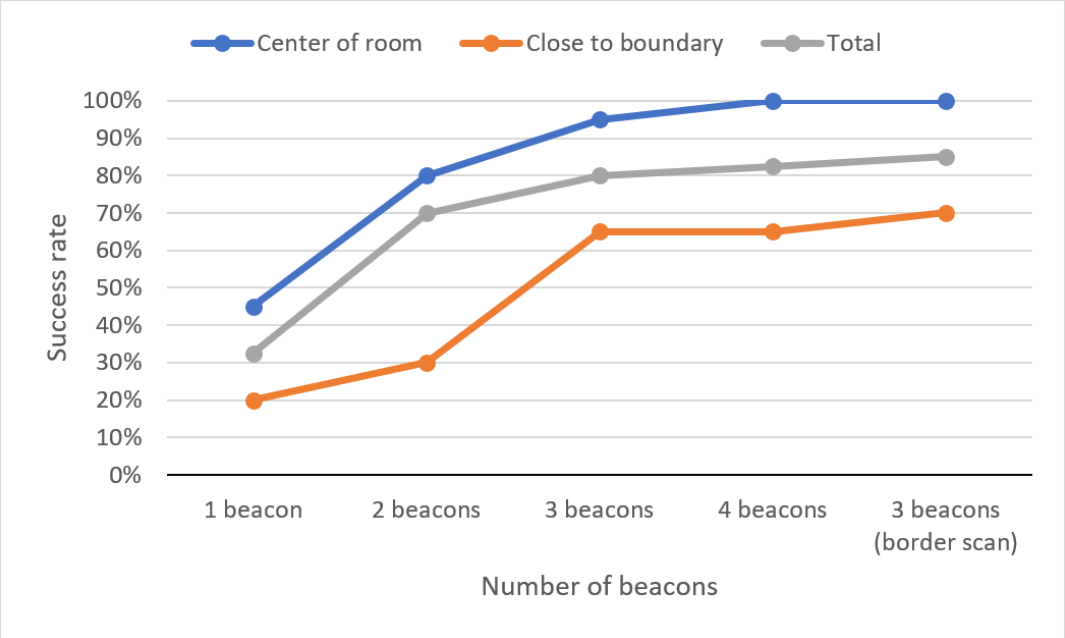


Figure 10.3: Graph for the testing results of the KNN algorithm

the measurements taken at the center of each room and for the measurements taken close to the border of each room. This was expected as a single RSSI value provides very little information to where the device is located - only that the device is a at certain distance from the beacon, not in which direction from the beacon the device is located. When two beacons are installed the total success rate of all the measurements increases from 33% to 70% which is a large improvement.

In Fig. 10.3 the success rate for the measurements taken close to the boundaries of the room can be seen. Here, the success rates ranges from 20% with only one beacon up to 70% with three beacons and a more specific room scan. For three beacons, estimating the location of users in five rooms 70% does not sound very precise. As Fig. 10.1 shows, these measurements were taken at the very edge of the rooms and for three out of five rooms, there were no physical border between the two rooms. Generally this is where the algorithm struggles a lot. In two out of the five rooms the measurements close to the boundaries were still enclosed by a physical boundary namely a closed door. In these cases the algorithm performs a lot better, which indicates that the algorithm actually prefers to have walls surrounding each room for better estimation. These problems generally only occur when the devices are placed closed to border as the table and graph shows in Fig. 10.2 and Fig. 10.3. For measurements taken at the center of the room, the algorithm performs very well when at least three beacons are installed in the system but the measurements were also taken at the very center of the room. When four beacons were installed or three beacons with a more specific room scan the algorithm is actually correct 100% of the time based on the experiment, when trying to estimate the user's room location.

10.2.1 Design Science

In Chapter 3 (Related Work) different algorithms for indoor location estimation were introduced and then analyzed and discussed in greater detail in Chapter 6 (Location Algorithm). In this section, the results of evaluating the KNN algorithm implemented in this system is compared to some of the results obtained in similar projects.

In their paper[21] Y. Wang et al. presented a system that, through a trilateration based algorithm, could estimate an area where users would most likely be located at. Thus, trilateration was not used to predict exactly where the user was located but rather a certain area where the user would most likely be located. In an open, square space of 7x7 meters and with 4 beacons at each corner they managed to get an accuracy of 93% to an area of around 1 square meter. It is difficult to compare this result directly to the results obtained from the experiments done in this project, as the environments are different. At the same time, in both projects only a certain area that the user might be located in is computed. For the Feedme system this area is the room the user should be located in, and for the project of Y. Wang et al. the area is of different sizes depending on the beacon setup. Although the results of their project are obtained in a very small space compared to the results of this project, it would be interesting to implement their algorithm and compare it directly to results of this project.

Xin-Yu Lin et al. have in their project[11] utilized the *Nearest Beacon* algorithm to estimate patients' location in a hospital. This algorithm builds upon estimating which beacon sends out the strongest signal, and then estimating the users' location to be the same as the beacon's location. A variation of this algorithm was also implemented in the Feedme system. In a building environment with 10 beacons set up in 10 different rooms, their system estimated the users' location correctly 97.22% of the time. In terms of overall goal, this project is very similar to the Feedme system as both projects have the intention of estimating users to be in a certain room known to the system – not their exact location. Even the environment of their experiment is very similar to the environment, where the evaluation results of the Feedme system was obtained, which makes the results comparable. Comparing the numbers directly, Xin-Yu Lin et. al have a higher success rate than what this project was able to achieve. However, in this project less beacons were used because beacons did not necessarily have to be installed in every room, contrary to Xin-Yu Lin's project. For the Feedme system, this was a clear goal, because it was determined to be too costly to have beacons installed in each room, both in terms of actual costs of the beacons, but also because generates more maintenance, as beacons needs to be registered to the system and replaced or recharged when their battery is low.

Overall, the results of the location estimation algorithm implemented in the Feedme system is highly comparable to what currently exists in other projects. The location estimation algorithm implemented takes advantage of the walls of each room, instead of getting weakened by them. As discussed in Chapter 3 (Related Work), obstacles have in some of the other BLE location estimation projects affected the results negatively, but for the Feedme system, stationary obstacles are turned into an advantage, which is fairly unique when looking at similar projects. Another thing that the system does well is, to fully utilize the capabilities of beacons. In the Feedme system, beacons are not required to be in every room which makes the system less costly compared to other similar projects. All of the signals obtained by the smartphone are used in the determination of the location, even though the beacons might be far away. This is not the case in projects that utilize the *Nearest Beacon* algorithm, where only the closest beacons are actually relevant. A disadvantage of the algorithm when comparing with similar projects is, that the algorithm requires a training phase where each room is scanned with the mobile application. Most other algorithms still require a setup phase e.g. a phase where beacons are calibrated or distances are measured, but in some of the other projects the setup phase is faster than for the Feedme system. However, as it was seen in the evaluation of the algorithm the scanning process can be reduced by focusing the scanning to parts of the room that are close to other rooms.

CHAPTER 11

Future Work

First and foremost, the most obvious future work would be to implement the missing user stories. Besides that, since the project has been developed such that it is scalable, basically all sorts of different features could be added. However, to finish the project in a perfect way, there could have been implemented more automatic tests for the mobile application and especially some sort of usability tests should be carried out. In this report, a software handbook is given, but a handbook would probably not be read by most users of the who uses a mobile application (unauthorized user in this particular project). Therefore, it would be highly valuable to make sure that the users actually know how to navigate the application and make sure that they feel like having a good experience when using it.

It was prioritized that the application should be as accessible as possible, so when opening the app the first time, a user token is generated through the API and that token is used to identify the user. This means that the user is anonymous, but if the user deletes the application or gets a new phone, the provided feedback is no longer associated with that user. Therefore, a future implementation could also be to have an optional user registration process for users who wants to save their history of given feedback, when switching to a new device.

Another way to improve the system would be to further optimize the location estimation algorithm. As presented in Chapter 10 (Evaluation), the system has been tested and evaluated in a real environment which has given some ideas as to how the algorithm could be adjusted. For the KNN algorithm it would be interesting to see how the algorithm performs with different values of k. It would also be interesting to test the algorithm with a different distance measure than the euclidean distance measure. As discussed in Chapter 6 (Location Algorithm), it would also be valuable to make some users confirm their room location which could then be used as training data for future users.

Lastly, some more abstract ideas were also discussed as potential further implementations. For example, when the admin is scanning a room, the signal strengths that are saved could be converted to a real distance unit, which could be used to draw shapes that represents the room and thereby automatically generate a map of the rooms that are scanned. This would allow the algorithm to not only estimate in which room the user is located but also in which part of the room. For indoor climate regulations this information would be valuable, because users closer to a window could have different perceptions of the indoor climate in comparison to users further away from windows.

CHAPTER 12

Conclusion

The main goal of the project was, to create a platform, that can establish the user's indoor location and based on that, receive feedback about the user's perception of the indoor climate. The feedback is generated by asking the user specific questions about the indoor climate. Those questions are associated with the rooms, and thus making it a platform driven by location based user feedback. The feedback's intended use is for optimizing the future indoor climate. When getting an overview of the system, it can therefore be concluded that those goals were reached and accomplished.

To establish the user's indoor location, Bluetooth Low Energy beacons have been used in combination with the machine learning algorithm, K-Nearest Neighbors. The success rate varies a lot depending on how many rooms there are per beacon, how much time was allocated to perform the scan of the room and where in the room a user is located. This means that the success rate spanned from around 20% in the absolute worst setup and up to 100% in the best set up. The time it takes for the user to get their room location estimated is roughly around 2 seconds.

For Feedme to work, a web server is developed. The web server is the back end of the system and for that purpose, there is also created a database that keeps track of the different elements of the system. In addition to that, an iOS application has been developed to fetch the signals broadcast by the beacons and pass them on to the web server, that runs the algorithm and returns the room in which the user is in. It was a high priority to minimize the time it takes for users to give feedback and therefore users are not required to register to use most of the functionality of the mobile application. An advantage to this was, that a user's identity is kept anonymous. A disadvantage was, that in the case that a user uses the application from a different device, their feedback is no longer associated with that user. For the administrators of the platform, a website has been developed with the purpose of making it easy to set up the system. The intention of the website is to make it convenient to set up the system and to be able to extract feedback in JSON format, such that the data can be used to optimize the future of indoor climate.

Bibliography

[1] Monika Adarsh. *11 Incredible beacon use cases every marketer should know in 2018 | Beaconstac*. 2017. URL: https://blog.beaconstac.com/2017/12/11-incredible-beacon-use-cases-from-2017-every-marketer-should-know-before-2018/.

[2] Leifur Bjornsson. *BLE Beacons for Indoor positioning : No Bull, Beacon review*. 2016. URL: https://locatify.com/blog/ble-beacons-no-bull-beacon-review/.

[3] Matthew Bon. *A Basic Introduction to BLE Security - Wireless - eewiki*. 2016. URL: https://www.digikey.com/eewiki/display/Wireless/A+Basic+Introduction+to+BLE+Security.

[4] Mike Cohn. *User Stories Applied*. Volume 13. Addison-Wesley, 2009. ISBN: 0321413091.

[5] Joseph Epstein. "Introduction to Wi-Fi". In: *Scalable VoIP Mobility*. Elsevier Inc., 2009, pages 101–202. ISBN: 9781856175081. DOI: 10.1016/B978-1-85617-508-1.00001-3.

[6] Jason Fitzpatrick. *The Difference Between WEP, WPA, and WPA2 Wi-Fi Passwords*. 2017. URL: https://www.howtogeek.com/167783/htg-explains-the-difference-between-wep-wpa-and-wpa2-wireless-encryption-and-why-it-matters/.

[7] US Government. *GPS Accuracy*. URL: https://www.gps.gov/systems/gps/performance/accuracy/#sa.

[8] US Government. *GPS Overview*. URL: https://www.gps.gov/systems/gps/.

[9] US Government. *Selective Availability*. URL: https://www.gps.gov/systems/gps/modernization/sa/.

[10] Paul Johannesson and Erik Perjons. *An introduction to design science*. Springer, 2014. ISBN: 9783319106328. DOI: 10.1007/978-3-319-10632-8.

[11] Xin Yu Lin et al. "A mobile indoor positioning system based on iBeacon technology". In: *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS* 2015-Novem (2015), pages 4970–4973. ISSN: 1557170X. DOI: 10.1109/EMBC.2015.7319507.

[12] You-Wei Lin and Chi-Yi Lin. "An Interactive Real-Time Locating System Based on Bluetooth Low-Energy Beacon Network". In: *Sensors* 18.5 (May 2018), pages 1637–1654. ISSN: 1424-8220. DOI: 10.3390/s18051637.

[13] Neha Mallik. *How Airlines can use Beacons to Enhance Travel | Beaconstac*. 2014. URL: https://blog.beaconstac.com/2014/07/how-airlines-can-use-beacons-to-enhance-travel-experience/.

[14] Yu Chi Pu and Pei Chun You. "Indoor positioning system based on BLE location fingerprinting with classification approach". In: *Applied Mathematical Modelling* 62 (2018), pages 654–663. ISSN: 0307904X. DOI: 10.1016/j.apm.2018.06.031.

[15] Brian Ray. *ZigBee Vs. Bluetooth: A Use Case With Range Calculations*. 2015. URL: https://www.link-labs.com/blog/zigbee-vs-bluetooth.

[16] Margaret Rouse. *internet of things (IoT)*. 2019. URL: https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT.

[17] Mike Ryan. “Bluetooth: With Low Energy Comes Low Security”. In: *USENIX Workshop on Offensive Technologies*. USENIX, 2013. URL: https://www.usenix.org/system/files/conference/woot13/woot13-ryan.pdf.

[18] Fazli Subhan et al. “Indoor positioning in Bluetooth networks using fingerprinting and lateration approach”. In: *2011 International Conference on Information Science and Applications, ICISA 2011* (2011), pages 1–9. DOI: 10.1109/ICISA.2011.5772436.

[19] Adel Thaljaoui et al. “BLE localization using RSSI measurements and iRingLA”. In: *Proceedings of the IEEE International Conference on Industrial Technology* 2015-June.June (2015), pages 2178–2183. DOI: 10.1109/ICIT.2015.7125418.

[20] Yankai Wang et al. “Indoor positioning system using Euclidean distance correction algorithm with bluetooth low energy beacon”. In: *2016 International Conference on Internet of Things and Applications, IOTA 2016* (2016), pages 243–247. DOI: 10.1109/IOTA.2016.7562730.

[21] Yixin Wang et al. “RSSI-Based Bluetooth Indoor Localization”. In: *Proceedings - 11th International Conference on Mobile Ad-Hoc and Sensor Networks, MSN 2015* (2016), pages 165–171. DOI: 10.1109/MSN.2015.14.

[22] Guochang Xu and Yan Xu. *GPS: Theory, Algorithms and Applications*. 3rd edition. Springer Nature. ISBN: 9783662503652. DOI: 10.1007/978-3-662-50367-6.

APPENDIX A

Glossary

- **User** A user of the system, which can have different roles depending on their usage of the system. A user could be a student giving feedback, it could be an administrator registering beacons to the system or even a system admin deleting users.
- **Room** A room is in this context an indoor space bounded by walls and/or doors for small to medium size, i.e. a room up to $150m^2$. Usually this would be an office, a class room or another type of room situated inside a larger building.
- **Building** A building is limited to an indoor environment where one or more rooms are located. The physical boundaries of a building usually also sets the boundaries of a building in the system, but it does not have to. This means a large building could potentially be broken down to two or more separate buildings in the system.
- **Beacon** A beacon is a small device that is capable of sending out a Bluetooth signal to be received by smartphones. It has a UUID code, that is sent out to the smartphones, which can be used to uniquely identify the beacon.
- **Question** A question is a given string of text with a defined set of answer options. A question is created by building administrators and is linked to a particular room in the system, making questions appear only to users located in that room. Questions can be active or inactive meaning a building admin can choose to show or hide questions for a room.
- **Answer** An answer is a string of text linked to question.
- **Feedback** Feedback is created when a user gives an answer to a particular question. Feedback is therefore an answer to a question, given by a certain user in a specific room.

APPENDIX B

Detailed Use Cases

B.1 Send location-detected feedback

Description: The user sends feedback where its room-location is automatically detected
Actor: User
Prerequisites:

- The user has opened the application and is on the give feedback tab
- The user is inside a room, that is registered by the system.

Main scenario:

1. The application successfully finds the room the user is located in.
2. The user answer's one or more questions about the indoor climate from the room
3. When all the questions are answered, the user receives a message with a confirmation

Alternative scenario:

- **1a.** The application could not determine the user's location
    1. A message is presented, informing the user that the location could not be determined.
- **1b, 2a, 3a.** User does not have internet access
    1. A message is presented, informing the user that internet access is required.

B.2 Check room

Description: User wants to see in which room it is located in
Actor: User
Prerequisites:

- The user has opened the application and is on the feedback or data tab.
- The user is inside a room, that is registered by the system.

Main scenario:

1. A message is presented, informing the user about the name of the room it is located in

Alternative scenario:

- **1a.** The application could not determine the user's location
    1. A message is presented, informing the user that the location could not be determined.
- **1b.** User does not have internet access
    1. A message is presented, informing the user that internet access is required.

B.3 Extract feedback-data

Description: Building admin wants to extract indoor climate data
Actor: System admin
Prerequisites:

- Admin is on homepage of Feedme

Main scenario:

1. Admin clicks button *admin page*
2. Admin is directed to login page
3. Admin types login information and presses button *login*
4. Admin is directed to new page with button *extract data*
5. A file with all feedback information with relevant format (csv, txt) is downloaded to the admin's computer.

Alternative scenario:

- **2a.** Admin has not put correct user information
    1. Admin is showed an error message requesting the admin to try again due to incorrect login info.

B.4 Extract feedback data about indoor climate

Description: A system admin wants to extract feedback-data about the indoor climate in order to improve and regulate the indoor climate.
Actor: System admin
Prerequisites:

- Admin is on Feedme web page for data extraction

Main scenario:

1. User presses room-location tab
2. User is presented with new screen displaying room location

Alternative scenario:

- **2a.** User is not connected to the internet

1. Room location is set to NA or – and a small message underneath the location text informs the user that internet is required to detect room location.

- **3a.** User is not situated inside a room

1. Room location is set to NA or – and a small message underneath the location text informs the user that the room location is unable to be detected and that the user should move closer to a room registered by the system.

B.5 Visualize feedback data about indoor climate

Description: User wants to see data about its own feedback or the feedback given by others.
Actor: User
Prerequisites:

- The user has opened the application and is on the *give feedback* tab

Main scenario:

1. User presses the *see data* tab
2. User is presented with the questions answered by the user based on what room it is located in.
3. User filters the question based on when the questions were given (today, last week, last month, all time) and who they were given from (itself or everyone), by pressing the buttons labeled with the filter options
4. User filters the question based on the room in which the questions are associated with, by pressing the room button in the top corner.
5. A pop up window appears, where the user can scroll through the buildings and rooms in which it has given feedback.
6. User presses ”save changes”
7. The pop up closes and the user is presented with the answered questions based on the filters applied.
8. User presses one of the questions
9. A pop up is shown, containing a pie chart information about the answer options of the pressed question and each answer option has a percentage of how many times it was answered.

Alternative scenario:

- **2a, 3a, 4a, 5a, 6a, 7a, 8a, 9a.** User is not connected to the internet

1. User is presented with a message, that there is no internet connection

- **2b** Location could not be determined

1. User is presented with the questions answered by the user based on all the rooms it has given feedback in.

- **2c, 3b, 4b, 5b, 6b, 7b, 8b, 9b.** There are no feedback based on the applied filters

1. User is presented with a message, that there was no given feedback and encourages the user to provide feedback

APPENDIX C

Back End Server Testing

All: 232 total, 232 passed 2.67 s

Collapse | Expand

- Should return 500 — passed — 5 ms
- **/api/auth** — 279 ms
  - **POST /** — 279 ms
    - should return 400 if email not set — passed — 35 ms
    - Should return 400 if password not set — passed — 9 ms
    - Should return 400 if password invalid — passed — 3 ms
    - Should return 400 if password wasn't correct — passed — 81 ms
    - Should return 200 if email and password valid — passed — 75 ms
    - Should return 400 if user did not exist — passed — 3 ms
    - Should return json web token that can be decoded to valid mongoose _id — passed — 73 ms
- **/api/beacons** — 58 ms
  - **POST /** — 45 ms
    - 400 if random parameter in body is parsed — passed — 18 ms
    - Should accept uuid value — passed — 15 ms
    - should return 403 if user with unauthorized role (0) tries to post beacon — passed — 6 ms
    - Should post new beacon with right parameters — passed — 6 ms
  - **GET /** — 13 ms
    - Should filter for building if query string parameter parsed — passed — 13 ms
- **/api/buildings** — 199 ms
  - **POST /** — 31 ms
    - 401 if json token not provided in header — passed — 7 ms
    - 400 if name not provided — passed — 6 ms
    - should have user as admin on newly posted building — passed — 13 ms
    - should return 403 if user not authorized with login role >= 1 — passed — 5 ms
  - **DELETE /:id** — 57 ms
    - Should return empty array of buildings after building was deleted — passed — 9 ms
    - Should also delete all rooms in that building — passed — 11 ms
    - Should delete reference to room for questions posted in that building — passed — 15 ms
    - Should not delete questions posted in other buildings — passed — 10 ms
    - Should still have room that does not belong to the deleted building — passed — 8 ms
    - Should return 403 if user was not admin on the building — passed — 4 ms
  - **GET /:id** — 32 ms
    - Should return only one building — passed — 6 ms
    - Should return 404 if building was not found — passed — 4 ms
    - Should return feedbackCount if withFeedbackCount query set — passed — 10 ms
    - Should set correct feedbackCount — passed — 12 ms
  - **Get /** — 79 ms
    - Should return building with room — passed — 6 ms
    - Should return array with rooms — passed — 7 ms
    - Should only return buildings that requesting user is admin on when query set — passed — 10 ms
    - Should return 403 if user was not admin but tried to get all buildings — passed — 4 ms

Should get all buildings when user was admin and did not set any query parameter	passed	10 ms
Should return 403 if non admin user tries to get buildings from where feedback from a different user was given	passed	7 ms
Should return buildings where another user has given feedback	passed	15 ms
Should get buildings where feedback was given	passed	9 ms
Should get building from another admin when proper query parsed	passed	11 ms

/api/feedback 1.19 s

POST / 50 ms

should return 401 if token not provided	passed	4 ms
should return 401 if invalid token	passed	2 ms
401 if invalid token	passed	3 ms
400 if roomId not provided	passed	3 ms
400 if question not provided	passed	4 ms
400 if questionId not set	passed	3 ms
400 if one question was not from same building as room	passed	12 ms
400 if question array length was not the same as room question array length	passed	0 ms
Should return 400 if question was from other room than feedback	passed	8 ms

should return 200 if all fields provided correctly	passed	11 ms

GET requests 1.14 s

GET / 1.07 s

Should return array with one feedback	passed	1.02 s
Should return 400 if bad query parsed	passed	6 ms
Should only return one length array when user query parsed	passed	7 ms
Should return only feedback from given room if query parsed	passed	8 ms
Should only return feedback within a month when time query parsed in url	passed	9 ms
should only return feedback within a year when year restriction set in query	passed	8 ms
should only return feedback within a week when week restriction set in query	passed	8 ms
should only return feedback from today when today-restriction set in query	passed	9 ms

GET /questionStatistics 30 ms

Should return array with answer object	passed	5 ms
Should return array with timesAnswered property	passed	7 ms
Should return valid timesAnswered for question	passed	10 ms
Should also regulate timesAnswered if user filter (query) parsed in url	passed	8 ms

GET /answeredQuestions/ 36 ms

Should return all unique questions with feedback	passed	6 ms
question in array should have value	passed	5 ms
Should have a property timesAnswered	passed	8 ms
Should return times answered	passed	9 ms
Should be possible to send queries and limit the result	passed	8 ms

/api/questions 280 ms

GET / 63 ms

Should return 401 if token not provided	passed	3 ms
401 if wrong token format sent	passed	3 ms
404 if user was not found	passed	5 ms
400 if roomId not provided	passed	3 ms
400 if roomId was wrong format	passed	4 ms
404 if room was not found	passed	4 ms
should return questions array with 1 length	passed	6 ms
should return 200 when getting questions array with valid parameters	passed	4 ms

should return question object with roomId field	passed	6 ms
should only return questions from detected room/building	passed	8 ms
Should return answer options	passed	5 ms
Should return answer options with name	passed	5 ms
Should return answer options with id	passed	7 ms

GET /active 8 ms

Should only return active questions	passed	8 ms

POST / 154 ms

401 if token not provided	passed	2 ms
401 if token not valid	passed	2 ms
404 if user was not found	passed	3 ms
400 if roomId not provided	passed	5 ms
400 if roomId not valid	passed	4 ms
400 if question posted in rooms of different buildings	passed	10 ms
404 if roomId not found	passed	4 ms
400 if value not provided	passed	4 ms

Should return 403 if user not authorized role (1)	passed	4 ms
403 if user not admin on building	passed	5 ms
Should return 400 if two or more answer options were not given	passed	4 ms
Should return 400 if answeroptions did not have minimum 2 elements	passed	3 ms
should return question object with proper room id	passed	28 ms
should only return 1 length array when posted question for two different rooms	passed	45 ms
Should automatically set isActive to false if not set	passed	8 ms
Should have answer options when question posted	passed	12 ms
Should trim answerOptions	passed	11 ms
PATCH /:id change isActive of question		11 ms
Should return 400 if isActive not provided	passed	4 ms
Should change isActive	passed	7 ms
DELETE /:id		44 ms
Should only contain one question in db after deleted one	passed	13 ms
Should return 403 if user was not admin on building with question	passed	10 ms
Should delete all answers for that question	passed	13 ms
Should return id of deleted question	passed	8 ms
/api/rooms		111 ms
POST /		30 ms
should return 401 if no token provided	passed	6 ms
Should return 403 if user not authorized with login role >= 1	passed	6 ms
should return room with proper building id	passed	9 ms
400 if random parameter in body is passed	passed	5 ms
should return 400 if name not set	passed	4 ms
Should return 403 if user not admin on building	passed	0 ms
GET rooms in building /fromBuilding/:id		8 ms
Should only return 1 room	passed	4 ms
Should return 403 if user was not admin on building	passed	4 ms
GET /		39 ms
Should return 403 if no query parsed and user wasn't admin	passed	4 ms
Should only return rooms that user has given feedback on, when query parsed	passed	9 ms
Should only return rooms, that another user is admin on	passed	7 ms
Should return 404 if user was not found	passed	5 ms
Should return 403 if user tried to get rooms another user is admin on but was not admin himself	passed	3 ms
Should only return rooms where user is admin if query admin=me parsed	passed	6 ms
should return array with length 2 of rooms when 2 rooms are posted	passed	5 ms
DELETE /:id		34 ms
Should return array of length 0 when room deleted	passed	6 ms
403 if user was not admin on building where room exists	passed	6 ms
Should delete questions that have only reference to deleted room	passed	10 ms
Should not delete references to other rooms	passed	12 ms
/api/signalMaps		243 ms
POST /		230 ms
Should not throw error	passed	44 ms
Should not throw error either	passed	50 ms
Should return 400 if neither roomId or buildingId provided	passed	5 ms
Should return new signalmap with one length array of beacons	passed	8 ms
Should have reference to room	passed	7 ms
Should return 400 if one of the beacons doesn't exist in the system	passed	4 ms
Should return 400 if one of the rssi arrays did not have the same length as the other's	passed	3 ms
Should set isActive to false by default if room not provided	passed	14 ms
Should set isActive to true if roomId provided	passed	8 ms
Should estimate room if roomId not provided	passed	18 ms
Should estimate correct room when nearest neighbor is a tie	passed	21 ms
Should not throw error if beacon was in client beacons array but not in servermap	passed	27 ms
Should return 400 if no signalmap was posted an a room estimation was requested	passed	6 ms
Should return 403 if roomId provided and user was not authorized	passed	5 ms
Should return 400 if room was not found	passed	4 ms
Should return 403 if user was not admin on building where signalmap is posted	passed	6 ms
PATCH /confirm-room/:id Confirm room		9 ms
Should return updated signal map with isValid = true	passed	5 ms
Should return 404 if signal map did not exist	passed	4 ms
GET /		4 ms
Should return array with correct beaconIds	passed	4 ms

/api/users 278 ms
- GET / 26 ms
  - Should not return password passed 16 ms
  - Should return 403 if user was not admin passed 7 ms
  - Should return 401 if token not provided passed 3 ms
- POST / 242 ms
  - Unauthorized user 13 ms
    - 400 if random parameter in body is passed passed 3 ms
    - Should have user role 0 when no email+password provided passed 4 ms
    - Should be a valid mongoose id decoded by returned json web token passed 6 ms
  - Authorized user 229 ms
    - Should create user with authorized role if valid email and password provided passed 75 ms
    - should return 400 if email invalid passed 2 ms
    - Should not allow two users to be created with the same email passed 77 ms
    - Should return json web token in header that can be decoded to valid mongoose _id passed 75 ms
- PATCH /makeAdmin 10 ms
  - Should return 403 if user was not admin on building passed 4 ms
  - Should return 401 if no token provided passed 1 ms
  - Should return updated user with adminOnBuildings updated passed 5 ms

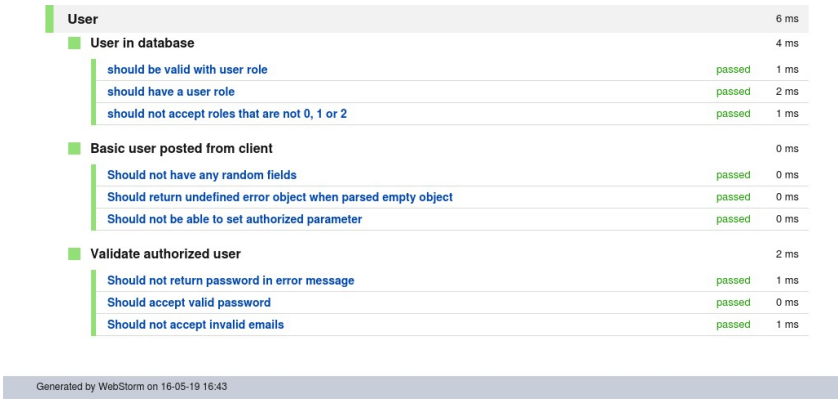
Answers 2 ms
- Answer in database 2 ms
  - should be valid with question passed 1 ms
  - should have a value passed 1 ms
- Answer from client 0 ms
  - Should not allow random parameters passed 0 ms
  - Should validate successfully when all parameters parse passed 0 ms

auth 0 ms
- admin 0 ms
  - should call status with 403 if user not admin passed 0 ms
  - Should call next when user admin passed 0 ms
- auth 0 ms
  - should call status 401 if id was not valid passed 0 ms

Feedback in database 3 ms
- should be valid with question, room and user passed 0 ms
- should have a question id passed 1 ms
- should have a user id passed 1 ms
- should have a user passed 1 ms
- should have an answer passed 0 ms

KNN algorithm manager 3 ms
- Initialization 2 ms
  - Should throw error if type is not a string passed 1 ms
  - Should throw error if dimensions below 1 passed 0 ms
  - Should throw error if some of the points did not have correct dimension passed 1 ms
  - Should throw error if initial points is not at least one-length array passed 0 ms
  - Should throw error if an initial point did not have both id and vector array passed 0 ms
  - Should throw error if id of a point is not set passed 0 ms
  - Should set points array if provided correct input passed 0 ms
  - Should not throw error even when vector with 0 value was parsed passed 0 ms
  - Should throw if k was not set to a value above 0 passed 0 ms
  - Should have points to at least the amount of k passed 0 ms
- Calculate euclidean distance between two points 1 ms
  - Should throw error if the one of the two points do not have the correct dimension passed 1 ms
  - Should calculate distance passed 0 ms
  - Should also calculate distance in 3 dimensions passed 0 ms
- Find nearest neighbors 0 ms
  - Should return array of length 2 passed 0 ms
  - Should consist of the the 2 closest neighbors passed 0 ms
- Find point with maximum distance to new point 0 ms
  - Should estimate the right point to be at max distance passed 0 ms

Find closest point index and distance		0 ms
Should return index of min distance point	passed	0 ms
Check if point has valid type		0 ms
Should throw error if type is not valid	passed	0 ms
Should not throw error if type is valid (set in initialPoints)	passed	0 ms
Estimate point type		0 ms
Should throw error if point type was already set	passed	0 ms
Should return the point type with minimal distances if the nearest points have different types	passed	0 ms
Should return type that most nearest points has	passed	0 ms
Question Validation		6 ms
Question in database		6 ms
Should have valid room ids	passed	1 ms
should have an array of rooms	passed	0 ms
Should have an array with at least one room	passed	1 ms
should have a value	passed	1 ms
Should have a list of answer options	passed	1 ms
Should have object id in answer options	passed	0 ms
Should have object id in answer options	passed	1 ms
Should have boolean isActive	passed	1 ms
Should validate successfully if all parameters parsed	passed	0 ms
Question from client		0 ms
Should be validated successfully if all parameters parsed	passed	0 ms
Should return error if value not provided	passed	0 ms
Room validation		2 ms
Room in DB		1 ms
should be valid with building, location and name	passed	0 ms
should have a building id	passed	1 ms
should have a name	passed	0 ms
Room from client		1 ms
should have a buildingId	passed	1 ms
should be validated successfully if buildingId and name provided	passed	0 ms
should have name	passed	0 ms
Location estimation algorithm		9 ms
Estimate room		7 ms
Should estimate room to be signalMap 1	passed	2 ms
Should also work with reversed order	passed	2 ms
Should find correct room with 3 signalMaps	passed	1 ms
Should return correct room when client beacons length is shorter than server beacon length	passed	1 ms
Should also locate the right room when client posts more beacons than server has	passed	0 ms
Should estimate the correct room even though the nearest point is for another room	passed	1 ms
Should not throw	passed	0 ms
update nearest neighbors		0 ms
Should return array with new neighbor	passed	0 ms
should return non-updated array when newNeighbor is further away	passed	0 ms
Find the room that most neighbors point to		1 ms
Should return the roomId that most neighbors point to	passed	0 ms
Should return correct roomId when room2 has most neighbors	passed	1 ms
If more neighbors have max count the method should return the element with the lowest distance	passed	0 ms
find index of max distance neighbor		1 ms
Should return correct index of max distance	passed	0 ms
Should return index of undefined neighbor	passed	1 ms
align and fill arrays		0 ms
should return aligned beacons array	passed	0 ms
Should return aligned beacon array with array length 3	passed	0 ms
Should fill array with -100 if an item wasn't in beacon id	passed	0 ms
Should throw if unaligned is not at least one length array	passed	0 ms
Should throw if beaconIds empty	passed	0 ms
Validation of url id parameter		0 ms
should return 400 if id not provided	passed	0 ms
should return 400 if invalid id	passed	0 ms



User		6 ms
User in database		4 ms
should be valid with user role	passed	1 ms
should have a user role	passed	2 ms
should not accept roles that are not 0, 1 or 2	passed	1 ms
Basic user posted from client		0 ms
Should not have any random fields	passed	0 ms
Should return undefined error object when parsed empty object	passed	0 ms
Should not be able to set authorized parameter	passed	0 ms
Validate authorized user		2 ms
Should not return password in error message	passed	1 ms
Should accept valid password	passed	0 ms
Should not accept invalid emails	passed	1 ms

Generated by WebStorm on 16-05-19 16:43

$\sqrt{17}$
$\Omega \int \delta e^{i\pi} = -1$
{2.7182818284}
ϑφερτυθιοπσδφγηξκλ
χ^2
$\sum$