Cross-platform Mobile Development and Internet of Things

Developing a cross-platform mobile application using web technologies to interact with smart things
Abstract

Today more and more objects in our daily lives are getting connected to the Internet. This phenomenon is called the Internet of Things and is a way for physical things such as cars, buildings or even bus stations to get access and communicate with other objects using the Internet. The problem is that for every Internet of Things device, an application is often needed in order to communicate with these devices. Developing mobile applications in a separate programming language for each operating system can be an expensive and time consuming task.

In this thesis, we implement and evaluate a cross-platform mobile solution for users to interact with smart things using the advantages of web technologies. To compare previous findings in this area, two literature reviews has been performed to find out which is the state of the art on cross-platform mobile development frameworks and smart-things technologies used for interacting with physical objects. The result is a mobile application developed using PhoneGap and jQuery Mobile that interacts with iBeacons, where students inside a university building can get directions and schedules for different rooms.

The application received good results from a couple of usability studies, and performed well when measuring its performance. The outcome shows that web technologies that exist today are a viable solution to native mobile applications in terms of interacting with smart things such as tagging technologies.

**Keywords:** Cross-platform, smart things, web technologies, iBeacon, beacon, Evothings, physical web, BLE, Bluetooth, NFC, QR code, PhoneGap, jQuery mobile, Cordova, mobile development, iOS, Android
Contents

List of Figures

List of Tables

1 Introduction 1

1.1 Background .................................................. 1
1.2 Problem Definition ........................................... 2
1.3 Motivating Scenarios and Context .......................... 3
  1.3.1 Objective .................................................. 3
  1.3.2 Technologies .............................................. 3
  1.3.3 Example scenarios ....................................... 3
1.4 Purpose and Research Questions ............................ 4
1.5 Scope and Limitations ....................................... 4
1.6 Outline ....................................................... 4

2 Methodological Approach 5

2.1 Literature Reviews ........................................... 5
  2.1.1 Tagging Technologies .................................... 5
  2.1.2 Cross-Platform Frameworks ............................... 5
2.2 Iterative Prototyping Approach ............................. 5
2.3 Usability Studies ............................................. 5
2.4 Selection of databases ....................................... 6
2.5 Selection of keywords ........................................ 6
  2.5.1 Tagging Technologies .................................... 6
  2.5.2 Cross-platform Frameworks ............................... 6
2.6 Search method ................................................ 7
  2.6.1 Tagging Technologies .................................... 7
  2.6.2 Cross-Platform Frameworks ............................... 8

3 State of the art 9

3.1 Tagging Technologies ........................................ 9
  3.1.1 General Information ...................................... 9
    3.1.1.1 QR Code ............................................. 9
    3.1.1.2 Near Field Communication .......................... 10
    3.1.1.3 Bluetooth Low Energy . ............................... 10
  3.1.2 Power Consumption ...................................... 11
3.1.3 Indoor Positioning ....................................... 12
3.1.4 Use Cases ................................................ 13
  3.1.4.1 BLE Beacons .......................................... 13
  3.1.4.2 NFC ................................................... 14
  3.1.4.3 QR codes ............................................. 15
3.1.5 Security Concerns ........................................ 15
3.1.6 Summary .................................................. 16
3.1.7 Challenges Discovered ................................... 18
3.2 Cross-platform frameworks .................................. 19
  3.2.1 Software development kits and frameworks ............ 19
  3.2.2 Development approaches ................................ 20
  3.2.3 Hardware accessibility .................................. 21
List of Figures

2.1 Research method for tagging technologies. 7
2.2 Search method for cross-platform frameworks. 8
3.1 The different ID’s of the iBeacon advertisement [3]. 10
3.2 Experimentally-derived RSS reduction, with and without attenuation of a human body. 12
3.3 The picture shows how Phonegap uses layered interfacing to create mobile web applications. 24
3.4 Titanium Studio’s architecture. 25
4.1 A high level view of the system. 32
4.2 System architecture. 33
4.3 A flowchart showing the flow for the prototype. 34
4.4 Overview of the application’s use cases. 35
4.5 Code snippet showing the function that scans for beacons. 38
4.6 First version of prototype running on Samsung Galaxy S3 (Android 4.3). 39
4.7 TimeEdit web page rendered on Samsung Galaxy S3 (Android 4.3). 40
4.8 Code snippet showing the function for displaying schedule data in a table. 41
4.9 Updated schedule page running on Samsung Galaxy S3 (Android 4.3). 41
4.10 Second version of prototype providing directions to B3032 running on iPhone 4S (iOS 8.0.1). 42
4.11 JavaScript code for updating directions and images to a specific room. 43
4.12 Code snippet from the jQuery Mobile page containing HTML elements that are accessed and changed by JavaScript. 43
4.13 Main menu of the second version of the prototype. 44
4.14 Updated view of the map. 44
4.15 Third version of prototype showing footer menu and information page. 45

List of Tables

1.1 Research questions. 4
3.1 A technical information summary of the three tagging technologies. 11
3.2 A summary of the findings made in the literature review on tagging technologies. 17
3.3 A table showing supportability of native features that can be accessed from a cross-platform tool [8, 23, 49, 50]. 21
3.4 Development tools with their IDE support and targeted platforms. 22
3.5 Development tools and the languages they are used in as well as their licences. 23
3.6 A summary of the findings from this literature review on cross-platform frameworks. 28
4.1 The functional requirements that were identified. 31
4.2 The non-functional requirements that were identified. 32
5.1 User feedback from the first usability test. 47
5.2 Tasks for the usability testing. 49
5.3 The answers to questions 1-9 from users who used the Android phone first, followed by the iOS phone. 51
5.4 Answers to questions 10-14. 52
5.5 Time it took for each user to complete the study. 52
5.6 The answers to questions 1-9 from users who used the iPhone phone first, followed by the Android phone. ........................................ 53
5.7 Answers to questions 10-14. .................................................. 53
5.8 Time it took for each user to complete the study. ...................... 54
5.9 Summary of the usability study. The mean score is a total mean value on questions 1-9. ......................................................... 54
5.10 Start up times for the four different devices, presented in milliseconds (ms). 54
1 Introduction

The Internet of Things is a concept in which everything from washing machines to cars is connected to the Internet. To allow these real world objects to connect to the Internet and be controlled, the objects uses well known web standards. This part of the Internet of Things is the so called Web of Things. The Web of Things can be seen as the application layer to the Internet of Things, where it uses existing Web standards to make real-world objects a part of the Web [42].

The Web of Things often refer to physical objects, but since regular mobile phones and tablets nowadays contains specific hardware such as cameras and accelerometers, they are a part of Web of Things as well. Existing technologies such as QR codes and Near Field Communication (NFC) are examples that can be used to connect physical objects to the Web. However, there are some constraints to these technologies. The most obvious limitation is that they only work on short distances. QR codes must be able to fit in a specific dimension, and NFC has a working range of approximately 10 centimeters [43].

In later years a new technology called Bluetooth Low Energy (BLE) has emerged as an extension to Bluetooth Classic. BLE maintains the same range as Bluetooth Classic while its power consumption is significantly reduced [44]. It also has the possibility to communicate without needing to pair devices together. Therefore by using BLE, a new area of use has emerged in the form of beacons. Beacons are able to broadcast data to other BLE compatible devices, which has opened the door to a large set of new use cases. These three technologies can all be used to "tag" physical objects.

But with a number of different platforms available on the mobile market, developing an application for all of them can be a very expensive and time consuming task. Mobile web applications are one solution to this problem [45]. Today there are a lot of different mobile development frameworks which allows access to the devices platform APIs [47]. This makes it possible to implement a web based mobile application that can access the native features on a mobile device.

1.1 Background

In 2020 it is predicted that up to 26 billion units will be installed and connected to the Internet, which is a huge increase compared to the 0.9 billion units installed in 2009 [1]. This makes the IoT a very relevant topic where a lot of development research is yet to be made.

There is a lot of discussions and predictions going on about what will be the lead technology to enable the IoT. QR codes have been used frequently for a while now, especially in Asia, but has had some problems gaining popularity in the rest of the world, and especially in the US [11]. Some of the reasons behind this are believed to be the requirement of an installed application together with issues of being in line of sight and getting focus on the codes [6]. NFC is emerging quickly in mobile devices, where research and user surveys are showing that NFC is a good candidate for making physical objects connected in tourism, museums and retail [38,39,48]. BLE beacons are the latest candidate of the three, and are seen as the most promising technology to many [6, 18].

With the introduction of iOS 7, Apple announced a new technology called iBeacon [3]. iBeacon uses BLE to broadcast a signal and can be used to determine if a device is within a certain area. When the device receives the signal it may trigger an app to perform a certain action, such as an advertisement or an offer on a specific product.
Developing a mobile application can mainly be done in three ways. As a native application, where you use the same code language as the operating system of which the phone is using [36]. As a web application, where often standard web technologies are used, such as HTML, CSS and JavaScript [46]. The third option is a hybrid approach, where the application is developed using web technologies, and then wrapped within a layer of native code that allows it to be run just like a native app [36]. The native approach can be very costly if the aim is to launch the application on several different platforms [16]. A web application is run through a web browser, which means that any device with a browser can run it [35, 37]. Though what native applications lack in portability they make up for in performance and their access to a device hardware components, such as camera or GPS [37]. In this thesis we will take the hybrid approach, where a literature review on mobile frameworks will decide which framework to use as a “wrapper” to our web application.

1.2 Problem Definition

There are a lot of benefits developing a mobile application in its native language. Properties such as great performance and customization abilities are two of the most used arguments of why a developer would use a native language to code an application. But one downside of native development is that the application only runs on one specific platform. This has a big impact on both developing time and cost.

Android and iOS, owned by Google Inc. and Apple Inc. respectively, combined have their operating systems installed on 92.3% of all existing mobile devices [4]. Since Android is not restricted to run on Google’s devices exclusively, it reaches a large amount of mobile device manufacturers and therefore has 70% of the market share. Apple does not allow any usage of their operating systems except on their own hardware, which makes them target only 17.3% of all operating systems shipped in 2013 [4]. Developing an application exclusively to iOS would therefore result in a huge loss of reachable customers. Knowing that, it is easy to understand that there definitely is a demand for a development approach that reaches multiple platforms.

With an increasing amount of IoT platforms it becomes harder for those who develop applications that uses these platforms, and the need for an open framework for the WoT is becoming more and more obvious [58]. The WoT enables physical devices to connect themselves to the Web, which allows them to share their services in an easy way [2]. The motivation behind this thesis is to investigate and compare a number of cross-platform mobile frameworks to see which one that is best suited for interacting with a tagging technology via the WoT. A tagging technology is defined as a technology used to "tag" physical objects, meaning you can put them on anything and they can then deliver information to users who wish to interact with it. By conducting two literature reviews, it is intended that we should get a better insight in cross-platform development and also how various existing technologies may be used to make regular objects “smart”. The literature reviews will be followed by development of a prototype using the state of the art cross-platform framework and tagging technology.

The main purpose of the prototype is to help students to get a better navigation and scheduling experience inside a university building. The problem with most of today’s existing navigation apps is that they just do not work indoors, often due to poor reception of GPS signals. Many universities still use printed schedules and maps to guide teachers and students to their specific classrooms. We will use the prototype to see if the navigation and scheduling experience can be improved by letting it interact with the state of the art
smart technology. A usability study will then be conducted to test and verify the prototype. Performance testing will also be done to compare the different operating systems and to ensure that the prototype provides a good user experience.

1.3 Motivating Scenarios and Context

Tagging technologies have a huge number of possibilities and usage areas. In a school building for example, they can be used to make a whole building “smart” by placing tags around the building and providing different services. Below are example scenarios describing how tags can be used in a school environment.

1.3.1 Objective

The objective of these scenarios is to help us identify problems that students and teachers may face when entering a school building they are not familiar with for the first time. Problems such as locating a specific classroom or knowing if a room is booked or not. With tags placed at strategic places around the buildings, users can locate a specific room and get directions to it, or get a real time view of a room’s schedule to see if it is booked or not.

1.3.2 Technologies

The technologies used could be any tagging technology (such as the ones mentioned in 1.1: NFC, BLE or QR codes) together with any relatively new mobile phone. The phones will of course have to be equipped with the technologies mentioned above. All map images should be stored locally on the phone, to reduce the amount of data needed. Schedules however needs an internet connection in order to fetch live versions of a room’s schedule. The data is obtained by web scraping the website on which the schedules are displayed.

1.3.3 Example scenarios

Below are four scenarios that show examples of how the application would function.

**Scenario 1**

A student is going to a lecture in a building for the first time. The student does not know where the lecture room is located. A tag is placed at the entrance of the building, and as the student enters the building he/she scans the tag which then provides information on where the room is located and how to get there.

**Scenario 2**

A student is walking in the building but is not sure where in the building he/she is currently located. The student scans a tag and asks for a map of the building, which will also display the student’s current location.

**Scenario 3**

A student needs to know where a room is located. Directions are not needed, as the student is already relatively familiar with the building. The student scans a tag and asks for a map of the desired floor, and is then presented with an overview of the rooms.
Scenario 4
A teacher wants to see if a specific room is available for use. He/she approaches the room and scans the tag, which provides a schedule of all bookings that day.

1.4 Purpose and Research Questions

The research questions are listed in the table below.

| RQ1 | What is the state of the art on cross-platform mobile web frameworks and technologies used for tagging physical objects? |
| RQ2 | How could these technologies be used in a school environment to improve indoor navigation and room scheduling? |
| RQ3 | Are there any differences in usability between different mobile operating systems when using a cross-platform development approach? |

Table 1.1: Research questions.

1.5 Scope and Limitations

In the literature review on tagging technologies, a couple of limitations were set. First, the whole technology of BLE will not be taken into the comparison, but focus will be on BLE beacons. The reason for this is that we want to focus on tagging technologies. And while BLE can do this, it can also do a lot more. So it would not be a fair comparison to the other technologies. The search will be limited to studies released from 2010 to 2015. The reason for this is that BLE was released in 2010.

A comparison to a native application would be ideal, but because of time constraints this will not be possible. It was also decided to limit the prototype to the two largest platforms today, Android and iOS.

1.6 Outline

The rest of the report is divided into the following chapters:

Methodological Approach: Here the research approach and the methods used for our literature reviews are described.

State of the art: The results of the literature reviews and challenges identified are presented.

Prototype Design and Implementation: In this chapter everything about the prototype is shown, from design to implementation.

Validation: The data obtained from the validation process is presented and analyzed.

Discussion: Reflections on the findings and contributions that this paper has achieved.

Conclusion: Recap of the whole report and concrete answers to the research questions.
2 Methodological Approach

The thesis mainly consist of two parts; a theoretical part and a practical part. The theoretical part contains two literature reviews. The research questions will be answered with the help of these literature reviews. That is, the results found in our reviews (and mainly the review on mobile web frameworks) will work as guidance to what frameworks we will use and how we will implement the application.

2.1 Literature Reviews

Two literature reviews will be performed in this thesis. The reason for doing a review on tagging technologies is to compare benefits and shortcomings on the before mentioned technologies. The other review, on mobile web frameworks, is made to identify which frameworks are optimal to our application that is to be developed, and can satisfy its requirements. RQ1 will be answered using the results from these reviews.

2.1.1 Tagging Technologies

The study will target three tagging technologies that is considered well known: BLE, QR codes and NFC. The goal of this study is to compare the three technologies in different aspects.

2.1.2 Cross-Platform Frameworks

The second study will focus on cross-platform frameworks. The goal of this study is to find one (or more) frameworks that suits our application’s requirements the best, so that optimal results can be achieved. Since there are a lot of different available cross-platform frameworks on the market at the moment, the survey will focus on those that uses web languages and still are under development and maintained.

2.2 Iterative Prototyping Approach

When developing the application an iterative prototyping approach will be used. A prototype is a smaller version of the final system that is being built [62], and should be seen as a proof of concept. This approach allows us to quickly receive feedback so that improvements can be made to our application.

2.3 Usability Studies

Usability testing is used to evaluate a product, with the goal of identifying any problems and measuring how satisfied the users are with it [63]. Usability studies will be performed to drive the development of the application forward, and also to validate the final version of the application.
2.4 Selection of databases

To be able to find relevant scientific literature and articles for usage with the literature review, a number of well known online databases were located and selected. We will use IEEE Xplore Digital Library and Association for Computing Machinery (ACM). By using student accounts provided by Linnaeus University, full-texts from both of the databases can be accessed without any need of extra registrations or payments. Thanks to recommendations from other students and the benefit of free access to all articles, we decided to use these two [7].

2.5 Selection of keywords

2.5.1 Tagging Technologies

The search was initially started using only the name of the technologies, “Bluetooth Low Energy”, “QR codes” and “Near Field Communication”. However, this resulted in a huge number of hits and we realized that additional keywords were needed. “Internet of Things” was combined with the name of a technology in an attempt to reduce the number of hits while still maintaining relevance. Searching the IEEE and ACM database with the keywords “Internet of Things” AND “<name of technology>” together resulted in 160 articles, where 26 were chosen after having read the title and abstract of each article.

2.5.2 Cross-platform Frameworks

To find relevant literature to the mobile cross-platform review, we started by using a large set of keywords: “cross-platform”, “mobile”, “web technologies” and “hybrid”. Using all of them together in one single search query resulted in no hits. Instead by limiting the amount of keywords to use one or at most two, the outcome became more satisfying. After some attempts with varied result, the most successful combination was to use the query: (mobile cross-platform) and search for Metadata only. The year range was set to only return articles written between 2000 to 2015. A total amount of 183 hits were found in the IEEE database, where around 100 abstracts were read and 10 enough relevant to be used for further reading and comparison. A similar query was used in the ACM database: Title: mobile AND Title: cross-platform which returned 146 hits, where 5 articles were used for further reading.
2.6 Search method

Below are the search methods described as flowcharts, one for each review.

2.6.1 Tagging Technologies

Figure 2.1: Research method for tagging technologies.
2.6.2 Cross-Platform Frameworks

Figure 2.2: Search method for cross-platform frameworks.
3 State of the art

This chapter will contain the two literature reviews. The first part is about tagging technologies, where a number of different aspects will be studied on three different tagging technologies. The second part is about mobile cross-platform frameworks, where five different frameworks will be compared and studied.

3.1 Tagging Technologies

This is the result of a literature review on three technologies currently used for tagging physical objects. Three different technologies were chosen:

- Bluetooth Low Energy (BLE) beacons
- Near Field Communication (NFC)
- Quick Response Codes (QR codes)

It should be noted that the information gathered about BLE is mostly information on Bluetooth 4.0, which was released in 2010. Bluetooth 4.1 was released in December 2013, and 4.2 was released in December 2014 [9], which is only a few months ago at the time of writing this thesis. Consequently, the amount of scientific information currently available on version 4.1 and 4.2 is very limited.

3.1.1 General Information

This section will give a brief background and introduction to each technology.

3.1.1.1 QR Code

The QR code is a so called matrix (or two-dimensional) barcode, and was created by Denso in 1994. In June 2000 it was approved as an ISO international standard (ISO/IEC18004) [10]. First used as a way to track parts in the automotive industry, it has grown to be a very popular way of tagging objects. The reasons for its popularity are many, but some of the biggest reasons include:

- Higher capacity compared to other bar codes
- Takes up a small space
- Free for everyone to create
- With the increase of smart phone users and QR code reading applications, the use cases increased beyond its original purpose

The way a QR code works is, a picture is taken using the camera on a mobile phone. The picture is then decoded using a QR decoding application, and an action is taken by the phone depending on what the decoded message contains. The message could for example be a URL, which redirects the user to said URL once decoded. Since the QR code is two-dimensional, it is read both vertically and horizontally. Compared to a one-dimensional barcode, a QR code only takes up about 10 percent of the space of a one-dimensional bar code containing the same data size [11].
3.1.1.2 Near Field Communication

NFC is a wireless technology using radio communication to exchange data. It was created as an attempt to bring the many different RFID standards together to make all technologies use the same standards. NFC supports three different types of operation modes:

- Peer-to-peer mode
- Card emulation mode
- Reader/writer mode

With the peer-to-peer mode, two NFC-enabled devices can exchange data with each other. The card emulation mode allows phones or other devices with NFC to act as smart cards, and work much like a regular smart card making it possible to do transactions such as purchases. Finally, the reader/writer mode works very much like a QR code does. An NFC tag can be placed on physical objects, allowing NFC-enabled devices to read the information stored in the tag [12].

3.1.1.3 Bluetooth Low Energy

Bluetooth Low Energy (BLE) was integrated into version 4.0 of the Bluetooth Core Specification in 2010. It is marketed as Bluetooth Smart, and was designed by the Bluetooth Special Interest Group (SIG). The major differences between Bluetooth Smart and Bluetooth Classic is the significantly lower power consumption while still being able to communicate over a similar or even larger range. Bluetooth 4.0 can be seen as a superset that includes both BLE and classic Bluetooth, among others.

With its low power consumption and relatively long range, together with BLE tag hardware becoming cheaper and cheaper, BLE is considered one of the main technologies enabling the IoT. It does everything that the aforementioned technologies do, and more. It is an emerging technology that makes use of the low energy and range of BLE. Beacons are small devices that broadcast data via BLE to other devices within its range. There are a couple of different approaches to applying the BLE technology on beacons.

iBeacon

The iBeacon protocol [3] has taken a leading role in the beacon industry, with several manufacturers applying the iBeacon protocol to their BLE beacons. The iBeacon broadcast message is made up of three different ID’s, see Figure 3.1 below.

<table>
<thead>
<tr>
<th>Field</th>
<th>Size</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>UUID</td>
<td>16 bytes</td>
<td>Application developers should define a UUID specific to their app and deployment use case.</td>
</tr>
<tr>
<td>Major</td>
<td>2 bytes</td>
<td>Further specifies a specific iBeacon and use case. For example, this could define a sub-region within a larger region defined by the UUID.</td>
</tr>
<tr>
<td>Minor</td>
<td>2 bytes</td>
<td>Allows further subdivision of region or use case, specified by the application developer.</td>
</tr>
</tbody>
</table>

Figure 3.1: The different ID’s of the iBeacon advertisement [3].
Physical web
Another approach to BLE beacons is the Physical Web [41], where a URL is broadcasted instead of an ID. Both have their advantages and disadvantages, but one of the main differences is that iBeacon requires an application to process the message, whereas physical web does not need one. For iBeacon this could become a problem in the future, where each IoT device would require its own application.

With millions of devices it will simply be impossible to have an application for each one. R. Want et al. means that the physical web could be the solution to this. With an extension of the Web that allows IoT devices to broadcast a URL, this could connect the Web with the IoT [6].

Below is a table that summarizes some technical information about the three technologies.

<table>
<thead>
<tr>
<th>Technology</th>
<th>BLE Beacon</th>
<th>QR code</th>
<th>NFC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Technology</td>
<td>Short-wave length UHF radio waves, ISM band</td>
<td>2D matrix code</td>
<td>Based on the RFID standards</td>
</tr>
<tr>
<td>Frequency</td>
<td>2.4 - 2.485 GHz</td>
<td>N/A</td>
<td>13.56 MHz</td>
</tr>
<tr>
<td>Range</td>
<td>~ 150m</td>
<td>~ 10cm</td>
<td>&lt; 10cm</td>
</tr>
<tr>
<td>Data rate</td>
<td>~ 300kbps</td>
<td>7089 characters (version 40)</td>
<td>~ 424kbps</td>
</tr>
</tbody>
</table>

Table 3.1: A technical information summary of the three tagging technologies.

3.1.2 Power Consumption

When it comes to power consumption these technologies all share one thing, and that is that they consume little to no power. QR codes does not need any power, for obvious reasons, and so there is little to say about that technology in this section.

NFC tags does not require any external power source either. When an NFC enabled device initiates communication with a tag, power is taken from that device [13]. So when it comes to power consumption and potential lifetime, BLE falls short. However, even though QR codes and NFC tags can easily outlive BLE beacons, BLE technology allows beacons to live for a relatively long time and it is not as much of a concern as it may seem.

BLE is constructed much like classic Bluetooth in many ways. But classic Bluetooth has a rather high power consumption, which makes it a bad choice for the new hardware technologies that are emerging, such as sensors, beacons and other hardware objects that are powered by batteries. Another difference is the amount of data that is transmitted. With classic Bluetooth constant interaction between devices is normal, for example a Bluetooth headset where audio streaming is necessary. This comes with high data transmissions as a result. BLE does not send the same amount of data and does not do it as frequently. This was another reason why BLE was developed. With BLE, they changed the physical layer of classic Bluetooth and simplified its state machine in order to allow asynchronous communication and lower power consumption [17].

Another thing that was changed in order to achieve lower power consumption was the data throughput, which is significantly lower than classic Bluetooth. BLE comes with
three different power consumption modes: ultra-low peak, average and idle, and can run for at least six months or even years on normal coin-cell batteries [13].

Estimote [19] promises a 3-5 year lifetime on their beacons, and Kontakt [20] says their beacon has an expected lifetime of up to two years on default settings (250 ms interval) on 24 hours daily usage, and up to six months when using iBeacon profile (100 ms interval).

### 3.1.3 Indoor Positioning

This section brings up how the technologies handle indoor positioning. It provides a comparative and rather abstract description of how the technologies deal with proximity, accuracy as well as advantages and disadvantages. Note that detailed information such as measuring principles and positioning algorithms is mostly outside the scope of this section.

There is a lot of research on indoor positioning being done, and when BLE beacons was introduced a lot of additional opportunities opened up. In comparison to WiFi, beacons can often be placed on any surface and does not take up a lot of space. WiFi networks are mostly concerned with getting a high coverage of a specific area, and is not optimized for indoor positioning. Beacons can be placed in such a way that optimizes indoor positioning.

The advertising mode of BLE, that was discussed in the previous section, allows devices to calculate rather accurate proximity to a location based on Received Signal Strength (RSS), especially when close to a beacon (within a metre). However, as the distance to the source becomes greater, the accuracy decreases. Because the signal strength decreases as the inverse square of the distance to the transmitter, as the distance increases accuracy will decrease, as shown in Figure 3.3 below [22].

![Figure 3.2: Experimentally-derived RSS reduction, with and without attenuation of a human body.](image)

It is the possibility for apps to measure the distance to beacons, together with the long range that is the main attributes which separates beacons from NFC and QR codes when it comes to positioning.

Faragher and Harle performed an experiment in which 19 BLE beacons were deployed in a 45m by 12m section of a building at the Computer Laboratory at the University of
In this experiment, an accuracy of 2.6 m was achieved 95% of the time. This was considered good results as they compared the accuracy they achieved using a WiFi tracking scheme. It should be noted that the WiFi infrastructure in that building was not optimized for indoor positioning, which would be the case in many buildings. But this just highlights one of the major advantages that comes with BLE beacons; that they can be placed anywhere in order to optimize their geometry and achieve best possible coverage for indoor positioning. They also argue that other factors contributed to an increased accuracy, such as their high power output and greater density of sources (beacons).

NFC works very much like BLE beacons in some aspects. The NFC tag can store data such as a URL, which is transferred to a mobile device once within range. It is then up to the mobile device to decide what to do with the data received. In [34], the authors propose an indoor navigation system using NFC tags, and motivates a number of advantages over other indoor positioning technologies. Cost is one major factor, as NFC tags are generally very cheap compared to WiFi and Bluetooth. It also gives a very accurate positioning, since it is easy to point out on a map where the tag is placed. However, it is not possible to provide dynamic, on-the-go positioning due to its very short range. Thus, an NFC indoor positioning system is not sufficient if tracking while moving is desired.

The same thing can be done with QR codes as with NFC when it comes to indoor positioning. There are however some differences that favor NFC. One advantage that NFC has is that it is possible to dynamically change what’s stored, and even cloud services that allows you to change it remotely. Whereas with QR codes if the data stored needs to be changed a new QR code has to be created. This means that if an NFC tag was to be moved to another location, it would be enough to edit the data stored. The QR code would however have to be replaced with a new one.

3.1.4 Use Cases

In this section we will discuss how tagging technologies may be (or are being) used. It will describe several use cases where these technologies have already been implemented, but also hypothetical use cases where we think they may be used one day.

3.1.4.1 BLE Beacons

BLE beacons has enabled a whole new way of interacting and communicating with users. With its accurate proximity capability beacons can offer customers a new experience in a number of scenarios and environments.

Apple was first with using their own iBeacon technology as they deployed beacons in all their 254 US stores in December 2013 [27]. Users who has the Apple Store app installed will now receive information, suggestions and offers as they enter the Apple stores.

In September 2013 Samantha Murphy Kelly wrote an article about how Major League Baseball (MLB) is adding BLE beacons to their stadiums in order to provide additional engagement for its visitors [28]. With iBeacon integrated into MLB’s ‘At the Ballpark’ app, it could give the users the possibility to display their ticket, give directions to their seat, give special offers, and much more. There are a lot of potential use cases that can be used within the retail section. The two examples mentioned above already demonstrates how beacons can be used to provide users with coupons, product information and advertising. In addition to all that, beacons can for example be used for remote payment and real-time messaging.

But beacons are not only for the customers; the owners of restaurants and retail stores
can make use of beacon technology too. By logging data on how customers move in a store, where the customer did or did not stop to look at a product, or offer quick feedback forms, owners can use this data to make their store more appealing for the customers.

While there is no doubt retail is the number one area of interest to deploy iBeacon (and other BLE-enabled) beacons, there are a number of other fields where beacons can be adopted. Public transport is one area where it could be utilized. By setting up beacons at bus stops or train stations, travelers can have easy access to the schedule. A payment option could also be implemented, allowing travelers to buy tickets directly from their phones.

Museums are already applying beacon technology to make a visitor’s visit more interactive. By putting up beacons next to a painting or a sculpture, the visitor could receive additional information once close enough to the beacon. It could for example play an informational video about the object.

Other places where beacons could be used includes:

- Airports
- Festivals
- Households
- Conferences
- and many more

3.1.4.2 NFC
NFC tags are today used in a number of different ways. And while NFC can also be used for wireless payment and peer to peer communication, focus will be on the NFC tags which includes the reader/writer mode of NFC.

What makes them so attractive on the market is their small size, cheap cost and fast interaction. Just like with BLE beacons, NFC tags can be placed on basically any surface. So called Smart Posters is one area where NFC tags has been explored. In [38], they created an application where NFC tags were placed on Smart Posters. One tag would contain the location of the poster, and other tags would contain points of interest, displaying directions and information about that point of interest. They performed a testing evaluation for several days in Córdoba city, where posters had information about companies located nearby. Users were asked to answer seven questions about their user experience with the application. Results showed that people were overall happy with their experience.

M. Blöckner et al. [39] developed a mobile application used as a museum guide that interacts with a dynamic NFC-display, and their user evaluation also showed positive results. In [52] NFC was tested on a university campus with a variety of use cases, including using it to buy coffee from a coffee machine and downloading a game from a poster. They received good response overall, but it was noticed that a lot of first-time users had problems figuring out how to use the technology.

Other research reports similar results with users experiencing problems when interacting with NFC tags for the first time. G. Broll et al [57] performed usability studies with users who had no prior knowledge about NFC. They compared execution time, number of errors and found that users given sufficient directions performed better compared to those who had no guidance.
NFC has another downside that should not be taken lightly. As of now, none of Apple’s iPhones support any NFC interaction, with the exception of iPhone 6 and iPhone 6 Plus which includes an NFC chip restricted to Apple Pay [53]. NFC is however restricted to Apple Pay only, and so they can not be used for interaction with NFC tags. This greatly reduces the number of users that can take advantage of the NFC technology, which of course is a big disadvantage for NFC.

3.1.4.3 QR codes
QR codes are mostly used as a way of advertising. It is not uncommon to see QR codes in newspapers, billboards, magazines and numerous products varying from food to electronics. QR codes have also been tested in museums, by placing the codes near artwork to give information about the artwork as well as information on where to go next [40]. In that paper, a two day trial was conducted. During this period 47 visitors scanned at least one QR code, and 18 of them scanned more than one. The overall response from the visitors was positive and thought they added more to the experience.

QR codes, together with NFC tags, have one advantage which BLE beacons cannot match, and that is that they can advertise in consumers’ homes. While BLE beacons are usually stationed in stores and other places, QR codes and NFC tags can be placed and printed in papers and magazines, effectively increasing their use cases. QR codes has the additional advantage of being free to produce, whereas NFC tags comes with a small cost which increase quickly when placing them in thousands of magazines. However, NFC seems to be prefered over QR codes in many ways. The need for an application to scan QR codes seems to be a big reason why many prefer NFC tags instead [48]. Another reason could be the to many less attractive aesthetics of the QR code, which is a bad thing for both the marketers and users.

3.1.5 Security Concerns
With all the potential use cases, beacons are expected to be used in a lot of places. Most of these places are public, and so the importance of security is crucial. The BLE specification contains a number of security features. Starting with the BLE specification, it has several security features included:

- Privacy protection
- Frequency hopping
- Mode control

With privacy protection, tracking devices for their location is not possible. Devices can still be detected, but it can not be identified or otherwise recognized by other devices that it has never been paired with [13].

Just like classic Bluetooth, BLE uses a frequency hopping scheme where the frequency is dynamically shifted between the 37 data transfer channels. It was designed to avoid as much interference as possible with other wireless technologies that also use the 2 GHz spectrum, but it also makes it harder for potential eavesdroppers to interpret the link, but not impossible.

Beacons does not collect data, they only broadcast data. In the case of iBeacon, it is the applications that is on a mobile phone that decides what should happen when it receives a beacon’s broadcast message. This means that the real security risks lies within the applications of the mobile devices, and not the beacons themselves. The Physical Web
project [41] takes a slightly different approach, where the beacon only broadcast a URL. This removes the risk of malicious applications, but adds the risk of false URL’s that can lead to bad websites.

QR codes and NFC share the same risk of malicious URI’s (Uniform Resource Identifiers). Both can easily be replaced if out in the open, and users should take great caution when scanning tags from an unknown source. In April 2012 David Maman did a simple experiment at Infosec UK, an information security conference. In the experiment, he placed a poster with the name of a famous security company offering the chance to win an iPad by scanning a QR code [51]. In total 455 persons scanned this code, and this was at a conference for security professionals. The code only contained a smiley face, but proved how easy it is to get people to scan potentially dangerous codes with a simple fake poster. In [54] the authors examine if a pop-up message that presents a warning to users of malicious QR codes will prevent users to proceed or if they will simply ignore it. A survey showed that most users would not ignore the pop-up message.

Some NFC tags have read/write memory, which allows smartphones to write data to a tag. The tag is initially writable by anyone, but to avoid other users from writing malicious data to tags, the user can place a security key on the tag. This means only the owner can write additional data or update it [59]. There are also NFC tags that does not allow data to be overwritten, which reduces the chance of someone altering the stored data. Eavesdropping on a passive NFC tag and an active NFC device is theoretically possible, but is only possible on very close distances [60]. This makes it very hard for others to eavesdrop on a conversation.

3.1.6 Summary

As a result of this literature review, we have gained a good understanding on how each technology works and how they can be used. There is no clear answer as to which technology is the superior one when it comes to enabling the IoT. Each of them has their own advantages and disadvantages depending on their use case. BLE does however have the widest range of use cases, and there is not much QR codes or NFC can do that BLE cannot. A lot of research on BLE beacons has targeted indoor positioning, and it is very likely that that is where BLE will be the most dominant technology. For more simple use cases, such as providing a single URL, NFC and QR codes are sufficient, and sometimes even prefered due to their unlimited life time. Both comes with some problems that may affect the number of users, such as the security risks with malicious QR codes and downloading QR readers from unreliable sources, and NFC’s incompatibility with the hugely popular iPhone brand.
<table>
<thead>
<tr>
<th></th>
<th>BLE Beacons</th>
<th>QR code</th>
<th>NFC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Power consumption</strong></td>
<td>Powered by coin cell batteries, beacons can function for several months up to several years, depending on broadcasting settings and type of battery.</td>
<td>Does not need any power</td>
<td>Passive NFC tags themselves are unpowered, but are powered by a radio frequency field generated by the device that initiates contact.</td>
</tr>
<tr>
<td><strong>Indoor positioning</strong></td>
<td>Possible to achieve good indoor positioning accuracy on moving targets, provided a sufficient amount of beacons and well implemented algorithms</td>
<td>Only static positioning by providing current position when scanning a code</td>
<td>Only static positioning by providing current position when scanning a tag</td>
</tr>
<tr>
<td><strong>Use cases</strong></td>
<td>Retail, public events, data collection, public transport, indoor positioning, airports, households, conferences, museums and many more</td>
<td>Advertisement (newspapers, magazines, billboards), museums, product information, business card, store a phone number</td>
<td>Retail, smart posters, magazines, product information, museums, households</td>
</tr>
<tr>
<td><strong>Security concerns</strong></td>
<td>In the case of iBeacon, only digits are broadcasted, and the security concerns lies with the application that reacts to these digits when received. For beacons broadcasting a URL, there is a risk of malicious URLs.</td>
<td>Both malicious codes and code reading applications exists. Methods, such as warning pop-up messages have been developed for minimizing the risk of scanning dangerous codes.</td>
<td>Tags can be locked, removing the risk of someone overwriting the real data with malicious data. However, the risk of someone replacing the tag is still present, just as with QR codes.</td>
</tr>
</tbody>
</table>

Table 3.2: A summary of the findings made in the literature review on tagging technologies.
3.1.7 Challenges Discovered

A number of challenges have been discovered during this literature review. The technologies each have different challenges, as they work in different ways. The challenges below are technology specific, meaning that a challenge may not apply to all three technologies. The following challenges were discovered:

First time users and usability
NFC is still an unknown technology to a lot of people, and a lot of research has reported usability issues with NFC tags. The biggest problem seem to be how to initiate contact with NFC tags. Due to their small size it can also be hard to discover tags when placed in non-optimal places. The other technologies does not face this challenge to the same degree. QR codes are today widely used, well recognized and simple to use. Beacons handles the initial contact automatically, and make use of the functionality of mobile applications which users is most likely already accustomed with.

Cross platform accessibility
While QR codes and BLE can be used on the majority of mobile devices, NFC is still not a part of Apple’s iPhone. An exception is the iPhone 6, but NFC is restricted to Apple Pay and is not capable of reading NFC tags. This results in huge loss of potential users, as discussed in 3.1.4.2.

Privacy
While interaction with QR codes and NFC is dependent on users making initial contact, beacons use the opposite approach. Abusing notification features on mobile devices can quickly become an unpleasant experience for a user if walking in to a store means being bombarded with notifications about special deals and offers. The ability of tracking users is also something that many users may feel invade on their privacy. However, some privacy concerns are addressed in the latest version of Bluetooth (4.2), which includes the need of users permission before tracking can be performed.

The need for an external application
The need of downloading an application for everything is something that repels a lot of users. It is thought to be one of the main reasons why QR codes are not as widely used as many thought it would be by now. Some BLE beacon implementations have tackled this challenge (Physical Web [41] for example), while others need applications in order to utilize their function (iBeacon). NFC can be used without any need of an application, which is a good thing for a lot of users.
3.2 Cross-platform frameworks

3.2.1 Software development kits and frameworks

A software development kit (SDK) is a set of tools that together can be presented in an environment when building applications for a certain platform or system. The included tools are often external libraries that are being used to communicate with specific parts of a system. Emulators and debuggers are also examples of tools that is included in the most common SDKs that targets mobile application development.

RhoMobile is a framework providing a set of products for building mobile cross-platform applications. The components Rhodes (Development), RhoConnect (Integration), RhoHub (Deployment) and RhoGallery (Manage) are together making up the framework which can be run on both Windows, Mac OS and Linux. RhoMobile’s main scope is to let the developer use a web programming language and by that be able to achieve a high level of productivity. To be able to make a mobile application, Singh and Palmieri [8] argues that it is necessary to have web experience whereas it is not needed to know any native programming languages or SDKs of each platform.

RhoMobile uses the architectural pattern Modern View Controller (MVC). An important part of the framework is the application generator which uses the MVC pattern to generate an initial application based on the developers model. It also creates subdirectories, template files and a controller for that specific model. The files are then compiled into a native executable application file that can be run on an emulator or a real mobile device. RhoStudio is the Integrated Development Environment (IDE) provided to users that do not want to use an already existing environment such as Eclipse, IntelliJ or Netbeans.

PhoneGap is another commonly used open-source mobile development framework owned by Adobe Systems. It was initially created and maintained by Nitobi Software but later bought by Adobe back in 2011. PhoneGap’s underlying software was developed under the name Apache Callback but later changed to Apache Cordova [15]. The main idea of PhoneGap is to use a common component between all mobile operating systems, which is the web browser. The framework therefore enables developers to build cross-platform applications using web languages instead of native languages. Unfortunately, PhoneGap does not provide a unique IDE to develop applications on all mobile operating systems. Developers have to execute their source-code on different IDEs, for example Eclipse when developing to Android and Xcode for iOS [9]. They do however provide an online “build service” where finished applications can be uploaded as a .zip archive. Compilation and packaging is then handled automatically online, so a compiled application will be ready for download in just minutes, ready for submitting to any app store [15].

Titanium, developed by Appcelerator, is a framework which is quite similar to PhoneGap. Applications created using Titanium have the possibility to reach native features in the same way PhoneGap does [24]. Titanium includes an SDK that provides different tools and an IDE named Titanium Studio. It is a very popular and widely used framework for building native applications using a web language [24].

Sencha Touch and jQuery mobile are two other frameworks that has a big focus on front-end development. Sencha Touch is a little bit more complex and powerful than jQuery Mobile since it contains SDK Tools that provides access to a subset of native features.
3.2.2 Development approaches

Today there are a lot of different mobile devices that are running different operating systems (OS). The operating systems are coded in different programming languages which means that applications has to be written in a language that is supported by that specific OS. Applications that are targeting the Android platform are written using Java. iOS uses two programming languages called Objective-C and Swift. Windows Phone is just like iOS compatible with multiple languages such as C# and C++.

There are three different approaches that can be used when developing applications for mobile devices. These approaches are:

- Native
- Web
- Hybrid

Building an application that directly targets a specific OS is called native development. Usually, the native code is compiled which makes the applications faster than using an interpreted language like JavaScript. If an application aims to target multiple different platforms, the native approach generates problems in forms of higher costs, more coding and therefor also more work [16]. In order to get rid of these problems, new development approaches have evolved.

Web and hybrid are two similar mobile cross-platform approaches that have become very popular in recent years, where both techniques uses web languages for building applications. A web application is primarily based on a JavaScript library, together with HTML5 and CSS3 to render the user interface [21]. A hybrid application is neither purely web-based nor native [15]. It means that the layout rendering is done in a web-view, and not by the native language of the OS. The hybrid application “wraps” in a web application into a native application [8].

Using a cross-platform approach offers many advantages to a developer. M. Palmieri et al. mentions some of the most important ones that these tools have brought [8]:

- Easier and faster development process compared to building a native application for each specific OS.
- Reduction of required knowledge and skills since most cross-platform frameworks uses easy to learn programming languages like HTML5 and JavaScript. No certain knowledge about OS specific application programming interfaces (APIs) is required.
- The ability of having one codebase and be able to compile the source code to multiple operating systems makes development and maintenance a lot easier. This reduces both time and cost.
- Possible to expand sales of applications to more markets and app stores to increase the gain for developers, business model owners, etc.

Applications developed by PhoneGap are hybrid [15]. Sencha Touch and jQuery Mobile creates web applications, but can work in conjunction with a hybrid app tool such as PhoneGap to package the result into into a native application [21]. Appcelerator Titanium compiles into fully native applications.
3.2.3 Hardware accessibility

GPS, accelerometer and camera are examples of hardware components that today are built-in to almost all mobile devices. These components has opened up the market for entirely new usage areas. Today, using a smartphone for navigation or photo purposes is just as common as making a phone call [8].

When developing applications in a native language, accessing these hardware features can be done easy through method calls directly to the underlying operating system. Using a web development approach, these kinds of calls can not be made that easy. Therefor several new technologies have evolved to close the gap between the hardware and software without needing to write any native code.

PhoneGap provides an environment where developers can create applications using HTML5, CSS and JavaScript and still be able to call native hardware features. This is done by a common JavaScript API that is included within the framework. It uses parts of native code to communicate with the operating system. The data that is returned from the system is then being passed back to the JavaScript and finally to the application that is running in a webview container [14].

PhoneGap also has the possibility to use external plugins. A plugin is the package containing injected code that enables the webview to communicate with the native platform that the application is running on [15]. An example of a PhoneGap plugin is Cordova Barcode Scanning, which uses the camera in a mobile device for scanning different barcodes such as a QR-code. The metadata encoded in the barcode can then be grabbed and processed by the application [15]. There is a wide range of different plugins available, specified on for example BLE and NFC communication. Most of them are open-source and available for free from Apache’s repository.

<table>
<thead>
<tr>
<th></th>
<th>PhoneGap</th>
<th>Titanium</th>
<th>Sencha</th>
<th>RhoMobile</th>
<th>jQuery Mobile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accelerometer</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>(✓)</td>
</tr>
<tr>
<td>Bluetooth Classic</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>(✓)</td>
</tr>
<tr>
<td>BLE</td>
<td>✓</td>
<td>△</td>
<td>✓</td>
<td>✓</td>
<td>(✓)</td>
</tr>
<tr>
<td>Camera</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>(✓)</td>
</tr>
<tr>
<td>Compass</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>×</td>
<td>(✓)</td>
</tr>
<tr>
<td>Contacts</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>(✓)</td>
</tr>
<tr>
<td>GPS</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>(✓)</td>
</tr>
<tr>
<td>File system</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>(✓)</td>
</tr>
<tr>
<td>NFC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>(✓)</td>
</tr>
<tr>
<td>Notifications</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>(✓)</td>
</tr>
</tbody>
</table>

Table 3.3: A table showing supportability of native features that can be accessed from a cross-platform tool [8, 23, 49, 50].

✓ = Direct native access

(✓) = Native access if combined with external development tool

△ = Requires a non-free external plugin
PhoneGap can be used together with Sencha or jQuery Mobile. Combining PhoneGap with one of these two frameworks opens up new ways of building applications. User interfaces (UI) can be created and rendered using Sencha’s API or jQuery Mobile and communication with native features will be handled by PhoneGap [21]. Titanium has its own specific API to connect with the native layer, so external frameworks such as PhoneGap is not needed [24].

3.2.4 Platform targeting

The table below shows each operating system the cross-platform development tools can be run on and which mobile platforms they targets [56].

<table>
<thead>
<tr>
<th>Development tool</th>
<th>IDE support</th>
<th>Platform targeting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phonegap</td>
<td>Windows, Mac OS X and Linux</td>
<td>Android, iOS, Windows Phone, BlackBerry, Symbian, WebOS</td>
</tr>
<tr>
<td>RhoMobile</td>
<td>Windows, Mac OS X and Linux</td>
<td>Android, iOS, Windows Phone, BlackBerry, Symbian, Windows OS</td>
</tr>
<tr>
<td>Appcelerator Titanium</td>
<td>Windows, Mac OS X and Linux</td>
<td>Android and iOS</td>
</tr>
<tr>
<td>Sencha Touch</td>
<td>Windows, Mac OS X and Linux</td>
<td>iOS, Android, BlackBerry, Windows Phone [25]</td>
</tr>
<tr>
<td>jQuery Mobile</td>
<td>Windows, Mac OS X and Linux</td>
<td>Android, iOS, Windows Phone, BlackBerry, Palm WebOS</td>
</tr>
</tbody>
</table>

Table 3.4: Development tools with their IDE support and targeted platforms.

3.2.5 Programming languages

As mentioned earlier, there are different approaches when developing mobile applications. Depending on which approach a developer decides upon to use, different programming languages will be used. When developing an application natively, it is very common that the language is based on object-orientation. Android applications are written in Java, iOS uses Objective-C and Windows Phone is compatible with multiple languages such as C# or C++ which are all object oriented. These languages often have a steeper learning curve compared to web languages.

Web browsers, both on mobile devices and desktops, today uses well known standards such as HTML5 and CSS3 [15]. These two technologies are the basis when developing web pages. The structure of a page is created using HTML, while the visual layout is made using CSS [64]. JavaScript, which is a browser-side programming language, can be used together with these technologies to make a web page dynamic.
The table below shows which programming languages each development tool supports and which licenses they are released under [29] [30].

<table>
<thead>
<tr>
<th>Development tool</th>
<th>Programming languages</th>
<th>License</th>
</tr>
</thead>
<tbody>
<tr>
<td>Phonegap</td>
<td>HTML5, CSS3 &amp; JavaScript</td>
<td>Free and Open Source under Apache License, Version 2.0</td>
</tr>
<tr>
<td>RhoMobile</td>
<td>HTML5, CSS3, JavaScript &amp; Ruby</td>
<td>Open Source under MIT License. (RhoElements, Cloud Build, Support, RhoConnect requires a non-free subscription.)</td>
</tr>
<tr>
<td>Appcelerator Titanium</td>
<td>HTML5, CSS3, JavaScript &amp; (AJAX)</td>
<td>Open source under Apache License, Version 2.0. (Enterprise and commercial licenses are available.)</td>
</tr>
<tr>
<td>Sencha Touch</td>
<td>HTML5, CSS3 &amp; JavaScript</td>
<td>Free Commercial License under GPLv3. (Sencha Touch Grid, Sencha Architect, and Sencha Cmd requires commercial license)</td>
</tr>
<tr>
<td>jQuery Mobile</td>
<td>HTML5, CSS3, JavaScript &amp; (AJAX)</td>
<td>Open Source under MIT License</td>
</tr>
</tbody>
</table>

Table 3.5: Development tools and the languages they are used in as well as their licences.

As the table shows, there are both free (open-source) and paid frameworks. Some of the frameworks do even provide different licenses depending on which features the developer needs. Using open-source as the development model for a project often attracts more developers who contribute with further improvement and bug fixes. Dedicated programmers often gather and create open communities. Most of the communities offer free community boards where active users can ask and answer questions. The boards are often very active and can be seen as way of offering free support. But to earn a profit on open-source software, the most typical business model is to sell support packages. Moreover, the earned money is often used to pay costs such as renting of web servers, online storage and traffic [31].
3.2.6 Design and architectural approach

PhoneGap’s architecture is mainly composed using a 3 layer approach. The picture is showing how its architecture uses “layered interfacing” [8].

![Layered Interfacing](image.png)

Figure 3.3: The picture shows how Phonegap uses layered interfacing to create mobile web applications.

The top layer called Web Application represents a mobile application and its source code. In this layer, development is made using HTML5, CSS3 and JavaScript. The layer below contains PhoneGap’s own JavaScript API and native APIs. All JavaScript code is written in the Web Application layer which means that the second layer is only responsible for interfacing between the JavaScript API and native APIs. The main purpose of this layer is to maintain relationships between each unique mobile operating system that uses a native API and the JavaScript API [8]. As mentioned earlier, the JavaScript API can then be used to access specific hardware features such as GPS, Camera or Bluetooth. Based on the operating system the application is targeting, the source code will be converted into native code such as Java or Objective-C.

Appcelerator Titanium has a similar architecture as PhoneGap. It also works like a bridge between the source code of an application and its native operating system. Figure 3.4 below shows Titanium Studio version 2.x architecture [32].
The top layer of the architecture contains the application itself. The application is written using pure JavaScript. At the most bottom layer, all different targeting operating systems are gathered. The application can either be compiled into a native application or a hybrid application using HTML5. Between the two layers is Titanium's SDK which contains APIs that can be accessed using JavaScript. A bridge named Kroll is then responsible for translating the calls into each operating system’s native function. For instance, when creating a button in Titanium, a real native button is actually created for each target OS [32].

RhoMobile uses the architectural pattern called Model View Controller (MVC). It divides an application into three different parts called model, view and controller. The model represents the business logic of an application. The data within a model can be presented in a view, while the controller handles requests from users. The purpose of the MVC is to separate all three layers from each other and at the same time to provide a “point of connection” between them [8]. RhoMobile therefore uses HTML, CSS and JavaScript for the view, but Ruby in the controller layer to make backend support [8]. A huge advantage of using MVC pattern is that each part can be modified independently which is great when you want to update one part of an application without affecting the others. Migration of legacy programs can also be done easier thanks to the separated layers [8].

Sencha Touch also supports building applications using a MVC structure, but it is not mandatory to do it [33]. Since jQuery Mobile only consists of front-end languages, it can be combined with other frameworks such as PhoneGap and be a part of its architecture.

### 3.2.7 User popularity & documentation

With the introduction of HTML5, new possibilities of creating mobile applications emerged. HTML5 supports interaction with various hardware sensors and features and can today provide a similar functionality as a native developed application. Since HTML5 also supports application cache and access to a device file system, an application can be run even when it is not connected to the Internet. Features like these are just a few examples of why HTML5’s popularity is increasing steadily among developers [35].

The capabilities of using HTML5 in the development process of building native mobile applications has opened up a new market for cross-platform development tools. Apache Cordova and its distribution named PhoneGap are examples of tools that uses HTML5.
They both have become quite popular to use and has a great documentation of technical literature [21][35]. The documentation is clearly structured and provides in most cases a quick- and a full example with source code. In some cases it also mentions problems with running specific methods on a certain platform [31]. PhoneGap is therefore a widespread choice for developers since it is very easy to use, has good flexibility and a straightforward architecture [21]. It is also open source, completely free to use and provides extras such as external plugins which has an impact on the popularity as well [37].

However, Titanium is just as popular to use even though it only uses JavaScript and not HTML5 to build applications. Titanium offers basic documentation freely available in Appcelerators developer center, while a subscription is needed to use their support. Additional functionality such as closed-source modules are also available with a licensed version. The quality of the documentation is good and contains numerous code examples, although most of them are minimalistic. Even though the documentation is good, the learning curve of the total framework is steep in the beginning due to a high degree of framework specific knowledge that has to be learnt before using it [31].

There is no actual leader on the market that fulfills all the general requirements of a cross-platform framework, but PhoneGap and Titanium are the ones that most developers prefers [21]. To build applications with RhoMobile, the developer requires knowledge of the programming language Ruby. Ruby is not as widely used compared to other languages such as PHP or Java. Another weakness with RhoMobile is that it only generates a native package and not native source code when building applications, which could cause limitations to further tweaking of applications [35]. Moreover, making changes to any parts of the JavaScript or HTML code requires a complete rebuild of the application [35], which is both time consuming and could possibly have a negative impact on user popularity. jQuery Mobile has some issues with its documentation. Although it presents both design options and all of its elements, there is hardly no examples with source code available which means that the developer in many cases requires to do further researching on its own [31].

3.2.8 Summary

By conduction this survey, we have got a better understanding of each framework’s capabilities and what makes them unique. It is clear that the mobile cross-platform approach have a lot of benefits to offer, although the ones compared in this literature review also have some disadvantages. Since all of them uses web languages, applications can be developed without any knowledge of native languages. They all compile to several different platforms which leads to less development costs and only one codebase in a single language. M. Palmieri et al. argue that the downside with using a JavaScript bridge between a native API is that is causes a loss of performance [8]. This could be critical when building highly demanding graphical- and 3D applications. Since the performance of the JavaScript APIs and the hardware components in devices have improved with time, the bridge-solution is now good enough to use for most applications that are being built today.
<table>
<thead>
<tr>
<th></th>
<th>Phonegap</th>
<th>RhoMobile</th>
<th>App. Titanium</th>
<th>Sencha Touch</th>
<th>jjQuery Mobile</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SDKs &amp; frameworks</strong></td>
<td>Builds applications using existing IDEs (Xcode and Eclipse)</td>
<td>Provides a set of different tools to build applications</td>
<td>Uses its own IDE and a set of tools to build applications</td>
<td>Provides SDK tools and a set of GUI components</td>
<td>Front-end framework providing pre-existing GUI components</td>
</tr>
<tr>
<td><strong>Development approaches</strong></td>
<td>Creates hybrid applications</td>
<td>Compiles into native applications</td>
<td>Delivers fully native applications</td>
<td>Web and hybrid (combined with PhoneGap/Cordova)</td>
<td>Web and hybrid (combined with PhoneGap/Cordova)</td>
</tr>
<tr>
<td><strong>Hardware accessibility</strong></td>
<td>Access to all common hardware features available</td>
<td>Does not support compass, otherwise full access</td>
<td>Access to all common hardware features except BLE (which requires external non-free plugin)</td>
<td>Full access to hardware features (if combined with PhoneGap/Cordova)</td>
<td>Full access to hardware features (if combined with PhoneGap/Cordova)</td>
</tr>
<tr>
<td><strong>Platform targeting</strong></td>
<td>Runs on the most common platforms today. (Android, iOS, Windows Phone, BlackBerry, Symbian, WebOS)</td>
<td>Targets the most common platforms today</td>
<td>Only targets iOS and Android devices</td>
<td>Limited to iOS, Android, BlackBerry &amp; Windows Phone</td>
<td>Targets the most common platforms today, except Symbian</td>
</tr>
<tr>
<td><strong>Programming languages</strong></td>
<td>Applications are being built using HTML5, CSS3 &amp; JavaScript. Based on open-source</td>
<td>Applications are being built using HTML5, CSS3, JavaScript &amp; Ruby</td>
<td>Applications are being built using HTML5, CSS3, JavaScript &amp; AJAX</td>
<td>Applications are being built using HTML5, CSS3 &amp; JavaScript</td>
<td>Based on HTML5, CSS3 &amp; JavaScript</td>
</tr>
</tbody>
</table>

Continued on next page
Table 3.6: A summary of the findings from this literature review on cross-platform frameworks.

<table>
<thead>
<tr>
<th></th>
<th>Phonegap</th>
<th>RhoMobile</th>
<th>App. Titanium</th>
<th>Sencha Touch</th>
<th>jQuery Mobile</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Design and architectural approach</strong></td>
<td>Uses a three layer approach where a JavaScript bridge communicates with native APIs</td>
<td>Uses modern-view-controller</td>
<td>Similar approach as PhoneGap. Uses a bridge to communicate with the OS</td>
<td>Supports modern-view-controller pattern, but not mandatory to use it</td>
<td>Uses pure front-end languages, but can be combined with Phonegap</td>
</tr>
<tr>
<td><strong>User popularity and documentation</strong></td>
<td>Very popular among developers. Has great documentation /technical literature that provides source-code examples</td>
<td>Requires knowledge of Ruby, which is not a very widely used language</td>
<td>Offers free basic documentation although a license is needed to use their support. Provides minimalistic code examples</td>
<td>A popular and widely used framework. Provides good documentation available for free at their website</td>
<td>Popular front-end framework to combine with external tools. Minor issues with documentation where users may have to research on their own</td>
</tr>
</tbody>
</table>

The following challenges were identified to the different frameworks:

- **Performance** - Even though JavaScript has increased its performance over time, the difference is still high compared to the native approach.

- **Development approaches** - Hybrid development is tied to each native OS specific browser. This means that whenever a user wants access to new features, they have to update the operating system which runs the app inside its webview. Taking the native approach instead forces the developer to learn several different programming languages if the application targets multiple different platforms.

- **User interfaces** - PhoneGap is hard to use for creating good user interfaces. Since it do not provide any ready-to-use GUI components, the user interface has to be completely written using pure HTML5 & CSS3. If using too many graphical animations or components, the app may start to act slow and becomes “choppy”. Combining it with jQuery Mobile could be a solution to this problem.

- **Licensing of tools** - Open-source frameworks often attracts more developers that contributes with fixing bugs and evolve the software. Sencha touch and Titanium are both frameworks that costs. Without a valid license, the developer has no possibility of getting any maintenance or support.
• **Access to hardware features** - Titanium do not provide access to as many native features compared to the other frameworks. A non-free plugin is required to access Bluetooth Low Energy. Sencha touch also has some limitations with hardware accessibility and lacks support of compass.

• **Platform targeting** - All frameworks targets the most common platforms available today, although there are some minor difference between them. When targeting multiple platforms, the user may have to do some operating system unique tweaks with the code, which sometimes could be pretty time consuming. Identifying platform specific errors could also be difficult for inexperienced developers.

• **Programming languages** - The majority of the frameworks that were investigated are using HTML5, CSS3 and JavaScript. RhoMobile though uses Ruby which is not a very commonly used programming language. The developer therefore needs knowledge and understanding of object orientation which is both time consuming and difficult to learn.

### 3.3 Choice of tagging technology and framework

From the study on tagging technologies, it is clear that BLE beacons brings more to the table than the other technologies. The prototype will therefore be based on this, and more specifically the iBeacon technology will be used. The choice of iBeacon is mainly due to the widespread use on a number of beacon manufacturers, with a lot of documentation available.

PhoneGap will be used to develop the prototype since it is open-source, has a large user base, provides good documentation and free support in forms of an open community. It is also very versatile and extendable by using any of the external plugins that are available for free. To make the prototype get a good native look-and-feel experience, jQuery Mobile will be used on top of PhoneGap as providing ready to use front-end components.
4 Prototype Design and Implementation

This chapter contains everything related to the prototype. This includes requirements, design, use cases, implementation details and other related information.

The scenarios in 1.3.3 together with the findings of our literature reviews are what is motivating the design and implementation of the prototype. A total of five beacons will be used for this prototype. Three beacons will be placed outside classrooms, and will from now on be referred to as classroom beacons. The two other beacons will be called direction beacons, as they will be placed in other parts of the building.

4.1 Requirements

From the literature review on tagging technologies the following requirements were identified:

- One of the biggest concerns with beacons is that users will be spammed with notifications, ultimately leading to uninstalling the application or turning off Bluetooth all together. It is important to keep these notifications to a minimum.

- One of the major advantages of BLE beacons is its longer range compared to other tagging technologies. This should be utilized to get the most out of the technology.

- How fast beacons should broadcast depends a lot on the use case. For indoor positioning the intervals should be very short, to get more precise measurements. For this prototype however, that is not required. The broadcasting interval of the beacons should be 1000ms, which is the time recommended by Sensorberg in their BeaCfg application.

The review on cross platform frameworks identified the following requirements:

- It is still tough to develop graphical user interfaces to PhoneGap with the look and feel of a native application using only web languages. A user running the application should despite that not be able to tell if it has been developed using a native or hybrid approach.

- Even though all cross-platform frameworks targets several different platforms, the application may behave differently between operating systems due to different implementations of their internal web browsers. The application should work similarly on the two most popular platforms today (Android and iOS).

- Some frameworks do not have access to specific native hardware features, or requires a non-free license to use them. It is necessary to have access to the most basic features such as network, notifications and vibration. Bluetooth Low Energy 4.0 is also required to be able to interact with smart things.

The requirements for the prototype application was identified from the scenarios described in 1.3.3:

- To get real time information about a room’s booking times, the application shall provide easy access to a room’s schedule.

- When entering a new building it can be difficult to know where everything is located. To address this, the application shall include maps of every floor of the building, as well as the possibility to get directions to rooms.
### 4.1.1 Functional Requirements

The table below contains a more formal view of all the functional requirements that has been identified.

<table>
<thead>
<tr>
<th>ID</th>
<th>Summary</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>System must be able to detect beacon signals up to 5 meters for direction beacons.</td>
<td>Upon entering a building, users should be able to detect a beacon without having to search for it.</td>
</tr>
<tr>
<td>2</td>
<td>System must be able to detect beacon signals up to 1 meter for classroom beacons.</td>
<td>Classroom beacons are easier to locate, and so a shorter range is required. The lower range is also needed to reduce interference with other beacons for classrooms that are located close to each other.</td>
</tr>
<tr>
<td>3</td>
<td>System must display schedule for a room where a beacon is placed.</td>
<td>To give the user a fast and easy way of accessing an up-to-date schedule as well as removing the need of using paper to display schedules.</td>
</tr>
<tr>
<td>4</td>
<td>System must display a map of the building, including room numbers and a pointer showing where the user is currently located.</td>
<td>To give a user the possibility to find a room of the user’s choice, and/or find out where the user is currently located.</td>
</tr>
<tr>
<td>5</td>
<td>System must display directions in the means of a drawn line on a map, from a user’s current position to a room of the user’s choice.</td>
<td>To give a user who is not familiar with the building an easy way of locating a room with the help of a suggested path.</td>
</tr>
<tr>
<td>6</td>
<td>System must provide clear instructions on how to operate the application.</td>
<td>With BLE beacons being a relatively new technology, novice users may find it difficult or confusing on how to use the application.</td>
</tr>
<tr>
<td>7</td>
<td>System must only use notifications when notifying that a user has arrived at its final destination.</td>
<td>Too many notifications can quickly become annoying, and should therefore be kept to a minimum.</td>
</tr>
<tr>
<td>8</td>
<td>The broadcasting interval of the BLE beacons should be 1000 ms.</td>
<td>A short interval will drain the battery quicker, and is mostly needed when indoor positioning is required. A too long interval however might make the user experience worse if a user has to wait too long for a device to find a signal.</td>
</tr>
</tbody>
</table>

Table 4.1: The functional requirements that were identified.
4.1.2 Non-functional Requirements

The table below contains all the non-functional requirements.

<table>
<thead>
<tr>
<th>ID</th>
<th>Summary</th>
<th>Rationale</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Should operate on Android and iOS.</td>
<td>These are the two main operating systems used today.</td>
</tr>
<tr>
<td>2</td>
<td>Should have the look and feel of a native mobile application.</td>
<td>In order to create the best user experience as possible.</td>
</tr>
</tbody>
</table>

Table 4.2: The non-functional requirements that were identified.

4.2 Design

This image shows a high level view of the entire system. The application receives data from iBeacons using Bluetooth and allows a user to interact with it in different ways depending on which beacon a user have approached.

A device that is running the application is connected to the Internet and communicates with a server hosted by import.io. Import.io sends requests to Linnaeus university web servers through HTTP and scrapes its response (HTML-data). The data is then converted into a JSON-API on the server side of import.io. The application fetches the JSON data and presents it in a standard HTML table.

![Figure 4.1: A high level view of the system.](image)

32
4.2.1 System Overview

The system is designed following the Model-View-Controller (MVC) architectural pattern. It consists of three parts: the model, the view and the controller. The model represents the data, the view is a visual representation of the data, and the controller manipulates the data.

The view consists of a single html file called index.html. This file consists of several different <div> tags. Only one <div> can be visible at any time, and it is the controller classes that decides which <div> should be displayed. The rest of the divs are hidden with the help from CSS.

The controller is made up of four JavaScript files. The app.js file contains most of the application’s logic: it scans for beacons, reading the UUID, Major and Minor values and decides which <div> should be displayed based on these values. It also controls the process of displaying maps of the building and giving directions to a user, making sure the right image is displayed depending on a user’s current location. The schedule files uses an Import.io API created manually by us. It fetches schedule data in JSON format, parses it and displays it in a table format.

The model contains the data, which consists of images (maps, directions) stored on the device, and the data for every schedule. The schedule data is retrieved in JSON format from Import.io which is stored on their cloud servers. Import.io gets the data from TimeEdit by extracting it directly from the website.

4.2.2 Use Cases and Prototype Flowchart

This section brings up the use cases for the prototype as well as a flowchart for the prototype. The flowchart in figure 4.3 shows the flow of the application and the different choices a user can make. It starts when the application is launched by the user. The application starts scanning for beacons and continues to do so until one is found. Once a beacon is found, the main menu will be displayed. If the beacon that is found is a classroom beacon, three options will be available in the menu: show schedule, show map and get directions. If the beacon found is a direction beacon, only show map and get directions will be available in the menu. When a button is pressed, the user will be taken...
to its respective page. The page will be displayed until the user presses the back-button, which will return the user to the first page and the application will start scanning for beacons again.

Figure 4.3: A flowchart showing the flow for the prototype.

For all of the use cases, the following preconditions are required:

- iOS device with Bluetooth 4.0 or later version (iOS)
- Android 4.3+ (Android)
- Bluetooth is activated
- The application is installed on the device
Use case 1: Enter beacon area
This use case starts when a user enters an iBeacon area. While the device is out of range of any beacon, the application will search for beacons. When within range, a menu will be displayed with two or three different options, depending on which beacon has been approached (see use cases 2-4). The steps below describes the use case.

1. User starts the application
2. The application scans for beacons
3. User moves within range of a beacon
4. A menu is presented on the screen
5. Use case ends

Use case 2: Show map
When the user selects the ‘Map’-button in the menu, a map of the building will be displayed with a mark indicating the user’s current position. Depending on which beacon the user interacts with, the appropriate map will be shown. The use case starts at point 4 from use case 1.

1. Use case 1 is executed
2. User selects the ‘Map’-option
3. A map is displayed on the device
   (a) User press the ‘back’-button, returning the user to use case 1, step 2
   (b) User exits the application
4. Use case ends

Use case 3: Show schedule
When the user selects the ‘Show Schedule’-button in the menu, a schedule of the room will be presented. Depending on which beacon the user interacts with, the appropriate schedule will be shown. This use case requires an active internet connection in order to fetch the schedule. The use case starts at point 4 from use case 1.
1. Use case 1 is executed
2. User selects the ‘Show schedule’-option
3. A schedule is displayed on the device
   (a) User press the ‘back’-button, returning the user to use case 1, step 2
   (b) User exits the application
4. Use case ends

**Use case 4: Get directions**
When the user selects the ‘Directions’-option in the menu, a pop-up menu will be displayed containing the rooms of the building. The user selects the room he/she wants directions to. A map of the building will be displayed, with a mark indicating the user’s current position. It will also have a drawn line showing where the user to go. This use case starts at point 4 from use case 1.

1. Use case 1 is executed
2. User selects the ‘Directions’-option
3. User selects a room
4. A map with current position and a drawn line indicating where the user should go is shown
5. The user arrives at the indicated location
   (a) If the location is the final destination, go to step 6
   (b) If the location is another beacon, use case is repeated from step 4
6. Use case ends

**Use case 5: Exit beacon area**
If the user is exiting a beacon area as the application is in menu mode, the menu will remain on the display. If the user is in a view mode of use case 2,3 or 4 (map, directions or schedule), that view will be displayed until the user goes back to the menu or exits the application.

1. Use case 1 is executed
   (a) User exits area before reaching step 2 of use case 2,3 or 4 (use case 1, step 2 begins)
   (b) User exits area while being in a view mode, (go to step 2)
2. The current view is displayed until
   (a) user goes back to menu (use case 1, step 2)
   (b) user exits the application (go to step 3)
3. Use case ends
4.3 Development

This section is about the development of the application.

4.3.1 Tools and Technologies

In addition to the cross-platform framework used, some other tools were also used.

Sensorberg Beacons
The beacons used for the prototype is manufactured by Sensorberg. The reasons for choosing Sensorberg was mainly for their relatively cheap price and good documentation. Also the fact that they are located in Europe was important, as we wanted to reduce the risk involved with shipping.

Evothings Studio
When developing a mobile application it generally takes some time between a change has been made and displaying the result on the mobile device. To reduce the waiting times a tool called Evothings Studio [54] was used, which makes any changes appear instantly on your device.

Import.io
At the time of development, there were no TimeEdit API’s available for extracting data. Import.io [55] was used to achieve this and proved to be a sufficient tool for our needs. With Import.io it is possible to create APIs out of any webpage and get the desired data from it.
4.3.2 Implementation

First version
When implementation started, the first goal was to simply make it possible for our mobile devices search and detect an iBeacon signal. This proved to be a rather simple task, and did not take very long. Except that we needed to create our own HTML page containing all the front-end components, most of the JavaScript code could be found from the example projects that Evothings provided and only minor changes had to be made.

```javascript
app.startScanForBeacons = function()
{
    // The delegate object contains iBeacon callback functions.
    var delegate = new cordova.plugins.locationManager.Delegate()

    delegate.didRangeBeaconsInRegion = function(pluginResult)
    {
        app.didRangeBeaconsInRegion(pluginResult)

        // Set the delegate object to use.
        locationManager.setDelegate(delegate)

        // Start monitoring and ranging our beacons.
        for (var r in app.beaconRegions)
        {
            var region = app.beaconRegions[r]

            var beaconRegion = new locationManager.BeaconRegion(
                region.id, region.uuid, region.major, region.minor)

            // Start monitoring.
            locationManager.startMonitoringForRegion(beaconRegion)
                .fail(console.error)
                .done()

            // Start ranging.
            locationManager.startRangingBeaconsInRegion(beaconRegion)
                .fail(console.error)
                .done()
        }
    }
}
```

Figure 4.5: Code snippet showing the function that scans for beacons.

The power of the beacon was set to low, as to avoid having to be far away from the beacon, thus making the testing process faster and easier. As the basics had now been established, requirement 1 was the first requirement to be addressed. Two pages were created, one which is showing whenever the mobile device is out of range of the beacon, and the other called menu is showed as soon as the device is within range (Figure 4.6). The first step was adding a link to the menu page, a schedule for a specific room on TimeEdit. This link appears as soon as a mobile device is in range of the beacon.
(a) Searching for beacons.  

(b) Beacon found.

Figure 4.6: First version of prototype running on Samsung Galaxy S3 (Android 4.3).

This proved to be very simple as well. However, the TimeEdit web page is not very mobile friendly, and it certainly does not give the application a native feel (as required by the non functional requirement 2).
Second version
To get a more native look on the application, the HTML5 based framework jQuery mobile was added. Since the schedule view provided by TimeEdit was not very suitable for mobile devices, the data would have to be extracted from the website so that we could design our own view. TimeEdit allows a lot of settings on a schedule view, and with the help from these settings it is possible to get an up-to-date version of the schedule with the wanted data every time the application is launched. Import.io was used to extract this data in JSON format, and then parsed into an array in JavaScript. The figure below shows how an array containing schedule data could be passed into a function to be displayed in a HTML table.
function organizeArrayData(arr) {
    var out = "";
    var today = "";
    var currentTime = new Date();
    today = '<b>Today ' + arr.results[0].timeright_value + ' - ' + currentTime.getHours() + ':' + currentTime.getMinutes() + '</b><br/>';
    out = '<table><thead><tr><th>Time</th><th>Name</th></tr></thead><tbody>
    for(i = 0; i < arr.results.length; i++) {
        out += '<tr><td>' + arr.results[i].time_value + '</td><td>' + arr.results[i].booker_name + '</td></tr>
    }
    out += '</tbody></table>';
    document.getElementById("schedule-table").innerHTML = out;
    document.getElementById("date-label").innerHTML = today;
    document.getElementById("roomnumber").innerHTML = 'B3032';
}

Figure 4.8: Code snippet showing the function for displaying schedule data in a table.

The picture below shows the updated schedule view. Only time and name of the person who booked the room is displayed, as we wanted a clean view without any unnecessary information. The real application will only display the schedule for the current day, but for testing purposes a whole week was included (as seen below) so that the table would always contain data.

Figure 4.9: Updated schedule page running on Samsung Galaxy S3 (Android 4.3).

Next, requirement 2 and 3 were addressed. The approach for them are almost the same, where both shows maps that are plain .png images but requirement 3 also shows the route to a specific room by adding lines on top of the map. It also provides directions in text to easier understand the route or if the user have problems interpreting the map.
The application dynamically switches images and text depending on where the user is located in a building. Each time a new iBeacon is being approached at the path, the device vibrates to alert the user that new information regarding the route is available. When the user finally arrives at the target destination, a notification is displayed and a short sound plays. Confirming the notification ends the navigation and brings back the user to main menu.

Updating navigation images and direction texts are handled by JavaScript. The application checks which page that is currently active on the screen by using a jQuery function: $.mobile.activePage.attr("mobile-page-id"); The returned data is stored in a variable called activePage. The image below shows code used for updating instructions to a user that is looking for directions to room B3032.
if(activePage == 'directionstob3032'){
  if(pageId == 'page-floor3'){
    hideDirectionPages();
    document.getElementById('dirb3032step2').style.display = 'block';
    document.getElementById('floorb3032').innerHTML = "Floor 3";
    document.getElementById('navtextb3032').innerHTML = "4. Go towards B3033 and turn left.";
  } else if(pageId == 'page-b3032'){
    hideDirectionPages();
    document.getElementById('dirb3032step3').style.display = 'block';
    document.getElementById('floorb3032').innerHTML = "Floor 3";
    document.getElementById('navtextb3032').innerHTML = "5. You have arrived at B3032!";
    navigator.notification.alert('You have arrived!', alertDismissed, 'B3032', 'Thanks!');
    function alertDismissed() {
      history.back();
    };
    navigator.notification.beep(1);
  }
}

Figure 4.11: JavaScript code for updating directions and images to a specific room.

<!--Direction page to B3037-->
<div data-role="page" id="directionstob3037">
  <div data-role="header" data-add-back-btn="true" data-back-btn-text="Back" data-position="fixed">
    <h1 id="floorb3037">Floor 1</h1>
  </div>
  <div data-role="content">
    <img id="dirb3037step1" src="images/navigation/entrance-to-floor3.png" width="100%" style="display: block;">
    <img id="dirb3037step2" src="images/navigation/floor3-to-b3037.png" width="100%" style="display: none;">
    <img id="dirb3037step3" src="images/navigation/b3037.png" width="100%" style="display: none;">
    <h1 id="navtextb3037"></h1>
  </div>
</div>

Figure 4.12: Code snippet from the jQuery Mobile page containing HTML elements that are accessed and changed by JavaScript.

As already shown in earlier images, changes were made to the styling of all pages including main menu. The jQuery mobile library was used for styling, and it was also used to add a header to the top of the page on every map and schedule view, where a back button was also included (see figure 4.10 above).
Third version
Some minor changes were made after conducting the first usability test (see 5.1). The map was rotated as suggested by one of the test users (see figure 4.14 below).

Figure 4.13: Main menu of the second version of the prototype.

Figure 4.14: Updated view of the map.
The map was also cropped to a better format, which makes the text instructions visible without needing to scroll the page. Some texts were also updated and are now displayed in bold for easier reading. These minor changes gave a clearer view and made it easier to follow directions.

A footer was added where you can navigate between scanning for beacons and an information page. The information page was added after receiving feedback from the first usability test (see 5.1).

Figure 4.15: Third version of prototype showing footer menu and information page.
5 Validation

In this chapter our work is validated, and the results will be analysed and presented. The validation process of the prototype consists of two different usability tests.

5.1 First Usability Study

The first usability test involved three persons that were allowed to test the application. This test was done in order to receive feedback on the prototype, so that changes could be made before the final, more formal usability test was performed.

The test consisted of a quick introduction on what beacons are, what the application does and how it works. The user was then introduced to a scenario, similar to scenario 1 in section 1.3.1. The user was asked to start at the entrance, where the first beacon was placed. Once within range of the beacon, the user was asked to locate room B3032 on the third floor by using the ‘directions’-feature on the application. A second beacon was placed in a stairwell, where a new direction image would be displayed. The third beacon was placed outside B3032. When arriving at the classroom, the user was asked to try the ‘show schedule’-feature.

After they had completed the scenario, the users were asked a few questions about the application:

1. What was good about the application?
2. Was there something you thought was bad or confusing with the application?
3. Is there something that could be improved? And if so, how?
4. Do you have any suggestions on additional features that can be added to the application?
<table>
<thead>
<tr>
<th>Question</th>
<th>Test person 1</th>
<th>Test person 2</th>
<th>Test person 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Everything was good overall.</td>
<td>The schedule feature was very good.</td>
<td>The schedule feature was very convenient and useful.</td>
</tr>
<tr>
<td>2</td>
<td>When arriving at the stairs, it was hard knowing where to go as the beacon was placed in a bad way. The instructions were not obvious to find (on iPhone), as one had to scroll down to see them.</td>
<td>The directions instructions was sometimes unclear.</td>
<td>The first directions image was rotated in a weird way, and was hard to read (user tried to rotate screen to get a better view).</td>
</tr>
<tr>
<td>3</td>
<td>The stairwell beacon could be placed in a better way. The instructions on where to go could be clearer, and also suggested placing instructions above the image.</td>
<td>Clearer instructions when arriving at the stairwell beacon.</td>
<td>Clearer instructions. Rotate map image to make it easier to read.</td>
</tr>
<tr>
<td>4</td>
<td>Could not think of any.</td>
<td>A feature to notify maintenance personnel about errors/faulty equipment in a particular room. A feature allowing you to book a particular room.</td>
<td>Adding the possibility to zoom the map images.</td>
</tr>
</tbody>
</table>

Table 5.1: User feedback from the first usability test.

The overall response from all three users were positive. We observed that there seemed to be some initial confusion on how it works, but this was expected due to the technology being very new. To address this we decided to add an ‘Information’ page, where the application’s functionality is explained in more detail.
Analyzing the provided feedback, these are the changes we decided to make based on the suggested improvements:

- Rotate the image at entrance when asking for directions.
- Make instructions clearer
- Add information page

These are the changes we tried to implement based on the suggested improvements, but did not work as expected:

- Zoom on map images. We attempted to implement this using two different solutions, but both had troubles working on Android. The first one was about zooming all content on a page using specific meta data containing scaling options in the HTML head tag. It worked fine on iOS but had no effect on Android. The second solution we tried was using an external JavaScript library to make the image zoomable inside a div box. Although it did work on both devices, the performance was very poor and the image was rendered in a lower resolution. We therefore decided not to use any of them.

- Placing instructions above the images. This resulted in bad user experience as one now had to scroll down to see the image. This improvement also became irrelevant after the map image was cut, which led to instructions now being visible without having to scroll down.

These features were never attempted to be implemented, due to time constraints:

- A feature to notify maintenance personnel about errors/faulty equipment in a particular room.
- A feature allowing you to book a particular room.
5.2 Second Usability Study

Part of the validation test was a user usability study, where the participants were students of Linnaeus University. The test was performed in June 2015, and was divided into two parts. In the first part users were asked to test the application by following a number of tasks provided (the tasks are listed under 5.2.1). Each user had to complete the tasks two times, one time with an Android device and one time with an iOS device. The order of the devices were selected randomly for each user.

After completing the tasks the first time, the user answered a number of questions about their user experience with the application (questions 1-9 in 5.2.2). The same questions were then answered after completing the tasks a second time with the other device. Finally, a few questions were asked about the application as a whole (questions 10-14 in 5.2.2).

5.2.1 Conditions

The testing took place in the B-building at Linnaeus University. The version of the application used during the usability testing was the version referred to as Version 3 in 4.3.2. The devices used was a Samsung Galaxy S3 and an iPhone 4S. Every user were handed the same tasks, which was printed on a piece of paper that the user had with him/her during the entire testing process. The tasks are shown in the table below.

<table>
<thead>
<tr>
<th>Task</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Go to the center of the entrance area and wait for the application to discover the beacon. A menu will appear on your screen.</td>
</tr>
<tr>
<td>2</td>
<td>When the menu has appeared, choose the ‘directions’-option and select any of the three rooms listed. Then find the next beacon by following the instructions provided.</td>
</tr>
<tr>
<td>3</td>
<td>Once you have reached the second beacon, follow the updated map and instructions.</td>
</tr>
<tr>
<td>4</td>
<td>Find the third and final beacon. (NOTE: you need to be within 1 meter of the last beacon). Once the arrival notification appears, press ‘Thanks!’</td>
</tr>
<tr>
<td>5</td>
<td>Press the ‘Schedule’-button to see all the bookings for today for the room that you have arrived at.</td>
</tr>
<tr>
<td>6</td>
<td>Press the ‘back’-button to return from the schedule view to the main menu, then press the ‘Map’-button to see a map and your current position.</td>
</tr>
</tbody>
</table>

Table 5.2: Tasks for the usability testing.
5.2.2 Survey

The survey consisted of 14 questions in total. Questions 1-9 was about the application’s performance and its usability, and could be answered on a 1-5 scale (1= Strongly disagree, 2= disagree, 3=Neutral, 4=Agree, 5=Strongly agree). They were answered twice by every user. Questions 10-14 was asked at the end of the survey and only answered once. The questions are listed below.

1. The application discovered the beacon quickly once you entered the entrance area.
2. The map with directions and instructions provided to find the second beacon was easy to follow.
3. The map and instructions was updated quickly after entering the second beacon area.
4. The map with directions and instructions provided to find the third beacon was easy to follow.
5. The notification that appeared upon arriving at the room was displayed quickly.
6. The directions feature was useful.
7. The schedule feature was useful.
8. The map feature was useful.
9. The application was easy to use.
10. Do you think any feature(s) is missing?
11. Do you think the application would be useful on a larger scale (i.e. with more beacons in the building)?
12. Which version did you prefer?
13. If you answered Android or iPhone on the question above, please motivate why.
14. Other comments?

5.3 Performance Testing

To compare the application’s performance on the two operating systems, we measure the time it takes for it to start up. This was performed on four different devices, two of which were tablets and two were phones. To get the difference in time, we used plain JavaScript and its Date().getTime() function at initialization and when the page has finished loading. The getTime() function returns the current time in milliseconds and was used to calculate the difference. The results are presented in 5.4.2.
5.4 Results

5.4.1 Results from the Second Usability Study

The answers from questions 1 to 9 are presented below. Table 5.3 contains the answers from users who used the Android phone first. The first number represents their answers after having completed the tasks on the Samsung Galaxy S3. The second number are the answers after completing the tasks using the iPhone 4S.

The mean value for Android are equal or higher on every question, with the exception of question 5. Android scored higher on questions 1 and 3 which was about finding the beacons. It was a noticeable difference between the two phones on the time of which it took the application to find the beacons as the user approached them. Question 5 however, which is also about discovering a beacon, has a lower score on Android. The difference is that the application is programmed to discover the first two beacons on a longer range, while the third (placed outside the rooms) beacon is supposed to be discovered when within a shorter range. It seems that Android is better at finding more distant beacons, but on close range they are almost equal.

It should also be noted that bluetooth signals can be rather unpredictable. User 1 answered with a 5 on question 3 when using the iPhone, whereas user 2 and user 3 gave it a much lower score. The overall score on both phones are on the upper half of the scale, and is seen as a positive result.

<table>
<thead>
<tr>
<th>User</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
<th>Q5</th>
<th>Q6</th>
<th>Q7</th>
<th>Q8</th>
<th>Q9</th>
</tr>
</thead>
<tbody>
<tr>
<td>U1</td>
<td>5/5</td>
<td>4/5</td>
<td>5/5</td>
<td>5/5</td>
<td>5/5</td>
<td>5/5</td>
<td>3/3</td>
<td>5/5</td>
<td>5/5</td>
</tr>
<tr>
<td>U2</td>
<td>5/5</td>
<td>4/3</td>
<td>5/2</td>
<td>5/4</td>
<td>2/2</td>
<td>4/4</td>
<td>5/5</td>
<td>3/3</td>
<td>5/5</td>
</tr>
</tbody>
</table>

Table 5.3: The answers to questions 1-9 from users who used the Android phone first, followed by the iOS phone.
For questions 10 - 14 the responses were positive. User 2 and 3 suggested a dynamic map as an additional feature. User 1 and 2 answered that they did not notice any differences between the two devices. User 3 preferred the Android version, and motivated this with the faster discovery of beacons. All three users also agreed that the application would be useful on a larger scale.

<table>
<thead>
<tr>
<th>User</th>
<th>Q10</th>
<th>Q11</th>
<th>Q12</th>
<th>Q13</th>
<th>Q14</th>
</tr>
</thead>
<tbody>
<tr>
<td>U1</td>
<td>Compass</td>
<td>Absolutely, more is better.</td>
<td>There was no difference between the two</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>U2</td>
<td>A map that turns in relation to the direction you're facing</td>
<td>Yes</td>
<td>There was no difference between the two</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>U3</td>
<td>The map updating your position as you move.</td>
<td>Yes</td>
<td>Android</td>
<td>Android seemed to update faster</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 5.4: Answers to questions 10-14.

In the table below, the times it took to finish the tasks are presented. As suspected, the first time the users tried the application it took a longer time to finish the tasks, compared to the second time they did them.

<table>
<thead>
<tr>
<th>User</th>
<th>Android</th>
<th>iOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>U1</td>
<td>2:05</td>
<td>1:46</td>
</tr>
<tr>
<td>U2</td>
<td>2:46</td>
<td>1:57</td>
</tr>
<tr>
<td>U3</td>
<td>2:36</td>
<td>1:56</td>
</tr>
</tbody>
</table>

Table 5.5: Time it took for each user to complete the study.

Below questions 1-9 are again presented, this time answered by users who used the iPhone first, followed by the Samsung phone. Again, Android has a slightly higher average than the iPhone.
Table 5.6: The answers to questions 1-9 from users who used the iPhone phone first, followed by the Android phone.

Questions 10-14, answered by users who used the iPhone first.

<table>
<thead>
<tr>
<th>User</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Q4</th>
<th>Q5</th>
<th>Q6</th>
<th>Q7</th>
<th>Q8</th>
<th>Q9</th>
</tr>
</thead>
<tbody>
<tr>
<td>U2</td>
<td>5/5</td>
<td>4/5</td>
<td>5/4</td>
<td>4/5</td>
<td>4/4</td>
<td>5/5</td>
<td>4/5</td>
<td>5/4</td>
<td>5/5</td>
</tr>
</tbody>
</table>

Table 5.7: Answers to questions 10-14.
Table 5.8: Time it took for each user to complete the study.

<table>
<thead>
<tr>
<th>User</th>
<th>iOS</th>
<th>Android</th>
</tr>
</thead>
<tbody>
<tr>
<td>U1</td>
<td>2:30</td>
<td>1:20</td>
</tr>
<tr>
<td>U2</td>
<td>2:34</td>
<td>1:27</td>
</tr>
<tr>
<td>U3</td>
<td>2:04</td>
<td>1:17</td>
</tr>
</tbody>
</table>

It’s clear that the users that had the Samsung phone as their second device had a shorter time in total between the entrance and their decided destination beacon.

<table>
<thead>
<tr>
<th>Mean score</th>
<th>Android (1st)</th>
<th>Android (2nd)</th>
<th>iPhone (1st)</th>
<th>iPhone (2nd)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean time</td>
<td>2:29</td>
<td>1:21</td>
<td>2:23</td>
<td>1:53</td>
</tr>
</tbody>
</table>

Table 5.9: Summary of the usability study. The mean score is a total mean value on questions 1-9.

5.4.2 Performance testing results

Performance testing was done on four different devices:

- Samsung Galaxy S3 (running Android 4.3)
- Samsung Galaxy Note 8.0 (running Android 4.3)
- iPhone 4S
- iPad Air

Each device started the application 10 times. A mean value was also calculated for each device. The results are presented in the table below. All tests were performed in the B-building of Linnaeus University using their WiFi network. Time is presented in milliseconds (ms).

<table>
<thead>
<tr>
<th>Try</th>
<th>Samsung Galaxy S3</th>
<th>Samsung Galaxy Note 8.0</th>
<th>iPhone 4S</th>
<th>iPad Air</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1039</td>
<td>697</td>
<td>760</td>
<td>206</td>
</tr>
<tr>
<td>2</td>
<td>1014</td>
<td>720</td>
<td>718</td>
<td>207</td>
</tr>
<tr>
<td>3</td>
<td>1049</td>
<td>660</td>
<td>663</td>
<td>206</td>
</tr>
<tr>
<td>4</td>
<td>995</td>
<td>684</td>
<td>738</td>
<td>207</td>
</tr>
<tr>
<td>5</td>
<td>986</td>
<td>688</td>
<td>654</td>
<td>205</td>
</tr>
<tr>
<td>6</td>
<td>1013</td>
<td>701</td>
<td>652</td>
<td>207</td>
</tr>
<tr>
<td>7</td>
<td>1003</td>
<td>769</td>
<td>656</td>
<td>207</td>
</tr>
<tr>
<td>8</td>
<td>1007</td>
<td>693</td>
<td>697</td>
<td>202</td>
</tr>
<tr>
<td>9</td>
<td>1027</td>
<td>634</td>
<td>656</td>
<td>207</td>
</tr>
<tr>
<td>10</td>
<td>1036</td>
<td>671</td>
<td>695</td>
<td>209</td>
</tr>
<tr>
<td>mean</td>
<td>1016,9</td>
<td>691,7</td>
<td>688,9</td>
<td>206,3</td>
</tr>
</tbody>
</table>

Table 5.10: Start up times for the four different devices, presented in milliseconds (ms).
6 Discussion

This chapter discusses how the problem was solved, as well as the results of the study.

6.1 Problem solving

During the literature review on tagging technologies, we found that not a lot of research had been made to compare all three technologies to each other, as most papers usually involved one technology only. In this thesis, we have summarized important differences between three of the most popular tagging technologies, and brought forth their respective strengths and weaknesses.

With iBeacon being a new technology on the market, there was a gap in research made in this area so far, and even less so using web technologies and cross platform development instead of native code. This project has contributed in the cross platform web development area combined with iBeacon technology. The validation also showed that students are positive to the idea of implementing beacons in school buildings on a larger scale.

6.2 Method reflection

The literature reviews gave us good knowledge and a solid background on everything we needed to know. We then used that knowledge to implement an application. The application was developed using a prototyping approach. By using a prototype approach, we could use feedback from the usability tests to iterate and create several versions with new functionality.

The final usability study gave us a good understanding of what students thought about the application, and helped us find differences in usability between the two operating systems.

6.3 Result reflection

It was quite unexpected that it was such a noticeable difference between iPhone and Android in how quickly the beacons were discovered. We believe that the hardware in the different devices was the cause of this.

We also noticed differences in how close a device had to be from a beacon to be able to detect it. When having the applications detection range configured to only react to beacons that were in the “near” area as we had set up with the entrance beacon, the distance could vary from 1 to 5 meters before detecting the beacon. An explanation to this may have been due to the low transmission rate the beacons were using, which was set to 1 broadcast per second. An increase could perhaps have improved the outcome, but would have had a negative effect on the battery power. There is another method of measuring distance to a beacon using RSSI, which could be more accurate rather than using the three predefined statuses; far, near and immediate. This method has not been investigated in this thesis, but could be interesting to look further into.

Our application only works when it is running in the foreground on the iOS platform, which means that a user will only be notified about beacons nearby if he or she has the application active on screen. This is caused due to limitations in PhoneGap, but none of the users in the usability study seemed to be negatively affected by that. Although it had
no negative effect on our application, there might be use cases where this is considered a flaw.

The time it took for users to complete all the tasks on their first try does not differ a lot when comparing Android and iPhone. On the second try however, users completed the tasks quite a lot faster on Android compared to iPhone. Drawing a parallel between the times and and the scores on each device, we can see that the use of Android on the second try had the highest score as well as the lowest time. The results from the usability study shows that the user experience on both phones were good, with a slight preference towards Android. We believe this is mostly due to the differences in the time it took for the application to register nearby beacons as well as the minimum distance required, where Android did better.

When we measured the performance of the application, it was clear that the iPad Air running iOS had the best rendering times with an average of 206.3 ms. The iPad Air do have the best hardware, although factors such as available memory, number of applications installed on the device and different versions of the operating systems could have affected the outcome slightly.

We are overall happy with the produced results. The application got a high score on both platforms, and the feedback provided by the users were positive. We do believe that using web technologies to develop mobile applications is a very viable option to native development, and can save a lot of time and money when the resources to implement an individual application for each platform does not exist.
7 Conclusion

In this chapter the research questions are answered and future work is discussed.

With a quickly growing digital society where more and more devices and things get connected to the Internet, the future of the iBeacon technology seems to be bright. There are so many more areas other than the ones mentioned in this thesis where iBeacon technology could be useful. Since both NFC and QR codes have their limitations in distance, BLE beacons seems to be the next technology with a lot more possibilities. Combining it with a cross-platform framework such as Cordova, an application could easily be built targeting many platforms without the knowledge of any native programming language which would otherwise be required. This project has reviewed the current state of tagging technologies and cross-platform frameworks. The results from the literature reviews led to a prototype based implementation of a mobile application that used iBeacon technology and a framework called Cordova/PhoneGap. This application was then evaluated by conducting a usability study. The study was performed in a school building at Linnaeus University in Växjö. The results showed that users were overall pleased with the application on both operating systems, and agreed that a large scale implementation would be a good idea.

7.1 Conclusions

The three research questions are formally answered below.

• **RQ1 - What is the state of the art on cross-platform mobile web frameworks and technologies used for tagging physical objects?**
  
  To answer this, two literature reviews were made. The results from this showed that iBeacon technology and PhoneGap were in the top of their respective fields. PhoneGap was chosen because it is completely open-source, has a large user base, provides good documentation and free support in their open community.

  The long range of BLE beacons as well as its availability on both platforms were two of the major deciding factors. jQuery Mobile was also recommended to combine with PhoneGap, which we did to easier create a responsive, touch friendly and modern looking user interface. This question helped us decide what tools to use in order to find an answer to RQ1, and also contributed in comparing different technologies in the area of tagging technologies and cross platform frameworks.

• **RQ2 - How could these technologies be used in a school environment to improve indoor navigation and room scheduling?**

  This question was answered by implementing a prototype that utilized the tools found in the literature reviews, and then tested in a usability study. The prototype, together with iBeacons, helps users to navigate in a school building and to find an up to date version of a room’s schedule. Our usability study shows that students found the application to be very useful, and they agreed a large scale implementation of the prototype would be a good idea and beneficial especially for new students.
• *RQ3* - Are there any differences in usability between different mobile operating systems when using a cross-platform development approach?

This question was answered through our usability study and from our performance testing. Both phones received an average score of above 4 out of 5 on the usability study, which is considered very good results. Some differences were discovered between the two platforms. The most notable one being how fast a beacon was discovered by the application, where Android did better. Android also received slightly higher scores overall, which was mostly due to this. This may however not be implementation related, but rather due to hardware and their respective bluetooth technology.

The performance and rendering times on our prototype were good on both operating systems, where only the Samsung Galaxy S3 had an average starting time above 1 second. And even though cross-platform frameworks are slower compared to natively developed applications, it clearly had very little to no negative effect on the user experience.

### 7.2 Future Work

Combining iBeacons with a cross-platform framework could definitely be a good solution to allow students to navigate inside an university building. Nevertheless there are still improvements that can be made to our prototype. Some of the users’ feedback such as dynamically update a user’s position on the map as he or she moves or showing a compass for easier navigation, could be interesting to investigate further. Since mobile devices often suffers from high attenuation and poor reception of GPS-signals when using them indoors, our implementation of using iBeacon technology could be a solution to that. It would be easy to implement extra functionality such as tracking and logging of how users move inside the university building when they have the prototype running on their devices.

Comparing our application with a native application would be an interesting thing to do, and compare both performance and usability differences. This would probably show the difference between the approaches in an even clearer way.

The problem of having to install an application for each IoT device still remains. One option is the physical web, where a URL is broadcasted instead. Google very recently revealed their solution to BLE beacons, Eddystone [61] (so recently we have not been able to include it in this thesis), which seems very promising. Since Eddystone is both open-source and offers more functionality such as being able to transmit multiple frame types, some very interesting research can definitely be done in this area.
References


[31] H. Heitkötter et al., ”Comparing cross-platform development approaches for mobile applications”, in 8th International Conference, WEBIST 2012, Porto, Portugal, 2012


