

## Criterion B: Design

### **Design Methodology**

#### **Bottom-Up Approach**

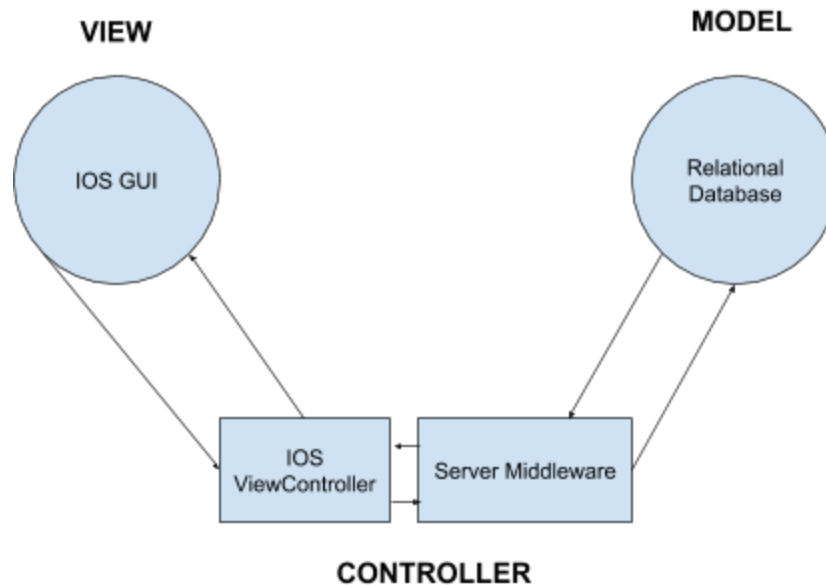
This application will be designed and developed using a bottom-up approach. With this approach, low-level systems are developed first and then pieced together; resulting in a complex system. I chose this methodology because I know the distinct elements of the system: the IOS end, the middleware and server, and the database; however, I still must link these elements together to form the final complex system. I will follow this methodology by first designing, and then developing the small, simpler pieces of software. Once these distinct elements have been determined to work alone, the pieces can then be put together to create the more complex system.

#### **Prototyping**

I will also make use of prototyping in my design methodology. I will use the prototypes to get feedback from my client before I finalize the project. I will be able to add features or change the design. This will result in a product that fits the client's needs better than without the prototyping.

### Design Pattern

The pattern that I have decided to use is the Model View Controller (MVC) design pattern. This serves to separate the software into distinct elements, which prevents coupling between the data and promotes reusability.



MVC design pattern diagram for this software. The view is the IOS GUI, the model is the relational database, and the controller is both the IOS ViewController and the server middleware. While there is also a MacOS, end the IOS end is the only set of views and controllers pictured.

<i>Model</i>
The Model of the software is the relational database used to store information about students and the student's location.
1. Relational Database

<i>View</i>
The View of the software is the IOS front-end GUI. It is the part of the software that directly interfaces with the user, and is distinct from the model.
1. IOS GUI for Students and Administrators 2. MacOS GUI for Administrators

<i>Controller</i>
The Controller of the software has two parts: the server middleware and the view controller in the IOS

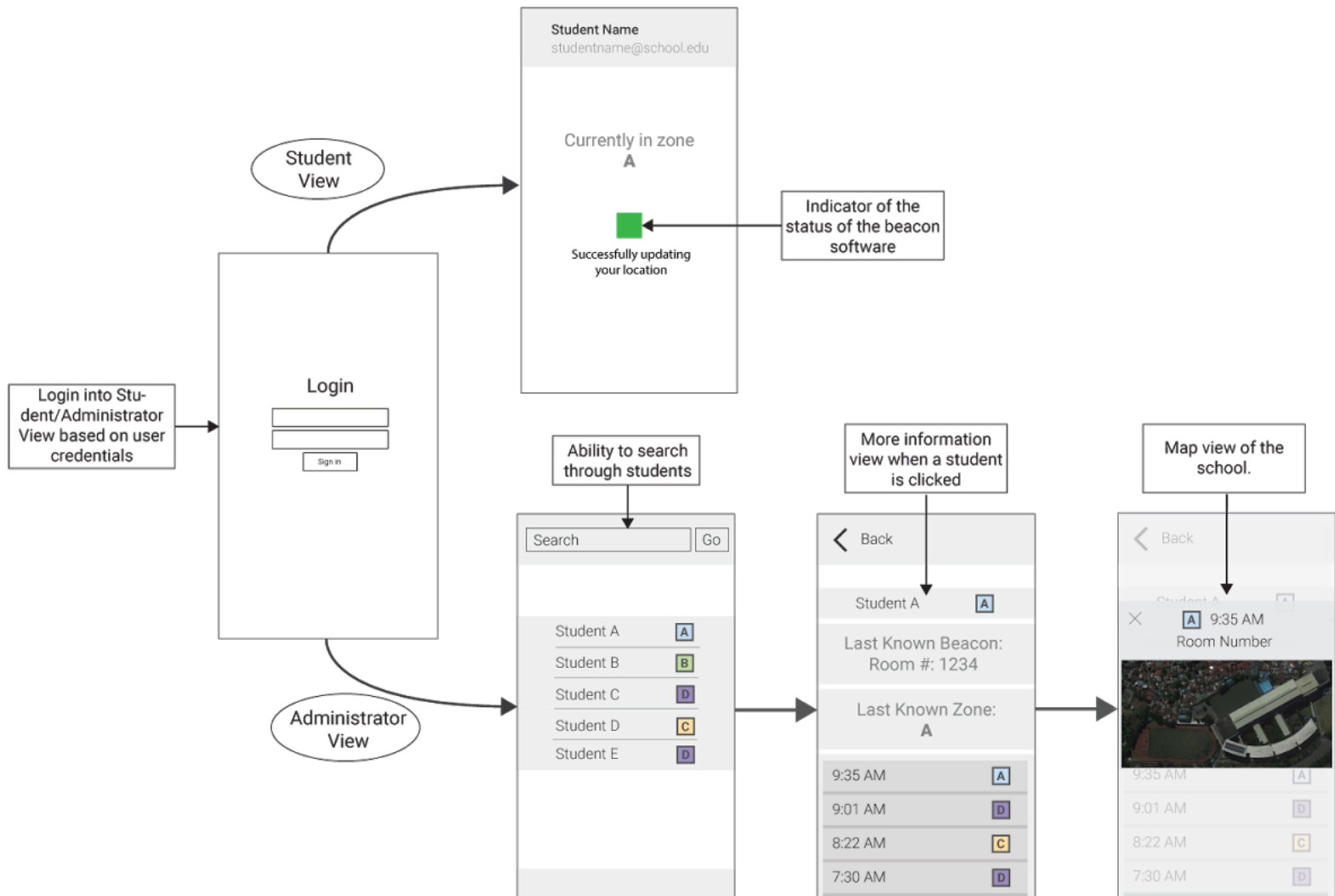
application. It serves as the line of communication between the model and the view.

1. ViewControllers in the IOS application
2. ViewControllers in the MacOS application
3. Middleware to interface between the database and the GUI: between the model and the view.

## Graphical User Interface Prototype

The graphical user interface (GUI) prototype intends to give a preliminary visual representation of the view of the computer software, before the software has been developed. It is structured by placing the opening view furthest to the left, and the sequential views to the right. The Student View is placed on top and the Administrator view is placed on the bottom. Short descriptions are also provided as text boxes to give brief elements of relevant information.

### 1. IOS Application



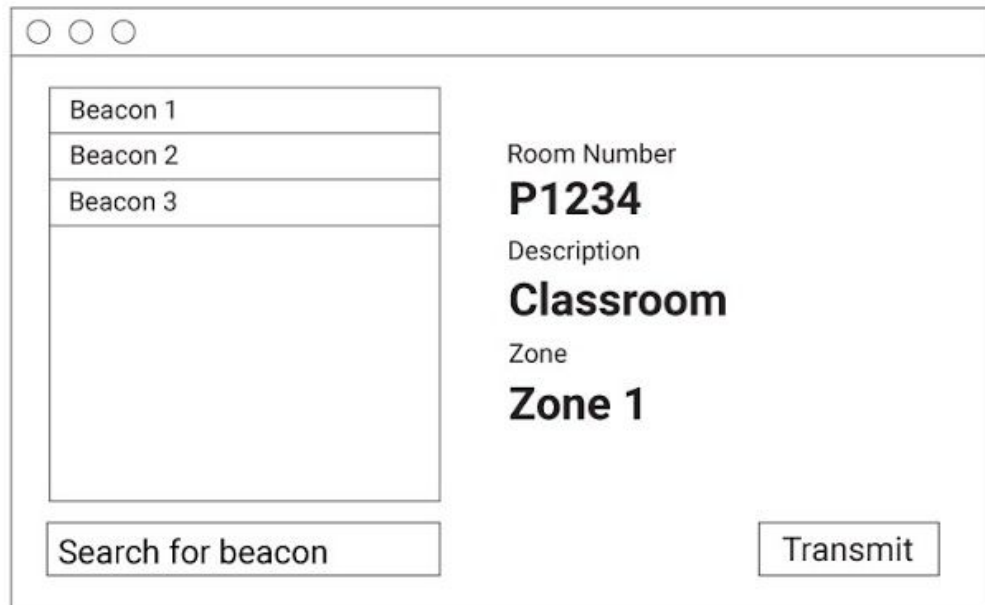
*Client Response to Prototype to initial prototype (refer to Appendix\_B\_Design Fig. 1; resulting in the new design above)*

I consulted with Mr. Dickinson about the prototype GUI, and he requested a map view of the school with the zones highlighted. This would allow administrators checking the location of students to know where the zone is, if they are unfamiliar with the system. This resulted in a new success criterion:

9	Administrator view (IOS): View the student's location by highlighting their location on a map of the school.
---	--

*Refer to a summary of the prototype discussion in 'Appendix\_A\_Consultations' and a recording of the consultation in the 'Consulation\_Evidence' folder.*

## 2. MacOS Application



A mockup of a MacOS application window. The window has a title bar with three standard macOS window control buttons (red, yellow, green) on the top left. The main content area is divided into two columns. The left column contains a list of three items: 'Beacon 1', 'Beacon 2', and 'Beacon 3'. Below this list is a large, empty rectangular box. At the bottom of the left column is a button labeled 'Search for beacon'. The right column displays information for a selected beacon. It starts with the label 'Room Number' followed by the value 'P1234' in a large, bold font. Below that is the label 'Description' followed by the value 'Classroom' in a large, bold font. At the bottom of the right column is the label 'Zone' followed by the value 'Zone 1' in a large, bold font. At the bottom right of the window is a button labeled 'Transmit'.

Beacon 1	Room Number <b>P1234</b>
Beacon 2	Description <b>Classroom</b>
Beacon 3	Zone <b>Zone 1</b>

Search for beacon

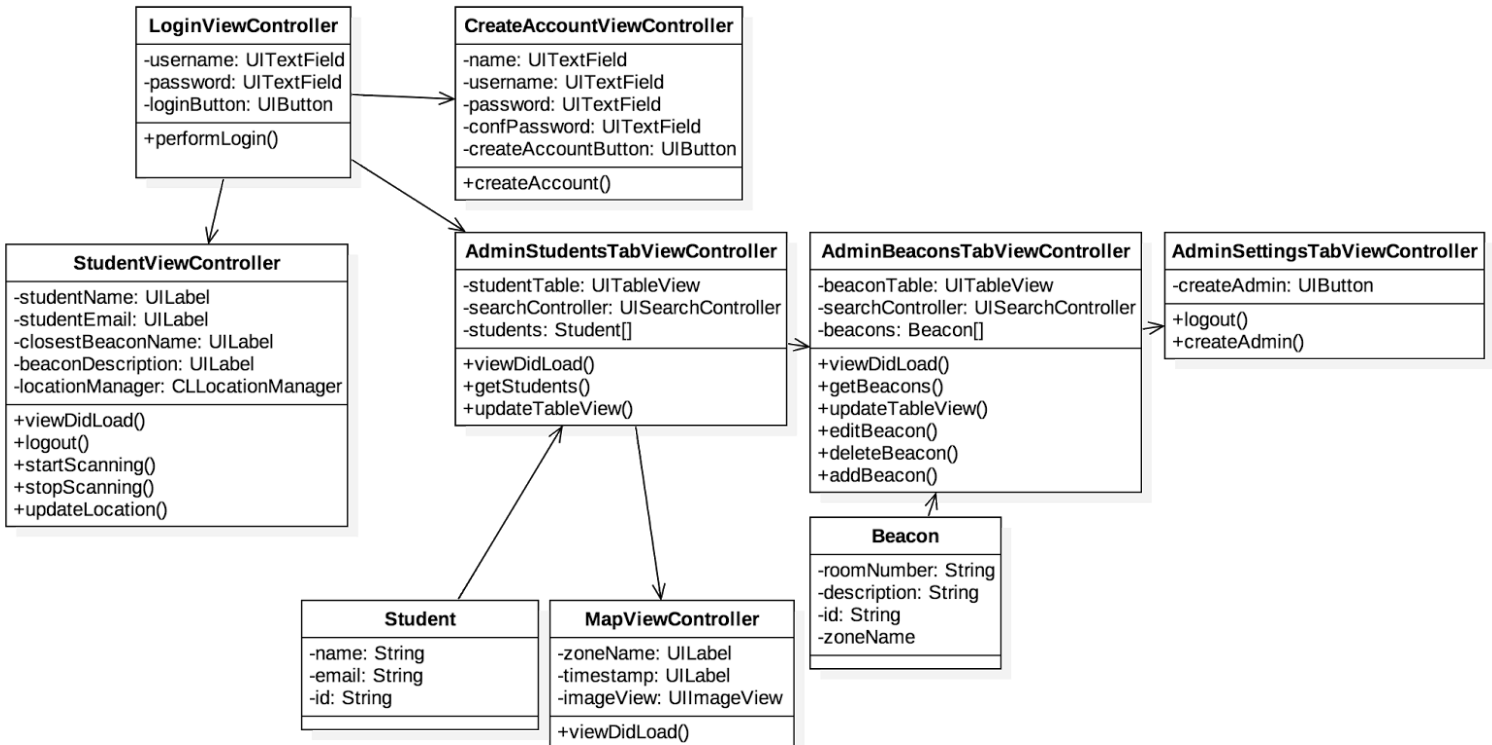
Transmit

## Elements of the System

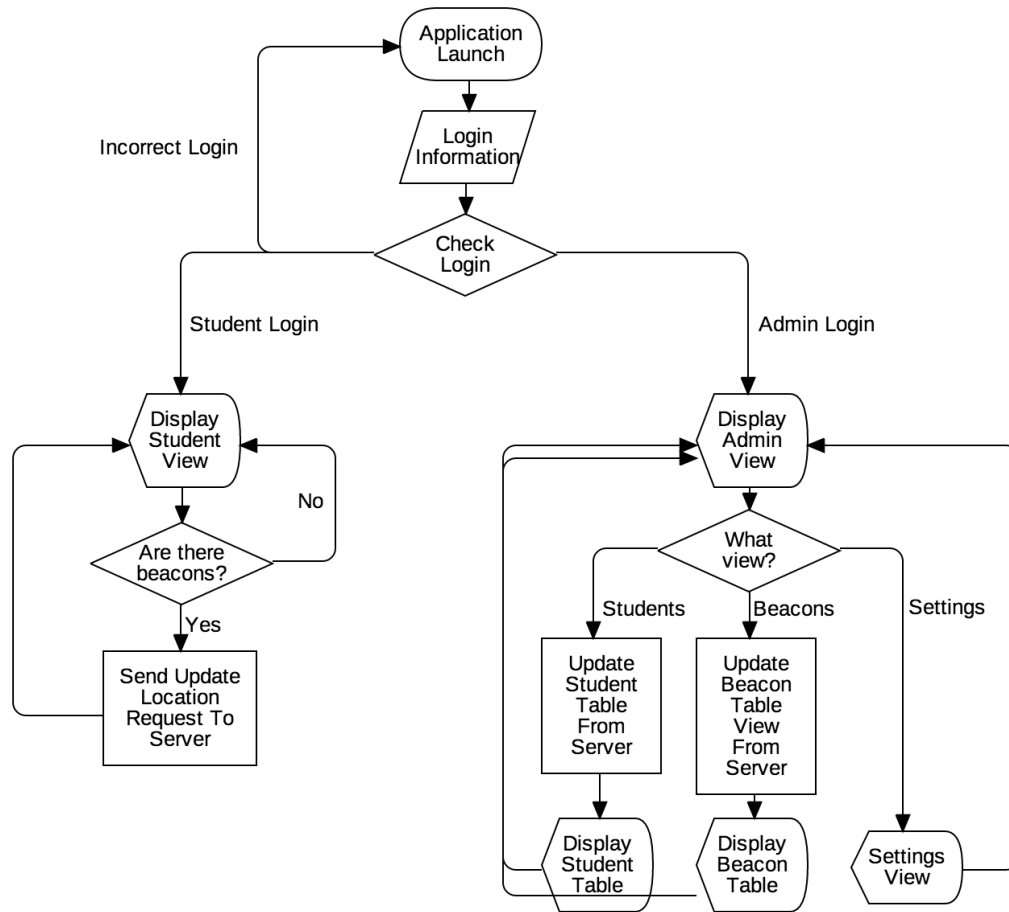
### 1. IOS Application

The Apple IOS end is the part of the application that is run on the phone of the student and the administrator. The application will be written in Swift, Apple's proprietary programming language. All storage here is done by instance variables.

#### UML Diagrams



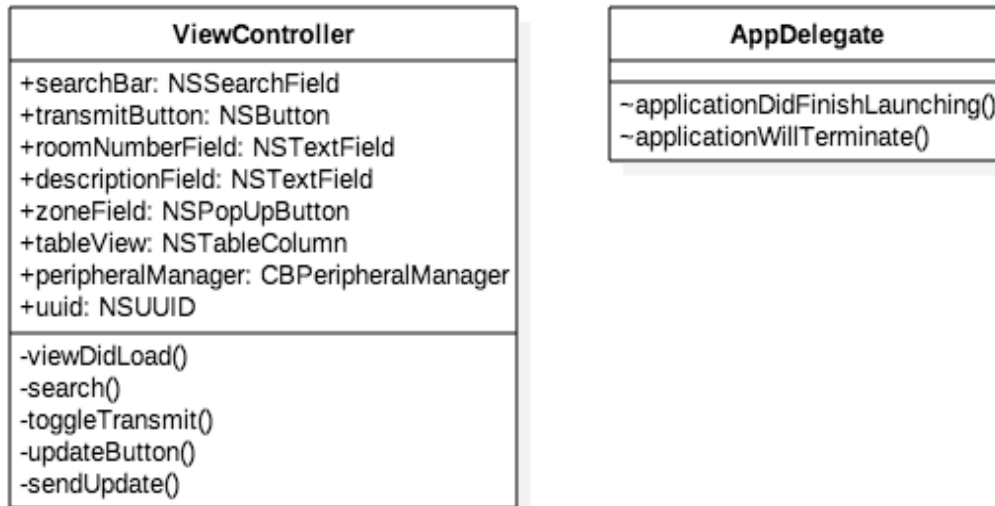
## Flowchart



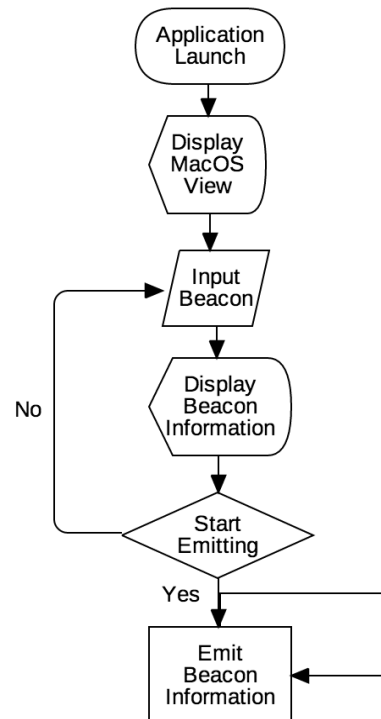
## 2. MacOS Application

The MacOS beacon server is the part of the application that will host the bluetooth 'beacons'. It will accomplish this by continually transmitting a packet of data, which, when received by a beacon client, will show the client which beacon they have connected to. All storage here is done by instance variables.

### UML Diagrams



### Flowchart

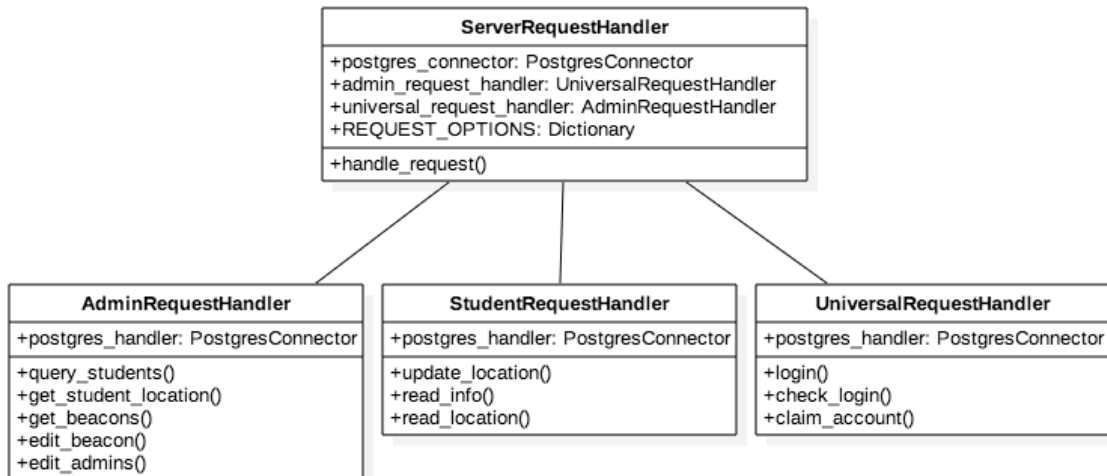




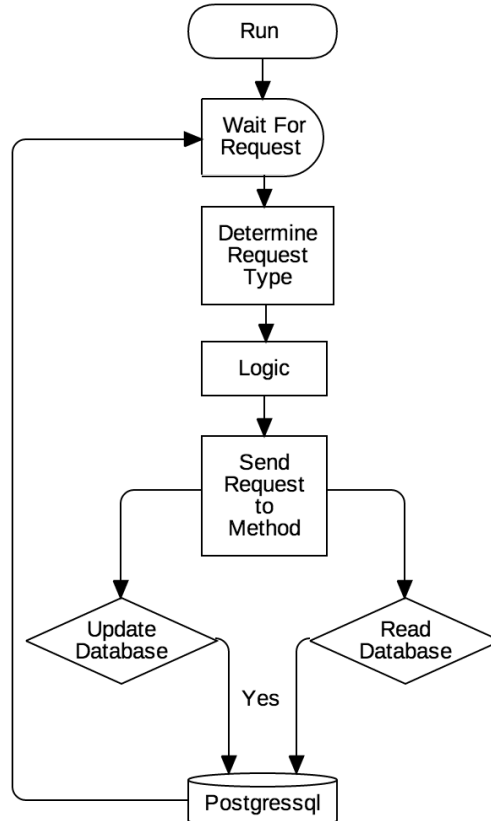
### 3. Python Middleware

The middleware for the bottle.py web server is the part of the application that will handle all incoming requests from the IOS end, deserialize them, and use the information to update the database.

#### UML Diagrams



#### Flowchart

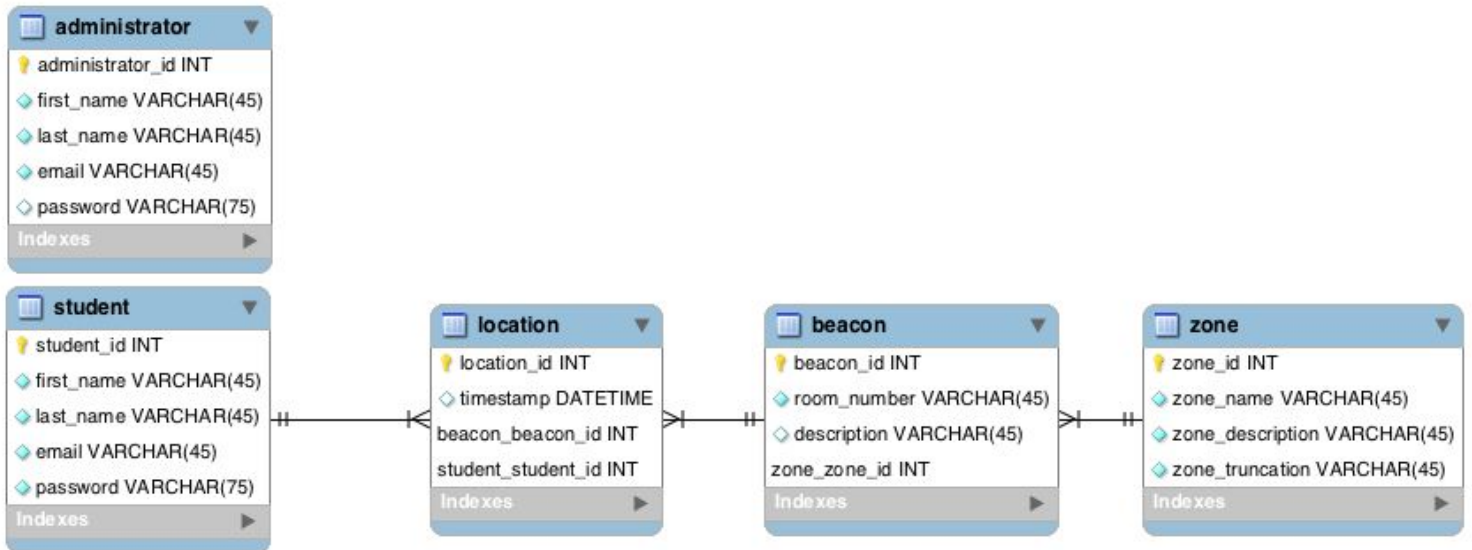


## Key Data Structures

### 1. Postgresql Relational Database

The relational database will be the primary persistent data structure in this software. This section serves to demonstrate the structure of the SQL database to be used in the program. The section outlines the necessary tables, fields, and connections in the database.

#### *Relational Database Diagrams*



#### Administrator

The administrator table serves to store administrators to the beacon attendance system. The table contains columns for the login information of the administrators: their email and hashed password.

#### Student

The student table, similarly stores the login information of the students, as well as the important primary key student\_id. This is the key to be used by the location table to find the current locations of students.

#### Location

The location table is the table to be searched when finding the location of a student. It has both the beacon\_id and student\_id as foreign keys, with a many location to one beacon and one student relationship.

#### Beacon

The beacon table serves to store the location and zone of each beacon in the school; hence, zone\_id is a foreign key. There is a many beacon to one zone (many to one) relationship.

#### Zone

The zone table stores each zone in the school where beacons are placed (e.g. Library, Cafeteria, etc.). Each beacon is assigned one of these.

## 2. Configuration File

The configuration file will be the secondary persistent data structure in this software. It will be formatted in JSON with the 'key: value' structure in a .JSON file. This exists to allow extensibility by preventing the hard coding of database server data.

### *File Format*

Keys stored
username password database port host

### **Testing Plan**

The testing plan section outlines the methods that will be used to test the success criterion decided in criterion A. The format will list the methodology for testing then the expected result of the testing.

<i>Criterion Number</i>	<i>Methodology For Testing</i>	<i>Expected Result</i>
1	Enter login details and tap sign in.	The user will be logged into their respective view: admins to the admin view, and students to the student view.
2	Moving between beacons.	The student's view will update with the new room number of the new beacon.
3	Look at the timestamps within the administrator view.	The time stamps will have a difference of five minutes between updates. Alternatively, the database will show a similar difference of five minutes.
4	Attempt to search through student table.	As the administrator searches with a string, students matching the search string will appear.
5	Tap on a student in the student table.	A detail view of the student should appear with a table of their recent locations.
6	Attempt to search through beacon table.	As the administrator searches with a string, beacons matching the search string will appear.
7	Attempt to use the add, edit, and delete buttons.	Buttons should perform their labeled function of adding, editing, and deleting beacons.
8	Select a beacon from the table and choose to begin emitting.	The computer the application is on should now be emitting a beacon with the major and minor key of that beacon.
9	Tap on location in a student's location table.	The detail location view should appear as a modal. The modal should display a map with the location of the zone highlighted.

### **Extensibility**

I will ensure the extensibility of my program by:

1. Using input validation to prevent errors if other programmers change data types.
2. Preventing coupling between distinct elements by encapsulating data. This means that each element of the system will be adjustable by other programmers without dangers of breaking the software. With the languages I am using, the private keyword does not exist; therefore, I should encapsulate simply by removing coupling between classes.
3. Using comments to make the code easier to read and therefore adjust in the future.
4. Allowing students and administrators to be easily added in the case that more students or administrators join the school.

***A further, more detailed description of the extensibility considerations for the software are listed in 'Appendix\_B\_Design' Fig. 2.***