

Qaiser Siddique

Context-aware digital services

Using BLE Beacons

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Thesis

19 April 2017

Author(s) Title	Qaiser Siddique Context-aware digital services using BLE Beacons
Number of Pages Date	48 pages + 1 appendix 19 April 2017
Degree	Bachelor of Engineering
Degree Programme	Information technology
Specialisation option	Software Engineering
Instructor(s)	Anssi Ikkonen, Head of Smart Systems and Software Engineering
<p>The goal of this final year project is to conceptualize context-aware digital services. During this project, proof of concept for a simple context-aware digital service called LetMeKnow was produced. In this report, context awareness, its use cases and implementation in digital world is discussed.</p> <p>Theoretical background covers the history of context awareness, Bluetooth low energy beacons and different beacon protocols. LetMeKnow digital service is an Android application that scans for Bluetooth low energy signal transmitted by a beacon. The service also consists of a backend system developed using The . The backend is responsible for collecting and providing contextual information of the service to users.</p> <p>Different kinds of contextual data were identified such as who is here, what is here and where you are. Using this contextual data, concepts for digital services that can connect the physical and the digital world to provide value to end users were created.</p> <p>It was concluded that collecting contextual data, analysing the data, optimizing constraints for the better user experience and most importantly providing a great value for the end users in a digital service is all possible in today's world.</p>	
Keywords	Beacons, Proximity, LBS, Firebase, Android, mobile, real-time database, Location Service, iBeacon, Eddystone

Contents

1	Introduction	1
2	Background	2
2.1	Ubiquitous Computing	2
2.2	Context Awareness	2
2.3	Location Services	3
3	BLE Beacons	4
3.1	Overview of Bluetooth	4
3.2	Bluetooth LE Beacons	4
3.4	Comparison between iBeacon and Eddystone protocols	5
3.4.1	iBeacons protocol	5
3.4.2	Eddystone Protocol	6
4	Use Cases of Beacons	9
4.1	Proximity and Location	9
4.2	Context awareness with Beacons	9
4.2.2	What is here?	10
4.2.3	Where are you?	11
4.2.4	Who is here?	13
4.2.5	Combination of Different Proximity Data Types	14
5	LetMeKnow Digital Service	15
5.1	Overview of the application	15
5.2	Technology selection	16
5.3	Architecture of the application	17
5.5	Components of the application	18
5.5.1	The Firebase Backend	18
5.5.3	Mobile Application	19
6	Implementation of LetMeKnow Service	21
6.1	Setting up an Android application project	21
6.2	Setting up the beacon	21
6.3	Beacons scanning	23
6.3.1	Permissions	24

6.3.3	Implementing AltBeacon Library	26
6.4	Setting up The Firebase SDK	30
6.4.1	Authentication with The Firebase	32
6.4.2	Database Design	34
6.4.3	Writing data	36
6.4.4	Reading Data	38
6.5	Implementing the Mobile UI	39
8	Observations	41
8.1	Capturing a Beacon signal	41
8.2	Spoofing beacons	42
8.3	Battery Optimization	43
8.3.1	Beacons battery	43
8.3.2	Battery of mobile devices	43
9	Discussions	44
9.1	Beacons technology	44
9.2	Context Awareness	44
10	Conclusion	45
	References	47
	Appendices	
	Appendix 1. Alkaline and lithium battery comparison	

Abbreviations

API	Application Programming Interface
BLE	Bluetooth Low Energy
EID	Ephemeral Identity
eTLM	Encrypted Telemetry
GPS	Global Positioning System
IDE	Integrated Development Environment
IoT	Internet of Things
ISM	Industrial Scientific and Medical devices
LBS	Location Based Service
OS	Operating System (OS)
PARC	Xerox Palo Alto Research Center
REST	Representational State Transfer
RSSI	Received Signal Strength Indicator
SIG	Bluetooth Special Interest Group
TLM	Telemetry
UC	Ubiquitous Computing
UI	User Interface
UID	Unique Identity
URL	Universal Resource Locator
USB	Universal Serial Bus
UUID	Universal Unique Identifier
UX	User Experience
Wi-Fi	Wireless Fidelity
XML	Extensible Markup Language

Table of figures

Figure 1 iBeacons advertising packet format [6]	5
Figure 2 Byte values of different Eddystone frame types [7]	6
Figure 3 Eddystone-URL frame type specification [7]	7
Figure 4 Eddystone URL scheme prefix Hex values [7]	7
Figure 5 Eddystone-URL HTTP URL Encoding [7]	8
Figure 6 Object around beacon high level architecture	10
Figure 7 Indoor navigation with beacons and floor plan	12
Figure 8 Simple architecture of for getting list of users nearby	13
Figure 9 Architecture diagram of different proximity data	14
Figure 10 Screenshot from LetMeKnow mobile application	15
Figure 11 High level architecture of LetMeKnow service	17
Figure 12 Authentication flow with The Firebase SDK	19
Figure 13 Beacon scanning flow with AltBeacon SDK	20
Figure 14 iBKS plus BLE Beacon	22
Figure 15 iBKS Config tool	22
Figure 16 Relation between transmission power and distance [9]	23
Figure 17 Requesting location permission in LetMeKnow application	25
Figure 18 Initializing a BeaconManager in Android application	27
Figure 19 RangeNotifier Implementation	29
Figure 20 Screenshot taken during the Firebase Android project setup	31
Figure 21 Enabling google authentication in the Firebase console	32
Figure 22 Authenticating a user with FirebaseAuthUI SDK in Android application	32
Figure 23 The Firebase UI Authentication flow for Android [12]	33
Figure 24 Screenshot from the Firebase database	35
Figure 25 User.java class in LetMeKnow Android application	36
Figure 26 Publishing data to the Firebase database	37
Figure 27 FirebaseRecyclerAdapter implementation in LetMeKnow mobile application	38
Figure 28 Wireframe of different screens in LetMeKnow application	39
Figure 29 UI updates for logged in users	40
Figure 30 Fluctuating visibility of a beacon [13]	41

Table of Listings

Listing 1 Installation of AltBeacon Android library with gradle [10]	26
Listing 2 Adding BeaconConsumer to Android Activity	27
Listing 3 Adding iBeacon layout to beacon manager	28
Listing 4 Three consecutive information log messages after a beacon was found	29
Listing 5 Unbinding beacon manager service	30
Listing 6 Adding the Firebase SDK to application gradle build file	30
Listing 7 Adding The Firebase UI SDK to application gradle build file	31
Listing 8 Getting user information from the Firebase Android SDK	34
Listing 9 Initializing the Firebase database instance in Android application	37

1 Introduction

The goal of this final year project was to conceptualize context-aware digital services. The use of hand-held devices over desktop is increasing. The user-centric approach to build digital services is the new trend. With mobile-first approach and user-first approach in the digital world, there are new challenges for software developers to solve. Developers today need to understand the user's context, where they are, what they need and how to solve their problems proactively.

The concept of contextual aware began with Mark Weiser's ground breaking paper "The Computer for the 21st Century" in 1991. On the basis of Weiser's research, Schilit used the term context-aware for the first time in 1993. Since then the internet of things (IoT) era began. In the last decade, development in technology like a small battery operated Bluetooth transmitter has made it possible for businesses to build context-aware services.

Google Inc. has launched a service called Google Application, previously known as Google Now. The main aim of this service is to understand the user's need in a context and provide them information before they ask for it. For example, Google application will inform their users about their flight schedules, gate numbers and other relevant information when it establishes that the user is traveling.

In this project, Bluetooth Low Energy (BLE) beacons were studied and used to determine the context of the user. A simple proof of concept Android mobile application was built. Possibilities of services that can be built on top of this proof were investigated. The basis of using beacons over other location services was analysed and evaluated before the project was started.

Furthermore, the project studied different kind of digital services that can be built using beacons. Examples of contextual data and context-aware services were also examined. The future and importance of beacons in building context-aware services was concluded.

2 Background

2.1 Ubiquitous Computing

Ubiquity is defined as ‘the fact of appearing everywhere or of being very common’. Mark Weiser invented the term Ubiquitous Computing (UC) around 1988, during his research as chief technologist in Xerox Palo Alto Research Center (PARC) [1]. Weiser claimed that computers in the 21st century will be ubiquitous, they will be so common that no one will notice, the network of electronic devices will be seamless [2].

The rapid pace development in the technology sector is making the technology itself meaningless. The metrics to measure is the relationship of the technology to us. Weiser predicted that the many computers would share each of us. With-in few minutes, we would access hundreds of computers and several embedded computers in appliances. Just like writing and electricity, which are now found everywhere, UC will become commonplace as well [3]. Weiser’s predictions were made in 1990s, and the era of UC is already here. In today’s digital world, we access multiple computers; for example, getting a list of your friends in any social media platform is done in matter of seconds. The process involves multiple computers in different locations.

2.2 Context Awareness

During the last decade, many services were introduced that are attempting to give users an experience based on their location; such services are called Location Based Services (LBS). The limitations with LBS is that it does not provide a comprehensive context of the user’s surroundings and needs. Many people are seen at the airport, some of them are travellers, workers and others are visitor to pick or drop their friends and family. Identifying that a user is at the airport is not enough to provide a better user experience (UX) in digital services; therefore, context awareness is needed for meaningful applications. Location information is just one part of the user’s context. When location awareness is extended to contextual awareness, it opens a new class of applications [4].

Schilit stated in 1993, that context-aware computing requires software that “adapts according to its location of use, the collection of nearby people and objects, as well as changes to those objects over time” [5]. This means that the location, surroundings,

state of a user and events happening around the user would comprises the set of elements that determine the context of a person; and having such knowledge in an application comprises a context-aware digital service.

There are four categories of context; according to a journal published in 2002 [4]. These categories are computing context, physical context, time context and user context. Computing context consist of the device information such as type of the device, battery information, display, sensors, computing power, network connectivity. Physical context comprises of the surroundings, environmental variables such as noise, light. Time context could be current time, day, week or even season. Lastly, the journal explains that user context holds the information about the current situation of a user such as their location, health and nearby friends [4].

Context can also hold information about the user's past and future activities, which can be obtained by their calendar and sensors in their devices. Whether a user is walking, driving or using public transport can be calculated by determining the changes and path consumed in recent location updates. With advanced machine learning and powerful cloud computers, Google application uses such information to inform the user about nearby attractions, real time traffic and public transport updates.

2.3 Location Services

Android is an Operating System (OS) mainly used in mobile phones and it offers a location service application programming interface (API). The location service API in Android offers three different kind of providers for the user's location which are GPS, Wi-Fi and Cellular Identity (Cell-ID). Android OS is using a combination of multiple providers to improve the accuracy of the user's location. There are several challenges such as battery optimization, varying accuracy of different providers and mobility of the user. This location service in combination with other geocoding service can help identify the surroundings of a user. Identifying that a user is in a shopping mall can easily be done with location service and Google Places API, as it is easy to map the geo coordinates with the places. Google Places API, returns a list of places such as shopping malls, library, offices, ATM machines. based on provided location information.

With location service providers, it is not easy to determine if the user is in an elevator ascending or descending, is she on the second or the third floor, whether the user is in

vegetables section or meat section in a grocery store. However, detailed contextual information can be identified using other external devices such as BLE beacons, discussed in chapter 4.

3 BLE Beacons

3.1 Overview of Bluetooth

Bluetooth is a wireless technology using radio waves for different communication between different devices. Spectrum for Bluetooth is set by an international agreement at 2.402 GHz to 2.480 GHz for industrial, scientific and medical devices (ISM) also known as ISM-band. Its specifications are maintained by Bluetooth Special Interest Group (SIG). Bluetooth Low Energy is a subset of Bluetooth protocol, which is transmitting radio wave signals on a very low power for quick connectivity and battery optimization. This protocol is aimed to work for a longer duration with smaller coin cell batteries.

3.2 Bluetooth LE Beacons

Beacons are a hardware equipment that transmits signals to provide different kind of information for various uses such as navigation for air crafts and ships, attracting attention and indicating weather. Bluetooth LE beacons are no different on the conceptual level, these are small devices that transmits a Bluetooth signals periodically. These are small devices which can be installed anywhere easily or embedded in different equipment. Now beacons are found also in key cards, size of a credit card, key chains and watches.

Unlike other Bluetooth peripheral devices, beacons do not require pairing between two devices; they are one-way transmitters, which allows any Bluetooth scanner to read the transmitted data without having to pair with the device. The first protocol for the beacons was developed by Apple Inc. in 2013, and later followed by other protocols like AltBeacon by Radius Network and Eddystone by Google Inc. Currently, iBeacon and Eddystone are the most popular protocols used for modern Bluetooth LE beacons.

3.4 Comparison between iBeacon and Eddystone protocols

Beacons transmit data packets periodically; the concept is similar for both iBeacon and Eddystone. Both beacons can be configured to transmit at different frequencies and transmission power. These attributes are limited by the factors such as manufacturers, pricing and battery life. The difference lies in the protocol and structure of data packets itself.

3.4.1 iBeacons protocol

iBeacon advertising data packet consists of a header and payload. Advertising header for iBeacons is practically fixed to length of 4 bytes followed by manufacturer identification byte and type of the beacon. For the applications, an important piece of the data is the actual payload that consists of the few identifiers described below.

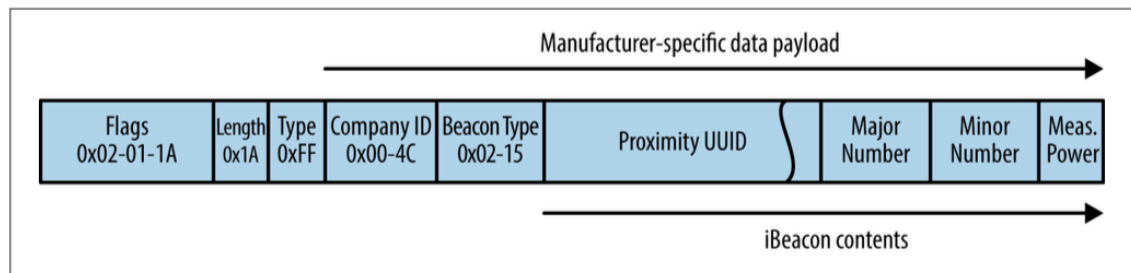


Figure 1 iBeacons advertising packet format [6]

The Figure 1 iBeacons advertising packet format shows content of the iBeacon payload. There are three main numerical identifiers in the iBeacon payload; which are 16-byte Universal Unique Identifier (UUID), 2-byte major number and 2-byte minor number. The UUID is the most important information in the iBeacon payload as it differentiates between different beacons, whereas major and minor numbers are not mandatory to monitor every use cases.

Receivers of beacon data can request for information about the beacons using the UUID from their own backend systems. Unlike other wireless specifications, iBeacon UUIDs is not centrally managed by any party. It is up to the application developers to manage their UUID for distinguishing between different beacons. Major and minor numbers can be used for hierarchal classification of additional information.

3.4.2 Eddystone protocol

The Eddystone protocol has four different packet frame formats unlike iBeacon protocol where only one format of packet is transmitted. The different kind of packets are Eddystone-UID, Eddystone-URL, Eddystone-TLM and Eddystone-EID. The frame types are identified by the byte value in a packet and parsed accordingly. Figure 2 below shows the byte values of the frame types.

Frame Type	High-Order 4 bits	Byte Value
UID	0000	0x00
URL	0001	0x10
TLM	0010	0x20
EID	0011	0x30
RESERVED	0100	0x40

Figure 2 Byte values of different Eddystone frame types [7]

The Eddystone-UID is consisted of unique 16-byte Beacon ID (UID) and it is composed of 10 bytes of namespace and 6 bytes of instance ID that allows to distinguish between different Eddystone beacons and an instance ID can function like the major and minor numbers in iBeacons protocol.

The Eddystone-URL frame type can broadcast a universal resource locator (URL) in payload. Application developers, marketers and other service providers can transmit a URL in this advertising packet and users choose to access URL without the need to install any application on their devices.

Byte offset	Field	Description
0	Frame Type	Value = 0x10
1	TX Power	Calibrated Tx power at 0 m
2	URL Scheme	Encoded Scheme Prefix
3+	Encoded URL	Length 1-17

Figure 3 Eddystone-URL frame type specification [7]

The Eddystone-URL is comprised of different fields in the packet, which is shown in Figure 3. The URL scheme prefix is represented with a 1 byte value to save some bytes in transmission data. The URL scheme prefix values are shown in Figure 3 below.

Decimal	Hex	Expansion
0	0x00	http://www.
1	0x01	https://www.
2	0x02	http://
3	0x03	https://

Figure 4 Eddystone URL scheme prefix Hex values [7]

In addition to the URL scheme prefix, the HyperText Transfer Protocol (HTTP) also requires a suffix; which is encoded in a single byte. This single byte is only decoded when

it is required to represent a URL for human eye. Examples of the URL encoding are shown in Figure 5 below.

Decimal	Hex	Expansion
0	0x00	.com/
1	0x01	.org/
2	0x02	.edu/
3	0x03	.net/
4	0x04	.info/

Figure 5 Eddystone-URL HTTP URL Encoding [7]

The Eddystone-TLM frame type broadcasts telemetry (TLM) information about the device. This information consists of different additional information that beacons can provide such as device temperature, battery voltage and advertised packet count. The Eddystone-TLM can also be encrypted before broadcasting, and only trusted devices or users can decrypt the information. The encrypted telemetry (e-TLM) frame can only be used with the Eddystone Ephemeral Identify (EID) frames.

The Eddystone-EID is the latest addition to the types of frames. It was introduced to the protocol in mid 2016. This frame is encrypted, it periodically changes its identifiers referred as rotating EID. Similar to the UUID in iBeacon, the UID in Eddystone is not centrally managed, there could be two or more beacons transmitting same UUID or UID in different locations. The mobile applications translate UUID to a specific object or place irrespective of its location, which can interrupt with the flow of the application. For example, a UUID of a beacon installed at airport entrance is used to welcome the visitors, if the same UUID is transmitted by another beacon in a different location such as a restaurant, it will also trigger the application to welcome the user to airport. The Eddystone-EID frame type can solve the spoofing problem as the real identity of the beacon is determined by a web service build by Google Inc. The developers register their Eddystone-EID beacons with Google's Proximity API, with proper permissions they can decrypt the EID to determine the real identity of the beacon.

4 Use Cases of Beacons

The aim of the project was to build a proof of concept mobile application that was capable of demonstrating the context awareness using Bluetooth Le beacons. There are different possibilities of context-aware services when using beacons. Contextual data can have multiple forms, which can be used individually or in combination with different forms of contextual information. It is important to understand the difference between proximity and location to evaluate what this experiment application is all about.

4.1 Proximity and Location

On a high-level, the beacon provides information about “what is near you” also called proximity data which is relative to an object, whereas location is an absolute value “where are you”, such as geographical coordinates, a street or a city. Both these terms are commonly confused, especially when the proximity technology such as beacons are used for determining the absolute location of a user; for example, in indoor positioning system. Since, beacons can be used to determine the location, it can be termed as micro-locations as accuracy of the location information is measured in centimetres unlike GPS where accuracy in ideal conditions is given in meters. [6]

4.2 Context Awareness with Beacons

Proximity data can be implemented in different ways to provide variety of contextual information. Services can be categorized in following three approaches: what is here, where you are and who is here.

4.2.2 What is here

Many museums are using “what is here” proximity data to provide a fully self-guided tour experience. When a user with hand-held devices approaches an artefact, proximity data triggers the mobile application to display or narrate the information about that artefact.

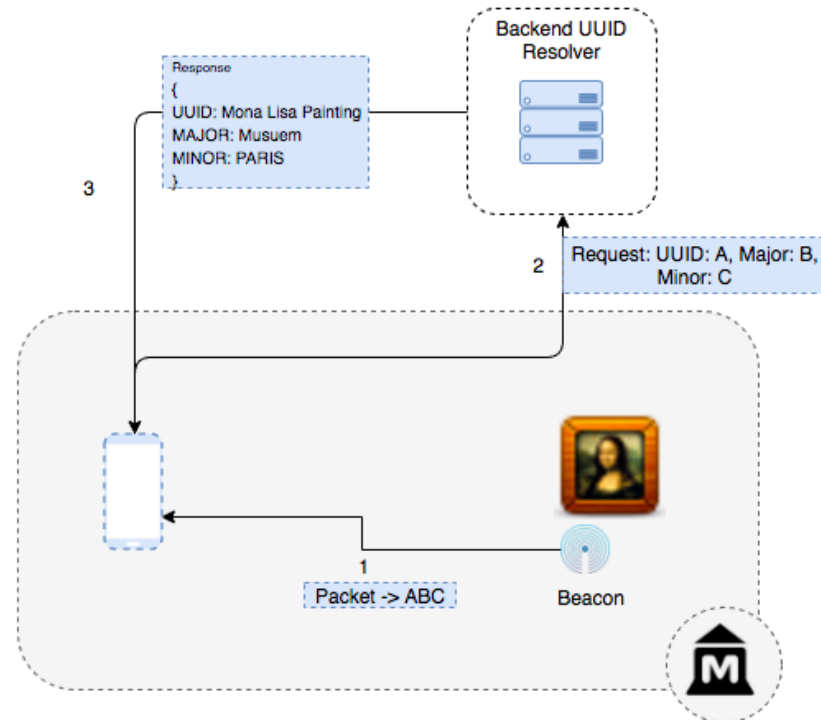


Figure 6 Object around beacon high level architecture

The high-level architecture of the museum use case is illustrated in Figure 6 where a beacon that is placed next to a painting is transmitting a packet. For simplicity, the packet is “ABC”. A mobile the device that is scanning for beacons can parse a transmitted data into UUID “A”, major number “B” and minor number “C”. To understand what this UUID means, either a mobile application need to carry this information with it, which is not a scalable solution as the UUIDs might change in the future or mobile applications request a backend system to resolve the data behind the UUID. The response from the URL resolver includes contextual data that is a painting in a museum in Paris.

The contextual data allows the mobile application developers to provide the experience that is limited by other location services. In this scenario, a mobile application understands an absolute location that is a museum in Paris and the more precise context that a user is in front of the Mona Lisa painting. Narration about the painting, textual description on the mobile user interface (UI) and even marking all the artefacts that are visited in the user's wish list can be done with such setup.

Similar architecture can be used in several different areas; for example, in a train station displaying upcoming trains information, in a train displaying next stations and many more. This example has used iBeacons and can also be replicated with Eddystone-UID. However, Eddystone-URL does not require a backend resolver, in fact, it transmits a URL, and mobile applications can open the URL for a certain artefact when a user is in its proximity.

4.2.3 Where are you

Indoor navigation systems using beacons are a good example of “where you are” services. As explained in the Proximity and Location (see section 4.1 above), beacons can be used to determine the micro-location.

This category of context is very useful for providing services like navigation where GPS, Cell-ID and Wi-Fi have some limitation in terms of fluctuating accuracy.

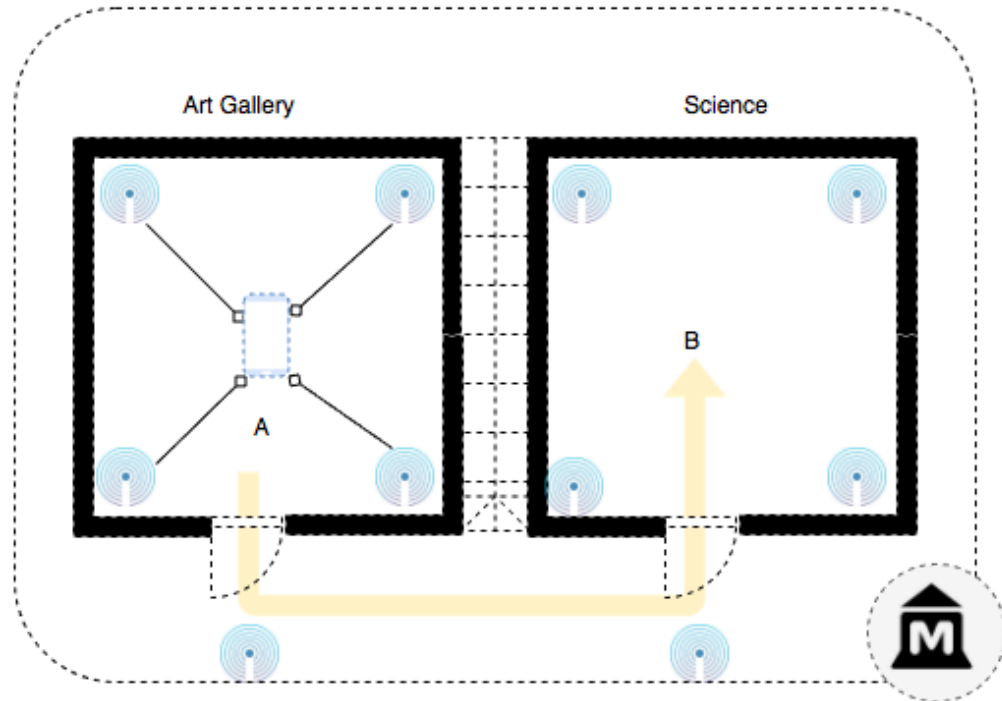


Figure 7 Indoor navigation with beacons and floor plan

In Figure 7 Indoor navigation with beacons and floor plan, it is illustrated that a user can be navigated from point A to point B. This setup requires an already known floor plan and installation location of the beacons. The receiver of Bluetooth packets, in this case a smart phone can determine the strength of the signal received using received signal strength indicator (RSSI) value. The signal is dependent on two factors the broadcasting power of a beacon and the distance. The transmission power (TX Power) of the beacons are configurable and is always a known value; therefore, with a simple equation, the distance can be calculated.

$$distance = 10^{\left(\frac{TxPower-RSSI}{20}\right)} \quad (1)$$

Equation 1 will yield the approximate distance of a receiver from the device, when TX Power and RSSI is known. The Position of the user's mobile the device is determined by calculating the distance between different beacons and mapping it on an already known floor plan.

4.2.4 Who is here

Users in a certain proximity can find each other with “who is here” kind of services, this requires having a backend web service which can serve the users with a list of people around. A simple architecture of such a system is shown in figure below.

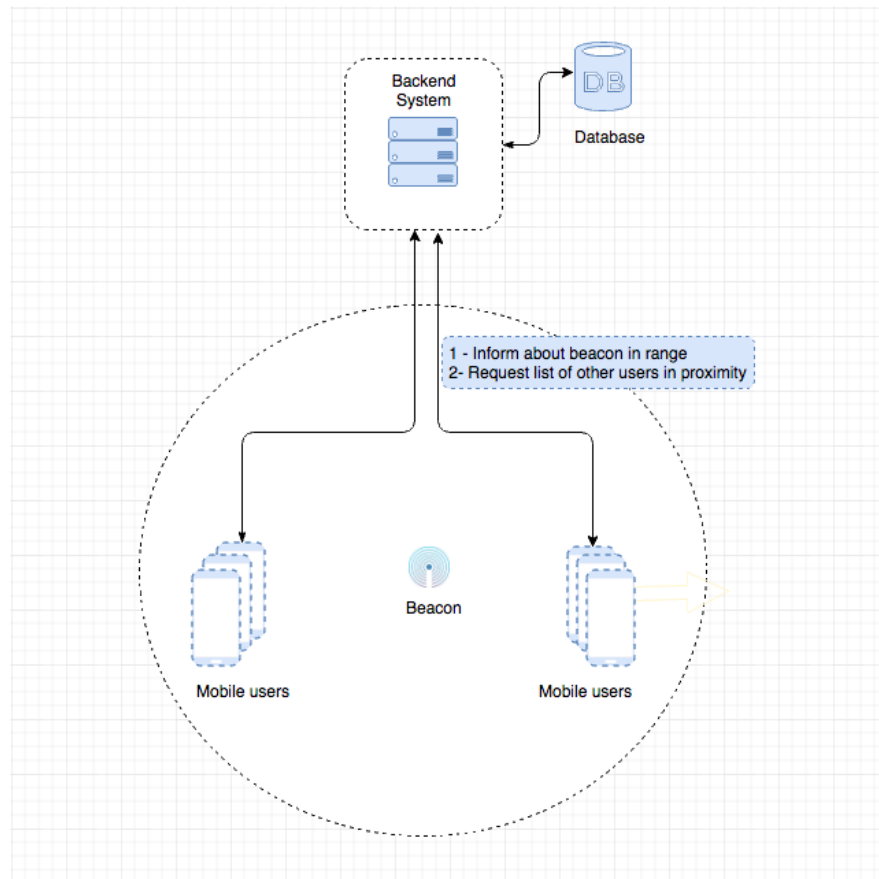


Figure 8 Simple architecture of for a getting list of users nearby

Mobile applications inform the backend system about the beacons in range, as shown in Figure 8 Simple architecture of for a getting list of users nearby. In response, a list of all the known users in the proximity of that beacon is fetched and returned to the users. There are several use cases of this contextual data when its combined with user profiles such as match making in a nearby location, find like-minded people, icebreaker in conferences and seminars. A proof of concept application is built for this kind of contextual information and detailed architecture, and the explanation is discussed in the following chapter.

4.2.5 Combination of Different Proximity Data Types

The Combination of different proximity data enriches the contextual information for mobile applications. A service that uses the proximity data explained in all the above sections of this chapter can produce a greater service. A diagram of more complicated architecture of a service is shown below.

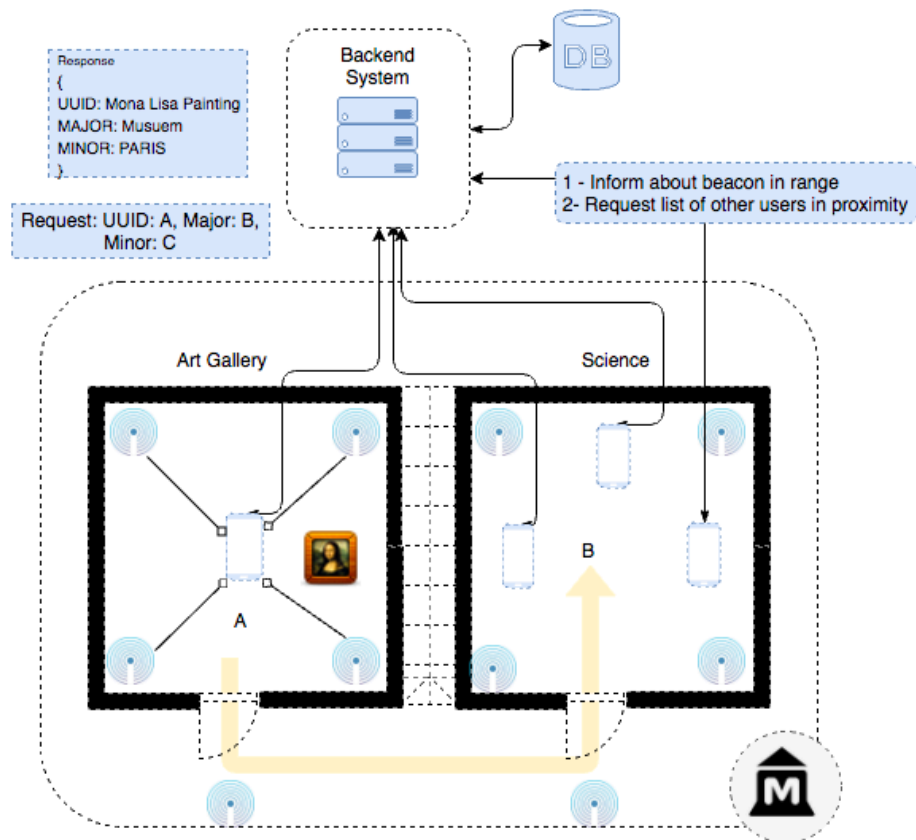


Figure 9 Architecture diagram of different proximity data

In figure 9, architecture for a complicated service is illustrated. A digital service using such components can provide real time information about your friends and other people around you. Moreover, users can navigate towards each other and share their comments about different artefacts they have visited.

5 LetMeKnow Digital Service

5.1 Overview of the application

LetMeKnow is a simple mobile application that was built during this project to demonstrate a context-aware digital service using Bluetooth LE beacons. The use case is “Who is here?” kind of proximity data briefly explained in section 4.2.4. Users of the application can login to the service with their Google accounts and scan for beacons around them. LetMeKnow application will only look for one predefined iBeacon UUID. All the users in the proximity of the same beacons are displayed in a list as shown in the figure below.

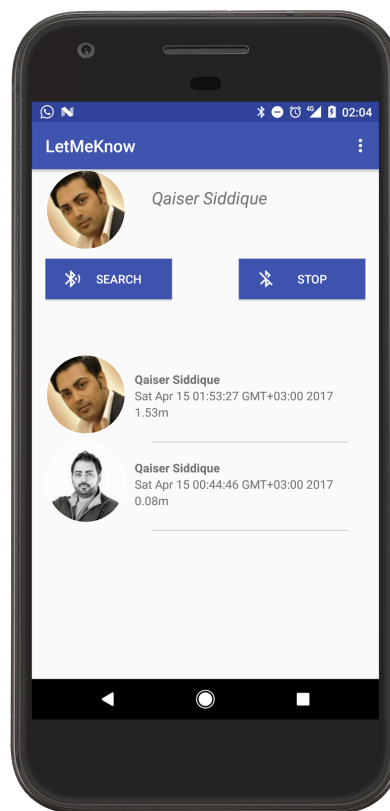


Figure 10 Screenshot from LetMeKnow mobile application

The user’s name, profile picture and distance from the beacon is displayed on the list view in the mobile application as illustrated in Figure 10. The distance from the beacon is updated in real time on all the user’s devices. Users can also stop scanning for beacons to preserve battery.

5.2 Technology selection

LetMeKnow, referred as the application hereafter in this document, is a mobile application for devices running Android OS 5 or above. The application was a proof of concept therefore the support for multiple platforms and OS versions was not considered. The Android OS 5+ is equipped with a better Bluetooth LE stack, both hardware and Bluetooth Le API, for developers. Therefore, low level scanning optimization was not required in this case.

The application requires a backend database for user management and data synchronization between users. The Firebase software development kit (SDK) is used as a backend for this application. The Firebase is selected because it provides both user authentication and real time databases for application using SDK. This eliminates the need for implementation of Representational state transfer (REST) APIs. Since it is a proof of concept for proximity data, the use of the Firebase makes the implementation simple and easy.

The beacon used this service is a iBKS beacon manufactured by Accent Systems. iBKS plus is a smarter beacon as it can transmit both iBeacon and Eddystone profile simultaneously. The cost of the beacon is 28 €, unlike some cheaper devices that can be purchased for as low as 5 € a piece. Accent System provides a configuration tool to configure parameters such as UUID, major, minor, transmission power and advertising interval. The tool is called iBKS Config Tool, and it is available Google Play Store. For scanning the beacons, AltBeacon Android library is used.

5.3 Architecture of the application

The LetMeKnow digital service requires a front end and a backend. The communication between different components of the application and Firebase is illustrated in a high-level architecture diagram of the service below in Figure 11. The Android application part in the diagram below consists of three major components; the Firebase SDK for communication with the backend, the AltBeacon SDK for scanning the beacons and the UI to provide digital touch points to the end user.

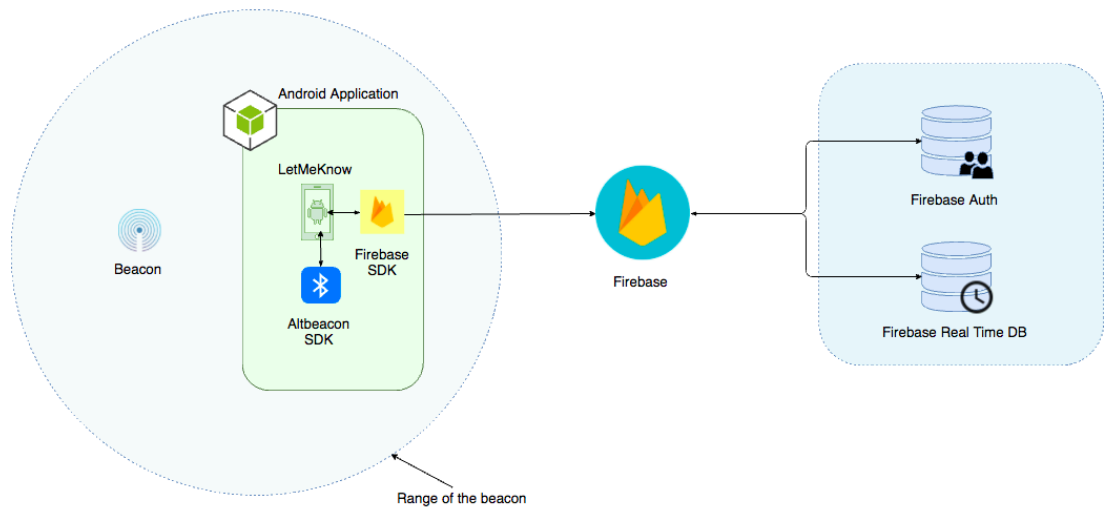


Figure 11 High level architecture of LetMeKnow service

The Firebase provides many features for mobile applications such as analytics, storage and push notifications. For the LetMeKnow service only two managed services from Firebase is used, which are real-time database and authentication service. The Firebase SDK on the mobile client takes care of most the communication between the two end points.

5.5 Components of the application

5.5.1 The Firebase Backend

As indicated earlier that the Firebase authentication is used to identify the users. The Firebase is integrated with multiple authentication providers such as Google, Twitter, GitHub, Facebook and more. In this application, Google authentication is used to get users credentials. These credentials are then verified by the Firebase authentication service and upon successful authorization a response is returned to the client application. This response contains the user's profile information including name and URL of their profile picture. The Firebase also manages the user's session in the application, which means that if users leave the application and come back again their logged in state will be persisted.

The Firebase real-time database is cloud-hosted managed service. Application developers are not required to maintain the database and servers. The data is stored as JavaScript Object Notation (JSON) tree, which is synchronized with all connected client in real time. Using the Firebase SDK for an iOS the device will also provide the same data and functionality to the users. Using such managed cloud service makes it possible for application developers to avoid complicated duplicate work. The Firebase is designed as a cross platform application's backend engine, which uses Google's cloud platform's heavy computation power to serve its users [8].

5.5.3 Mobile Application

The mobile application has a simple architecture. The User Interface is implemented with the Android native code using Extensible Markup Language (XML) and FirebaseUI SDK. The basic UI for triggering the authentication of the user, scanning beacons and navigating between different screens is using the native code. However, authentication flow and rendering the list of users from The Firebase real time database is using FirebaseUI SDK. In figure 12, steps involved to authenticate the user are illustrated.

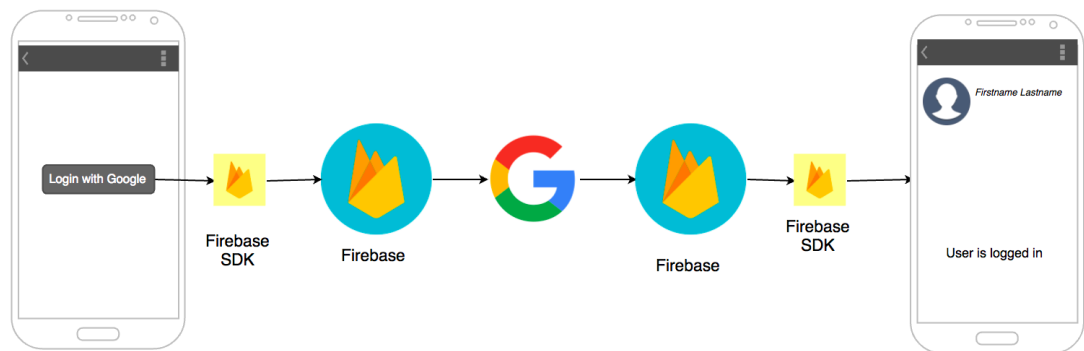


Figure 12 Authentication flow with The Firebase SDK

When a user clicks on the sign in button on the UI, The Firebase's Authentication UI SDK (AuthUI) is triggered. AuthUI provides a complete set of user interface to complete the authentication flow. A combination of AuthUI and FirebaseAuth SDK, LetMeKnow performs the authentication for the end user. The communication flow between logged in and logged out state of the application is shown in the Figure 12. Similarly, The Firebase database is used to manage the user's proximity information, which is discussed in 6.4.2 below.

The third element of the mobile application is implementation for scanning the beacon. AltBeacon SDK is used to accomplish this capability. This library can scan for beacons and notifying the application. The library does the scanning work on a different thread; therefore, managing the usage of main UI thread is not required. With the use of this library low level optimization of Bluetooth stack is not required. The following figures illustrates getting the list of beacons using AltBeacon SDK.

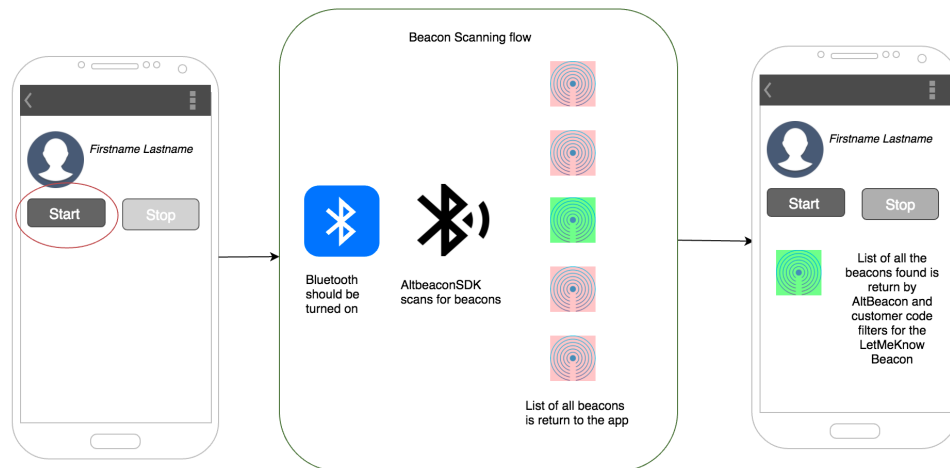


Figure 13 Beacon scanning flow with AltBeacon SDK

When user clicks a button to trigger the Bluetooth scanning, the first thing that needs to be active is the Bluetooth radio hardware. Once the Bluetooth is turned on then the AltBeacon SDK starts scanning for nearby Bluetooth devices. The library will filter the beacons from other peripheral devices. A list of all the nearby beacons is returned to the application. The application then looks for a specific beacon to trigger its functionalities. The work flow used in the application is also illustrated in in figure 13.

6 Implementation of LetMeKnow Service

6.1 Setting up an Android application project

To build the application Android Studio integrated development environment (IDE) was used. Android studio is developed and managed by Google Inc. in corporation with JetBrains. Android studio is based on IntelliJ IDE developed by JetBrains. The IDE provide a simple setup a project wizard to setup a new application. Configurations used to setup LetMeKnow project is listed below.

- Platform: Phone and Tablet
- Minimum SDK: API 21, Android 5.0 (Lollipop)
- Boilerplate: A blank activity and black layout resource file

Initial setup for the project was very simple, however more configurations for external libraries and SDKs were added to the project which are explained later in this chapter.

6.2 Setting up the beacon

The beacons used for this project is iBKS plus manufactured by Accent Systems. In order configure the beacon iBKS config tool, an Android application, provided by the vendor was used. To complete this setup, an Android the device with Bluetooth low energy and a iBKS Plus beacons is required. The beacon has a button on the top, which was used to turn on the beacon. Same button when pressing and holding for 30 seconds starts the configurable mode.



Figure 14 iBKS plus BLE Beacon

Figure 14 shows a button in the middle of the beacon, which can be used to set different modes of the beacon. iBKS config tool was downloaded and installed from Google play store. It can be accessed using the following link https://play.google.com/store/apps/details?id=com.accent_systems.ibks_config_tool . The config tool was used to scan for beacons as shown in figure 15 below. The beacons that are in configurable mode are indicated in the config tool with white background as shown in the figure below.

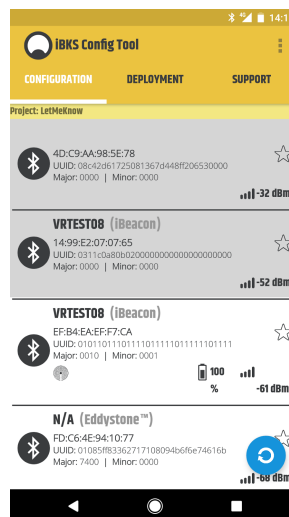


Figure 15 iBKS Config tool

The beacon was selected from the list to configure. In the configurations, there are multiple protocols. The settings for iBeacons was opened. This beacon model can transmit multiple iBeacon packets, for this use case on one of the slots is required. Slot 1 was selected to configure the beacon and the UUID 01011011-1011-1101-1111-

011111101111, major 0010 and number 0001 values were set. However, for this simple implementation major and minor numbers are not used.

In other configurations, transmission power was set to 0 dBm and advertising interval was set to 300 milliseconds. Advertising interval defines the time interval between transmitted packets. Transmission power effect the battery life of the beacon and is a major factor for signal range as explained in equation 1. The figure below illustrates the relation between transmission power and distance covered by the signal.

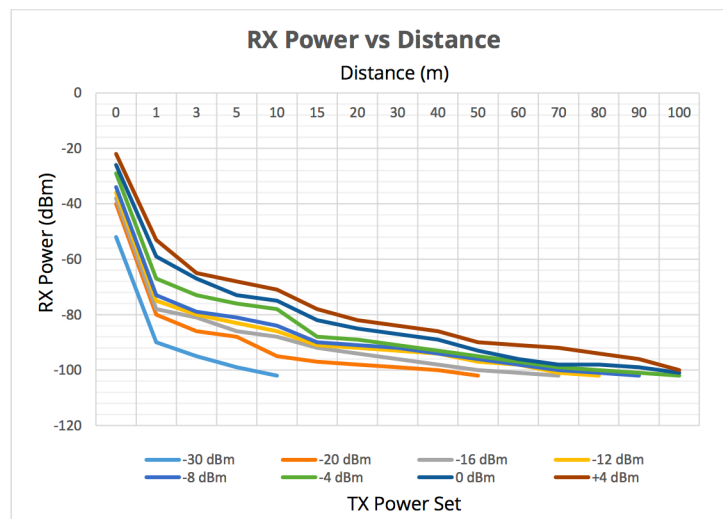


Figure 16 Relation between transmission power and distance [9]

After configuration of beacons are entered in the and saved, the beacons setup is completed. Looking at Figure 16 above, transmission power of 0 dBm will allow the mobile devices to scan within a radius of at least 70 meters in ideal conditions.

6.3 Beacons scanning

For scanning the beacons, all applications required permissions from the device users to access and use their Bluetooth hardware. This is a requirement set by Android OS to give more control to the device owners. With the right permissions, Bluetooth hardware can be used to scan for beacons. Custom implementation for Bluetooth stack and APIs was out scoped for this project. An external library AltBeacon, which is designed specifically for beacons scanning was implemented.

6.3.1 Permissions

Prior to Android OS 6.0 (Marshmallow), all the required permissions were added in Android manifest XML file. The application stores parse the requested permissions from this file and displays it on the application download screen to inform users about the set of hardware and services any application is used in the application. Since Marshmallow, developers are required to ask for runtime permissions. This means whenever a certain service is needed, users should be prompted with a proper permission request. The change in the permissions implementation allowed several users to download and test the apps without worrying about different permissions. Secondly, it gives more control to the users to allow a limited set of permissions. It is easier for users to decide if they know the purpose for a permission request.

In this application, Bluetooth hardware is needed. The permission for using Bluetooth is grouped with location permission. Therefore, when user clicks on the start scanning button, the application need check if user has granted the permission as illustrated in figure below.

```

@RequiresApi(api = Build.VERSION_CODES.M)
private void requestForLocationPermission() {
    if (this.checkSelfPermission(Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED) {
        final AlertDialog.Builder builder = new AlertDialog.Builder(this);
        builder.setTitle("This app needs location access");
        builder.setMessage("Please grant location access so this app can detect beacons.");
        builder.setPositiveButton(android.R.string.ok, null);
        builder.setOnDismissListener(new DialogInterface.OnDismissListener() {
            @Override
            public void onDismiss(DialogInterface dialog) {
                requestPermissions(new String[]{Manifest.permission.ACCESS_COARSE_LOCATION}, PERMISSION_REQUEST_COARSE_LOCATION);
            }
        });
        builder.show();
    } else {
        startScanningForBeacons();
    }
}

```

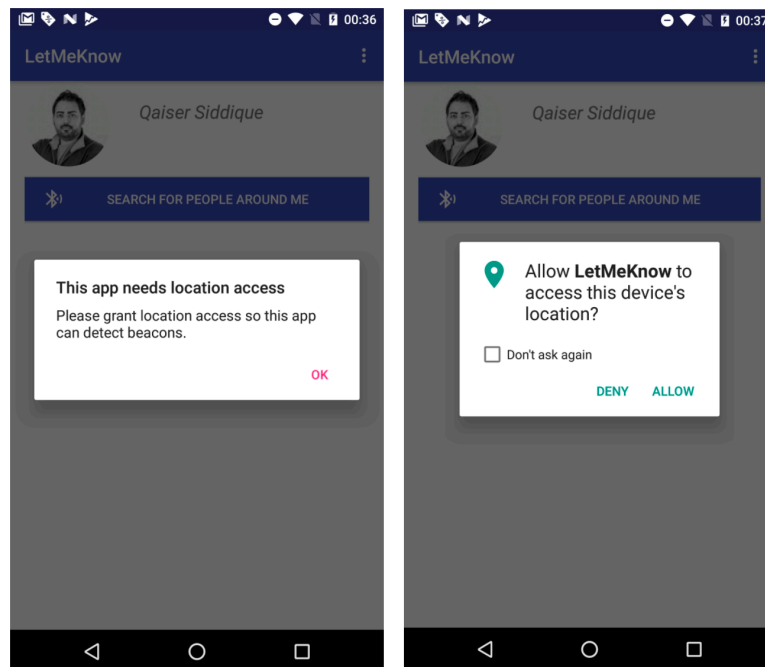


Figure 17 Requesting location permission in LetMeKnow application

In Figure 17, it is illustrated that the code is required only for Android OS 6.0 and above. A customer message that explains location access is required for beacons detection and a default OS prompts with actions allow and deny is provided by the Android OS.

6.3.3 Implementing AltBeacon Library

AltBeacon library is installed in the application using the gradle configurations. The SDK version used in this project is 2.9.2. Installation instructions are listed below.

```
JCenter
Add JCenter to your build file's list of repositories.
repositories {
    jcenter()
}
to use the JCenter Repository
dependencies {
    ...
    compile 'org.altbeacon:Android-beacon-li-
brary2.9.2}'
    ...
}
```

Listing 1 Installation of AltBeacon Android library with gradle [10]

After adding the configurations and syncing modified gradle build files, the library is available to use in the code. The library provides a java interface for Android activities to connect to beacon service. This interface is called beacon consumer. A Beacon consumer is responsible for notifying an activity with a call back when the beacon service is ready to use. The interface is used in conjunction with Beacon Manager object that allows to interact with beacons [10].

The following code snippets illustrates the setup that enables beacon scanning.

```
public class UsersActivity extends AppCompatActivity
implements BeaconConsumer {
...
@Override
    public void onBeaconServiceConnect() {
        ...
    }
...
}
```

Listing 2 Adding BeaconConsumer to Android Activity

When the service is connected a callback `onBeaconServiceConnect()` is received. This call back works on a different thread than the main UI thread. In this callback BeaconManager is required to interact with the scan results. Beacon manager is first initialized and globally referenced in the main UI thread as shown in the figure below.

```
beaconManager = BeaconManager.getInstanceForApplication(this);
beaconManager.getBeaconParsers()
    .add(new BeaconParser().setBeaconLayout("m:2-3=0215,i:4-19,i:20-21,i:22-23,p:24-24"));
beaconManager.setForegroundScanPeriod(1100L);
beaconManager.setForegroundBetweenScanPeriod(1100L);
beaconManager.setBackgroundScanPeriod(1100L);
beaconManager.setBackgroundBetweenScanPeriod(1100L);
beaconManager.bind(this);
```

Figure 18 Initializing a BeaconManager in Android application

Scan intervals and scan period both in foreground and background are configurable, in this example a value 1100 milliseconds are used. This means that the library will scan for 1100 milliseconds, after the scan is completed application will pause scanning for 1100 milliseconds. Different values can be used depending on the use cases to preserve mobile the device's battery life. Beacon manager is then bound with the consumer, see Figure 18.

Beacon manager by default scans for AltBeacon protocol specification; therefore, a BeaconParser is added to the manager to configure scanning for iBeacon protocol. Listing below shows how iBeacon protocol is added to beacon manager.

```

...
beaconManager.getBeaconParsers()
    .add(new BeaconParser()
        .setBeaconLayout
            ("m:2-3=0215,i:4-19,i:20-21,i:22-23,p:24-24"));
...

```

Listing 3 Adding iBeacon layout to beacon manager

Where the prefixes below are defined in the official documentation and the values for beacon layout was taken from the sample application [11].

m - matching byte sequence for this beacon type to parse (exactly one required)

s - ServiceUuid for this beacon type to parse (optional, only for Gatt-based beacons)

i - identifier (at least one required, multiple allowed)

p - power calibration field (exactly one required)

d - data field (optional, multiple allowed)

x - extra layout. Signifies that the layout is secondary to a primary layout with the same matching byte sequence (or ServiceUuid). Extra layouts do not require power or identifier fields and create Beacon objects without identifiers.

Taking a brief step back, so far, a BeaconConsumer is added to an activity to connect to the service and a BeaconManager is initialized to interact with the beacons. Now the application needs a notification when a beacon is in range. This callback is received by implementing a RangeNotifier in onBeaconServiceConnect() callback as shown in the figure below.

```

185     @Override
186     public void onBeaconServiceConnect() {
187         beaconManager.addRangeNotifier(new RangeNotifier() {
188             @Override
189             public void didRangeBeaconsInRegion(Collection<Beacon> beacons, Region region) {
190                 if (beacons.size() > 0) {
191                     final Beacon beacon = beacons.iterator().next();
192                     if (beacon.getId().toString().equalsIgnoreCase(LET_ME_KNOW_BEACON_UUID)) {
193                         Log.i(LogTag.BEACONS, "Demo beacon found " + String.format("%.2f", beacon.getDistance() / 100000) + " meters away");
194                         UsersActivity.this.runOnUiThread(new Runnable() {
195                             public void run() {
196                                 updateBeaconInfo(String.format("%.2f", beacon.getDistance() / 100000));
197                             }
198                         });
199                     }
200                 }
201             });
202         });
203         try {
204             beaconManager.startRangingBeaconsInRegion(new Region("myRangingUniqueId", null, null, null));
205         } catch (RemoteException e) {
206             Log.d(LogTag.BEACONS, e.getMessage());
207             e.printStackTrace();
208         }
209     }

```

Figure 19 RangeNotifier Implementation

In Figure 19, the RangeNotifier interface provides a callback called `didRangeBeaconsInRegion()`. This callback will return a collection of beacons in range. Parsing through the list of beacons a comparison is made to look for a beacon with specific UUID. Logs below in the listing 4 shows the distance between the device and the beacon.

```

04-14 14:07:31.959 5434-6122/beacons.qaiser.solinor.com.letmeknow I/beacons: Demo beacon found 5.22 meters away.
04-14 14:07:34.183 5434-6172/beacons.qaiser.solinor.com.letmeknow I/beacons: Demo beacon found 4.62 meters away.
04-14 14:07:36.359 5434-6230/beacons.qaiser.solinor.com.letmeknow I/beacons: Demo beacon found 4.74 meters away.

```

Listing 4 Three consecutive information log messages after a beacon was found

Scanning for the beacons is triggered with a button on the UI as shown in Figure 13 earlier. However, when user clicks the stop scan button, application need to unbind the consumer and disconnect the service. If service is not unbound then the it will run in the background and consume significant amount of mobile phone's battery. To accomplish this a simple method is illustrated in the listing below.

```

private void unbindConsumer() {
    if (beaconManager.isAnyConsumerBound()) {
        beaconManager.unbind(this);
    }
}

```

Listing 5 Unbinding beacon manager service

The method in listing 5 is also called in application lifecycle methods such as `onStop()` and `onDestroy()`. Lifecycle methods are managed by Android framework, these are triggered when user stops using the current activity or the application processes are killed.

6.4 Setting up The Firebase SDK

The Firebase is also installed with gradle configurations. In addition to gradle a JSON configurations file provided by The Firebase for each project is needed to connect the mobile application to the Firebase project. The Firebase SDK version used in this project is 10.2.1. The listing below shows the installation of SDK.

```

dependencies {
    ...
    compile 'com.google.firebase:firebase-core:10.2.1'
    ...
}
apply plugin: 'com.google.gms.google-services'

```

Listing 6 Adding the Firebase SDK to application gradle build file

The Firebase dependencies and google play services plugin are added and synced in the application gradle build file as shown in listing 6. In addition to the core module, this application is using the Firebase UI SDK to benefit from authentication flow UI and database change UI, explained in the following sections.

```
dependencies {
...
    compile 'com.firebaseui:firebase-ui-database:1.2.0'
    compile 'com.firebaseui:firebase-ui-auth:1.2.0'
...
}
```

Listing 7 Adding the Firebase UI SDK to application gradle build file

The Firebase UI SDK version 1.2.0 is used in this project as shown in Listing 7 above. To connect the Firebase mobile application project and the Firebase backend application project google-service JSON is downloaded from the Firebase backend console. The file contains all the configurations required for the system to work. The developers do not require any addition configuration in the mobile application. The file is available during the setup of the project in the Firebase console as shown in figure below.

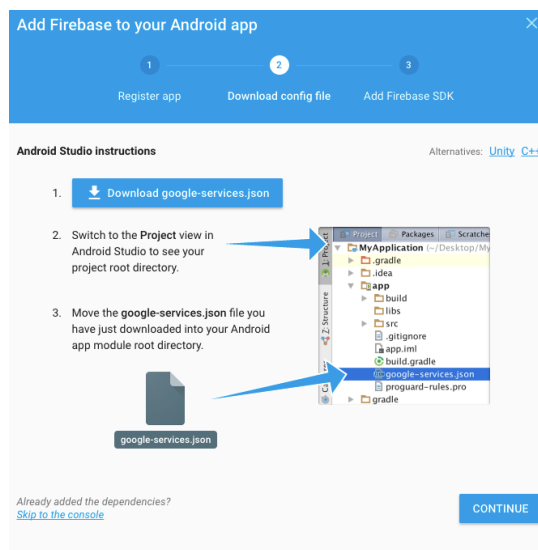


Figure 20 Screenshot taken during the Firebase Android project setup

After configuring the application gradle build file, project gradle build file and downloading the JSON configuration file in correct folder, the Firebase setup is complete. However, it is important to understand that the Firebase is available only for Android devices that are running Android version 4.0 or newer and Google play services 10.2.1 or higher.

6.4.1 Authentication with The Firebase

The Firebase authentication is enabled from the console. The Firebase console is a web interface to configure and monitor the application and backend system. In develop section of the Firebase console, authentication settings can be accessed. In this application, Google authentication is required which is enable by a simple toggle button as shown in figure below.

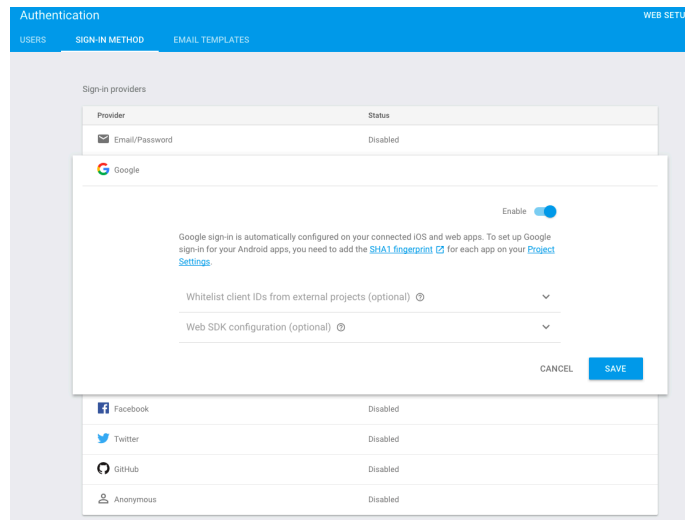


Figure 21 Enabling google authentication in the Firebase console

The Firebase supports other authentication providers as shown in figure above. After the initial installation of the SDKs and enabling authentication in the console implementing the authentication is as simple as calling an IntentBuilder provided by AuthUI SDK. When user clicks the sign in button following figure illustrates how AuthUI can be triggered to authenticate the user.

```
Button signInButton = (Button) findViewById(R.id.sign_in_button);
signInButton.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View v) {
        startActivityForResult(
            AuthUI.getInstance().createSignInIntentBuilder().setProviders(Arrays.asList(
                new AuthUI.IdpConfig.Builder(AuthUI.GOOGLE_PROVIDER).build()
            ))
                .build(),
            RC_SIGN_IN);
        Log.d(LogTag.FirebaseAuth, "onAuthStateChanged:signed_out");
    }
});
```

Figure 22 Authenticating a user with FirebaseAuthUI SDK in Android application

The code snippet above will initiate the Google authentication flow. Using AuthUI SDK makes it very simple for the developers to authenticate their users. The application developers do not need to implement the complicated flow with all the security and user experience constraints. Google has explained the complete flow in the following figure.

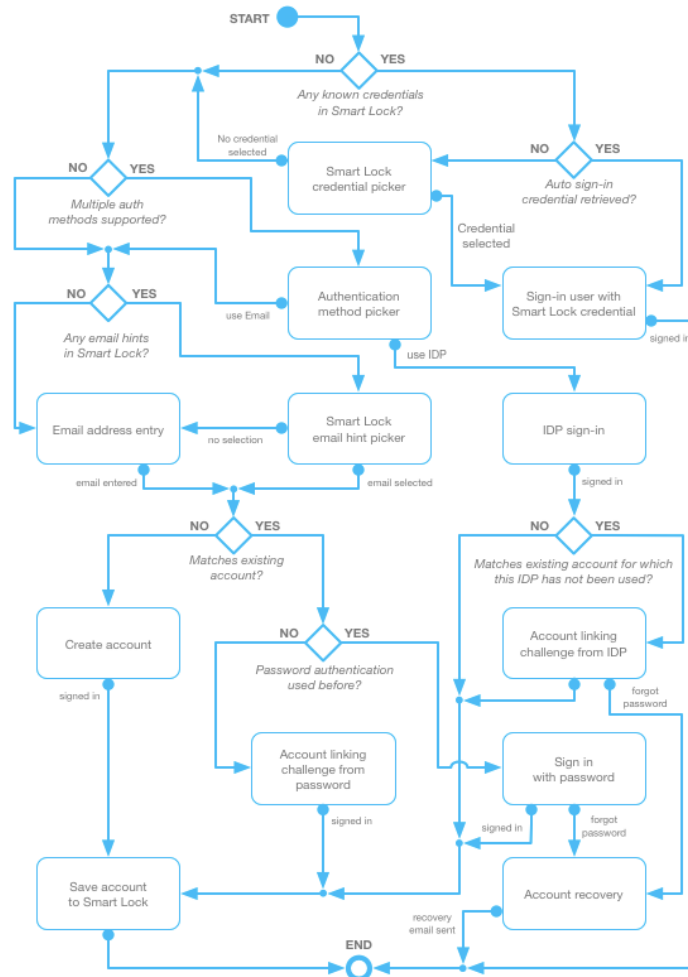


Figure 23 The Firebase UI Authentication flow for Android [12]

The figure 23, above illustrates a complete flow chart for authenticating a user. The implementation is completed for Android SDK due to the use of Smart Lock. SmartLock is a service built by Google that maintains a list of trusted applications and devices. This allows the users to quickly access their data in different application or keep access their mobile the device with other trusted the device without having to remember their passwords.

Going through the complete process the AuthUI will return one of the following messages back to the application.

- User is successfully authenticated
- User has cancelled sign in process
- No network connectivity
- Unknown error

On successful sign in LetMeKnow application navigates the user to next screen as illustrated earlier in Figure 12 Authentication flow with The Firebase SDK12. Other response cases are dealt by displaying the error messages on the UI. Logged in the user's basic profile can be accessed from The Firebase User object. The Firebase User object contains information such as user name, email address, URL for display picture. In the application only user name and profile picture URL is retrieved as illustrated in the figure below

```

FirebaseUser user = FirebaseAuth.getInstance()
    .getCurrentUser();
if (user != null) {
    userName = user.getDisplayName();
    photoUri = user.getPhotoUrl();
}

```

Listing 8 Getting user information from the Firebase Android SDK

The profile details user name and photo is displayed on the UI. The photo is downloaded and displayed on the UI in a background thread which is done using Glide library.

6.4.2 Database Design

At this point, the application can scan for beacons and understands the basic information about the user. This means the application is in a state where it knows "Who is nearby?" and "What is nearby?". This is the proximity data that will be pushed to backend database. All users can access the same data and see a list of people around a certain beacon. This data sharing is enabled by the Firebase database.

There are no configuration or database modelling required for the Firebase. The Firebase real time data structures all the received data as JSON Tree. JSON is a format that consists of key value pairs. JSON objects or a tree of JSON objects can be parsed by client applications when the data structure and keys are known. When data is pushed to database, new record is added or updated in existing node if available otherwise a new node will be created.

In REST model, a client application requests for a data such as give me list of all users. To update the list client applications are required to make a new request. The Firebase does not use REST model, it uses publish-subscribe (pub-sub) software architecture. A pub-sub is a messaging pattern where publishers push the data to backend without caring who if there are any listeners and subscribers can subscribe to certain topics. A subscriber receives an update whenever data is changed in their topics.

In this application, there is only one node named “users”. When a beacon is found in the range of mobile the device, user information is published to users node in the Firebase database. All other client applications are automatically subscribed to the same node, by design, and they will receive the notification of data change. The illustration below is a screenshot from the LetMeKnow the Firebase database.



Figure 24 Screenshot from the Firebase database

The users node in the database contains a JSON array of user objects. Each user is identified with a key as shown in Figure 24, which is their UID retrieved from authentication. The user's record consists of five key value pairs listed below.

1. Distance: That is their calculated distance from the beacon in meters
2. Name: User name from their Google profile
3. Photo URL: The URL to download their Google profile photo
4. Present: It is a Boolean attribute that shows if users are currently active
5. Timestamp: Time in milliseconds to indicate when the record was updated

If users are scanning for the beacons, their distance and timestamp are updated to users node. The Firebase database viewer highlights the updated fields as shown in figure above.

6.4.3 Writing data

The Firebase SDK does not require the raw data JSON format. The application developers can push a Java Object to a node. In this application, User object is created and published to the users node. The user Java Object is a simple java class with five fields and two constructors as shown in figure 25 below.

```
public class User {
    private String photourl;
    private String name, distance;
    private long timestamp;
    private boolean present;

    public User() {
    }

    public User(String name, String photourl, long timestamp, String distance, boolean present) {
        this.name = name;
        this.photourl = photourl;
        this.timestamp = timestamp;
        this.distance = distance;
        this.present = present;
    }
}
```

Figure 25 User.java class in LetMeKnow Android application

The default empty constructor is required by the Firebase to create the user objects from snapshot of database. The snapshot of database is received by the subscribers of users node when a change in data is observed. As fields of user object in java class and keys

of user object in JSON tree are same as illustrated in the Figure 24 and Figure 25. This mapping is done automatically by the Firebase.

For writing a record to JSON tree, FirebaseDatabase object is initialized in onCreate method of an activity as shown in the listing below.

```

...
private DatabaseReference mDatabase;
...
@Override
protected void onCreate(Bundle savedInstanceState) {
...
mDatabase = FirebaseDatabase.getInstance().getRefer-
ence();
...
}

```

Listing 9 Initializing the Firebase database instance in Android application

The database reference contains the information about the endpoints of the Firebase project and its database. This reference object can be used to write to any node in the database. For example, in case of a car dealer store, vehicle of a specific make can be added to its own node such as Database Reference > Car Dealer > Vehicles > Motor cycles > Honda > add new object. In this application, there is only two nodes that are users > UID. These nodes are also termed as child of a parent node. The following method is used in the application to publish new data to correct node.

```

private void updateBeaconInfo(String distance) {
    User user = new User(firebaseUser.getDisplayName(),
        firebaseUser.getPhotoUrl().toString(),
        System.currentTimeMillis(),
        distance,
        true);
    mDatabase.child("users").child(firebaseUser.getUid()).setValue(user);
}

```

Figure 26 Publishing data to the Firebase database

A FirebaseDatabase reference object has a child called “users” which further has a child that is UID of that user. The Java user object created in Figure 26 is used as a value of UID child node. This can simply be expressed as, A user object is created and set as a value in a designated path. This process will write a new record if UID does not exist in the Firebase database or update an existing record with same UID. The Firebase guarantees a unique UID for all the users of an application irrespective of authentication provider.

6.4.4 Reading Data

Reading data from database is a subscription on the Firebase node. The Firebase SDK provides ValueEventListener, which is an API to subscribe to certain nodes or topics with a known path. For example, in LetMeKnow application, a user John could subscribe to users > UID of Jane. It means whenever Jane enters in range of the beacon John will notified. This feature was out scoped for the project. However, all the users are subscribed to “users” node as mentioned earlier.

Since the application is using FirebaseUI SDK; therefore, ValueEventListener is not required. The FirebaseUI provides a recycler adapter, that links the data with recycler view, a UI widget for displaying lists in Android application. The Firebase recycler adapter API can listen to a specific node, parse the data snapshot into a java object and populating a list view on the UI with customer design. The figure 27 below shows a Firebase recycler adapter implementation in LetMeKnow application.

```

RecyclerView recycler = (RecyclerView) findViewById(R.id.recycler_view);
recycler.setHasFixedSize(false);
recycler.setLayoutManager(new LinearLayoutManager(this));

mAdapter = new FirebaseRecyclerAdapter<User, UserHolder>
    (User.class, R.layout.users_list_item, UserHolder.class, FirebaseDatabase.child("users").limitToLast(30)) {
    @Override
    public void populateViewHolder(UserHolder userHolder, User user, int position) {
        userHolder.setName(user.getName());
        userHolder.setDistance(user.getDistance());
        userHolder.setLastSeen(TimeAgo.getTimeAgo(user.getTimestamp()));
        userHolder.setImage(UsersActivity.this, user.getPhotourl());
    }
};

recycler.setAdapter(mAdapter);

```

Figure 27 FirebaseRecyclerAdapter implementation in LetMeKnow mobile application

The adapter is listening to FirebaseDatabase child “users” and requesting last 30 records as shown in above illustration. The record limit is handful addition to API. Application developers can manage smooth user experience by setting the right limits based on number

of records they have in the node. The Firebase recycler adapter eliminates the need to separate implementation of several steps to get the results on the UI. Traditionally, a REST API call was made to backend server, the JSON data received was parsed and java objects were created based on the data received. The data was then compared with existing data on the UI and it was updated according to the requirements of the application.

6.5 Implementing the Mobile UI

The user interface is built with the Android native XML code. There are two main screens to represent the logged in and logged out of the user. Finally, an options menu is used to allow the users to log out of the application as shown in interactive design illustration below.

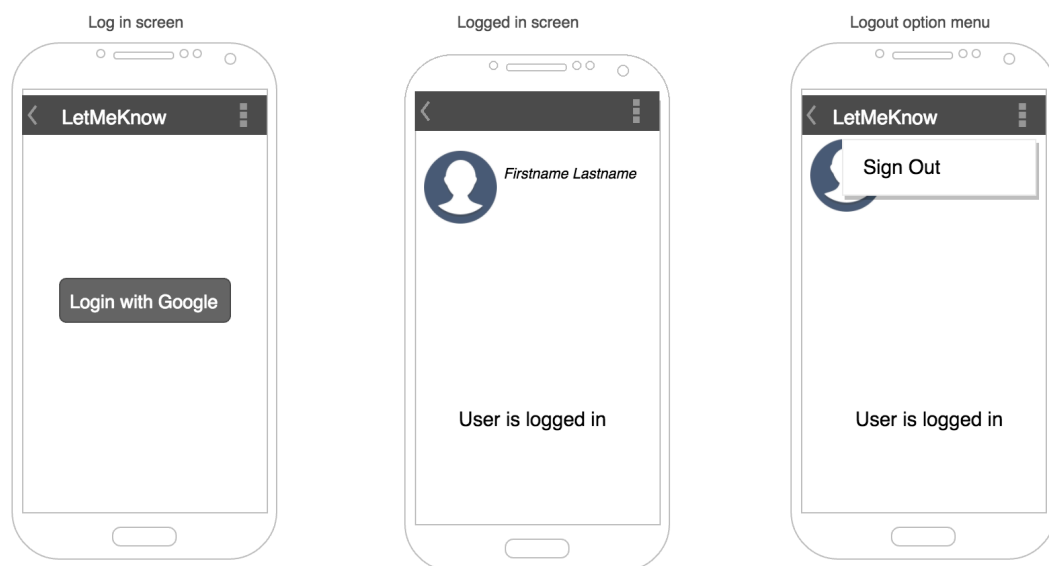


Figure 28 Wireframe of different screens in LetMeKnow application

The logged in users can perform beacons scanning by simply pressing a button to trigger the scan. Location permission is requested as explained in 6.3.1 above.

The logged in state screen also implements the Firebase recycler adapter, which requires a list view, to bind the real-time database updates on the UI as shown in the figure below.

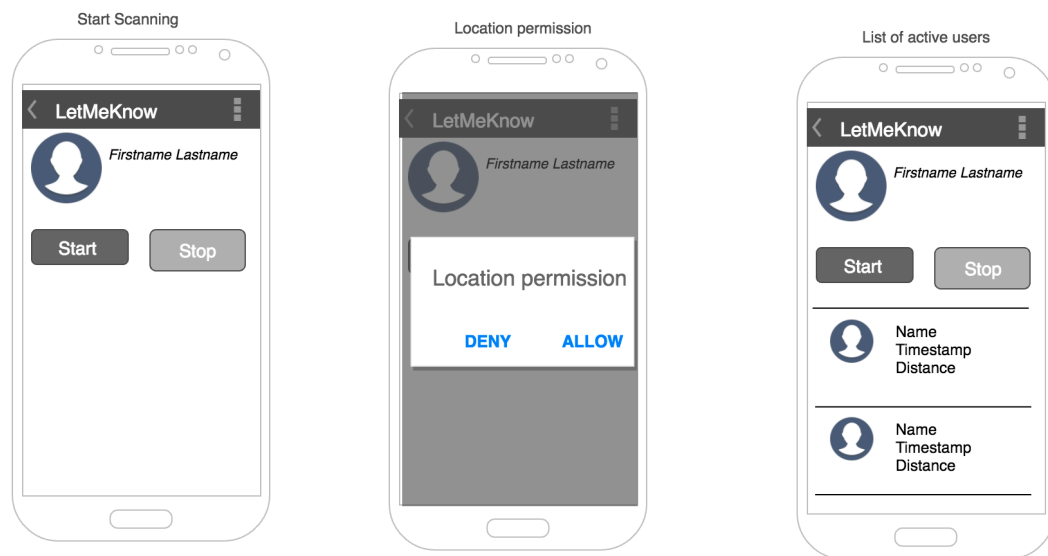


Figure 29 UI updates for logged in users

The Firebase recycler adapter requires the reference to the custom layout. This reference is passed as an Android layout resource ID in the parameter. In Figure 27 FirebaseRecyclerAdapter implementation in LetMeKnow mobile application above, the resource ID is "R.layout.users_list_item". A recycler view holder binds the User object to UI widgets in the layout for each list item. The list is automatically updated by the adapter when changed to the database is observed.

The image of the user is downloaded in the background using the Glide Android library. The circular image view is not a native component. It is a custom UI widget developed by Henning Dodenhof which was installed from the following link <https://github.com/hdodenhof/CircleImageView>.

8 Observations

The goal of the project was to understand possibilities of context-aware services and produce an Android application to demonstrate a use case. The mobile application Let-MeKnow was produced. The details of the produced application are already discussed in the previous chapters. However, there were few observations made and studied during the project which are discussed in this chapter.

8.1 Capturing a Beacon signal

Beacons transmit a signal at a predefined interval just like a receiver pauses between its consecutive scans. There is a possibility that a receiver will not see a beacon during its scan if a beacon is not advertising during that time. This means that the visibility of a beacon can fluctuate despite being in the physical range of a beacon. The figure below illustrates the possible miss and hit scenario.

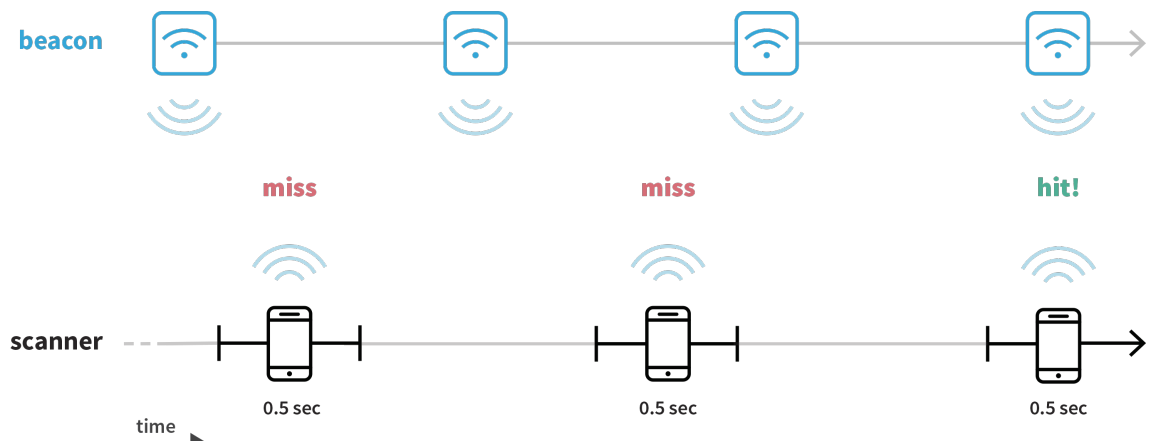


Figure 30 Fluctuating visibility of a beacon [13]

The figure above a beacon is transmitting a data packet once per second and a mobile the device is scanning half a second before pausing for one second. In this case, it may take up to 3 scan attempts to see a beacon.

When a beacon is first seen, application triggers an action. In LetMeKnow application beacons triggers the database record update in the backend. It is very important to make sure, if the user has is longed in the proximity of a beacon. This can be achieved by

combining multiple scan results and taking an average of beacon hits. There are other possibilities such as, if a beacon is not seen in consecutive 5 or 10 scan results, this could indicate that the user is not in the range of the beacon. The number of consecutive scans depends on the advertising interval, scan period and scan interval.

In this project, the problem was observed and solved by increasing the duration of scan interval to 1.1 seconds and reducing the advertising interval to 0.3 seconds. During this scan period the beacon will transmit a signal at least three times. These configurations are not very optimal for battery consumption. The tests, observations and conclusions were made together with the mobile application team from Solinor Oy in September 2016.

8.2 Spoofing beacons

Beacons UUIDs are managed by the application developers and they are visible to all scanners. This design allows beacon spoofing, meaning that anyone can configure a beacon with same configurations and application may cause unwanted behaviour such as a beacon with UUID-A in a museum can be spoofed and placed in a train station. The application using UUID-A may assume that the user is in a museum. Apple Inc. wanted to keep the beacons as simple as possible; therefore, they left UUID management to developers. Since the simplicity of beacon packet makes it very easy to implement in existing infrastructure; therefore, a length of data packet is kept to maximum of 20 bytes. The available length can formulate enough combinations to mark all the devices in the world.

The spoofing problem can be solved by using an Eddystone-EID protocol or similar solutions for iBeacons offered by manufacturer. A decryption key is embedded in the beacon which is synced with a backend system when a user registers a beacon. For Eddystone-EID, Google has launched a Beacon Config Tool. Users log in to the tool with their account, scan for the nearby beacons they own and register it to proximity database managed by Google. During this process both backend and beacon will share the decryption key and synchronize rotation intervals. Beacons that are configured to transmit Eddystone-EID can rotate their UUIDs as frequently as 10 seconds per ID. The scanners will require a permission from the developers to decrypt a UUID of the beacon with pre-shared key. This setup prevents the application from misunderstanding the context.

8.3 Battery Optimization

8.3.1 Beacons battery

High transmission power consumes more battery in beacons. Battery replacements efforts may cost more in logistics than to replace a beacon, especially in large infrastructures and for small coin cell sized beacons. Depending on the requirement there are multiple battery options available from the manufacturer. Alkaline batteries are very common among the beacons. However, possibility to add lithium battery to iBKS is offered by Accent Systems. Battery consumption comparison between alkaline and lithium batteries for iBKS plus model is published by Accent Systems, see appendix 2. Beacon that operate on universal serial bus (USB) and equipment with power supply cord can also be used if the deployment location has access to power supply.

8.3.2 Battery of mobile devices

In mobile the device Bluetooth scanning period and scan interval effects the battery life of devices. If an application that is looking for beacon all the time, this will drain the battery of the phone. Consuming massive amount of battery when it is not needed will damage the user experience. The users tend to uninstall applications that requires a lot of battery to function.

The device battery optimization can be achieved by scanning for beacons when there is a need. This approach limits the automatic triggering of actions for many services. Google NearBy API can also be used to scan for beacons. This API provides different strategies for beacon scanning such as search only for BLE devices. Applications can subscribe to messages from nearby devices in both foreground and background. The scanning is automatically managed by OS to conserve battery. The developers may lose full control over scanning customization for their needs; for example, hit and miss logic can define use cases of an application.

Geofencing can be used to conserve battery and maintain control over beacon scanning. This approach is used by Google's Awareness API. When a user enters, or exits a pre-defined geographical area, a notification is triggered to its subscribers. For example, in the museum use case entering a geofence around the museum can trigger automatic

beacon scanning. Users may use the application for indoor navigation, messages based on their location inside the museum and more. When leaving the museum, the application can automatically stop scanning for beacons.

9 Discussions

9.1 Beacons technology

Beacon technology is getting popular every day, not as predicted when launched in 2013 by Apple Inc. It is such a simple device that many service providers are still unable to master the best use out of it. Therefore, applications with beacons are still considered in its testing years. The power in these small battery-operated devices can be a data goldmine for businesses. As the industry is revolving around big data and machine learning, most of the services that are built with modern machine learning techniques are around contextual awareness. Users are still trying to grasp the idea of being tracked by beacons and other applications in the name of context awareness. The application providers must provide a great deal of value to the users to encourage them to use the applications. In Bluetooth 4.2 specification, users can opt-in to be tracked by Bluetooth device. This gives more control to the end users.

In Bluetooth 5, which is recently launched; the range is claimed to be doubled and Bluetooth low energy specifications are optimized for internet of things applications. This shows great improvements and future of the technology itself. However, the best uses of this technology are yet to be seen.

9.2 Context Awareness

In chapter 4, we discussed three of the five Ws, what, where and who. There are two more Ws yet to be discussed When and Why. When as in the future, when this will happen again. We can collect as much data as we want, but it is meaningless unless analysed and used properly. The prediction when something will happen in the future and when someone will need something, brings the richest context awareness to the applications. Having understood 5 Ws in a context can provide a great user experience and value to the end users.

A user of Google application gets a notification about the traffic situation at a specific time. The traffic situation and time is personalized to all users. A person commuting with train between his house and work around 4 and 5 pm will get the timetable, changes in schedules and other updates around the same time. A person driving during this time with same route receives a notification about estimated time of arrivals for alternative routes to the destination. This service is a result of contextual information when and why: When a user needs an information, why she needs to commute, and how a user commutes is another added contextual information.

10 Conclusion

The goal for this final year project was to conceptualize context-aware digital services. A simple application was built to demonstrate the implementation of context-aware digital services. It was concluded that Bluetooth LE beacons can be used for context awareness in the mobile application. There are some limitations such as battery optimization, fluctuating signals, inaccuracy of distance and reliability of information. However, beacon technology is still considered young and rapid pace development especially with Eddystone, launched in 2015, and Bluetooth 5 launched in 2016, the future of beacons seems very strong. ABI research forecasts 400 million beacons deployment in year 2020. In 2016, it was reported that around 5 M millions are deployed. [14].

The application produced in this project demonstrates that developing contextual awareness application can be as simple as scanning for beacons. Other discussion about different kind of contextual information can also be gathered easily. There are main public APIs provided by Google and other vendors, which makes developing context-aware applications very simple for the user. There were many use cases discussed, few of the are already in use. One use implemented in Korea shows the simplicity of beacons. Pregnant ladies are given a beacon, which triggers “Empty the seat” light in public transport. In Korean culture, asking for a seat may be considered impolite by some people. In Romania, a service is built to guide the visually impaired people about the buses and bus stops nearby, a beacon also triggers a notification to bus driver that a visually impaired person might need assistance.

LetMeKnow Android application can be further improved using Eddystone protocol to prevent spoofing, reducing the fluctuating distance accuracy by using one of the methods discussed in chapter 7.2 and improving the UI and UX. Furthermore, configuration of

beacon UUID to search nearby can be added, and database nodes for each beacon can be created. This could allow filtering of the users based on beacons.




























Collecting contextual data, analysing the data, optimizing constraints for the better user experience and most importantly providing a great value for the end users: in digital service is all possible in today's world.

References

1. Weiser M, Gold R, Brown JS. The origins of ubiquitous computing research at PARC in the late 1980s. IBM Systems Journal. 1999; 38(4): p. 693-696.
2. Weiser M. The Computer for the 21st Century. Scientific American. 1991 September; 3(3): p. 94-104.
3. Weiser M, Brown JS. The Coming Age of Calm Technology, The Next Fifty Years of Computing New York: Springer New York; 1997.
4. Park A, Steffen L, Wilhelm M. Location Based Services for Context Awareness-Moving from GSM to UMTS. Proc. SSGRR. 2004 January.
5. Schilit BN, Theimer MM, Welch BB. Customizing mobile applications. InProceedings USENIX Symposium on Mobile & Location-independent Computing. 1993 August; 9.
6. Gast MS. Building applications with iBeacon. 1st ed. United States: O'Reilly Media, Inc.; 2014.
7. Google Inc. Github. [Online]. 2016 [cited 2017 April 16. Available from: <https://github.com/google/eddystone/blob/master/protocol-specification.md>.
8. Google Inc. Firebase Documentation. [Online]. 2017 [cited 2017 April 16. Available from: <https://firebase.google.com/docs>.
9. Accent Systems. Accent Systems. [Online]. 2016 [cited 2017 April 17. Available from: <https://accent-systems.com/product/ibks-plus/>.
10. Radius Network. Altbeacon Specifications. [Online]. 2016 [cited 2017 April 17. Available from: <https://github.com/AltBeacon/Android-beacon-library>.
11. Radius Network. Github, Official Java API Documentation. [Online]. 2014 [cited 2017 April 17. Available from: <http://altbeacon.github.io/Android-beacon-library/javadoc/index.html>.
12. Google Inc. Github - FirebaseUI-Android documentation. [Online]. 2017 [cited 2017 April 17. Available from: <https://github.com/firebase/FirebaseUI-Android/>.
13. Solinor Oy. Solinor Blogs. [Online]. 2016 [cited 2017 April 18. Available from: <https://solinor.fi/2016/11/22/building-services-with-ble-beacons/>.



























14. Business wire. [Online]. 2016 [cited 2017 April 18. Available from: <http://www.businesswire.com/news/home/20160126005779/en/Beacons-Track-Hit-400M-Deployed-2020-Reports>.

Appendix 1. Battery life estimation for iBKS plus Alkaline

Slots Enabled	Tx Power (dBm)							
	-30	-20	-16	-12	-8	-4	0	+4
 iBeacon	91	91	90	88	88	87	84	77
 UID Eddystone	90	90	90	88	88	86	84	77
 URL Eddystone	90	90	90	88	88	86	84	77
 EID Eddystone	92	92	92	90	90	89	87	81
 iBeacon  TLM Eddystone	85	85	85	83	83	82	79	73
 iBeacon  URL Eddystone	69	69	68	66	66	64	61	55
 UID  TLM Eddystone Eddystone	85	85	84	83	83	81	79	73
 UID  URL Eddystone Eddystone	69	68	68	67	66	64	61	54
 URL  TLM Eddystone Eddystone	85	85	84	83	83	81	79	73
 EID  URL Eddystone Eddystone	70	70	69	67	67	66	63	56
 EID  TLM Eddystone Eddystone	92	87	86	86	85	84	81	76
 iBeacon  URL Eddystone  TLM Eddystone	69	68	68	66	66	64	61	54
 UID  URL Eddystone Eddystone  TLM Eddystone	68	68	67	65	65	65	61	54
 EID  URL Eddystone Eddystone  TLM Eddystone	70	69	69	67	67	65	62	56

Notes: Battery Life in months. Battery Capacity 4 x LR6 Alkaline: 5400mAh

Appendix 1. Battery life estimation for iBKS plus Lithium

Slots Enabled	Tx Power (dBm)							
	-30	-20	-16	-12	-8	-4	0	+4
 iBeacon	103	102	102	102	101	100	98	93
 UID Eddystone	102	102	102	101	101	100	98	93
 URL Eddystone	102	102	102	101	101	100	98	93
 EID Eddystone	104	104	103	103	102	101	100	96
 iBeacon	99	99	99	98	97	97	95	90
 TLM Eddystone								
 iBeacon	87	86	86	85	84	83	80	74
 URL Eddystone								
 UID Eddystone	99	99	98	98	97	96	94	90
 TLM Eddystone								
 UID Eddystone	86	86	86	85	84	83	80	74
 URL Eddystone								
 URL Eddystone	99	99	98	98	97	96	94	90
 TLM Eddystone								
 EID Eddystone	87	87	87	86	85	84	81	75
 URL Eddystone								
 EID Eddystone	100	100	100	99	99	98	96	92
 TLM Eddystone								
 iBeacon	86	86	86	85	84	83	80	74
 URL Eddystone								
 TLM Eddystone	86	86	85	85	84	82	80	73
 UID Eddystone								
 TLM Eddystone	87	87	86	86	85	84	81	75
 URL Eddystone								
 EID Eddystone	87	87	86	86	85	84	81	75
 TLM Eddystone								

Notes: Battery Life in months. Battery Capacity 4 x LR6 Lithium: 10400mAh