2016

Context Aware Application Using Beacons

Abdullah Nasim
Bard College

Context Aware Application using Beacons

A Senior Project submitted to
The Division of Science, Mathematics, and Computing
of
Bard College

by
Abdullah Nasim

Annandale-on-Hudson, New York
May, 2016

Abstract

With the growing popularity of mobile computing, as well as the increasing demand for location-based services, this paper introduces a design and implementation of a location-aware system using Beacons. This project comprises three parts: a web application, back-end server and a mobile application client. The web application serves as a content management system by allowing users to manage projects, input data and create associations between Beacons and data. The backend server contains application logic and a database. The mobile application client interacts with the Beacons and provides location-specific services.

Contents

List of Figures

Dedication

To my parents and my homeland, Pakistan.

Acknowledgments

I would first like to thank Ms. Heidi Knoblauch for giving me the chance to work with Experimental Humanities on this project. I also want to thank Ryan Sablosky for his constant patience and help throughout this process. I also want to extend my heartfelt gratitude to my supervisor Sven Anderson whose guidance and encouragement were very vital in completing this project.

1
Introduction

1.1 Problem Description

In current times, the internet and mobile technology allow anyone with a smartphone to access a vast amount of information. Recent studies show that in the U.S. web surfing on mobile devices accounts for 52% of the time spent consuming digital media [1]. Another study reveals that the ownership of smartphones exceeds that of desktops and laptops around the world [2]. With increased ownership of smartphones and high-speed internet, the social trends and habits of consumers have significantly changed. Consumers demand information on their mobile devices when and where they need it. This has led to development and research in the field of mobile computing, in particular, context-aware mobile applications. Context-aware mobile applications can serve to anticipate the needs of the consumer and provide services based on their location. There have been several systems developed in the past few decades aiming to provide location of consumers and objects. Currently, the Beacon technology has become one of the major emerging technologies that is being used around the world to enable mobile applications to provide location-based services.

1.2 Project Goals

The goals of this project are to create an open-source Android mobile application and a web application. This project is under the department of The Experimental Humanities (EH) at Bard College. We plan to create an interactive exhibition about Mass Incarceration using Estimote Beacons. Estimote Beacons are tiny computers that broadcast radio signals and can be used to produce context-aware mobile applications. Smartphones within range of the beacons can detect and receive the broadcasted signals. Mobile applications can be developed that act upon these signals and push notifications, images, text, sound, and videos depending on the functionality of the app. We intend to develop a mobile application that will allow participants to receive information regarding Mass Incarceration as they explore the installation. The supporting web application serves to manage the project and content which is delivered by the mobile application. The installation will provide digital stories about the impact of incarceration on New York City neighborhoods and spark conversation about how Beacons can be used to raise awareness about social issues.

The ultimate goal is to produce an open source system consisting of a web application and a mobile application. The web application will allow users to self-host their own Beacon-based projects and the mobile application will be automatically configured to provide location-based services according to the project setup. Therefore the system can be replicated and used by others to create context-aware mobile applications with little or no modification aside from the input dataset.

1.3 Contributions

The idea of the project was conceived by Heidi Knoblauch who is the Digital Coordinator of EH. Heidi has worked tirelessly to obtain approval and funding to establish the project. Ryan Sablosky, the web developer of EH, has setup the backend server which consists of

the Flask application, Web API and the database. As part of my senior project, I worked on the mobile application, the front-end templates and views in the backend.

2
Background

In 2013, Pew Research Center reported that 74% of adults ages 18 and older used their smartphones to get directions or other information based on their current location [3]. This number which has surely increased over the years as the ownership of smartphones increases. These statistics are indicative of the immense need and potential in location-based services through mobile computing.

2.1 Context Awareness

Imagine you enter a departmental store and your smartphone pushes a welcome notification and information regarding current deals and discounts. As you pick up an object on display your phone vibrates and lets you know of the price, the colors available, and the availability of sizes in the store. Instead of having to find an employee and asking them, your smartphone will provide you with this information. This is an example of a context-aware system using a mobile application.

A formal definition of Context-aware computing is defined by Gartner IT as "a style of computing in which situational and environmental context about people, places and things

is used to anticipate immediate needs and proactively offer enriched, situation-aware and usable content, functions and experiences." [4]

Some important aspects of context that are used in these systems are based on [6]:

- Identity (Who)
- Activity (What)
- Time (When)
- Location (Where)

In this paper we will focus on using location as context which in its broader sense is not just limited to the geographical location but also the objects that build up the surrounding environment. Mobile applications which are context aware can discover nearby objects and take advantage of this contextual information. Such applications can anticipate the potential needs of the user and act accordingly in advance thus providing the user real-time information when and where they need it without having to manually search for it.

2.2 Location Systems

Currently, there is no standard technology which is used for providing accurate location indoors and outdoors, and many various techniques and solutions have been proposed. The most popular technique is to use Global Navigation Satellite Systems (GNSS). GNSS are systems of satellites that are used to pinpoint the geographic location of a user's receiver anywhere in the world. Using information from a number of satellites the system can triangulate the location of the user to a precision of a few meters [7]. There are currently two operational GNSS: The United States' Global Positioning System (GPS) and Russia's Global Orbiting Navigation Satellite System (GLONASS). There are also two

GNSS under development: China's BeiDou Navigation Satellite System and the European Union's Galileo [8].

Perhaps, the most widely used GNSS is the GPS. The United States Department of Defense created the system initially for military use in 1978. Later, it was made fully operational for all civilians to use in 1995 [8]. Since then GPS has made a significant impact on our lives. It has led to the development of hundreds of applications in several domains such as transportation, construction, farming, mining and many others.

However, there are a few challenges with GPS when it comes to using it as a sensitive location context provider. GPS provides location to a precision of 5 meters [8] and therefore it can be used to pinpoint a user at a neighborhood block but it can not definitively be used to tell if the user is in a particular aisle in a store or standing near a particular object. Another drawback of GPS is that its accuracy can be affected by obstacles in its signal path such as atmospheric conditions, buildings and trees [9]. Therefore GPS does not work accurately as a reliable positioning system in indoor and underground locations. Lastly, since GPS requires communication with several satellites it can be a huge battery drain on devices such as smartphones [9]. Hence GPS is not a feasible system when a smartphone needs to constantly monitor location to provide services to it's users.

Besides using systems that rely on satellites for navigation, companies have done extensive research on creating alternatives that are more sensitive than GNSS and can also work indoors.

Perhaps the first location system was developed by AT&T Cambridge in the 1990s called the Active Badge location system. It is an indoor location system that serves to locate individuals and/or objects within a building by determining the location of the active badges on them. These badges emit infrared carrying a globally unique identifier every 10 seconds. These signals are detected by IR sensors around the room and sent to a central server using a network which in turn provides the location information. The system

was a significant first step however it had problems due to the usage of infrared. Infrared detection requires the emission to be within the line of sight thus the positioning of the sensors is very important and can easily be obstructed by moving objects. Infrared from the sun and fluorescent lights also interfere with the infrared emitted from the badges and thus cause inaccuracies in the system. Also, the set up and maintenance of the system is costly and require several badges, sensors and a strong network which prevents the system to be employed in larger rooms [10].

Another system called Active Bat location system employed similar techniques as above utilizing ultrasounds. AT&T researchers in 1999 employed the techniques used by bats in dark caves to locate objects. Users and objects carry Active Bat tags which emit a unique ID via ultrasound. These signals are received by ceiling-mounted receivers which compute its distance from the Bat. The system can locate these bats to within 9 cm of their true position for 95 percent of the measurements. However such a system requires a large ceiling sensor infrastructure and is thus expensive and not scalable to larger areas [11].

These systems are hard to implement mostly due to the need for additional hardware to detect IR and ultrasound signals and to compute the location. Smartphones do not come with built-in sensors for these signals and thus systems based on these frequencies are not feasible for location-aware mobile computing. Therefore we turn to systems that are based on technologies that are compatible with recent smartphones.

A Microsoft Research group developed RADAR which is a building-wide tracking system using IEEE 802.11 WaveLAN wireless networking technology. The system includes wireless devices which emit radio waves and a base station which detects and measures the signal strength. The base station then uses this data to calculate the location of the wireless devices within the building. This system has a few advantages over the systems mentioned above. Since it uses RF the system has a large range and does not require line of sight. It also utilizes the building's wireless network to operate thus there is no need for additional

infrastructure. Also smartphones come equipped with the ability to connect to wireless LAN. However, like any system there are drawbacks to this approach. In order to locate an object, the object must not only support wireless LAN but also be connected to it thus limiting the smartphone to indoor settings. Also, RADAR locates objects to a precision of about 3 meters of their actual position and thus cannot be used as a sensitive location provider [12].

Since the creation of Bluetooth in 1994 the potential that Bluetooth communication offers has drastically increased. Bluetooth is a wireless technology standard for exchanging data over short distances using short-wavelength UHF radio waves [13]. Bluetooth has played a vital role in helping to make devices wireless and allow fast communication across several types of electronics. Nowadays, Bluetooth can be found in almost every mobile device, computer, television and even cars. Bluetooth devices communicate with each other by becoming paired. A device can be set to be "discoverable" which allows it to be discovered by other bluetooth-enabled devices. After the devices are paired they can transmit data to each other. Once the pairing is established it is stored in memory which allows the devices to be automatically paired in the future when they are near each other [14].

In 2010, the release of Bluetooth Low Energy (BLE) put Bluetooth technology on the forefront of technologies to provide context. Compared to older versions of Bluetooth which could transmit large amounts of data such as music and videos, BLE serves to transmit small packets of data hence minimizing power consumption. This allows for smaller devices such as smart watches to utilize BLE and operate on coin-cell batteries [15]. Nowadays BLE is supported by most mobile and computer operating systems including iOS, Android, Windows, OS X, Linux, and Windows 8.

In 2013, during the WorldWide Developers Conference, Apple introduced the iBeacon protocol which uses BLE to transmit an advertising packet. iBeacon is built upon the core

of BLE and serves to standardize the advertising packet transmitted by BLE. Therefore, all smartphones that detect BLE can also detect iBeacon. The iBeacon advertising packet consists of:

Field	Size	Usage
Universally Unique Identifier (UUID)	128 bit	distinguishes a Beacon (or a group of Beacons) from another
Major	16 bit	optional number used to distinguish a subqroup
Minor	16 bit	an optional number used to distinguish individual Beacons within a subgroup
Measured Power	8 bit	a factory-calibrated constant for the expected RSSI measured 1 meter from the Beacon

Table 2.2.1. iBeacon advertising packet

Currently, iBeacon has created a huge buzz in the technology industry and is widely being used and experimented with. This has led to the development of Beacons which utilize iBeacon to provide location as context and help develop context-aware applications.

Beacons have gained popularity in serving as a sensitive location provider since it can provide context to smartphones to a precision of a few centimeters which trumps all previous solutions mentioned above. Since it does not rely on large supplementary hardware, Beacons can be easily installed both outdoors and indoors. A comparison of all the solutions mentioned in this section is summarized in Appendix A.

2.3 Beacons

According to Wikipedia, a beacon is an intentionally conspicuous device designed to attract attention to a specific location [5]. The word is widely associated with images of lighthouses serving as navigational aids for ships near coastlines.

In the realm of technology, Beacons are small wireless devices which constantly transmit iBeacon signals at regular intervals. Mobile applications can detect these signals, identify the beacon by the identifier and trigger a location-based action on the device. Apple introduced iBeacon protocol but do not manufacture the beacons. The physical beacons are made by third-party manufacturers such as Estimote, Kontakt, Swirl and several others.

Mobile Applications detect and interact with Beacons in two main ways: Region Monitoring and Ranging. Region Monitoring involves defining a region of one or a set of beacons using the UUID and/or major and/or minor values of the beacons. Similar to geofencing, region monitoring enables the application to push notifications when the smartphone enters or exits the defined region [16]. The unique ability of this interaction is that even if the application is not running, the application is launched in the background and the notification is pushed. An example of its real-world application includes a user being instantly notified about the current deals as they enter the store.

While monitoring operates with a region, ranging involves detecting individual signals and estimating their proximities to the smartphone. It is important to note that iBeacon provides information about proximity and not exact location; iBeacon can answer what items are nearby but not where you are. Absolute location is a fixed location defined by longitudes and latitudes or some form of coordinate system. GPS is a system that provides absolute location information. However, proximity refers to the distance in relation to some object (in our case a beacon) and is capable of determining if an object is nearby but cannot determine where exactly it is [22]. The benefit of proximity is that mobile applications can identify nearby items of interest which are often independent of location and may also be mobile. For instance, in a store a user can be notified when an employee is near them.

Since most smartphones now support detection of BLE, it can estimate proximity to the beacon by using the signal strength to calculate Received Signal Strength Indicator (RSSI). The stronger the signal, the closer the beacon is to the smartphone. Since BLE does not require the smartphone to pair with the beacon, it can receive signals from multiple beacons. It infers which beacon is nearest to it by measuring and comparing their individual RSSI. Using iBeacon and RSSI, smartphones can also approximate the distance from the beacon. iBeacon signal transmits Measured Power (8 bits) which is a factory-calibrated read-only constant for the expected RSSI measured 1 meter from the beacon. Using this constant and the RSSI, an estimate of the distance between the receiver and the beacon can be calculated [22]. This calculation of the distance can be inaccurate if there are a lot of obstructions within the room which can dampen the signal strength.

2.4 Current Applications of Beacons

BI Intelligence, tech research, claims that beacon technology is the fastest-growing in-store technology since mobile credit card readers and predicts installation of 4.5 million active beacons in the U.S by the end of 2018 [17]. Industries that seek to improve customer experiences and maintain strong relationships with their clients have started implementing Beacons all over the world. Beacons have been deployed and tested in many sectors especially retail, entertainment, travel and tourism.

Perhaps the industry that will reap the most benefits (and profits) from using Beacons is retail. One prime example is Macy's deployment of 4000 beacons in stores throughout the U.S during the holiday season in 2014. Using beacons, retailers can send customers notifications about product specifications, videos of how to use products, coupons and deals. Retailers can also use the technology to monitor movements and patterns of costumers within the store in order to improve product placements and personalize notifications based on past purchases [18].

Airports and airlines have also taken up the technology. Virgin Atlantic set up beacons in the Upper Class Wing at Londons Heathrow Airport in May 2014. They offered a variety of services including personalized duty-free and currency exchange offers to passengers and alerting passengers to prepare their boarding passes as they near security check-ins [18]. Japan Airlines also continued the trend in Tokyos Haneda Airport Domestic Terminal 1. By setting up an indoor positioning system, the airlines staff is equipped with smartphones allowing the airlines to locate staff members and assign location-specific tasks to them [18]. American Airlines also deployed a trial run at the Dallas Forth Worth Airport in June 2014 to help passengers navigate the airport by providing them with information about distances to gates, boarding times and security check lines.

In the sporting world, Major League Baseball (MLB) took up the opportunity to provide location based advertising as well. In March 2014, it set up beacons in 28 of its 30 ballparks around the U.S. Through the MLB application, sports fans receive offers for merchandise coupons, information on concession stands and history of the venue, ability to check-in and even video clips of game replays. MLB reported that the number of check-ins tripled from 2014 to 2015 and that it is looking for ways to add to the application to further enrich the experience of the fans [18].

Starwood Hotels & Resorts is running beacons in 30 of its hotels and resorts. The beacons are helping the concierges in greeting guests by name and accelerating the check-in process for frequent guests. It has also implemented a pilot program that will allow Starwood guests at two of their U.S. hotels to skip the check-in process and along with a partner application unlock their room door using their smartphones [18].

After being popularized by Apple and big brands, local businesses and institutions have also started using Beacon technology to aid in their services. Craft brewery Schlafly Beer, a bar in St. Louis, MO, is utilizing beacon technology to better connect with their customers. Beacons are located on beer taps to push information about drink specials and new beer

releases to engage with customers at the key moment when they are about to order a drink [19]. In Bucharest, Romania, public transport is employing beacon technology to make the city more accessible for visually impaired citizens. Instead of having to rely on a friend for help, visually impaired citizens, using a mobile application, can now be notified by voice output when their bus pulls up and be directed to it through the station [20]. Beacons have also become popular in providing engaging experiences to visitors by acting as a virtual tour guide. In the famous Antwerp museum in Belgium, visitors are directed towards exhibits and notified about stories behind the exhibits. The beacons also stimulate interaction with visitors through interactive trivia questions [21].

It is clear with the increase in usage and applications around the world that beacon technology is a powerful tool in helping provide location as a context. Beacons are not only limited to promoting advertisements but can also serve to deliver valuable information to people on a daily basis.

2.5 Challenges with Beacons

Like any emerging technology, Beacons have drawbacks:

The main challenge of working with Beacons is the interference in signals due to obstructions in the area which can significantly affect the range of the beacons. Therefore, positioning of the beacons is important and they should preferably be placed on ceilings or on walls at a height higher than that of an average adult.

Another issue that has arisen is regarding the security issues with Beacons. Although beacons do not transmit any sensitive information, they do however transmit their identifier without any encryption. These identifiers can be detected by any smartphone and used by a competitor or malicious group in their applications. For instance in a store a user can receive notifications about better deals from a rival store simply because the rival

store piggybacks on the beacon network set up in the store. For this case to happen the user needs to have the application from the rival store installed [24].

In large spaces a considerable number of beacons are required to ensure continuous coverage. Conversely, in small spaces overlapping of beacons can be a problem especially if signal strengths are similar causing the mobile application to continuously toggle between the beacons [23].

On the mobile side, if the application requires fetching data from a server then Internet capabilities must be available which may not always be the case in the outdoors. Also, constant use of cellular data or Wi-Fi can quickly diminish battery life which serves to be counter-intuitive as BLE is preferred for its low energy consumption.

Since beacons are small physical devices that are usually attached to walls and objects, it is very easy for them to get stolen or vandalized [23].

Although beacons operate on BLE which enables the small devices to last for a long time, the batteries do run out eventually and require manual replacement which can be very timely and expensive depending on the number of beacons deployed.

2.6 Debunking the Myths

Similar to the advent of most new technologies, iBeacon has met with hesitation and reluctance which results from the lack of knowledge among people regarding the technology. The main concern consumers have is that Beacons track their locations and access their private information without their consent. In fact, Beacons simply transmit a unique identifier which is detected by a smartphone and causes mobile applications to react accordingly. Beacons do not receive data and thus do not contain any data about the users. Any malicious act such as tracking a customer's location and habits is done by the mobile application. There are several steps that the user undertakes to grant mobile applications consent to perform such acts. A user has to download the mobile application, accept the

terms and services agreement and activate their Bluetooth [29]. Thus Beacons themselves do not pose any threat to the user's privacy. The mobile application can but the user has complete control whether they consent to such acts or not. At any time they can reverse the decision by deleting the application or turning their Bluetooth off.

2.7 Alternative to iBeacon

Other companies have taken up a similar venture as Apple and created their versions of BLE standards as alternatives to iBeacon. Currently, the main competitor is Eddystone which was launched by Google in 2015. The main benefit of using Eddystone over iBeacon is that it has added to the iBeacon specification. One main difference is that Eddystone contains an additional identifier called Eddystone-URL which, as the name suggests, transmits a URL which opens up on the smartphone's built-in web browser and thus the smartphone does not require a mobile application [24]. Since Eddystone is relatively new, there are not many manufacturers who are producing Beacons with the Eddystone specification. Although Eddystone has been released, Google is still working on improving Eddystone and adding more identifiers that can be used for different purposes. Hence, iBeacon continues to be the popular specification used to manufacture Beacons and these iBeacon-based beacons are being used in several running applications and projects.

3
Project Requirements

3.1 Definitions

- **Beacons**: Beacons that provide context to smartphones
- **Item**: A document containing text, image and/or video
- **Project**: Based on a topic, a project consists of one-on-one associations between a set of beacons and items
- **Users**: Users create and maintain projects on the web application
- **Participant**: A person who downloads the mobile application and enters an installation
- **Installation**: An exhibition set up in a physical space by users using beacons to provide participants location-specific information.

3.2 Use Case Diagrams

To clearly illustrate the requirements and models of a system, Use Case Diagrams are used. Use Case Diagrams consist of Actors which represent external agents and Use Cases

which represent the actions or sequence of actions that the Actors can take. In our project there are two external agents: Users and Participants.

1. Users:

   Users manage projects and content by using the web application. Users can have three roles: Admin, curator and collector. These roles differ in terms of access and permission to the projects and its contents. The roles and their respective permissible actions are shown in Figure 3.2.1.

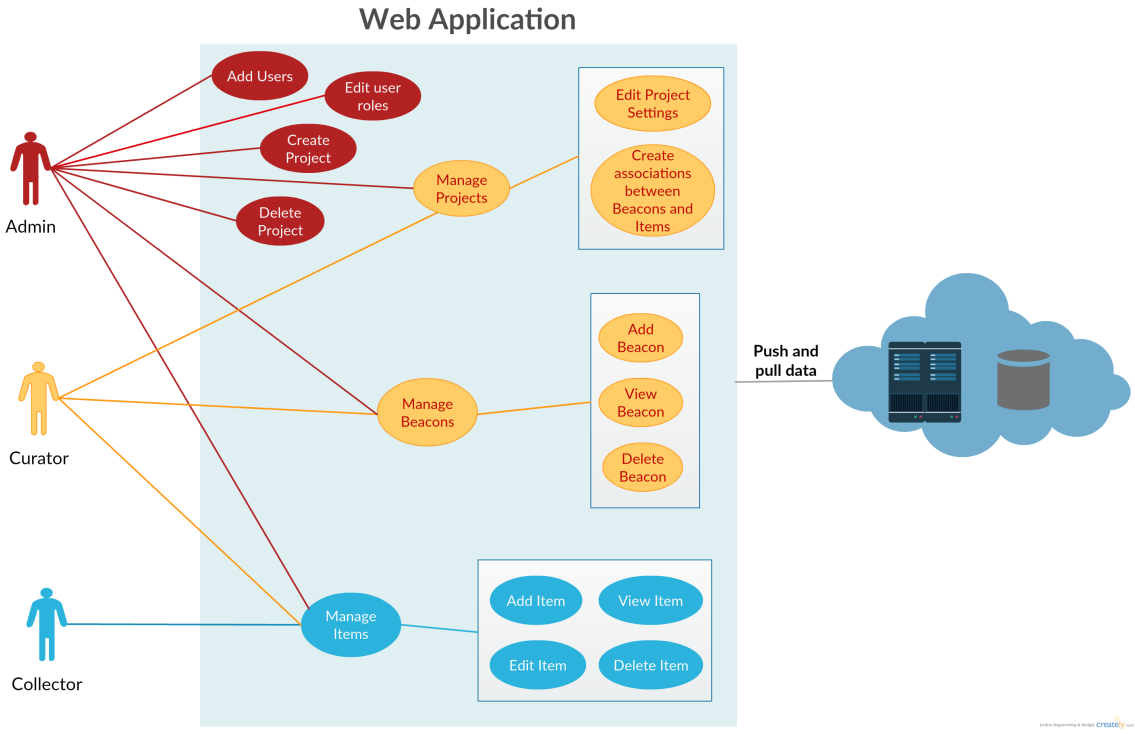


Figure 3.2.1. Use Case Diagram for Web Application

- **Admin**, labeled in red, has the most control. Admin can create new projects, add curators and collectors, assign users roles and permissions to projects, change the roles a user has, and manage the project in every way.

- **Curators**, labeled in yellow, have the second most control. Curators can manage the projects that they are authorized for, however, can not create projects or edit user roles.
- **Collectors**, labeled in blue, have the least control. Collectors can only manage items for projects that they are authorized for.

The users set up the physical installation by placing Beacons on walls and objects based on the items they have associated with the Beacons in the web application.

2. Participants:

   Participants interact with the physical installation by using their mobile application. They move within the installation and based on their proximity to a Beacon the mobile application displays the associated item.

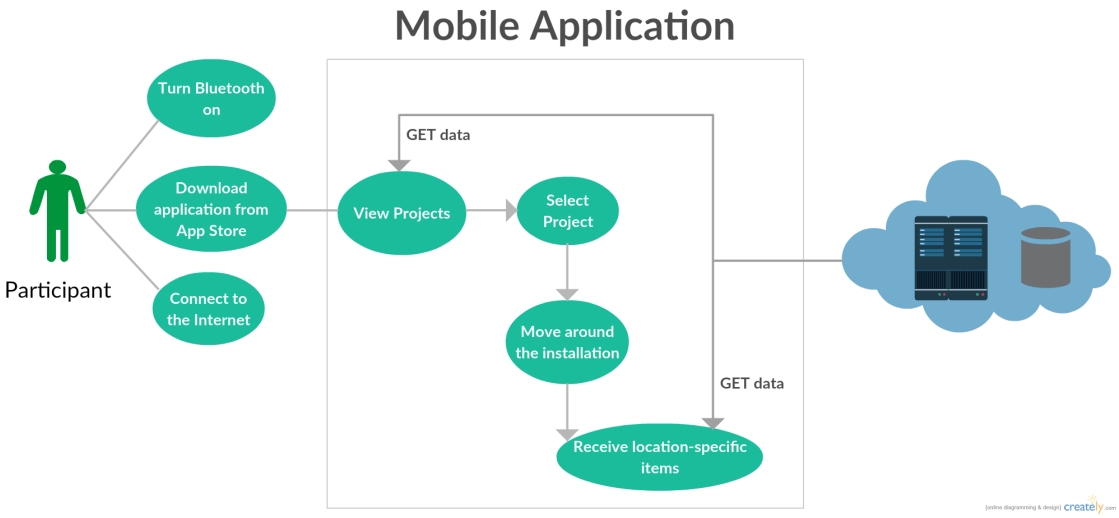


Figure 3.2.2. Use Case Diagram for Mobile Application

In both the Use Case Diagrams there is an external data source which is used to store and retrieve data. With these diagrams, we have established the Actors in our system and the requirements that our system should provide the Actors. In the next chapter we design the architecture of the system that will accomplish these requirements.

4
System Design

The architecture of this system is designed on a client-server model. The client-server model is a distributed application structure that partitions tasks between the providers of a service, called servers, and those who request the services, called clients [25]. Client-server model helps divide the different processes in the application to optimize work done at the client and server side. It also reduces redundancy since the data is stored on the server instead of each client.

There are two main types of architecture systems based on this model: 2-tier and 3-tier. The 2-tier architecture involves only a client and a database server as seen in Figure 4.0.1. In this architecture, clients are called thick because they hold all the application logic and directly connect to the database to retrieve information without involving any intermediary structure. The main disadvantage with the 2-tier approach is difficulty in scalability and upgrades [26]. Therefore most systems now implement a 3-tier architecture.

In a three-tier architecture the software is divided into 3 different tiers: Presentation tier, Application tier, and Data tier (Figure 4.0.2). The client is part of the presentation tier which contains presentation logic and provides user interfaces. The Application tier which

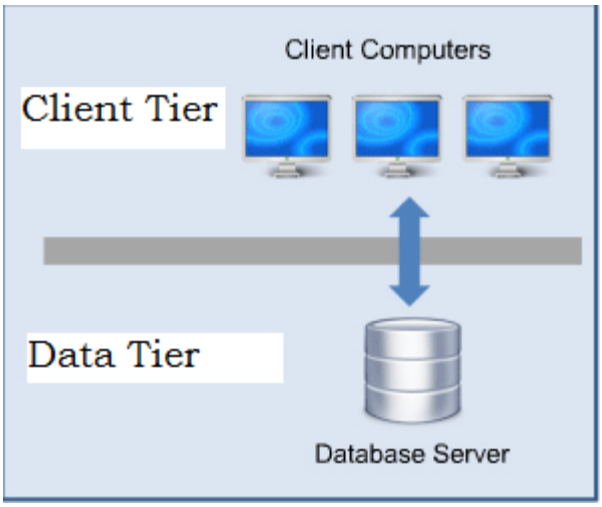


Figure 4.0.1. 2-Tier Architecture

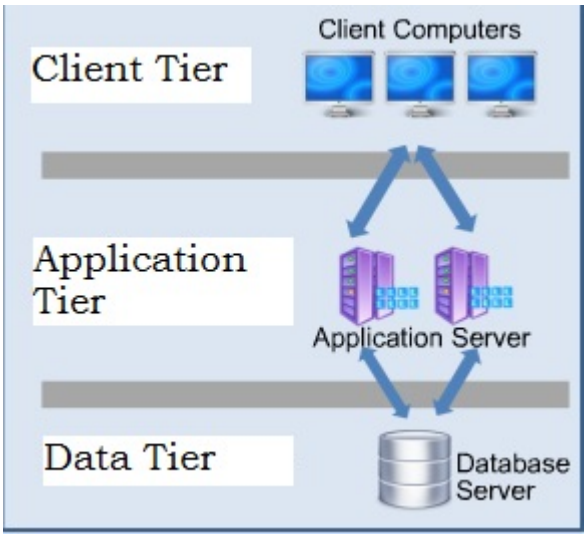


Figure 4.0.2. 3-Tier Architecture

holds most of the application logic serves as the intermediary between the Presentation and Data tiers. The Data tier is the database server that stores and retrieves data. Since the presentation tier holds little application logic it serves to get user inputs and send requests to the application tier, and receive responses from the application tier and accordingly update the interfaces. The application tier processes requests and in turn calls on the data

tier to provide it with the required data [26], and it processes said data and sends the response back to the client. The advantage of a 3-tier architecture is that the modularity increases scalability and flexibility by allowing upgrades and replacements to be done independent of other components.

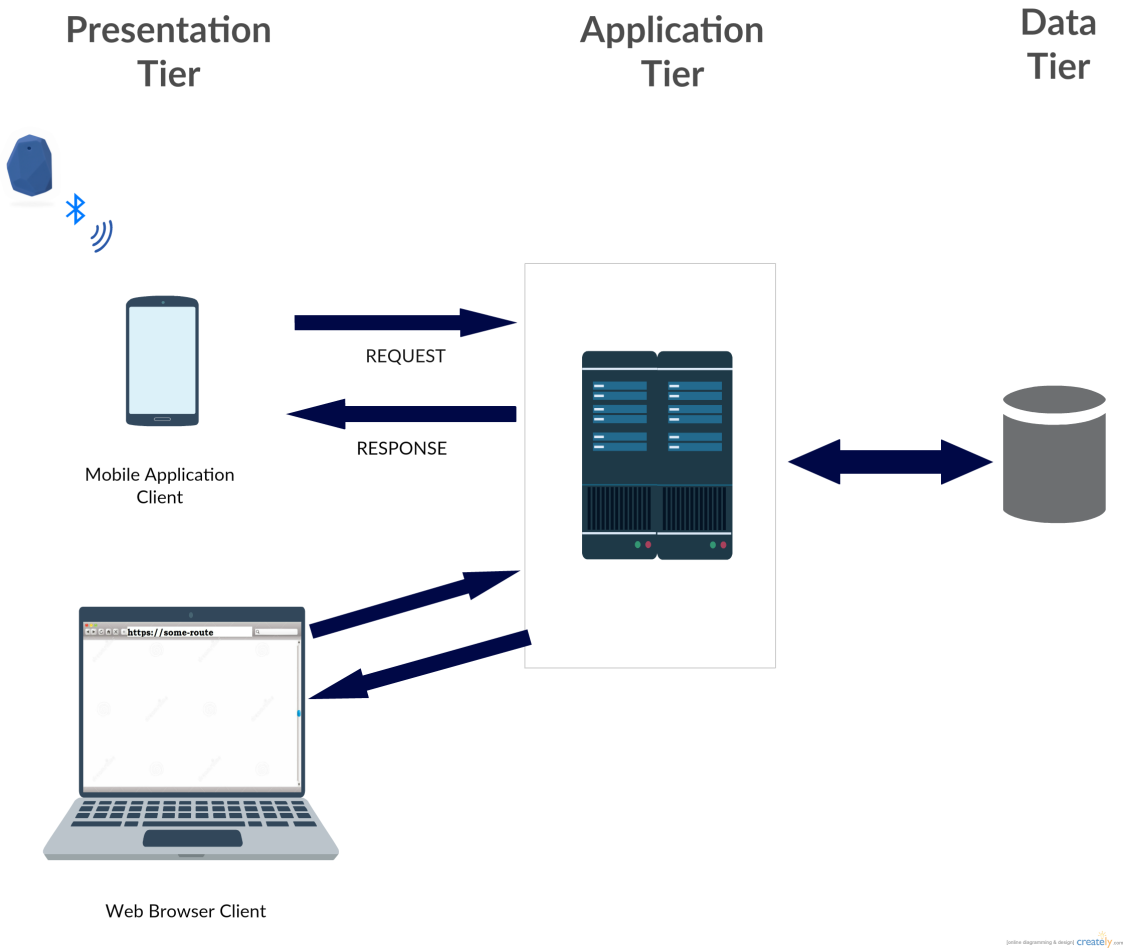


Figure 4.0.3. Initial System Design

Our project employs a 3-tier client-server architecture system as shown in Figure 4.0.3. The presentation tier includes the mobile application that the participants download and use to interact with the installation, and web browsers which render the web application that users use for content management. The application tier holds the application logic and receives requests from clients and responds accordingly by pushing to and pulling

from the data tier. The Data tier comprises of the database which stores information about users, projects, beacons, items and the associations between beacons and items.

5
Technical Implementation

In this chapter we expand our initial system design and add technical details to each tier which includes the technologies used and specific file names in our project. The modified design is shown in Figure 5.0.1.

The chapter is divided into sections which elaborate each tier, its respective components, application development environments and relevant technical code.

5.1 Estimote Beacons

There are currently several third party companies that manufacture Beacons based on Apple's iBeacon protocol. There are a few factors that need to be considered when deciding which Beacon to purchase and use in the project. Perhaps two of the most important factors are the battery life and the companies' supporting software for Beacon management. Estimote is by far the most popular company in the market currently. Using coin size battery Estimote promises upto three years of battery life. Estimote also provides a Beacon management system where users can view their Beacon settings such as battery life and edit the iBeacon advertising packet such as major and minor values. Because of

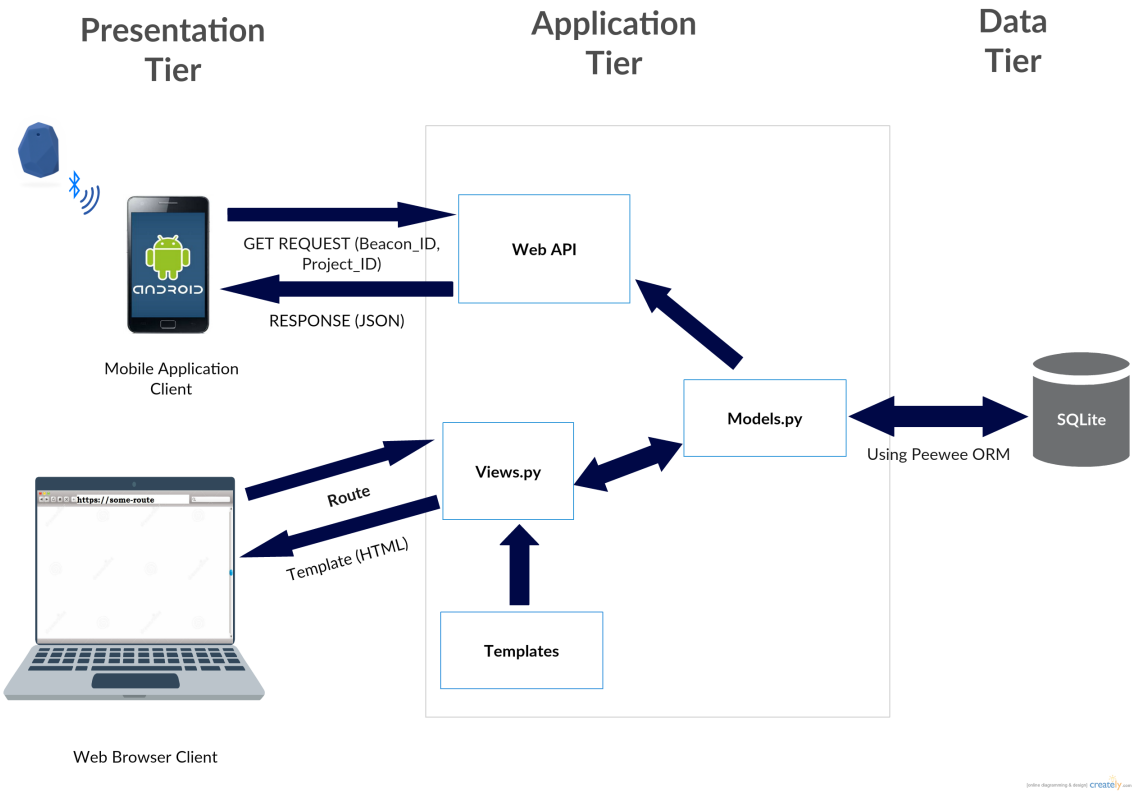


Figure 5.0.1. Final System Design

these reasons we decided to purchase Estimote Beacons and use them in the project. I worked with a starter development kit of 3 Estimote Beacons.

5.2 Data Tier

The database used is SQLite which is a relational database management system. SQLite provides high performance and simplicity since the entire database consists of a single file making it great for testing and development purposes. The database is managed and tested using DB Browser for SQLite which is an open source tool and provides simple visuals to view and edit the database.

Appendix B shows the schema of the database. The tables are organized and related in a manner to reduce redundancy and simplify querying across multiple tables. There are

several many-to-many relationships in the database for instance from projects to items where many projects can share relationships to many items. The same holds for projects to beacons and projects to users, thus, there are through-tables to store these many-to-many relationships.

- **User:** stores all the users registered in the system along with their personal information. The primary key is the ID field which is used to reference users in other tables.
- **Role:** stores all the possible roles of the users- Admin, curators and collectors.
- **UserRoles:** stores all the users along with their respective roles.
- **Project:** stores all the projects along with its information. The primary key is the ID field which is used to reference projects in other tables.
- **Beacon:** stores all the Beacons along with their iBeacon fields (UUID, Major, Minor). The primary key is the ID field which is the unique Beacon ID and is used to reference Beacons in other tables.
- **Item:** stores all the items along with their information. The primary key is the ID field which is used to reference items in other tables.
- **BeaconItemMap:** stores all the associations between beacons and items along with their respective projects.
- **Project_User_Through:** tracks many-to-many relationships between projects and users.
- **Project_Beacon_Through:** tracks many-to-many relationships between projects and beacons.

- **Project_Item_Through:** tracks many-to-many relationships between projects and items.

5.3 Application Tier

The application tier consists of the Flask web application which is rendered on the web browser client and is used by users to manage projects, and the Web API which publishes RESTful services containing data from the database which is fetched by the mobile application client to display content.

5.3.1 Web Application

The web application is built using Flask. Flask is a micro web framework written in Python based on Jinja2 template engine. Flask serves to keep the web application development simple and extensible [27]. Flask supports the addition of several extensions which add extra functionality to the application. Jinja2 is the default templating language for Flask. Jinja2 serves to combine a template written in HTML and CSS with logic and data structures to render dynamic web pages.

The web application is deployed and tested locally on the computer using Virtualenv. Virtualenv is a useful tool that creates isolated Python development environments. The application code is written and managed using Microsoft Visual Studio which provides a powerful editor for both python and HTML files.

Flask, like most other web frameworks, employs the Model-View-Controller (MVC) architectural pattern to modularize code by separating the data model (models.py), application logic (views.py) and user interface (templates folder) [28].

5.3.1.1 Models.py

In large projects with complex database schema querying the database using SQL becomes tedious thus using Object-relational mappers (ORM) is preferred. ORM maps data

stored in relational database tables into objects. Usage of ORM leads to huge reduction in code and allows developers to extend object oriented programming to dealing with data from databases as well. The main benefit of ORMs is that by mapping relational tables to classes, developers can refer to objects of the classes and their fields throughout the application without having to query SQL CRUD (Create, Read, Update, Delete) logic each time. The ORM used in the project is Peewee which is a simple and easy to use ORM. Flask-Peewee is a Flask extension that adds support for Peewee to the Flask application.

The file *models.py* establishes connection to the database and defines model classes. Model classes map to relational tables in the database, fields (or attributes) map to columns in a table and therefore an object refers to a row in a table. For example the Project table in the database is mapped to:

```
class Project(BaseModel):
    """Model for Projects"""
    name = TextField(unique=True)
    description = TextField()
    beacons = ManyToManyField(Beacon, related_name="projects")
    items = ManyToManyField(Item, related_name="projects")
    approved_items = ManyToManyField(Item, related_name="projects_approved")
    users = ManyToManyField(User, related_name="projects")
```

Due to this class definition, an instance of the *Project* model class can be used to refer to a particular project from the database and its attributes like name and description can be obtained in the same manner as in object-oriented programming:

```
project = Project.get(id=project_id)
name = project.name
```

Any modifications to the objects and their fields throughout the application are reflected in the database thus *Models.py* serves to connect to and manage data in the database using the Peewee ORM.

5.3.1.2 Views.py

Views.py contains controller methods which are associated to a route. When the user enters a URL, the application matches the URL with the predefined routes and runs the associated controller method. Within the controller method, if necessary, the model classess are used to retrieve data from the database. The data is stored in a structure for instance a list or a dictionary and then passed to a view which renders the HTML page.

```
@bp.route("/profile")
def profile():
    projects=Project.select()
    return render_template("profile.html", projects=projects)
```

The image above shows the profile controller which is run when the user types or is directed to the */profile* URL. It accesses the Project class shown in section 5.3.1.1 and retrieves all the objects in a list. It then passes the list to the *profile.html* template which is then rendered and displayed on the web browser client.

5.3.1.3 Templates Folder

The templates folder contains all the HTML web pages which are rendered along with data structures passed by the controller methods using Jinja2. The main benefit of using Jinja2 templates is that the template can contain variables, statements and logic expressions which are evaluated and replaced with corresponding values when the page is rendered. Statements and logic expressions such as if statements and for loops are placed within {% %}, and variables are placed within {{ }}. We continue with the example above of the profile controller that passes a list named projects to the *profile.html* template.

```
{% for project in projects %}
<tr>
    <td>{{loop.index}}</td>
    <td>{{project.name}} </td>
    <td>{{project.description}}</td>
```

The image above shows a portion of the *profile.html* where a for loop iterates over all the projects and enters the name and description fields of each project into a table.

Another benefit of using Jinja2 is using inheritance within templates. Instead of reusing existing code in different templates (for example header and footer) templates can inherit other templates. *profile.html* inherits the *layout2.html* template. In order to do so textit-layout2.html must specify a labeled block that can be overwritten by other templates:

```
<div>
{% block content %} {% endblock %}
</div>
```

profile.html must include:

```
{% extends "layouts/layout2.html" %}
```

and the labeled block content:

```
{% block content %}
        <div class="content">···
       </div>
{% endblock %}
```

Similarly, there can be several blocks placed within a template that can be overridden by another template that inherits it for example blocks to specify the title, content, caption etc for the web page.

In templates, static files for CSS and JavaScript are also required. The standard approach is to create a static folder which contains all style sheets and other static content that the templates may require. Referencing these files in the templates is achieved by using the Flask *url_for* which generates a URL for the static file as seen in the image below.

```
<link href="{{ url_for('static', filename='css/layout.css') }}" rel="stylesheet">
```

5.3.1.4 Flask-Security

Flask-Security is an extension that adds security mechanisms like user authentication, authorization, registration and password recovery to a Flask application. It is also used to track the current user, their login activity and their respective role.

Flask-security is a very powerful tool that handles most of the authentication for the developers. It provides a module named *current_user* which can be accessed in templates and python files. It refers to the current user who is logged in and is an object of the *User* model class in *models.py*. Therefore, the fields of the *current_user* can be retrieved in the same object oriented manner as described in section 5.3.1.1. Unlike other objects *current_user* is a global variable throughout the project and does not need to be passed to templates. Hence, rending user specific information on web pages become simple. For example, if the *current_user* is logged in then the webpage will show their email and the option to Logout, otherwise it will show the option to Login.

```
{% if current_user.is_authenticated %}
<div class="col-lg-4 col-lg-offset-2">
    <a href="/admin/profile">{{ current_user.email }}</a>
</div>
<div class="col-lg-4">
    <a href="{{ url_for('security.logout') }}">Logout</a>
</div>
{% else %}
<div class="col-lg-4 col-lg-offset-4">
    <a href="{{ url_for('security.login') }}">Log in</a>
</div>
{% endif %}
```

Flask-Security also keeps track of the role of the *current_user* and thus this can be used in templates to display certain parts of webpages to users depending on their roles and privileges. For example, only an admin is able to view the Add Project link.

```
{% if current_user.has_role("admin") %}
<div class="vd_info tr"> <a>  Add Project </a> </div>
{% endif %}
```

In views.py, we can restrict users access to a given webpage by using decorators on controller methods that render the webpages:

- @*login_required*: ensures that if the current user is not logged in then they cannot access the web page rendered by the controller's route.

- @*roles_required("admin")*: ensures that if the role of the current user is not admin then they cannot access the web page rendered by the controller's route. Similarly @ *roles_required* can be used for curators and collectors depending on their privileges.

5.3.1.5 Flask-WTF

Forms are an important part of user interfaces in web applications. In this project, forms are used by users to manage content like create and edit projects, items and beacons. WTForms is a python library which provides flexible form rendering and validation. Flask-WTF is a Flask extension that extends the functionality of WTForms into the project.

WTForms in a Flask application employs a similar MVC approach as discussed before for rendering. *Forms.py* is a python file which creates classes for each unique form. Each form class contains class variables which represent form fields. The image below shows an example of a *UserPrefs* class which represents a form that is used by users to edit their personal settings. These settings are represented by the class variables like name and occupation.

```
class UserPrefs(Form):
    """Form for user-changeable things."""
    name = StringField("Name")
    occupation = StringField("Occupation")
    description = StringField("Describe Who You Are")
    city = StringField("Current City")
    birthdate = StringField("Date of Birth")
    image_html = StringField("Enter URL link for your image")
    password = PasswordField("Password")
    password_confirm = PasswordField("Confirm password")
```

In *views.py*, the controllers can, instantiate a form object, just like models, and pass it to the template to be rendered by the view. Another useful aspect of using WTForms is form validation and submission which is handled easily and reduces the amount of boilerplate code a developer needs to write. As shown in the image below initially the form object of the *UserPrefs* class is instantiated. If the form is for editing, like in the image, then the form fields are loaded with preexisting stored values. Therefore when the form is rendered it displays the current values of the variables. When the user submits a form, it is validated and the values of the fields are stored. In the image, it is stored with the *current_user*, however, in other forms it is stored in the objects of the model classes in consideration. Unlike rendering a template, after a form submission the controller method redirects to another controller method in *views.py*. In the image the controller redirects to the URL of the profile controller which in turn renders the *profile.html* template displaying the edited values.

```python
@bp.route("/user/me", methods=["GET", "POST"])
@login_required
def user_settings():
    """Edit user settings.
    """
    form = forms.UserPrefs()
    form.name.data = current_user.name
    form.occupation.data = current_user.occupation
    form.city.data = current_user.city
    form.description.data = current_user.description
    form.image_html.data = current_user.image_html
    form.birthdate.data = current_user.birthdate

    if form.validate_on_submit():
        if (form.name.data != current_user.name):
            current_user.name = form.name.data
        if (form.password != current_user.password) and form.password.data:
            current_user.password = form.password.data
        if (form.occupation.data != current_user.occupation):
            current_user.occupation = form.occupation.data
        if (form.city.data != current_user.city):
            current_user.city = form.city.data
        if (form.description.data != current_user.description):
            current_user.description = form.description.data
        if (form.image_html.data != current_user.image_html):
            current_user.image_html = form.image_html.data
        if (form.birthdate.data != current_user.birthdate):
            current_user.birthdate = form.birthdate.data
        current_user.save()
        flash('Your settings were updated successfully')
        return redirect(url_for("Admin.profile"))

    return render_template("edit_self.html", form=form)
```

In the template, we continue to use the HTML form tag however the fields are represented by the variables of the form object passed to the template.

```
<form method="POST" action="">
    {{ form.hidden_tag() }}
    {{ form.name.label }} {{ form.name (class="form-control") }}<br />
    {{ form.occupation.label }} {{ form.occupation (class="form-control") }}<br />
    {{ form.birthdate.label}} {{form.birthdate (class="form-control")}}<br/>
    {{ form.description.label }} {{ form.description (class="form-control") }}<br />
    {{ form.city.label}} {{form.city (class="form-control")}}<br/>
    {{ form.image_html.label}} {{form.image_html (class="form-control")}}<br/>
    {{ form.password.label }} {{ form.password (class="form-control") }}<br />
    {{ form.password_confirm.label }} {{ form.password_confirm (class="form-control") }}<br />
    <input type="submit" class="btn btn-primary" value="Save Preferences" /><br/>
</form>
```

A key point to note in controller methods is that the route defaults to GET requests meaning that the browser has access to GET information from the server and the controller method thus provides that information. However, in the example of form submission the controller method needs to explicitly contain the methods allowed in the route:

```
@bp.route("/user/me", methods=["GET", "POST"])
```

This permits the browser to POST requests to the server and update the information stored.

5.3.1.6 Summary

The MVC architecture described in this chapter serves to render dynamic templates for all controller methods and HTML files in the project. Appendix C lists all the HTML files along with their respective views and roles within the project. Appendix D illustrates the templates rendered on the web browser showing from top to bottom the sequence of web pages that leads the user from their profile to the dashboard where they can manage the project, beacons and items.

5.3.2 Web API

The application tier also consists of a Web API which uses representational state transfer (REST) to communicate data between different components over the World Wide Web. Currently, REST is the standard for publishing web services and web APIs mostly due

to its simplicity and stateless architecture. Clients can request resources simply by using the resource identifier (URI) without any knowledge of the state of the resource or its origin. RESTful APIs do not require a specific format for the data however the most popular format is JSON. In our project RESTful API publishes information about projects and, beacon and item associations in JSON format. The main advantage of formatting the response in JSON format is that it is lightweight and can be processed by most programming languages and thus our system can be expanded to incorporate other clients such as iOS devices.

Flask, using a similar MVC approach as before, is used to publish RESTful Web APIs as well. The Web API primarily serves to respond to the requests sent by the mobile application on the defined URLs. The controller methods in the API have defined URLs which publish JSON data. The image below shows an example of a controller that publishes all the projects in the system.

```
@bp.route("/projects")
def list_projects():
    """List all projects.
    """
    projects = Project.select().order_by(Project.id)
    plist = []
    for project in projects:
        plist.append({
            'id': project.id,
            'name': project.name,
            'description': project.description,
            'beacons': [x.id for x in project.beacons],
            'approved_items': [x.id for x in project.approved_items]
            })
    return jsonify(projects=plist)
```

The controller uses the *Project* model class to obtain all of the projects in the system and enters the projects along with certain fields into a JSON object.

This JSON object is then published on the server as shown below:

127.0.0.1:5000/api/v0/projects


```
{
  "projects": [
    {
      "approved_items": [
        1,
        2,
        3
      ],
      "beacons": [
        "B9407F30-F5F8-466E-AFF9-25556B57FE6D:17587:13997",
        "B9407F30-F5F8-466E-AFF9-25556B57FE6D:50714:31884",
        "B9407F30-F5F8-466E-AFF9-25556B57FE6D:64917:10860"
      ],
      "description": "This project serves to provide participants with stories of prisoners in NYC",
      "id": 1,
      "name": "Mass Incarceration"
    }
  ]
}
```


Similarly, another controller method which is used to respond to requests from the mobile application is *project_beacon*. It responds to the request by publishing a JSON object containing information about the item associated with the Beacon id sent as a parameter in the URL.

```
@bp.route("/project/<int:project_id>/beacon/<beacon_id>")
def project_beacon(project_id, beacon_id):
    try:
        item = (Item.select()
                .join(BeaconItemMap)
                .where((BeaconItemMap.project == project_id)
                       & (BeaconItemMap.beacon == beacon_id))
                .get()
                )
    except Item.DoesNotExist:
        return abort(404)

    return jsonify(item={
        'id': item.id,
        'created_at': item.created_at,
        'html': item.html
        }
    )
```

The controller method, as shown in the image above, retrieves the associated item by using the *Item*, *BeaconItemMap* and *Beacon* model classes. This example shows how the Peewee ORM simplifies querying across multiple tables. The JSON object containing information about the item is published onto the server as the response.

5.4 Presentation Tier

The mobile application is built for Android devices using Android Studio which is the official integrated development environment (IDE) for Android platform development. The application is deployed and tested on the Google Nexus 7 tablet.

The mobile application serves two main purposes:

1) Detecting and interacting with Beacons using Ranging and Monitoring.

2) Sending HTML requests to access resources available on the server using URLs constructed dynamically within the mobile application.

5.4.1 Beacon Ranging and Monitoring

Estimote has a powerful SDK for Android which provides the necessary modules to interact with Estimote Beacons. From the SDK we need to import BeaconManager which handles

all interactions with Beacons, Region which is used to define a region of Beacons using Beacon IDs, and Beacon which represents a single Estimote Beacon.

```
import com.estimote.sdk.Beacon;
import com.estimote.sdk.BeaconManager;
import com.estimote.sdk.Region;
```

As mentioned before, Beacon monitoring is used to notify the mobile device when it enters or exits a region of Beacons. One of the key features of Beacon Monitoring is that it functions even when the application is running in the background. BeaconManager provides built-in methods that aid in monitoring with beacons as seen in the image below:

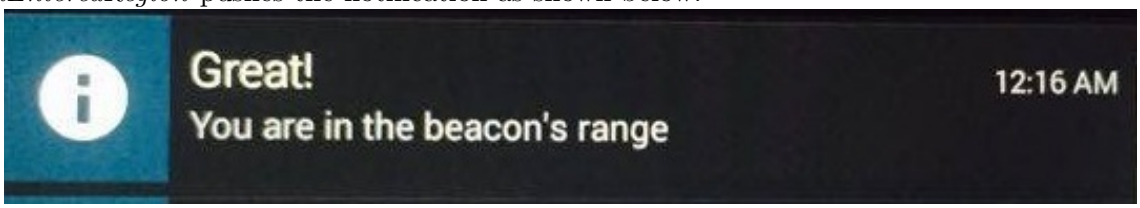
```
public class Monitoring extends Application {
//For Beacon monitoring. Push notifications even when application is not open on screen.
    private BeaconManager beaconManager;
    @Override
    public void onCreate() {
        super.onCreate();
        beaconManager = new BeaconManager(getApplicationContext());
        beaconManager.setMonitoringListener(new BeaconManager.MonitoringListener() {
            @Override
            public void onEnteredRegion(Region region, List<Beacon> list) {
                showNotification("Great! ", "You are in the beacon's range");
            }
            @Override
            public void onExitedRegion(Region region) {
                showNotification("Stop! ", "You are leaving the installation.");
            }
        });
        beaconManager.connect(() -> {
                beaconManager.startMonitoring(new Region("monitored region",
                        UUID.fromString("B9407F30-F5F8-466E-AFF9-25556B57FE6D"), null, null));
        });
    }
```

1. *startMonitoring* and *stopMonitoring* serve to schedule monitoring. *startMonitoring* is always running which allows the application to constantly scan for Beacons in the background. *stopMonitoring* is only used when the battery of the mobile device is in consideration. In this example, the region is defined using the UUID which is the same for all of our purchased Beacons. In real-life installations the region will be defined by those beacons that are present near the entry and the exits.

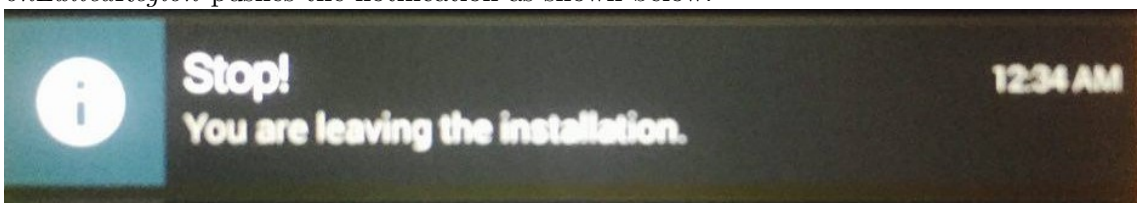
2. *setMonitoringListener* scans for Beacons in the area which are included the predefined region

3. The Listener passes the event to one of the following methods: *onEnteredRegion* or *onExitedRegion* which pushes a notification to the mobile device.

onEnteredRegion pushes the notification as shown below:



onExitedRegion pushes the notification as shown below:



Beacon ranging involves identifying individual Beacons and their proximities to the mobile device. Similar to Monitoring, BeaconManager provides *setRangingListener* which identifies Beacons within a predefined region and returns a list of Beacons ordered by decreasing RSSI values. From that list, the first element is selected since that represents the nearest Beacon to the mobile device. From the *nearestBeacon*, the unique Beacon ID is obtained which is then used to construct a URL to communicate with the server to obtain the associated item.

```
beaconManager = new BeaconManager(this);
beaconManager.setRangingListener((region, list) -> {
        if (!list.isEmpty()) {
            Beacon nearestBeacon = list.get(0);
            beacon_id = nearestBeacon.getProximityUUID() + ":" + nearestBeacon.getMajor() + ":" + nearestBeacon.getMinor();
            beacon_id = beacon_id.toUpperCase();
            URL_ITEM = "http://127.0.0.1:5000/api/v0/project/" + project_id + "/beacon/" + beacon_id;
            sendRequestForItem(URL_ITEM);
        }
});
```

As the participant moves around the installation, the mobile application constantly calculates the nearest beacon and according to that fetches data from the server. Estimote

SDK hides most of the technical details for example calculating RSSI values and scanning Beacons within a region, therefore, providing developers an easy approach to interact with Estimote Beacons.

5.4.2 GET requests to server

With Android devices, the current approach to send HTTP requests over the network is via using Volley. Volley is an HTTP library that integrates easily with RESTful APIs and is compatible with data in JSON format. It primarily functions by creating a RequestQueue and passing Request objects to a given URL. Volley provides built-in support for sending requests and therefore reduces large chunk of boilerplate code and allows developers to write more application-specific code.

In our mobile application, only GET requests are sent to the server. Currently, the system only permits the mobile application to GET information from the server and not POST, DELETE or UPDATE.

With the current implementation of the mobile application, the *ProjectActivity* presents all the projects in the system to the participant to select from. In *sendRequestForProjects* a GET request is sent to the *URL_PROJECTS*. The *URL_PROJECTS* is a fixed variable and does not need to be dynamically altered during the application.

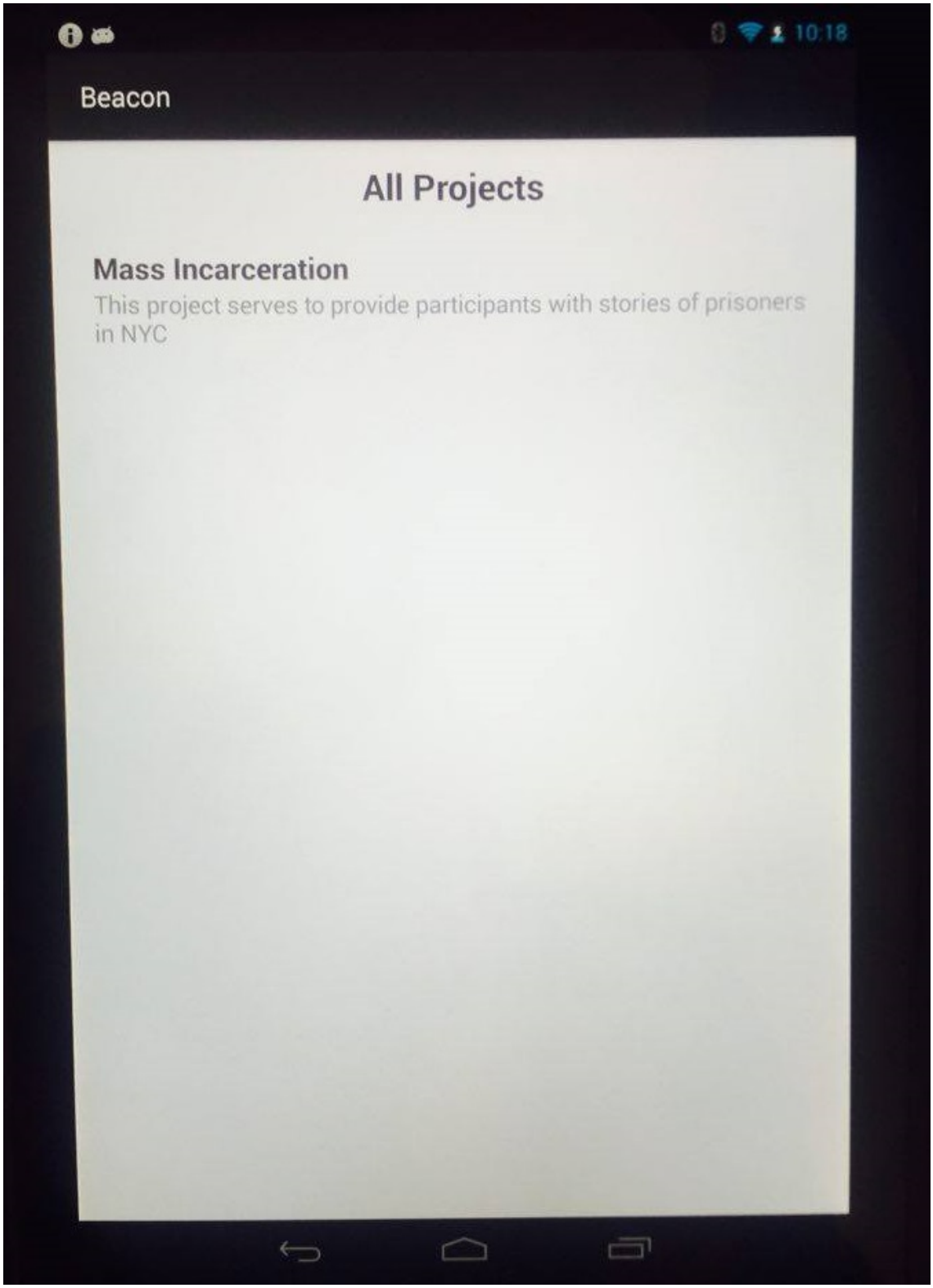
```
public static final String URL_PROJECTS="http://127.0.0.1:5000/api/v0/projects";
```

After the server receives these requests, it matches the given URL with the defined routes of the controllers and executes the *list_projects* controller as shown in section 5.3.2.

On the mobile side, after the request is made if the response is received successfully in JSON format then the *onResponse* method is called, otherwise the *onErrorResponse* is called. In *onResponse* the JSON object is parsed and the information is entered into an Android ListView widget.

```
//sends request to server using URL_PROJECTS expecting a JsonObject. JsonArray held within Json Object.
private void sendRequestForProjects() {
    JsonObjectRequest projectRequest = new JsonObjectRequest(URL_PROJECTS, null,
            new Response.Listener<JSONObject>() {
                @Override
                public void onResponse(JSONObject response) {
                    try {
                        JSONArray projectArray = response.getJSONArray("projects");
                        for (int project = 0; project < projectArray.length(); project++) {
                            project_id = projectArray.getJSONObject(project).getString("id");
                            project_name = projectArray.getJSONObject(project).getString("name");
                            project_description = projectArray.getJSONObject(project).getString("description");

                            projects.add(new Project(project_id,project_name,project_description));
                        }
                        populateListView();

                    } catch (JSONException e) {
                        e.printStackTrace();
                    }
                }
            }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            VolleyLog.e("Error: ", error.getMessage());
        }
    });
    requestQueue = Volley.newRequestQueue(this);
    requestQueue.add(projectRequest);
}
```

The List View presents all the projects to the participant who then selects the project they are interested in as shown in the image below:


10:18
Beacon
All Projects
Mass Incarceration
This project serves to provide participants with stories of prisoners in NYC

After the participant selects a project, an intent is created to the *MainActivity* along with the project id.

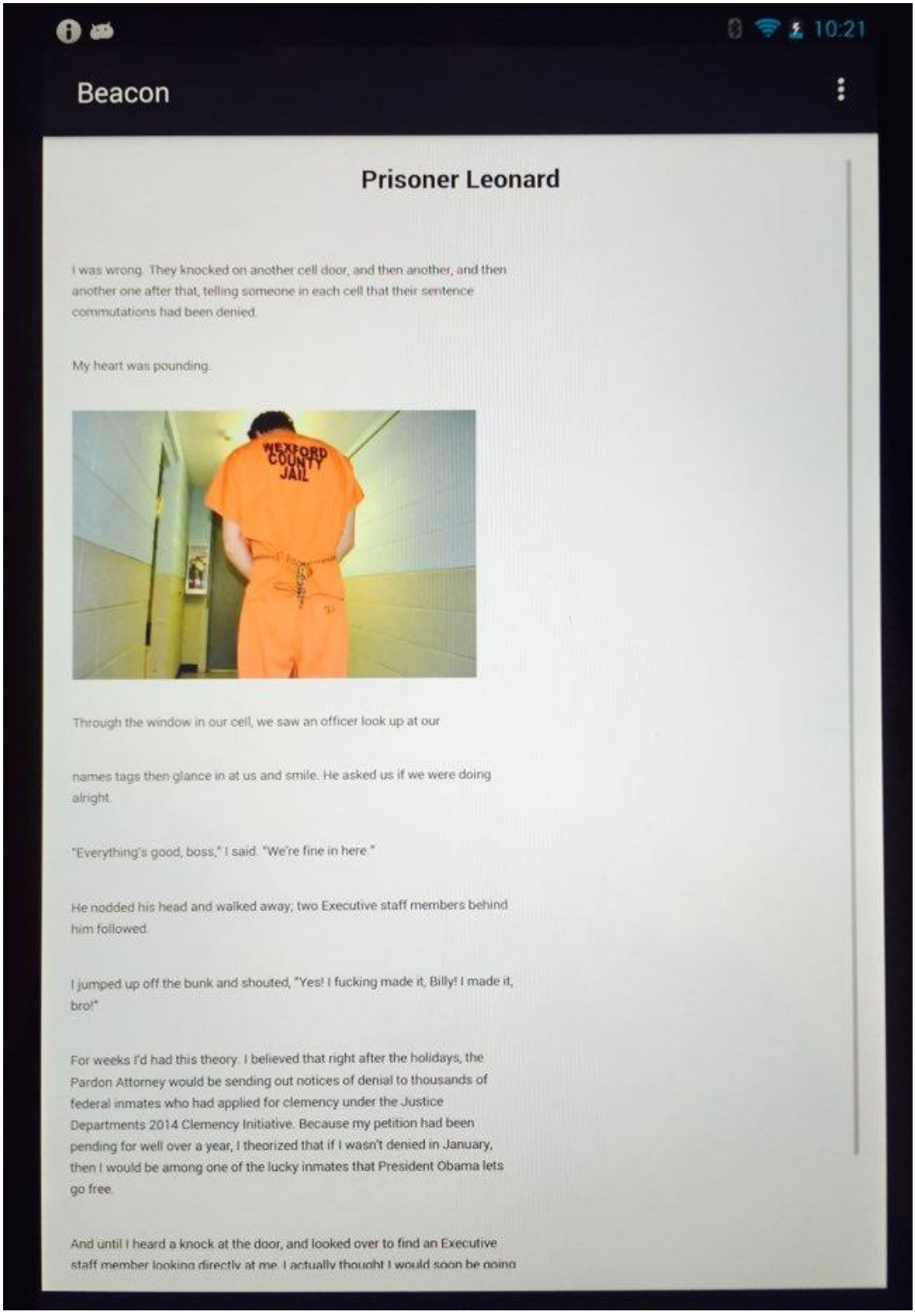
In the *MainActivity*, the application begins Beacon ranging. Beacon ranging constructs a *URL_ITEM* dynamically containing the project id that the user selected and the id of the beacon nearest to the mobile device.

```
URL_ITEM = "http://127.0.0.1:5000/api/v0/project/" + project_id + "/beacon/" + beacon_id;
```

In *sendRequestForItem*, the *URL_ITEM* is used to send a request to the server to obtain the JSON object which contains information about the item associated with the nearest beacon in the current project. The server matches the *URL_ITEM* with the routes of the controller methods and executes the *project_beacon* method shown in section 5.3.2.

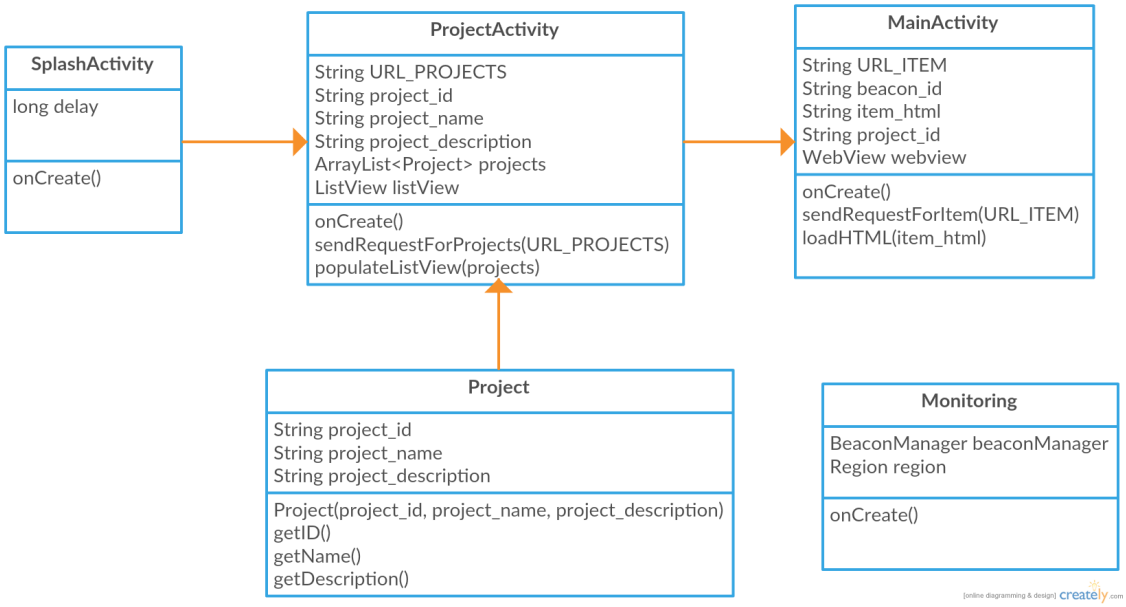
```
private void sendRequestForItem(String URL_ITEM){
    JsonObjectRequest itemRequest = new JsonObjectRequest(URL_ITEM, null,
            new Response.Listener<JSONObject>() {
                @Override
                public void onResponse(JSONObject response) {
                    try {
                        item_html = response.getJSONObject("item").getString("html");
                        loadHTML(item_html);

                    } catch (JSONException e) {
                        e.printStackTrace();
                    }
                }
            }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            VolleyLog.e("Error: ", error.getMessage());
        }
    });
    requestQueue = Volley.newRequestQueue(this);
    requestQueue.add(itemRequest);
}
```

After the JSON object is parsed, the html of the item is extracted and it is displayed by the application using an Android WebView widget as shown in the diagram below:


10:21
Beacon
Prisoner Leonard
I was wrong. They knocked on another cell door, and then another, and then another one after that, telling someone in each cell that their sentence commutations had been denied.
My heart was pounding.
WEXFORD COUNTY JAIL
Through the window in our cell, we saw an officer look up at our
names tags then glance in at us and smile. He asked us if we were doing alright.
"Everything's good, boss," I said. "We're fine in here."
He nodded his head and walked away; two Executive staff members behind him followed.
I jumped up off the bunk and shouted, "Yes! I fucking made it, Billy! I made it, bro!"
For weeks I'd had this theory. I believed that right after the holidays, the Pardon Attorney would be sending out notices of denial to thousands of federal inmates who had applied for clemency under the Justice Departments 2014 Clemency Initiative. Because my petition had been pending for well over a year, I theorized that if I wasn't denied in January, then I would be among one of the lucky inmates that President Obama lets go free.
And until I heard a knock at the door, and looked over to find an Executive staff member looking directly at me, I actually thought I would soon be going

5.4.3 Overview

The class diagram for the mobile application is illustrated below:



Class	Purpose
SplashActivity	Temporary page at the beginning of the application which displays a short overview and purpose of the application
ProjectActivity	Gets projects from server and allows participants to select a project and begin their tour
Project	Project class used to instantiate Project objects in ProjectActivity
MainActivity	Displays associated item of nearest beacon from the project selected by the participant
Monitoring	Runs on the UI thread and pushes notifications when user enters/exits the beacon's range (even when the application runs in the background)

Table 5.4.1. Purpose of each Mobile Application class

6
Results

The project successfully incorporates the system design mentioned earlier. The web application allows management of projects, beacons and items and the data is stored within the database. The mobile application sends requests to the server, receives the required data and displays it.

6.1 Testing

The date for the exhibition on Mass Incarceration is September 2016. For this project, I worked with three beacons and developed the web application locally on my computer. Therefore, the applications were tested locally and not on a larger scale.

The key components that were monitored and tested include:

1. **Database Storage**: It is important that the data entered by the users in the web application is stored correctly in the database. Using the DB Browser for SQLite I was able to verify that the data was indeed being stored in the correct tables and in the correct format.

2. **GET Requests to server**: The Web API publishes services on the local server and therefore the mobile application on the tablet is not able to send requests to the local server bound on my physical computer. I used www.myjson.com which is a free hosting service to publish the JSON responses on the internet. In my mobile application I hard coded the URLs from myjson.com to test whether my mobile application was able to GET data from a server, parse the JSON object and output the correct values.

3. **Beacon ranging**: By viewing the *BeaconItemMap* table in the database I am able to verify that the item displayed by the mobile application was indeed the one associated with the nearest beacon.

6.2 Evaluation

The evaluation requirements of the web application set by the client (EH) are included in the Appendix E. After viewing my progress the client has decided to move forward with the project without my contributions.

There were a couple of aspects of my contribution that the client were unsatisfied with:

- JavaScript in the front end templates
- Absence of Map View, User Admin Page and Item filtering
- Edits of the backend particularly the *views.py* file which interferes with other backend code written by Ryan.
- Less changes in the templates
- Glitches in the project that cause it to crash

As I reflect on the feedback that was given to me by the client, I understand the ways in which I could have worked differently to produce the product that they wanted. Since, I

am the first computer science senior that has worked on implementing a technical project with a client, I will discuss the changes that I would make if I had the opportunity to recreate this project and hopefully these changes can be used in the future with seniors that will complete similar projects.

The first change would be on the communication front of my interaction with the client. There was miscommunication about the project layout and the expectations on both mine and the client's end. Weekly meetings and check-ins with both the clients and my advisor together would have helped decrease these misunderstandings.

I would also ask the client to make the requirements document more clear and explicit, especially with the layout and sequence of pages as this was left open for my interpretation. Ideally, I would have preferred an overview of the project, the setup, the different variables used and detailed write-up of the individual components that I had to implement.

This has been a great learning experience that I am grateful for both academically and in terms of teamwork skills. I have learned that collaboration on software projects is difficult since we are each contributing portions of a code into a larger project and we all have different approaches to coding.

As a student, I have enjoyed learning and understanding the overarching architecture and the different technologies used in the project. I desired to understand the underlying concepts behind the different components in the project and that is why I spent time researching and understanding those concepts.

7
Conclusion

Estimote's Steve Cheney : "The online world that we are used to has tons of available information but the physical world is this place we are trying to navigate at all times, and your phone does not have any context about it. Beacons are one way to solve that." [30]

The Beacon technology is a great improvement on the past solutions that have aimed to provide location as context. Beacons are small, cheap and do not require additional hardware. Most companies that produce Beacons provide a supporting management system which can be used to modify Beacon settings. In section 2.4 we discussed the current implementations of projects using Beacons around the world. The main market for Beacons is surely advertising and marketing. However, its potential has been extended to other fields as well for instance in museums and small businesses. This is indicative of the widespread benefits of Beacons to serve users in providing location-based services. There are definitely drawbacks in using Beacons (loss of signal strength) however the potential it possesses only seems to grow. The technology is constantly being improved upon and integrated with existing technologies to satisfy the need of the consumers to receive information when and where they need it.

7.1 Limitations of the system

During testing I realized that since Beacons broadcast signals every 10ms the mobile application continuously receives the signal and sends a GET request. This causes the content to be loaded continuously. Based on the installation and need for content update, the broadcasting interval needs to be changed in the Estimote Beacon management system. Another concern is the range of Beacons. Beacons general have long ranges which depends on both the defined signal strength and the battery power of the Beacon. These factors can cause Beacons of varying proximities to have similar RSSI values causing inaccuracies in the application. Especially if the Beacons are close to each other the mobile application can fluctuate between several nearest Beacons. Therefore when setting up the installation Beacon ranges, battery levels and broadcasting intervals need to be monitored, tested and accordingly changed in the Estimote Beacon management to provide the best possible experience to the participants.

The mobile application requires the availability of Internet to deliver data to the participants. Therefore, this project requires not just Beacons but also availability of Wi-Fi or cellular data. Also, fetching data from the server continuously can have a significant effect on the battery of the mobile device.

7.2 Future Work

Currently, the web application successfully creates a project and can be used to manage its contents. However, more functionalities need to be added and glitches in the system need to be handled. On the mobile side, the concerns listed above need to be addressed as well.

There is currently research being done on creating location-aware systems that incorporate both Wi-Fi and Beacons. Meraki, a division of Cisco, is working on producing access

points that incorporate Wi-Fi and Beacon technology [31]. These access points can serve as regular Beacons which transmit iBeacon advertising packets and interact with mobile applications. These access points also serve as routers allowing consumers to connect with the internet. Therefore instead of having separate hardware for both technologies Meraki is aiming at an integrated access point that can serve as a router and as a beacon in order to provide faster and more efficient location-based services.

Appendices

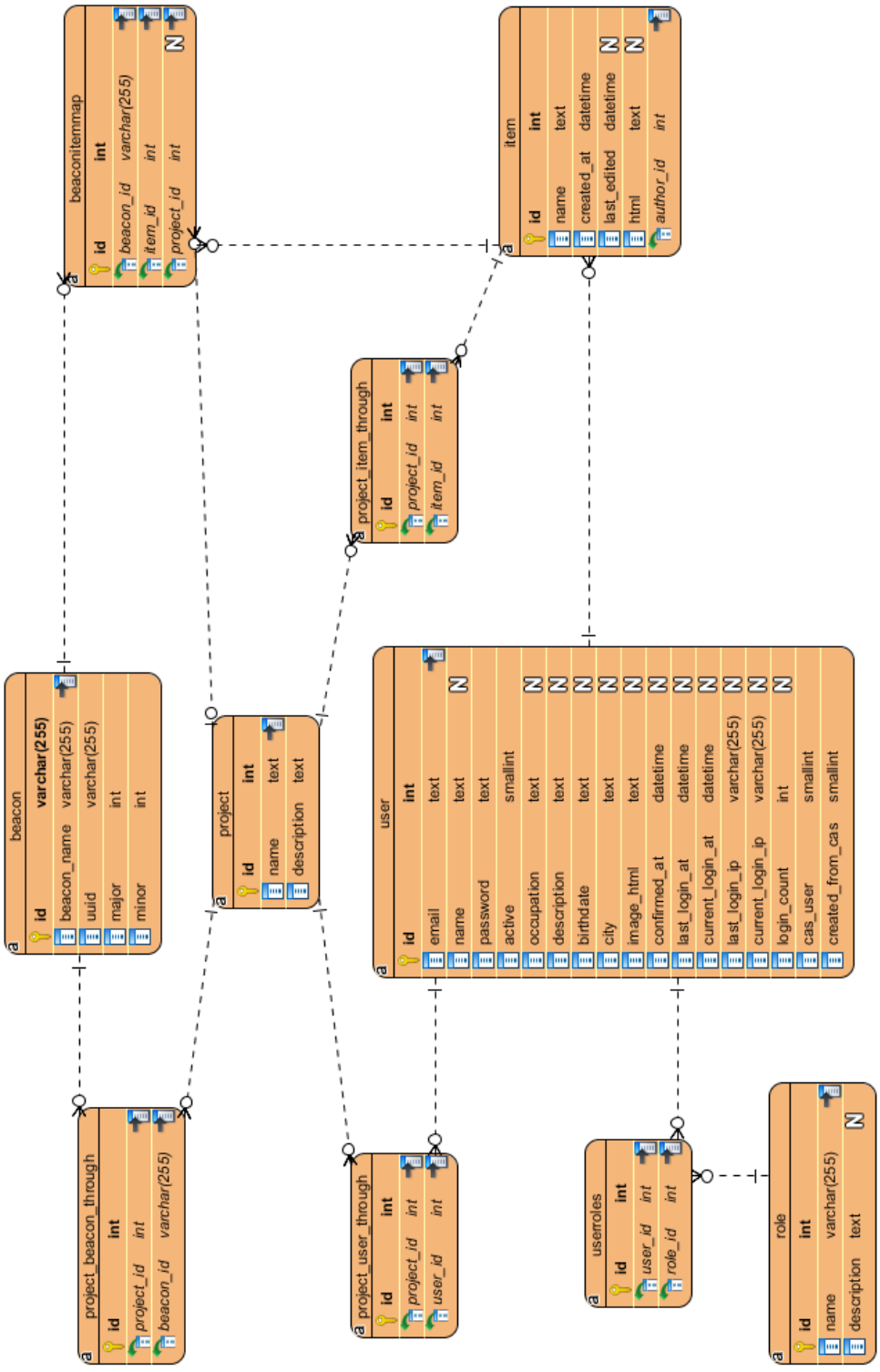
Appendix A

Comparison of Location Systems

	A	B	C	D	E	F	G
1	Technology	Method	Positioning	Smartphone Power Usage	Cost	Precision	Limitations
2	GPS	Satellites	Outdoors	High	Low	approximately 5 metres	Can not be used indoors
3	Active Badge	Infrared	Indoors	N/A	High	Several feet	Obstructed by sunlight
4	Active Bat	Ultrasound	Indoors	N/A	High	Few centimetres	Requires large external infrastructure and maintenance
5	Radar	WLAN	Indoors	High	Low	Few metres	Only used indoors with weak precision
6	BLE	Bluetooth	Indoors and Outdoors	Low	Low	Few centimetres	May require internet depending on the application

Appendix B

Database Schema


beaconitemmap
id int
beacon_id varchar(255)
item_id int
project_id int
item
id int
name text
created_at datetime
last_edited datetime
html text
author_id int
project_item_through
id int
project_id int
item_id int
beacon
id varchar(255)
beacon_name varchar(255)
uuid varchar(255)
major int
minor int
project
id int
name text
description text
user
id int
email text
name text
password text
active smallint
occupation text
description text
birthdate text
city text
image_html text
confirmed_at datetime
last_login_at datetime
current_login_at datetime
last_login_ip varchar(255)
current_login_ip varchar(255)
login_count int
cas_user smallint
created_from_cas smallint
project_beacon_through
id int
project_id int
beacon_id varchar(255)
project_user_through
id int
project_id int
user_id int
userroles
id int
user_id int
role_id int
role
id int
name varchar(255)
description text

Appendix C

Table showing all the HTML files and respective controller methods in views.py

	Templates	views.py	Role
Beacon Management	view_beacon.html	view_beacon(project_id, beacon_id)	Rendered page displays settings of a specific beacon
	view_all_beacons.html	view_all_beacons(project_id)	Rendered page displays all the beacons currently part of the project
	view_available_beacons.html	view_available_beacons(project_id)	Rendered page displays all the beacons that are not part of the project but are available to use
Item Management	view_item.html	view_item(project_id, item_id)	Rendered page displays the item
	view_all_items.html	view_all_items(project_id)	Rendered page displays a list of all items currently part of the project
	add_item.html	add_new_item(project_id)	Rendered page displays a form that allows user to add an item by using a HTML editor
	edit_item.html	edit_item(project_id,item_id)	Rendered page displays a form that shows saved item HTML and allows users to change it
Project Management	add_project.html	add_project()	Rendered page displays a form that allows user to create a project, select beacons, select already exisiting items from other projects and create associations between the beacons and items
	edit_project.html	edit_project(project_id)	Rendered page displays a form that allows user to edit an existing project
	view_project.html	project_overview(project_id)	Rendered page displays the fields of the project

Appendix D

Images of Web application rendered on browser

BEACON PROJECT AN5666@BARD.EDU LOGOUT

ABDULLAH NASIM

STUDENT

I am a Computer Science Senior at Bard College

Last Visited May 4, 2016

Profile Projects

Edit

Name:	Abdullah Nasim	Email:	an5666@bard.edu
City:	Annandale on Hudson	Birthday:	10/22/1993

BEACON PROJECT AN5666@BARD.EDU LOGOUT

ABDULLAH NASIM

STUDENT

I am a Computer Science Senior at Bard College

Last Visited May 4, 2016

Profile Projects

Add Project

#	Name	Description	Actions
1	Mass Incarceration	This project serves to provide participants with stories of prisoners in NYC	

BEACON PROJECT

Abdullah Nasim

Project Information

View Project

Edit Project

Beacon Management

Item Management

MASS INCARCERATION

This project serves to provide participants with stories of prisoners in NYC

BEACONS

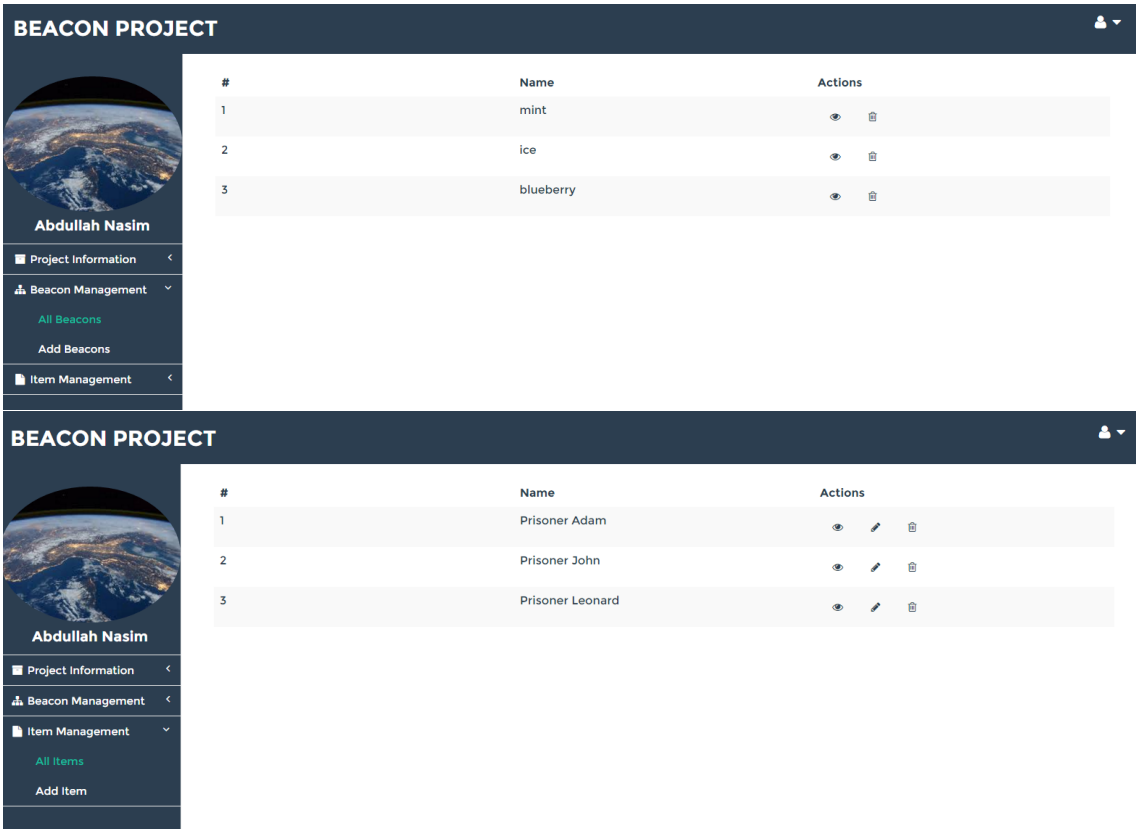
- mint
- ice
- blueberry

ITEMS

- Prisoner Adam
- Prisoner John
- Prisoner Leonard

BEACON-TO-ITEM ASSOCIATIONS

Beacon	Item
mint	Prisoner Adam
ice	Prisoner John
blueberry	Prisoner Leonard


BEACON PROJECT
Abdullah Nasim
Project Information
Beacon Management
All Beacons
Add Beacons
Item Management
Name Actions
1 mint
2 ice
3 blueberry
BEACON PROJECT
Abdullah Nasim
Project Information
Beacon Management
Item Management
All Items
Add Item
Name Actions
1 Prisoner Adam
2 Prisoner John
3 Prisoner Leonard

Appendix E

Web Application Requirements Document

Objective criteria

- **Responsive** -- it must scale smoothly to screens of varying size, from a big-screen TV to a smartphone.
- **Enhances progressively** -- the design should be functional on approximately 95% of browsers. Key, basic, functionality should not rely on JS (or should have a non-JS fallback that kicks in automatically).
- **Low-latency** -- to the extent technically possible no aspect of the frontend should introduce an undue lag on the user's experience.
- **Accessible** -- to the extent practical, usable by people not using a traditional web browser/keyboard/mouse combo. (ARIA compliant)
- **Integrated with the rest of the project** -- both code and (open-source) licensing
- **Semantic markup** -- where applicable, takes advantage of more modern, semantic tags like <author>, <article>, etc. as well as metadata
- **Valid** -- the resulting markup pushed to a browser should pass a modern HTML validator check (https://validator.w3.org/ and/or https://validator.nu) with few-to-no warnings and no errors.

Subjective criteria

- **Not ugly** -- should look nice.
- **Reasonably interactive** -- the user should get feedback for their actions -- but not too much.
    - Small animation on a vital notification to grab a user's attention: good.
    - Having a lightbox pop-over notification that swings to-and-fro to let the user know that they've added an item to the database: bad.
- **Lightweight** -- should not download 15Mb from some CDN just so you can get a particular fade or font.
- **Intuitive UX** -- in an ideal world, this project will be used by people of diverse levels of technical understanding. The way to do basic tasks, as well as what task the user should do next, should be obvious.

Definitions

- **Beacon**: estimote beacon
- **installation:** a set of beacons
- **Project**: a particular exhibit associated with an installation of beacons, e.g. gentrification, mass incarceration, sexual assault, hate crimes. Basically just an association between a (sub)set of beacons, approved items, and users authorized to add items to the project.
- **Item**: a story, image, video, or other document.

Required parts:

- User management:
  - Login page
  - User administration overview
  - Single user administration page
  - Personal settings page
- Beacon management:
  - Beacon overview
  - Settings page for a single beacon
  - Map view
- Item management
  - Upload new item page
  - Edit item page
  - Item browser (w/ filtering: view all items associated with a project, view new items...)
- Project management
  - New project page
  - Edit project page
    - Edit associations between beacons and items
  - Project admin overview

Timeline:

1. **Feb. 12: First draft of design mockup due.**
   - We'll give Abdullah feedback.
2. **Feb. 19: 2nd draft**
   - We'll either approve the design or give more feedback
3. **March 1: If deemed necessary, third draft of mockup due. If not, progress check-in**
4. **March 18: First working prototype for design due.**
   - This should implement the majority of the required parts. There can be rough edges and parts that are in-progress, but at this point the contours of the design and most major functionality should be set.
   - We'll test it out and give Abdullah feedback
5. **April 1: Second prototype due**
   - Every part should be implemented. The rough edges should be sanded, if not painted and clear-coated. Everything after this should be polish.
6. **April 14: Progress check-in**
7. **April 29: Final design due.**

Tech specs

- The webapp is written in Python. We're using Flask as the framework for the webapp. The template language used is Jinja.
- I'm happy to make changes to the backend to accommodate Abdullah's needs if necessary.
- After completion code will be released under an open source license

Bibliography

[1] Sarah Perez, *Majority Of Digital Media Consumption Now Takes Place In Mobile Apps*, http://techcrunch.com/2014/08/21/majority-of-digital-media-consumption-now-takes-place-in-mobile-apps/.

[2] Dave Chaffey, *Mobile Marketing Statistics compilation*, http://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/.

[3] Kathryn Zickuhr, *Location-Based Services*, http://www.pewinternet.org/2013/09/12/location-based-services/.

[4] Gartner IT, *Context-Aware Computing*, http://www.gartner.com/it-glossary/context-aware-computing-2/.

[5] Wikipedia, *Beacon*, https://en.wikipedia.org/wiki/Beacon.

[6] Anind Dey and Gregory Abowd, *Towards a Better Understanding of Context and Context-Awareness*, ftp://ftp.cc.gatech.edu/pub/gvu/tr/1999/99-22.pdf.

[7] Margaret Rouse, *GNSS (Global Navigation Satellite System)*, http://searchnetworking.techtarget.com/definition/GNSS.

[8] Wikipedia, *Satellite Navigation*, https://en.wikipedia.org/wiki/Satellite_navigation.

[9] Devika Girish, *iBeacon vs NFC vs GPS: Which Indoor Location Technology will your Business Benefit from?*, http://blog.beaconstac.com/2015/07/ibeacon-vs-nfc-vs-gps-which-indoor-location-technology-will-your-business-benefit-f

[10] Roy Want, Andy Hopper, Veronica Falcao, and Jonathan Gibbons, *The Active Badge Location System*, http://alumni.media.mit.edu/~dmerrill/badge/Want92_ActiveBadge.pdf.

[11] AT&T Laboratories Cambridge, *The Bat Ultrasonic Location System*, http://www.cl.cam.ac.uk/research/dtg/attarchive/bat/.

[12] Suzanne Ross, *Wherever You Go, There Is Connectivity*, http://research.microsoft.com/en-us/projects/radar/.

[13] Wikipedia, *Bluetooth*, https://en.wikipedia.org/wiki/Bluetooth/Origin.

[14] Jimbo, *Bluetooth Basics*, https://learn.sparkfun.com/tutorials/bluetooth-basics.

[15] Greg Sterling, Jules Polonetsky, and Stephany Fan, *Understanding Beacons: A Guide to Beacon Technologies*, https://fpf.org/wp-content/uploads/Guide_To_Beacons_Final.pdf.

[16] Apple Inc., *Getting Started with iBeacon*, https://developer.apple.com/ibeacon/Getting-Started-with-iBeacon.pdf.

[17] Cooper Smith, *THE BEACONS REPORT: Growth Forecasts For The Most Important Retail Technology Since The Mobile Credit Card Reader*, http://www.businessinsider.com/beacons-are-the-most-important-new-retail-tech-2014-7.

[18] Trip Reddy, *15 Companies From Airports to Retail Already Using Beacon Technology*, https://www.umbel.com/blog/mobile/15-companies-using-beacon-technology/.

[19] Chantal Tode, *Schlafly Beer brews beacon marketing program using beer taps*, http://www.mobilemarketer.com/cms/news/software-technology/22557.html.

[20] David Nield, *iBeacon technology powers a new Smart Public Transport project in Bucharest*, http://www.gizmag.com/ibeacon-smart-public-transport-bucharest/37901/.

[21] Serena Chu, *Apples iBeacon Tech Brings Art Gallery To Life*, http://www.psfk.com/2014/01/apple-ibeacon-rubens-house.html.

[22] Galen Gruman, *Beacons are harder to deploy than you think*, http://www.infoworld.com/article/2983166/bluetooth/beacons-are-harder-to-deploy-than-you-think.html.

[23] Craig Gilchrist, *iBeacon security: understanding the risks*, http://blog.estimote.com/post/104765561910/ibeacon-security-understanding-the-risks.

[24] Ron Amadeo, *Meet Googles Eddystonea flexible, open source iBeacon fighter*, http://arstechnica.com/gadgets/2015/07/meet-googles-eddystone-a-flexible-open-source-ibeacon-fighter/.

[25] Wikipedia, *Client-server model*, https://en.wikipedia.org/wiki/Client-server_model.

[26] Jakob Jenkov, *Client-Server Architecture*, http://tutorials.jenkov.com/software-architecture/client-server-architecture.html.

[27] Tutorials Point, *Flask Tutorial*, http://www.tutorialspoint.com/flask/.

[28] Alex Coleman, *Model-View-Controller (MVC) Explained – With Legos*, https://realpython.com/blog/python/the-model-view-controller-mvc-paradigm-summarized-with-legos/.

[29] Matthew S.Gast, *Building Applications with iBeacon: Proximity and Location Services with Bluetooth Low Energy*, O'Reilly Media.

[30] Noah Elkin, *Beacons to Shine a Light on Consumers Murky Path to Purchase*, `http://streetfightmag.com/2015/11/24/beacons-to-shine-a-light-on-consumers-murky-path-to-purchase/`.

[31] Shubbi Mittal, *Wi-Fi meets iBeacon Technology*, `http://blog.beaconstac.com/2015/12/wi-fi-meets-ibeacon-technology/`.