

Software Engineering - Spring 2019

Report 2

GlassHome

Group 12

Harshil Parekh, Adarsh Gogineni, Shivum Mehta, Shaan Parikh, Andy Guo,

Avi Patel, Nathan Silva, Parth Patel, Kyle Abed

March 17, 2019

sites.google.com/scarletmail.rutgers.edu/softwareengineeringspring2019/

14:332:351 Software Engineering Report 1
Department of Electrical and Computer Engineering
*The State University of New Jersey,
Rutgers*

Contribution Breakdown of Report 2

| Responsibility | Harshil Parekh | Adarsh Gogineni | Shivum Mehta | Shaan Parikh | Andy Guo | Avi Patel | Nathan Silva | Parth Patel | Kyle Abed |
|---|----------------|-----------------|--------------|--------------|----------|-----------|--------------|-------------|-----------|
| Project Management | 26.11% | 17.78% | 15% | 26.11 % | | 8.33% | | | 6.67% |
| Section 1: Interaction Diagrams | 11.11% | 11.11% | 11.11% | 11.11 % | 11.11% | 11.11% | 11.11% | 11.11% | 11.11% |
| Section 2a: Class Diagram | 100 | | | | | | | | |
| Section 2b: Data Types and Operation Signatures | | | 100 | | | | | | |
| Section 2c: Traceability Matrix | | | | | | | | | 100 |
| Section 3a: System Architecture and System Design | | | | 100 | | | | | |
| Section 3b: | | | | 100 | | | | | |
| Section 3c: Mapping Subsystems to the Hardware | | | | | 100 | | | | |
| Section 3d: Persistent Data Storage | | | | | | | 100 | | |
| Section 3e: Network Protocol | | 100 | | | | | | | |
| Section 3f: Global Control Flow | | | | | | 100 | | | |
| Section 3g: Hardware Requirements | | 50 | | | | | | 50 | |
| Section 4: Algorithms and Data Structures | | | | | | | 50 | | 50 |
| Section 5: User Interface and Implementation | | | 50 | 50 | | | | | |
| Section 6: Design of Tests | | 45 | | | | | 45 | 10 | |
| Section 7: Project Management and Plan of Work | 50 | | | | 50 | | | | |

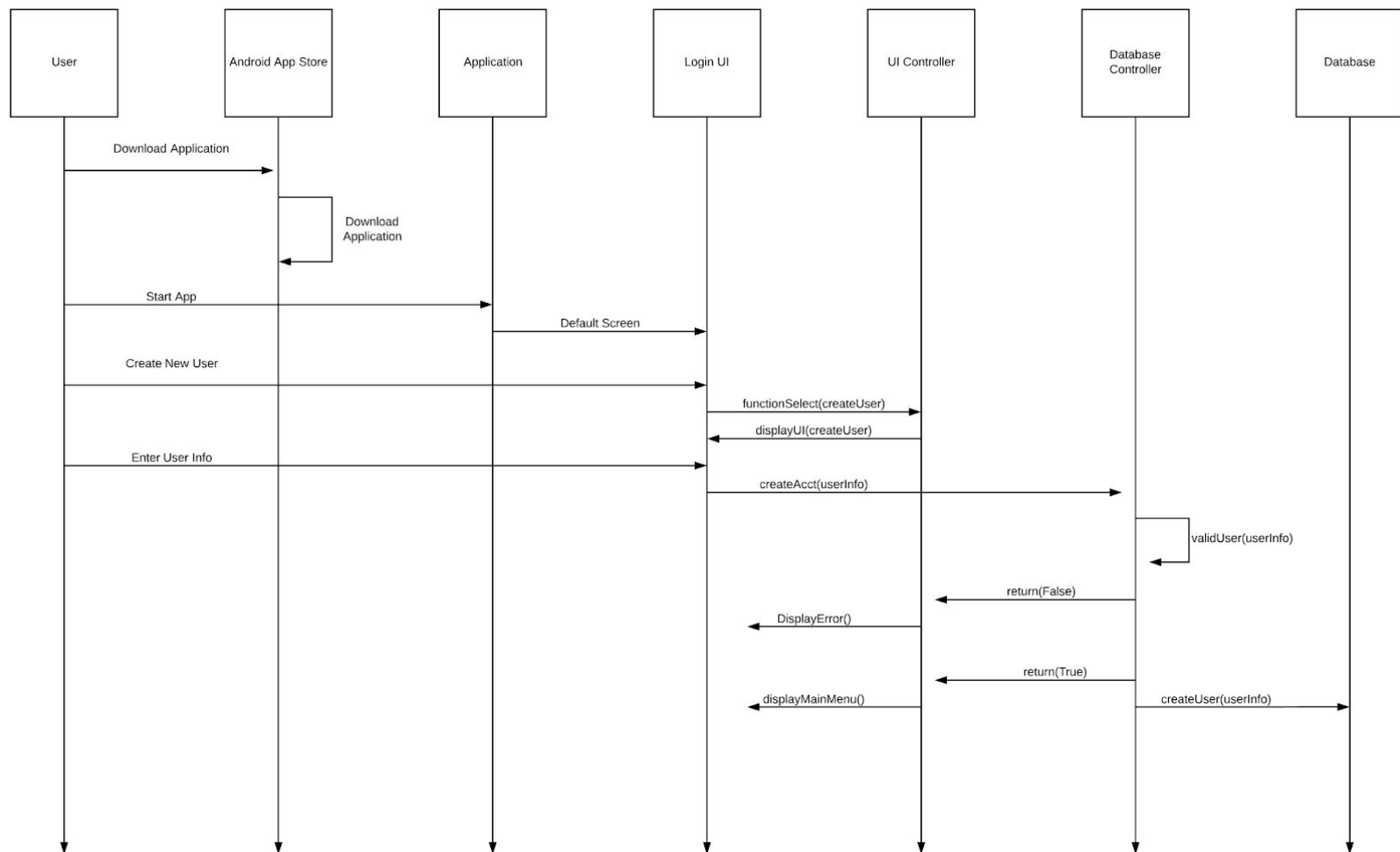
Table of Contents

| | | |
|------|---|----|
| 1. | Section 1: Interaction Diagrams | 4 |
| 2. | Section 2: Class Diagram and Interface Specification..... | 14 |
| 2.1. | Class Diagram | 14 |
| 2.2. | Data Types and Operation Signature..... | 15 |
| 2.3. | Traceability Matrix..... | 20 |
| 3. | Section 3: System Architecture and System Design..... | 21 |
| 3.1. | Architectural Types | 21 |
| 3.2. | Identifying Subsystems | 22 |
| 3.3. | Mapping Subsystems to Hardware | 23 |
| 3.4. | Persistent Data Storage | 24 |
| 3.5. | Network Protocol | 26 |
| 3.6. | Global Control Flow | 27 |
| 3.7. | Hardware Requirements | 27 |
| 4. | Section 4: Algorithms and Data Structures | 28 |
| 4.1. | Algorithms | 28 |
| 4.2. | Data Structures | 28 |
| 5. | Section 5: User Interface Design and Implementation | 29 |
| 6. | Section 6: Design of Tests | |
| | 33 | |
| 7. | Section 7: Project Management and Plan of Work | 39 |

| | | |
|------|--|----|
| 7.1. | Merging the Contributions from Individual Team Members | 39 |
| 7.2. | Project Coordination and Progress Report | 39 |
| 7.3. | Plan of Work | 40 |
| 7.4. | Breakdown of Responsibilities | 41 |
| 8. | References | 41 |

Section 1: Interaction Diagrams:

User Case 1- Download:

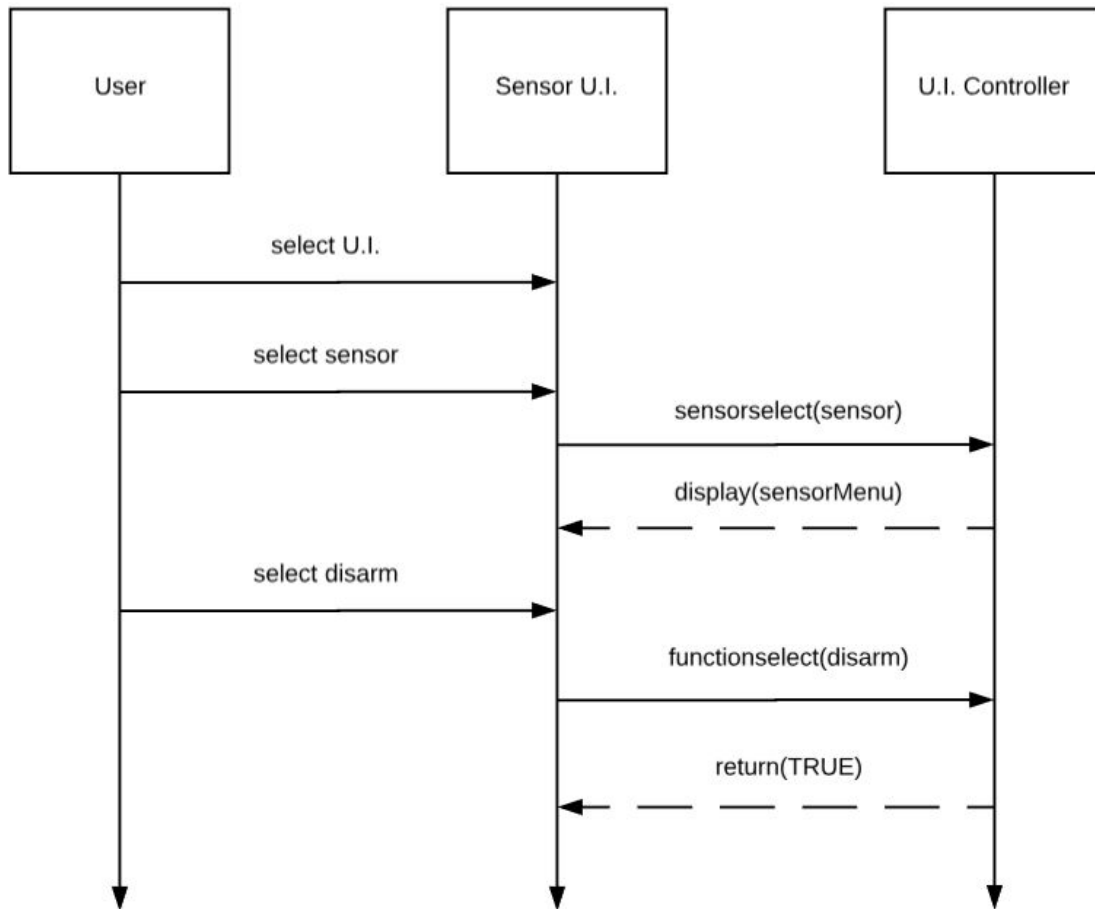


****Planning to Remove this Use Case ****

User Case 2 - Connect:

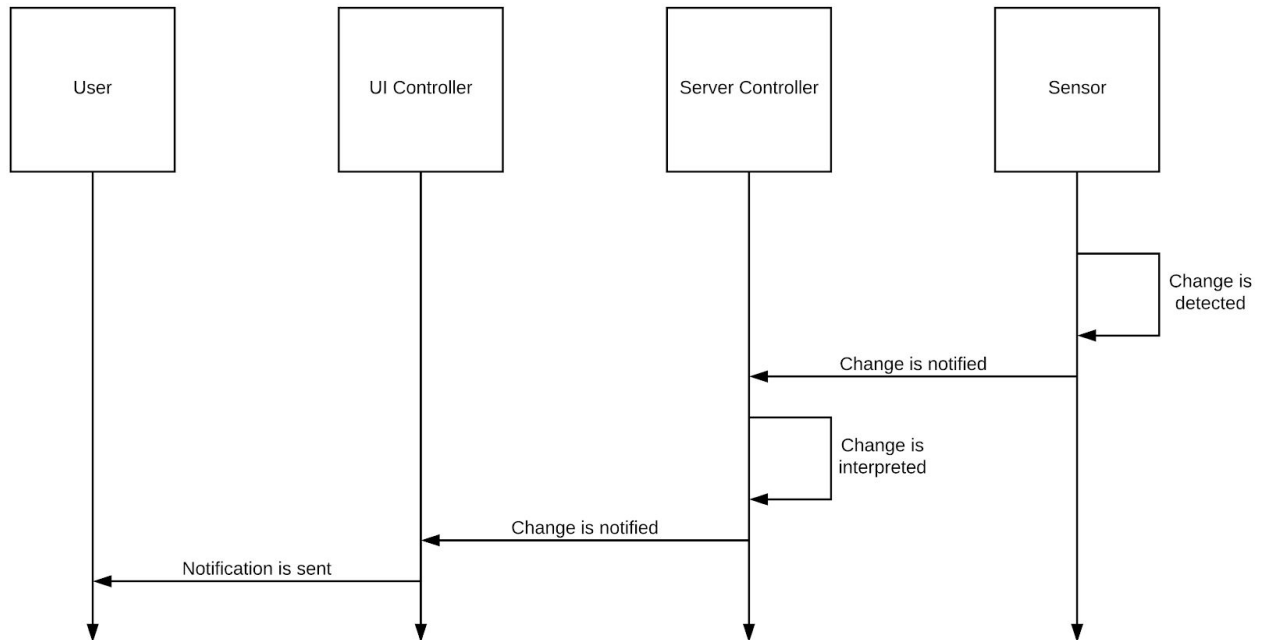
****Planning to Remove this Use Case ****

User Case 3 - Disarm:



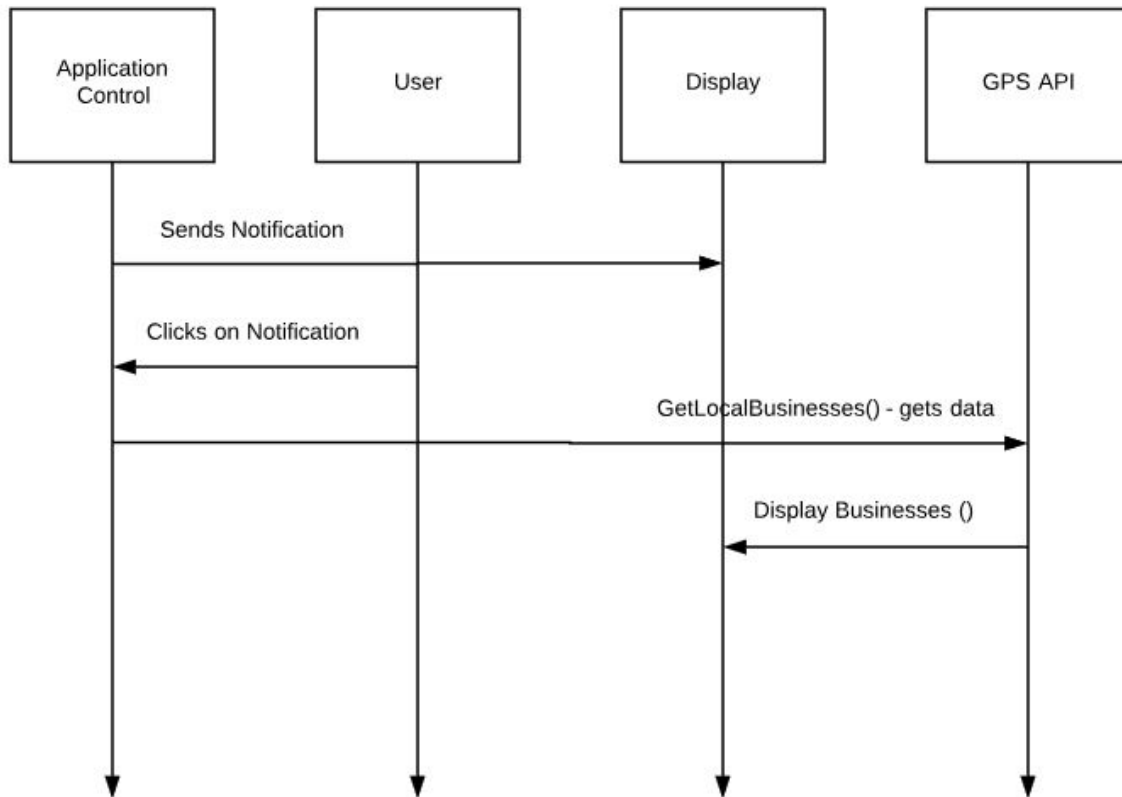
In times in which a user does not require the sensor to notify them, they can opt to disarm the sensor. For this the user will need to go to the sensor user interface in the app where all the sensors are listed out and are displaying their statuses. Once there the user will select a sensor and the user interface controller will display a drop down menu. The user will have the option to disarm or remove the sensor. For our purpose we want to select disarm. This will tell the U.I. controller to disarm the sensor and return a true value to indicate that the sensor has been successfully disarmed. While creating the responsibilities I had thought of including another user interface for the individual sensor's menu; however, I realized that this could be added to the existing user interface through a drop down menu. I had also thought of using a display and touch indicator.

User Case 4 - Signal:



This user case allows the application to push a notification to the user once the sensor detects a signal (or a change). A signal is sent to the server of the application, and the user is subsequently notified. This diagram is designed in a way that allows the user to be identified almost immediately of any change that occurs with the sensors themselves. In our case, the 'change' is interpreted by the server controller and is then implemented into the UI controller which notifies the user. An alternate design could be that the User is notified immediately, and the server controller then interprets the alert and re-notifies the user of the potential hazard. This would enable to immediately know of a potential hazard and could make a difference in allowing the user more time in making a decision. The reason this design was chosen was because it was believed that the interpreted change and appropriate contact listed would be the most user-friendly option, while also not compromising the user's time too much. All in all, this design is both practical and informative for anyone using the service.

User Case 5 - Connect:



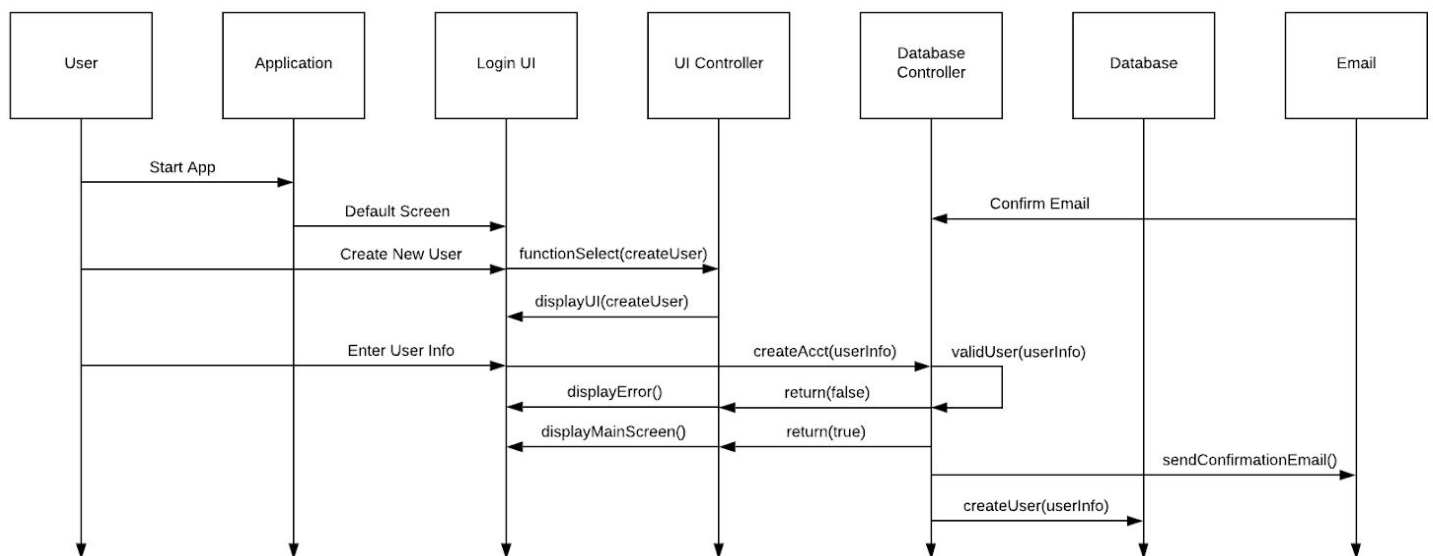
This user case informs the user of contacts of any of the local businesses that are available in order to fix the issue that is detected. For example, if the sensor picks up change in water levels in the basement and detects a flood, a notification will inform the user about the flood and give details on who to call to fix this issue. For this to happen, the Application Control will first send the notification to the display of the phone. Through this, the User will be able to interact with the notification. When clicked, the application control will then use the GPS API to fetch the data of the local businesses around, which is then displayed onto the phone. The reason this design was chosen is because the four main objects in all communicate with each other in order to make for a smooth user interface.

User Case 6 - Notifications:

**** Planning to Remove this Use Case ****

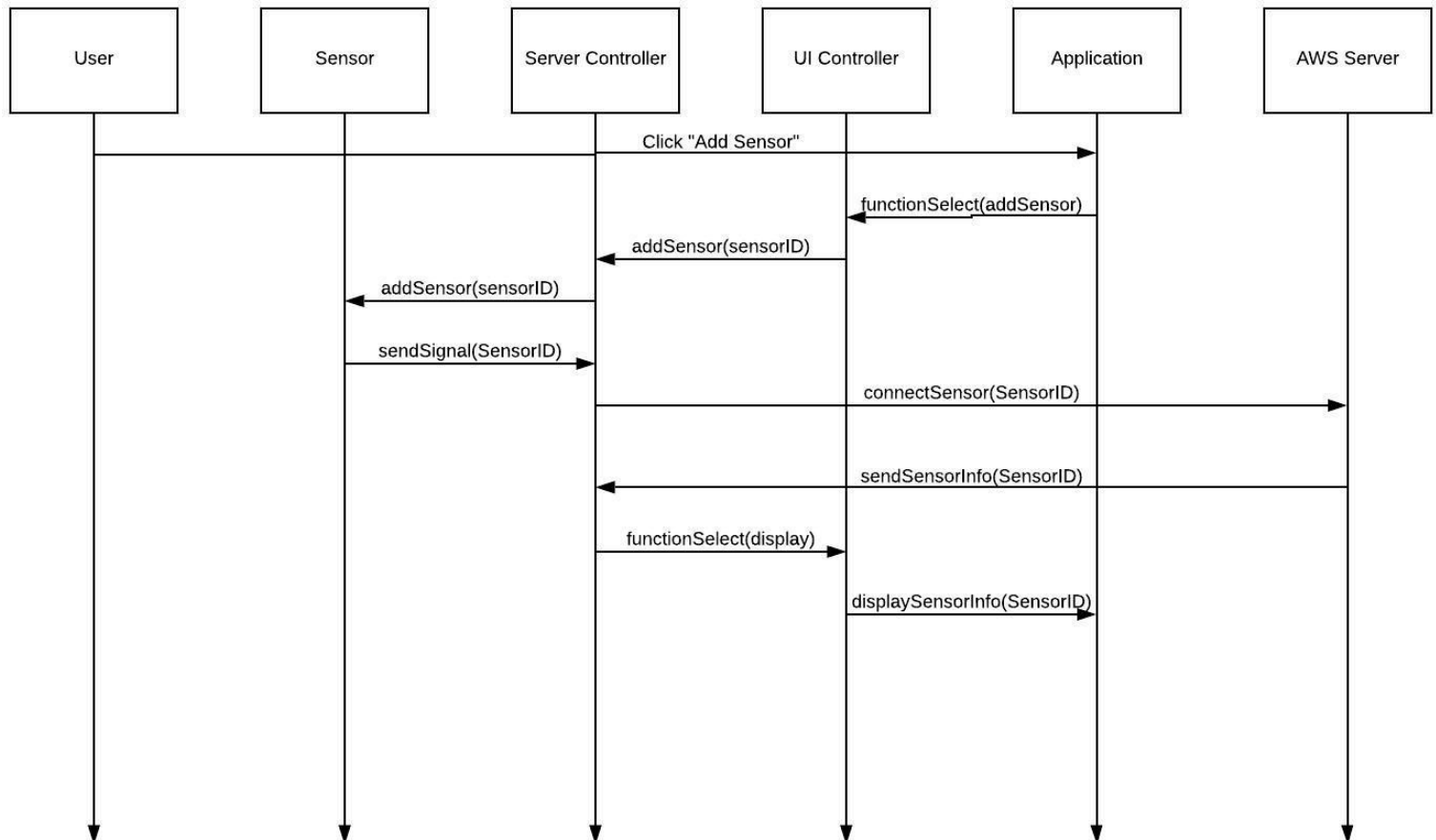
User Case 7 - New Accounts:

Creating a new account would require for the user to already have installed the application. Launching the app will bring the user to the default screen where they have the option of logging into or creating an account. The user would hit the “Create a new account” button and the Login UI would notify the UI Controller which in turn presents the user with the registration screen.



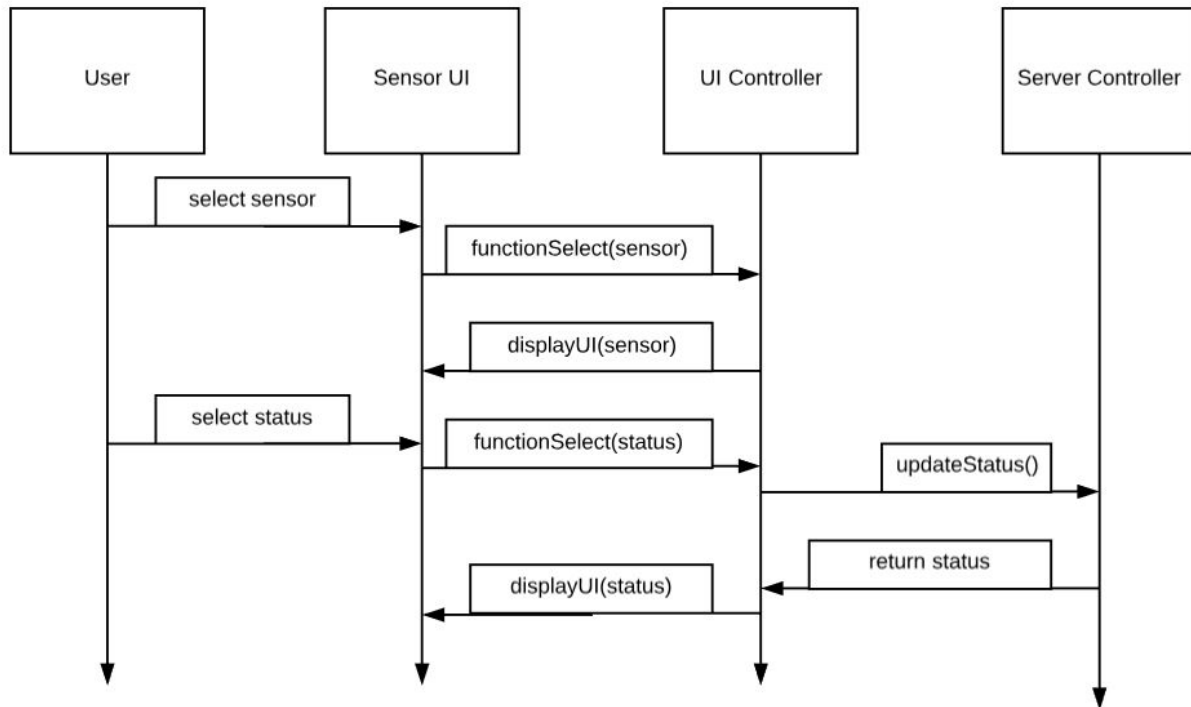
The user enters all the necessary information which the Login UI relays to the UI Controller. The information is brought to the Database Controller which makes sure the information provided by the user is valid. If it isn't, the Database Controller returns a false boolean which will notify the user that an error has occurred, giving them the option to try again. If it is valid, the Database Controller will return a true boolean which allows the user to access the app's features. It will also store all this user's information in the Database and send a confirmation email to the user which can be confirmed at anytime.

User Case 8 - Add:



To add a new sensor, a user would first click a button on the application, which is where the user would be able to see all of the sensor information. Because of this, the Application is an important responsibility. The application will not display anything, however, without a User Interface Controller to manage the operations needed to display the correct information, which is why this is another important responsibility. The Sensor is the foundation of our GlassHome product, which is why it is a major responsibility, and to properly connect to the application, a Server Controller as well as the AWS Server is needed to send and receive important Sensor data. All of the operations between the User Interface, Sensor, and AWS Server will be performed by the UI and Server Controllers, which will then display all of the information to the Application, allowing the User to connect a new sensor and receive its data.

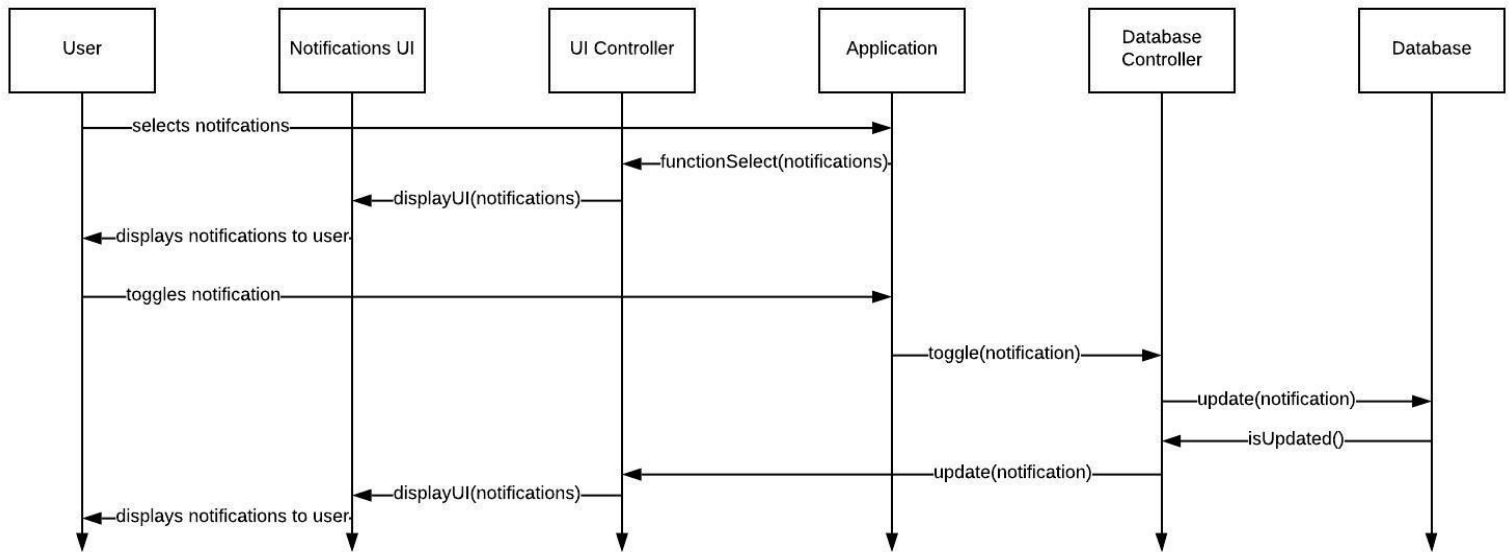
User Case 9 - Check:



To check the status of the sensor the user will go to the sensor user interface, here all the sensors are listed out. From there the user can select the status option and the application will request it from the UI controller. This will send a signal to the server to update the status of the sensor. The updated status will be sent back to the UI controller which will update the display to the new screen dashboard UI for the sensor.

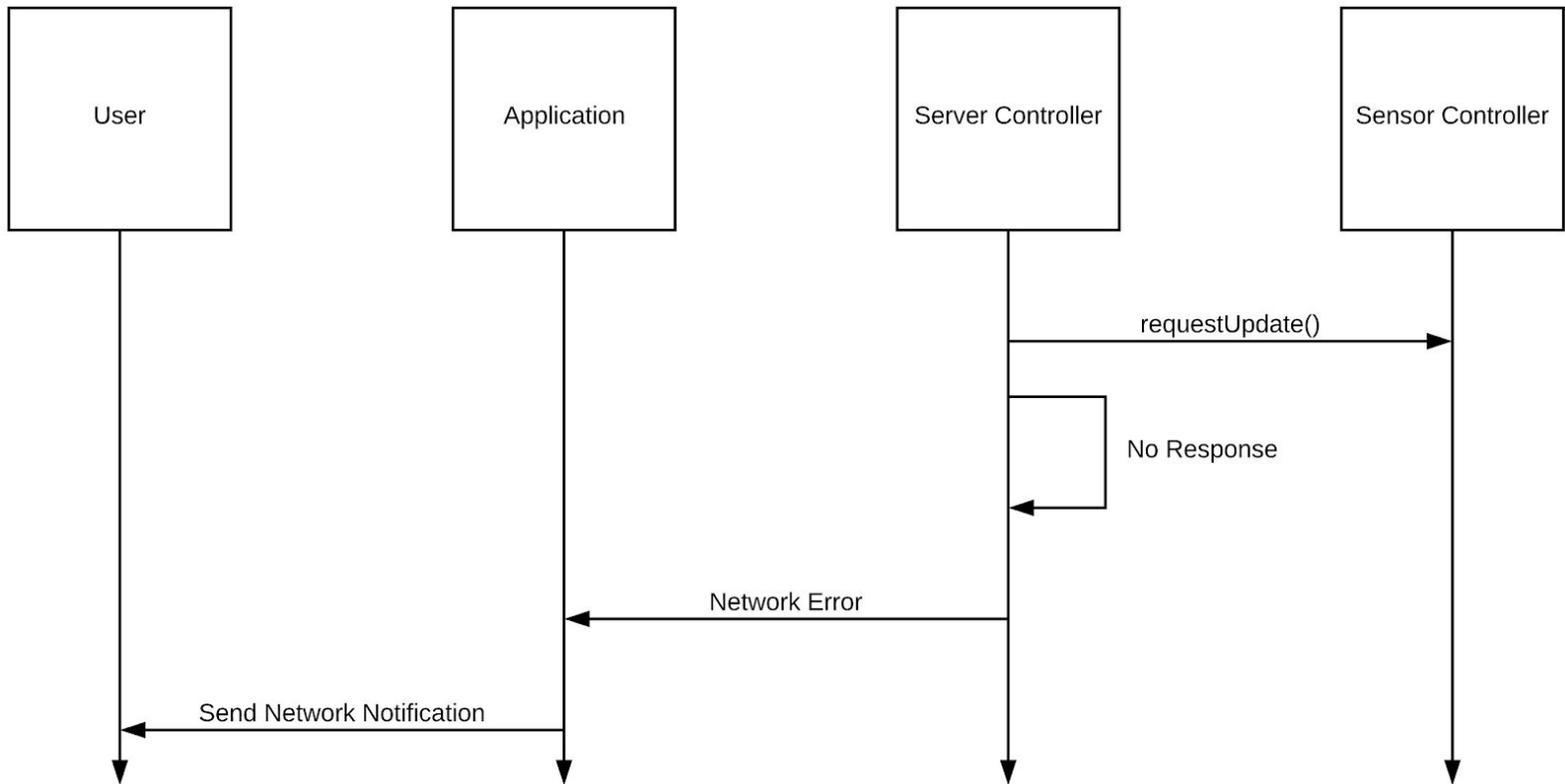
While creating the responsibilities, there could've been another UI for the settings. However, it was more efficient to include it in the sensor UI via the drop down menu.

User Case 10 - Control Alert:



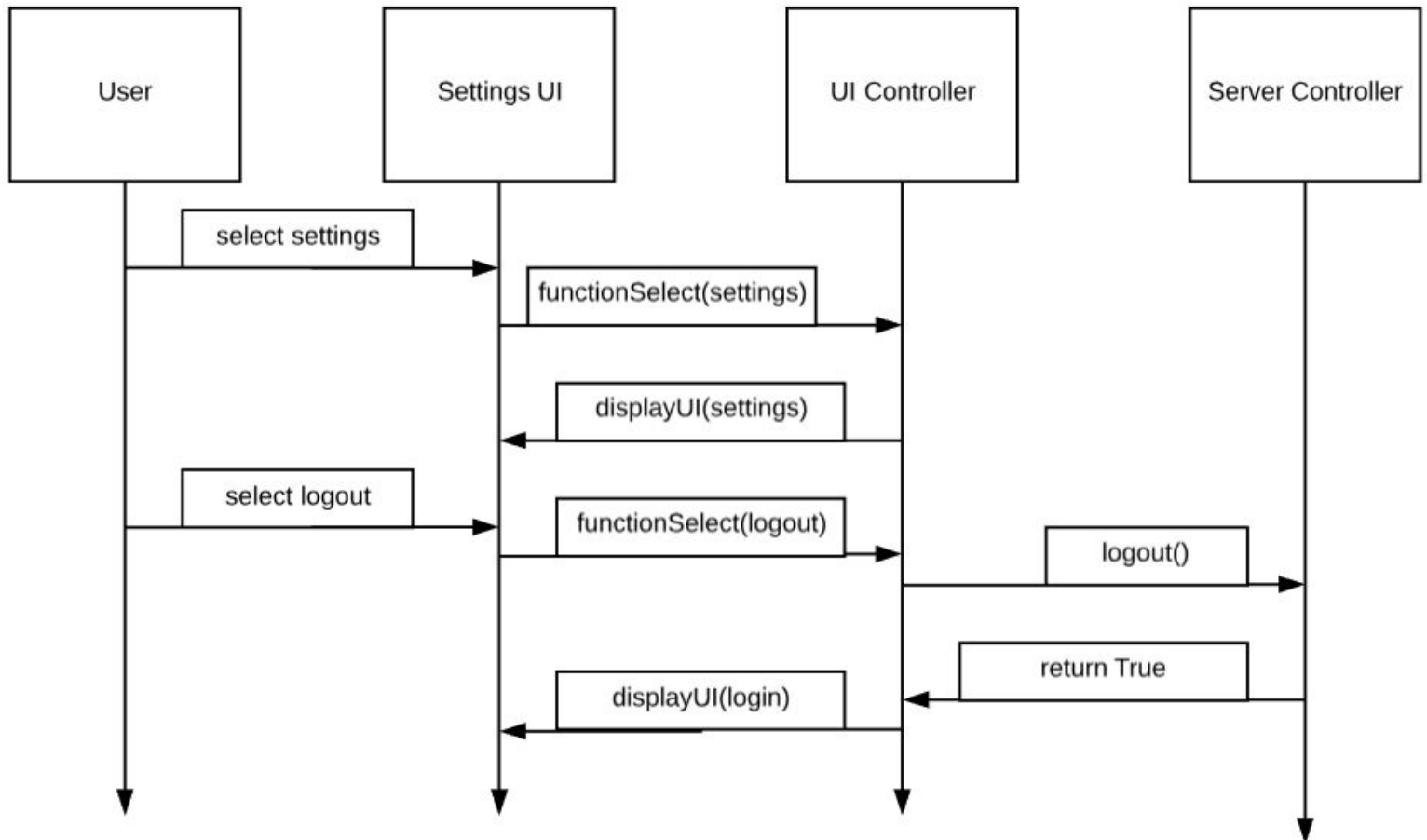
To update notifications setting, the user will first select “notifications” on the application. The application will this request to the UI Controller in which the UI Controller displays the current notification settings to the user (through the Notifications UI). Now the the user has the notification status of all the sensors displayed to them, they can now toggle a notification from on to off or off to on. The application will then tell the database controller this user action and the database controller will then send an update to the actual database. After that, the database will acknowledge to the controller that it has been updated. The database controller makes the UI Controller aware of this update and displayed the updated status of the sensor notification to the user (again, through the Notifications UI).

User Case 11 - Network Error:



In order to have an instant notification the server controller will be constantly requesting an update from the sensor controller. If the sensor controller doesn't send back any data then the server controller will send a network error to the app, which will notify the user that there is a network error.

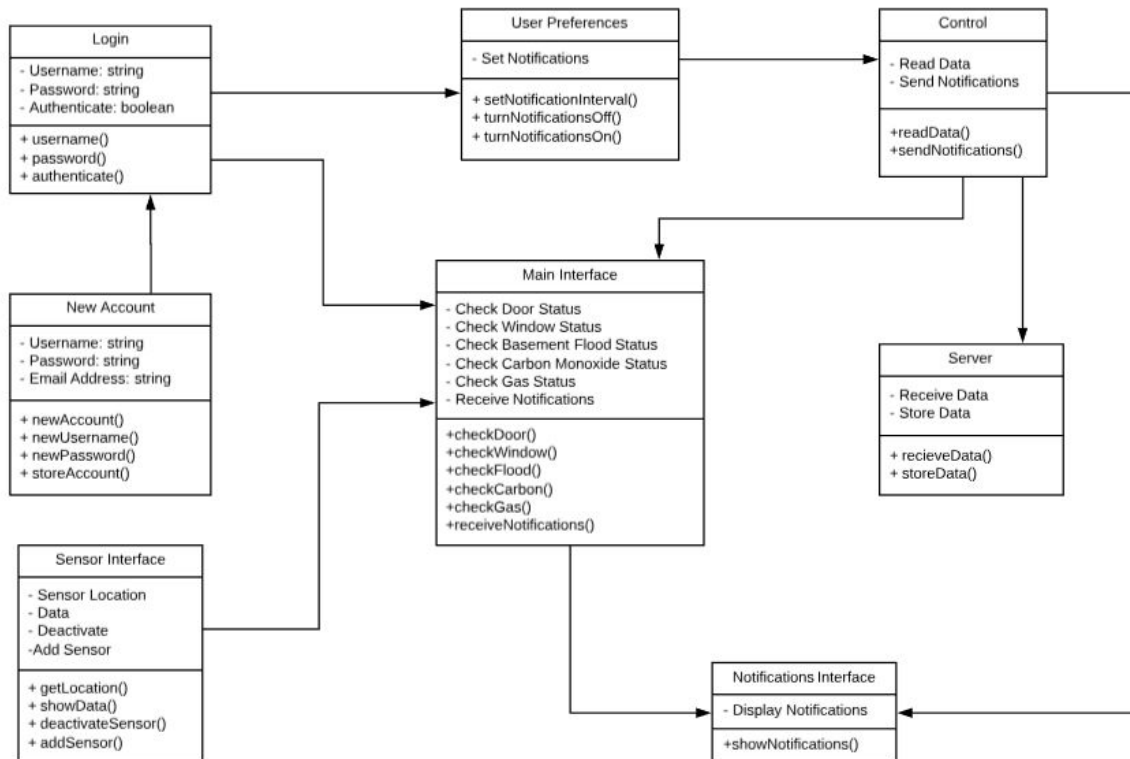
User Case 12 - Log Out:



To log the user out of the application the user will go to the settings user interface and select the log out option. The user interface controller will then log the user out of the server. To notify a successful log out the server controller will return a true value and the user interface controller will then send the user to the login user interface. I thought of adding a display UI as another responsibility, however it is more efficient the do it this way because everything is through the settingsUI and the controllerUI rather than the displayUI. By making it with these four main responsibilities it will be the most efficient for the application allowing the user to log out easily.

Section 2: Class Diagram and Interface Specification

a) Class Diagram

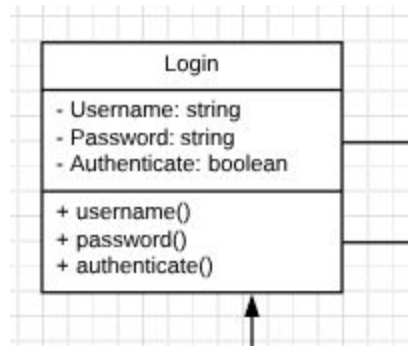


In this figure, we can break the classes down into three main User Interfaces: New User/Account, Sensor Interface, and Notification Interface. The class diagram shown illustrates these interfaces and highlights the necessary usage functions needed in order to get the home security system running. The application starts off by prompting a login/new account mode. This then logs the user in and they are presented with a new interface where they can view sensor statuses (Main Interface) or change their preferences (User Preferences Interface). The sensors also communicate to the main interface as shown in the diagram by updating their status periodically for the user to view. This is done in a two-step process: the wifi module first receives the data from the hardware component in the arduino template, where it then pushes an alert to the AWS server. Then, the AWS server interprets this message and decides a notification to push to the application relevant to the user account. This then leads to the notification

interface, or the control, where users will receive alerts or notifications that allow the statuses to be pushed to them with a vibration or sound. The control is also connected directly to the AWS server, which is useful for storing the data and receiving new sensor status data. The entire setup can be summarized in these three GUI's and are necessary for the working condition of this system.

b) Data Types and Operation Signatures

1)



Username: string

Password: string

Authenticate: boolean

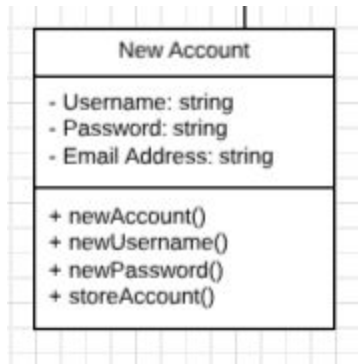
username(): string

password(): string

authenticate(): boolean

Class determines if the user can successfully authenticate and login to the application given a username and password as input. Login data is sent from the New Account class (shown next). The supplied credentials from the user will be matched with the stored data to determine if a match is made. If so, authenticate() will be value True, and it will be value False if the login does not match. If the login is successful, the user will be able to see their main interface and/or preferences. If not, the user must try again or create a new account.

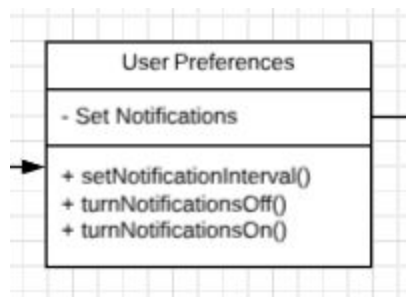
2)



Username: string
 Password: string
 Email Address: string
 newAccount: void
 setUsername(): void
 setPassword(): void
 storeAccount(): void

Class is used for creating new accounts. A user will enter their desired username, password, and email address so that the data can be forwarded to Login. newAccount() is executed when the user initiates creating the new account and storeAccount() is executed when the account is created and pointing toward a Login class. The other 2 methods/functions are used to actually adjust the attributes of the class. After creating a new account, the user can try to login to their account, which was shown in the Login class.

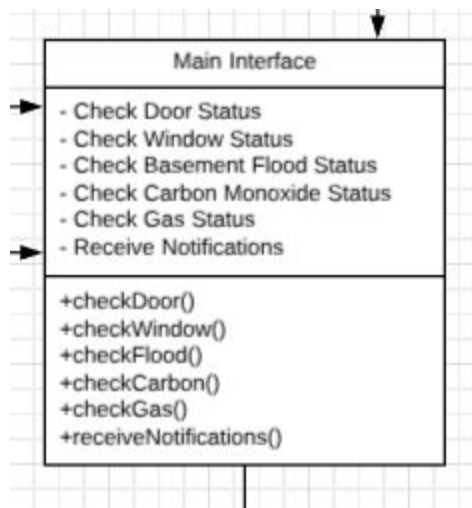
3)



Set Notifications: string
 setNotificationInterval: int
 turnNotificationsOff(): boolean
 turnNotificationsOn(): boolean

Class is used when one wants to adjust their notification settings for the various sensors. As there are multiple sensors and scenarios a user may or may not want to have a sensor activate, this screen allows them to toggle between on/off. `turnNotificationsOff()` will turn off all notifications for a specific sensor. `turnNotificationsOn()` will turn on all notifications for a specific sensor. Additionally, the user has the option to set a notification interval. The set interval is a period of time in which it checks if a sensor is returning an issue and will send a notification to the user.

4)

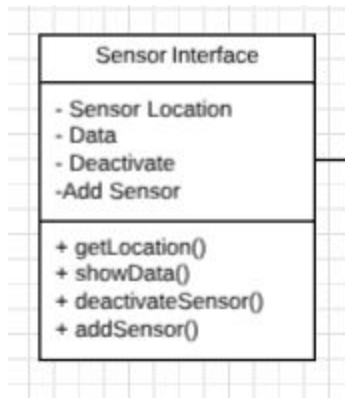


Check Door Status: boolean
 Check Window Status: boolean
 Check Basement Flood Status: boolean
 Check Carbon Monoxide Status: boolean
 Check Gas Status: boolean
 Receive Notifications: string
 checkDoor(): boolean
 checkWindow():boolean
 checkFlood(): boolean
 checkCarbon(): boolean
 checkGas(): boolean
 receiveNotifications(): string

The Main Interface is arguably the most important class. This is the class that is constantly checking if one of the sensors are being “activated.” The Main Interface will know who is currently logged in, check to see if the sensors are detecting any issues, and be able to send notifications to the user when there is a problem (based on their preferences from the previous class). This class will check for open/closed doors, broken windows, basement

flooding, and traces of carbon monoxide and gas. All of these checks are of boolean type since they are either “True” which means the issue was detected or “False” which means everything is how it should be. The “True” values will cause potential notifications to be sent to the user.

5)



Sensor Location: string

Data: string

Deactive: boolean

Add Sensor: boolean

getLocation(): string

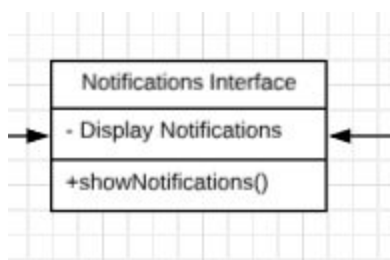
showData(): string

deactivateSensor(): void

addSensor(): void

Class act as the interface for the various sensors a user may want to have associated with their account. The attributes list here is the sensor location, the corresponding sensor data, and the boolean values for Deactivate and Add Sensor. The user will be able to call the method/functions getLocation() and showData() to show those respective attributes, but they will also be able to add and deactivate sensors. A user will addSensor() to begin tracking a new sensor and potentially receive notifications. However, they also have the option to deactivate a currently active sensor if they wish to longer monitor that specific situation.

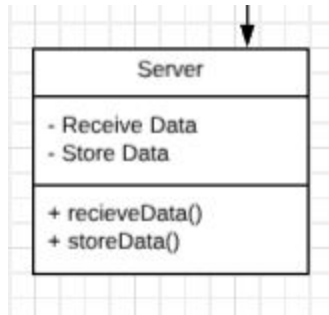
6)



Display Notifications: string
showNotifications: string

The Notification Interface class is fairly simple; it displays the notification that the Main Interface sends due to a “True” sensor picking up an issue. A string value will be stored here as an attribute that will be shown when showNotifications() is called upon.

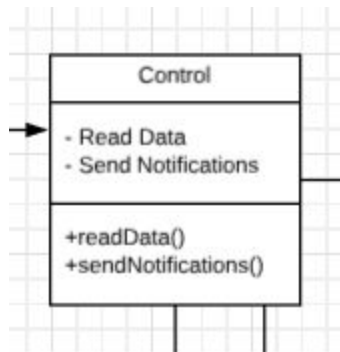
7)



Receive Data: string
Store Data: string
receiveData(): string
storeData(): string

The Server class is where all of the data associated with the application is stored. Whether it be the user’s login information, preferences, or current sensor statuses, the Main Interface will have to speak to the Server to both send AND receive data. All data will be of string type and the two methods/functions that will be called is receiveData() and storeData(). Having a reliable and efficient server is essential to the entire operation of the application.

8)



Read Data: string
Send Notifications: string

```
readData(): string  
sendNotifications(): string
```

This class is responsible for sending the notifications to the user within the AWS server. The control interprets the new data from the user or the sensors and is responsible for populating the notifications interface, allowing a clean and efficient method of pushing notifications based on not only the specific sensor (home hazard or home security) but also the user's notification preferences set up in the preferences interface of the application.

c) Traceability Matrix (explanation)

As seen in the Class Diagram above, the different classes all interact with each other in different ways. The Main Interface acts as this sort of “main hub” for the application. Users create and login to accounts to reach the Main Interface to adjust sensors, change preferences, and send/receive notifications. While we have already provided specific details regarding each specific class above, the purpose of this section is to provide a written explanation from a high-level view on how all of the class work together and “trace” to each other. Since this application requires sensors to be associated with specific users, consumers must create and login to an account that is unique to them - this is where the New Account and Login class come into play. Once there user logs in, they will redirected to the aforementioned Main Interface. Additionally, the user will be able to adjust their preferences (User Preferences) in terms of what sensors are on/off and notification intervals. The purpose of the Control Class is just to send/receive data between the Main Interface and the Server. Control acts as a “middleman” between the two. Finally, Control will also send notifications (Notification Class) based on the status of the different sensors in Main Interface.

Section 3: System Architecture and System Design

a) Architectural Styles

Our product GlassHome is structured to have multiple tasks and functions simultaneously running in parallel. Each part of our system works independently on its unique functions, and then sends the data collected to our server. To make this development manageable, we have multiple people working on the different parts of this system so that everyone is able to work on a crucial step. Our GlassHome system can be divided into three main parts: the Sensor, the AWS Server, and the Mobile Application. The architectural styles needed to ensure smooth functioning of our product are:

- a) Event Driven architecture
- b) Database architecture
- c) Client and Server Architecture

Our system interacts with itself through its Event Driven architecture. The output that a user receives on his/her mobile application is based on what information is detected, produced, collected, and sent by the Sensor, AWS Server, as well as Mobile Application. The sensors can detect a change in movement or a door opening/closing. This will trigger the sensor and will cause it to send a signal to our AWS Server, which will then send the data to the Mobile Application. The user will receive a notification when an event occurs due to each part of the system working together based on the specific event that occurred.

Since there are many unique functions which must operate in parallel with one another and process the same information, each component of the system must be able to communicate with the other subsystems. Data must also be stored for and sent to the correct user. This is where we need a Database architecture. A user will first register his/her account on GlassHome, and this information will be stored in our AWS Server and Database. The Sensors that the user adds to his/her device will also be sent to the Server and Database in order to collect the information as well as store it for both safety and management reasons. Private home information as well as account information will be stored in the database, which is why it is a crucial architecture that needs to be implemented in order to have a successful and user friendly product.

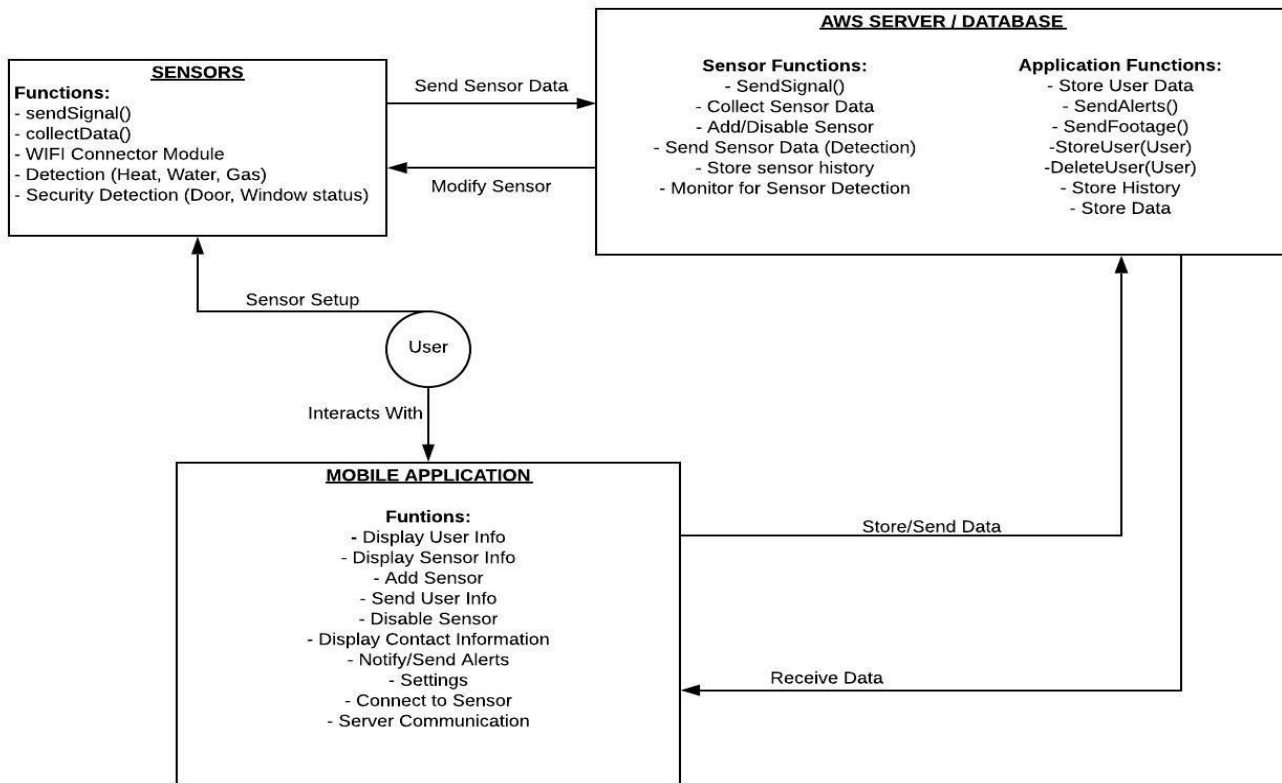
The Client and Server Architecture is what will allow proper communication between the user and the Sensor information as well as Application notifications. In our system, we want different users to be able to access similar information and feel connected to the sensors, which are connected to the AWS Server. We want to create a Client and Server Architecture to allow

the user to have access to all of the features included in GlassHome. After the user inputs their relevant information, it is the server's job to process all of the information given in order to display what the user needs.

b) Identifying Subsystems

Users will interact with our GlassHome system from the mobile application. There are many subsystems, however, that allow GlassHome to provide the proper information to every user.

- The first subsystem is the User - Sensor subsystem. The user first interacts with when buying the product in order to set up the monitoring and security system. It includes functions such as sensor setup, detection, and sharing data with the Server subsystem.
 - The second subsystem is the AWS Server, which will directly interact with the Sensors and Mobile Application, and with the user indirectly. This will hold all of the data sent by the Sensors as well as user information. The server will also be the middle child in our subsystems, sending data to and from both the Sensors as well as the Mobile Application.
 - The third subsystem is the User - Mobile Application subsystem. This is what the user will use to view all of his/her information, as well as perform functions such as view sensor information, add/disable sensors, turn on/off notifications, view emergency contact information, and much more. This is one of the most important subsystems because it will allow GlassHome to work directly with the user. The network protocol would be HTTP.
-
- UML Package Diagram:



c) Mapping Subsystems to Hardware

Our system will require running on multiple computers to work successfully. As mentioned previously, our key subsystems include a user running our application, sensors set up wherever our client desires (home, workplace, etc.), and some sort of a server/database that will connect everything together.

The mobile application subsystem will run on a smartphone. Right now, we're developing exclusively for android devices. These devices are built to connect to the internet and so connecting to the server/database and sensor subsystems should be possible.

The sensors subsystems will run on sensors we build for our customers. They will be built on arduino boards and will have wifi modules installed. This will allow for successful communication with the other subsystems.

The server/database subsystem will run on AWS. AWS offers services that will allow us to store the data of our customers as well as relay this data along with data from the sensors between the other subsystems.

d) Persistent Data Storage

Glass Home requires a very simple storage of database. One case is the constant storage using the login and logout features. When a user requests for an account log in or logout the system will send this information to the server and compares it to the data that was saved in the database. The database that will store all this information is very simple and can be stored in a table such as the one below:

| ID# | Name | Username | Password | Login Attempts |
|------|---------------|------------|----------|----------------|
| 1234 | John Smith | Smith123 | 12345 | 0 |
| 2345 | Michael Brown | MBrown245 | 12345 | 0 |
| 3456 | Sarah Miller | Miller!182 | 12345 | 0 |

In the table above the database stores the login username and password that the user was able to successfully create. When the user creates an account, the system will create an ID number for the user and store the information within that ID number. Things such as the user's full name, password, username, and the number of failed login attempts are saved.

We have other important datas that will need saved into the database. Such keeping a history of all the actions and a live feed recording from the cameras installed. Under each ID number, the user will be able to see the full history of the sensors. One such is if the motion sensor picks up that there was movement, the camera will pick up and start a recording. This data is then send to the database and saved there. The time and the date of the event will also be recorded as well. An example of this database can be shown in the table below:

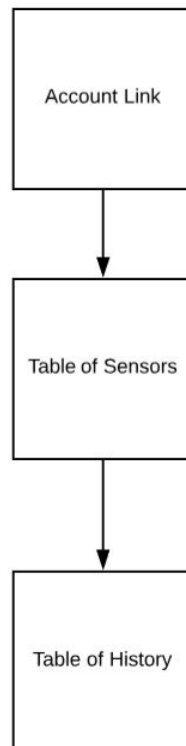
| ID# | Sensor ID | Status | Report Status |
|------|-----------|--------|---------------|
| 1234 | 1111 | online | Viewed |
| 1234 | 1112 | online | Not Viewed |
| 1234 | 1113 | online | Viewed |

This table shows the database of all the sensors and shows if the sensor is on or off. Essentially this shows the status of all the sensors.

| ID# | Sensor ID | Time | Message | Footage |
|------|-----------|--------------------|-------------------------------|---------------|
| 1234 | 1111 | 1/10/19 @ 12:40 AM | Movement detected | Not Viewed |
| 1234 | 1112 | 2/11/19 @ 1:30 PM | Water hazard detected | Not available |
| 1234 | 1113 | 2/21/19 @ 3:45 PM | Sensor successfully installed | Not available |

This table shows the history alert history of all the sensor. If a motion sensor detects movement it will be recorded and the time as well as a footage of the event will also be recorded.

As we can see, the three databases are all related to each other, we can conclude that this is a relational database to organize all three of the tables together. The logic relationship of the three database table can be shown below:



e) Network Protocol

Our system, GlassHome, runs on multiple machines working together coherently. This includes the sensors, the AWS server, and the mobile application. The sensors receive various signals from around the house (water leaks, gas leaks, doors open, etc.). Once they pick up a specific signal, they send this data through the WiFi module to the AWS server which then sends the notification to the mobile application. Since the system works simultaneously with multiple parts, communication between the three is very important. To understand this we can look at what network protocols are in use.

Our system uses the ESP8266 WiFi Module for Arduino coupled with the sensors, in order to create a line of communication from the sensors to the AWS Server over the internet. There are multiple protocols within this connectivity, called the stack of protocols. This stack has multiple layers that are important in order to connect and communicate with the other objects in the system:

- The Link Layer : This layer contains the physical link between two devices. In our system, this is done using a WiFi connection. Once the WiFi link is established the ESP8266 is part of the local area network (LAN) and can communicate with all devices on the LAN.
- The Internet Layer : Every device on the network has a personal Internet Protocol (IP) address. This allows each sensor to send a message to a specific address. The ESP uses these IP addresses to know where it should send the data. With this layer, it is possible that the device can send packets over the internet.
- The Transport Layer : The transport layer in the ESP8266 uses a protocol called Transmission Control Protocol (TCP). This protocol makes sure that all packets are received, and that the packets are in order, and that the corrupted packets are re-sent. It solves the issue of any corrupt packages sent and checks that everything is in order.
- The Application Layer : The application layer is mainly for understanding the language that the data is sent through. We use the HyperText Transfer Protocol (HTTP) in this layer in order to communicate. It uses text to perform requests and interpret responses from the client to the server and back.

Representation of the TCP/IP Stack in the ESP9266 WiFi Module:

| Layer | Protocols |
|-------------|-------------------------------------|
| Application | HTTP, FTP, mDNS, WebSocket, OSC ... |
| Transport | TCP, UDP |
| Internet | IP |
| Link | Ethernet, Wi-Fi ... |

f) Global Control Flow

Event driven system: An event can be described as a significant change in state. Our system GlassHome monitors the house continuously and is always checking to see whether there is a case of an event through the sensors, which are laid out throughout the home, building, restaurant, etc. When combinations of sensors are triggered GlassHome sends a notification to the user and also depending on the situation notifies any emergency hotline. Considering that our system is an event driven system our system is of an event response type.

g) Hardware Requirements

Our system relies on several different sensors, an arduino board to connect them to, a wifi module to connect to the WiFi, and a stable internet connection.

Sensor Requirements:

- Hazard Prevention : Carbon Monoxide Sensor, Water Sensor, Smoke Sensor, Gas Sensor
- Home Security: Motion Sensors coupled with Sound Sensors, Magnetic Read Sensors

The sensors included will detect various signals around the house and send these signals to the real time firebase database using the Arduino and the NodeMCU WiFi module. The signals include, open/closing doors, gas leaks, floods, window breaks, as well as carbon monoxide leaks.

Hardware Requirements:

- Arduino UNO
- NodeMCU WiFi Module

The NodeMCU will connect the sensors to the firebase and send real time updates when turned on. This is paired with the arduino and connected to the sensors to pick up the signals. The NodeMCU will be connected over a stable WiFi connection supplied by an access point or router.

The application interface is downloadable on an Android Smartphone, updated to the current version of Android OS. There are no restrictions on screen resolution and it will require minimal space to install, (<15 Mb).

Section 4: Algorithms and Data Structures

a. Algorithms

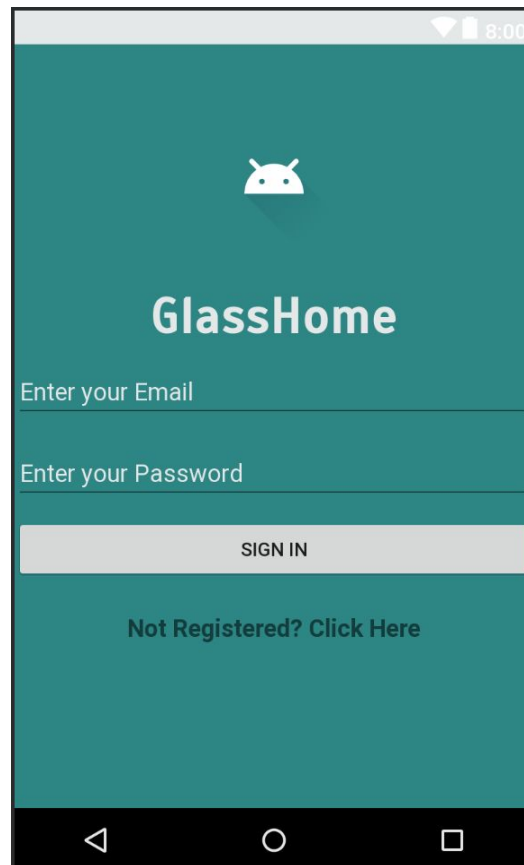
The current plan for our project will not make use of any specific algorithms. The only algorithms that *may* come into play later on in the project is searching (for logins and hash keys). In those situations, the complexity will be $O(n)$.

b. Data Structures

Our project will make use of a few data structures to store information. First, the database is essentially a collection of tables that can be accessed to obtain a various information. Information includes attributes such as username/login, password (would have to encrypt), email address, phone number, etc). The other idea is to use Hash Tables to store user objects in. A user object will have many of the above attributes all stored and accessible via unique hash key. Hash Table will be used within the application itself while the database will always be up and running regardless if the application is or not. Our final report will have more in-depth details regarding the data structures that were used in the process. We must fully expose ourselves to building the application to know what other data structures may be needed along the way.

Our decision to work with Firebase was made because of the platform's compatibility with our ideas and flexibility. It made sense to use Firebase as it is a Google product and we are developing for Android devices; the two go hand in hand. Beyond this, Firebase provides us with not only the data structures we needed but the server we wanted to use to bridge together the application and the sensors. The alternative would be to use two platforms that each do one of these things, but in an effort to work smarter, not harder, using only Firebase should make our application and sensor integration a much smoother process.

Section 5: User Interface Design and Implementation



Use Case: New Accounts.

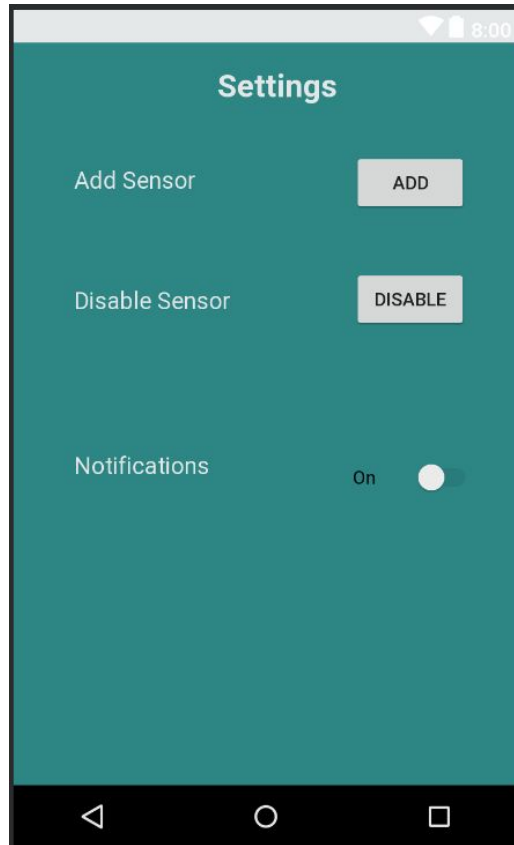
To create a new account, a user can open the application once it is downloaded and click “Not Registered? Click Here”. This will direct the user to the same page, but instead of the Sign In option, the user will be able to register an account. After this is done, the user can log in with an Email and Password as shown above. We have modified this from our previous version by adding a feature to create accounts, and sign in all on the same page.



Use Cases: Check (Status), Log Out.

This is the dashboard where the user can check the status of the different home appliances by clicking on the specific button. Once clicked, the user will be directed to a page showing all of the information for that specific sensor and area of the home. We have modified this by adding specific areas of the home that the user can click to see more information for. Once the user installs sensors, he/she will also be able to customize this page to view different information.

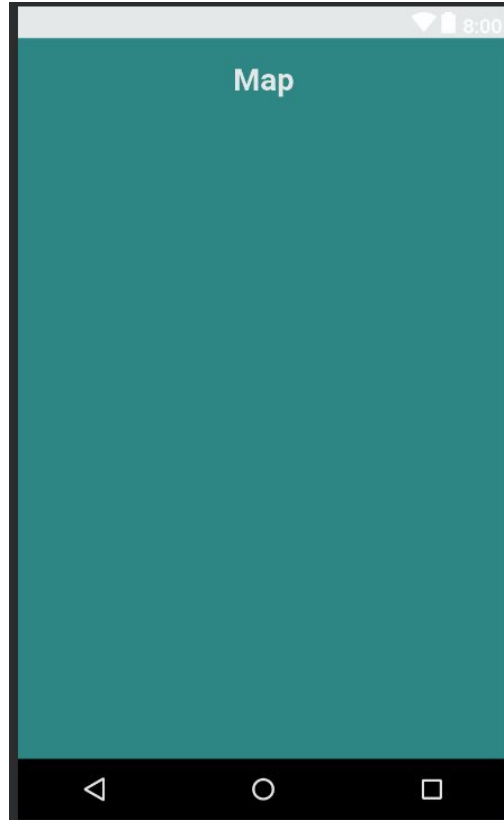
The user will also be able to log out by clicking on a tab on top of the application which is not visible, until clicked. This will give them an option to go to a different page or log out.



Use Cases: Add, Disarm

This use case allows the user to quickly add a new sensor to their set of existing sensors, or to add their first sensor to the list. Once logged into the application, a list (empty at first) will be presented to the user, and the user can click on the “three dot” drop down menu to be presented with a list of pages. The user can then click on “Settings” and subsequently press “ADD SENSOR”. The user will then be presented with a new screen that scans the existing server and lists all possible sensors that the application can connect to. In order for this to work, the user must be in range of the sensor and both the application and the arduino module must be connected to the same WiFi network. Once the user clicks the appropriate sensor they want connected, the screen will present the user with a “Connect” or “Not Connected” status. Users can then repeat this process for multiple sensors or keep any amount of sensors they already registered.

In order to disable a sensor, the User could go through a similar set of steps in order to get to the settings screen but instead of clicking “ADD”, they can click “DISABLE”. This would present the user with a list of already connected sensors to the application, and they can click on one of them in order to disarm a sensor from their dashboard. The user can repeat this process for any number of sensors they had connected.



Use Case: Contact

In this User Interface, the user can view a map of nearby businesses as well as the contact information for these locations. In order to access this section, the user can go to the “3 dot menu” and select “Map”. From there, a color-coded pinpointed location of certain particularly useful selection of businesses and help information will be presented to the user. More specifically, based on the user’s home location, nearby businesses based on a certain radius are going to be listed. If the user clicks on one of the pinpoints, a popup explaining the name, contact info, and services of the particular business will be displayed. The user then has the option of moving on to another business or writing down the contact info of this particular business in order to call them.

Section 6: Design of Tests

Note that for this report you are just *designing* your tests; you will *program and run* those tests as part of work for your first demo, [see the list here](#).

- A. List and describe the test cases that will be programmed and used for unit testing of your software.
 - B. Discuss the test coverage of your tests.
 - C. Describe your *Integration Testing* strategy and plans on how you will conduct it.
- Describe also your plans for testing any algorithms, non-functional requirements, or user interface requirements that you might have stated in your Report #1.

- A) This test we will attempt to connect our sensors with a server for our own app. The sensor we plan on testing is the gas testing or door alarm.
- B) The gas testing we will see if the sensor will be triggered by a gas leak and then send an alert to the user. Similarly with the door alarm, once the door is open or triggered, there should be a signal that travels to the server and then to the application which will send an alert notification.
- C) For the tests we will attempt to break the sensor and see if there will be connectivity with the server and application to send an alert notification.

Unit Testing:

| | |
|---|---|
| Test Case Identifier: | TC - 1 |
| Use Case Tested: | UC - 1 (Disarm) |
| Pass/Fail Criteria: | -Test passes if disarm option is clicked and notifications stop |
| Input Data: | -Disarm option (in sensor settings) |
| Test Procedure: | Expected Result: |
| Step 1: Select sensor to disarm | Application shows the sensor options and status |
| Step 2: Select disarm setting | Application stops sending the notifications to the phone |
| Step 3: Trigger sensor to see if notification is sent through | No notification is sent to the user's phone |

| | |
|---|--|
| Test Case Identifier: | TC - 2 |
| Use Case Tested: | UC - 2 (Signal) |
| Pass/Fail Criteria: | -Test passes if a sensor sends a signal through WiFi to the database and the database records this change when an event occurs |
| Input Data: | -Sensor Signal |
| Test Procedure: | Expected Result: |
| Step 1: Create a triggering event for any sensor | Sensor sends the signal to the database |
| Step 2: Check to see if the database receives and records this change | The database updates in real time to show the current signal |

| | |
|---|---|
| Test Case Identifier: | TC - 3 |
| Use Case Tested: | UC - 3 (Contact) |
| Pass/Fail Criteria: | -Test passes when the user is notified and recommended a list of correct local businesses to resolve the issue detected by the sensor |
| Input Data: | -Sensor Signal |
| Test Procedure: | Expected Result: |
| Step 1: Create an emergency situation by triggering sensors | User is sent a notification to alert about the situation |
| Step 2: Click on recommended businesses | App analyzes the change in the database and lists possible local businesses to resolve the situation |

| | |
|------------------------------|------------------------|
| Test Case Identifier: | TC - 4 |
| Use Case Tested: | UC - 4 (Notifications) |

| | |
|-----------------------------------|---|
| Pass/Fail Criteria: | -Test passes when an event is recorded by the sensor and a notification is sent to the user's phone |
| Input Data: | Sensor Signal, Database Change |
| Test Procedure: | Expected Result: |
| Step 1: Create a triggering event | Sensor will send a signal to the database to record a change |
| Step 2: Check phone | The change in the database will trigger the application to send a notification to the user |

| | |
|---|---|
| Test Case Identifier: | TC - 5 |
| Use Case Tested: | UC - 5 (New Account) |
| Pass/Fail Criteria: | -Test passes if user can create a new account and is able to log in with that account. |
| Input Data: | -Username, Password |
| Test Procedure: | Expected Result: |
| Step 1: Type in a username and and a password | System saves account information into database. |
| Step 2: Log into new account | System checks to see if the username and password are correct; if correct, user is logged into the account. |

| | |
|------------------------------|--|
| Test Case Identifier: | TC - 6 |
| Use Case Tested: | UC - 6 (Add) |
| Pass/Fail Criteria: | -Test passes if a new sensor is added and shows up on the dashboard of the application |
| Input Data: | -Sensor signal |
| Test Procedure: | Expected Result: |

| | |
|---|--|
| Step 1: Add a new sensor to the account | System saves the new sensor to the accounts database. |
| Step 2: Check sensor list to see if the sensor has been added | Sensor is shown in the list of sensors under that account. |

| | |
|--|--|
| Test Case Identifier: | TC - 7 |
| Use Case Tested: | UC - 7 (Check) |
| Pass/Fail Criteria: | -Test passes when user checks a certain area of the house and sees the real time status of each sensor in that particular area |
| Input Data: | --Select Sensor UI |
| Test Procedure: | Expected Result: |
| Step 1:User selects the sensor UI and chooses an area of the house | App shows the sensor UI and lists the sensors in that particular area as well as the real time status of each sensor. |

| | |
|--------------------------------------|--|
| Test Case Identifier: | TC - 8 |
| Use Case Tested: | UC - 8 (Network Error) |
| Pass/Fail Criteria: | -Test passes when app senses a loss in network connection and sends a notification |
| Input Data: | -Lack of data from database |
| Test Procedure: | Expected Result: |
| Step 1: Disconnect sensors from WiFi | App recognizes a lack of data being sent from database of particular sensors and notifies user that those sensors are offline. |
| Step 2: Disconnect app from WiFi | App recognizes no connection to database and sends user a notification. |

| | |
|---|---|
| Test Case Identifier: | TC - 9 |
| Use Case Tested: | UC - 9 (Log Out) |
| Pass/Fail Criteria: | -Test passes when user selects the log out button in settings and is brought to the log in UI |
| Input Data: | -User Input (Click Logout) |
| Test Procedure: | Expected Result: |
| Step 1: User clicks Log Out in settings | App sends user to the log in UI |

Test Coverage:

Acceptance Tests:

- Acceptance tests focus on the “big picture” or the system as a whole. For our system in specific, acceptance tests show us that the system as a whole is coherently working together. Each individual part of the system has its own functionality that combines to make a fully functional system.
 - TC - 3 (Contact) - App must analyze the combination of signals to list out correct businesses to contact
 - TC - 4 (Notification) - The app relies on information from the sensors as well as the database for this task
 - TC - 7 (Check) - The app relies on the information from the sensors and the database for this task

Unit Tests:

- Unit tests test focus on the “individuality” or the details of each component of the system. For our system in specific, unit tests show us that each component of our system is working as it should be on its own.
 - TC - 1 (Disarm) - Must check each individual sensor to see if they are being disarmed when selected
 - TC - 2 (Signal) - Must check each individual sensor to see if they are sending a signal when an event occurs
 - TC - 5 (New Account) - Adding a new account adds to the overall functionality

- TC - 6 (Add) - Adding a sensor is a component of the bigger functionality
- TC - 8 (Network Error) - Checks for WiFi, and adds to the bigger functionality
- TC - 9 (Log Out) - Logs out of the account and adds to the overall functionality

Integration Testing:

Integration testing is a level of software testing where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units. For our system there are 3 key integrations that must be tested.

The first is the integration between all the sensors, the database and the app. This must be flawless as we want the user to have a real time alert for emergency situations. If the app is not optimized to be quick and reflexive then our system is failing in providing the user a sense of safety for their home. In order to test this we must rigorously test each sensor individually and confirm that the interaction between them and the database is perfect. Then we must test the interaction between the app and the database and see if the data is being read properly. This ensures that the reactivity of the whole system is how we want it.

The second is the integration between different types of sensor signals. The app must be able to make an accurate prediction of what type of situation is occurring and also provide key numbers to call in case of emergencies or even smaller problems in the home. In order to test this we must trigger multiple sensors at the same time or in a reasonable span of time. The app must then be able to read these various signals and predict what type of situation is going on and then recommend the proper emergency contacts, or in the case of a non-emergency problem a proper business, to resolve the problem.

The third key integration is the interaction inside of the app and between the settings of the sensors. The transition from one UI to another UI as well as the interaction between UI's and the sensors, should be without as much lag as possible so that the user can have a smoother experience.

Section 7: Project Management and Plan of Work

(a) Merging the Contributions from Individual Team Members

Compiling everyone's work to ensure consistency, uniform formatting, and appearance for our report brings up many issues. But to avoid these issues and keep up efficiency, we first started by dividing up the workload early on. We used Google Drive folder shared with all team members. This way everyone had access to everyone else's work. And during the final compilation, all parts were easily accessible by one sole group member to put together and submit.

During the beginning of the week, we would have a group meeting where we would discuss the status of our project, what we were working on currently, challenges we faced thus far, and what we wanted to accomplish next. Afterwards, we would look at the sections and sub-sections that are apart of the report this week, and divide them up to groups of team members. If some sections seemed to be more work, we would assign more team members on it. This way, the work was dispersed evenly. We would also assign a personal deadline, around mid-week, so we had enough time to account for any issues/ problems that may have arised. This also made the contribution breakdown easy to know what each team member had done.

Near the end of the week and before the weekend, we would have another meeting and take a look if everyone had completed the parts that were assigned. If not, they were directed to do so as soon as possible. On Saturday or Sunday, the final submission document was put together with the cover sheet, contribution breakdown table, table of contents, project management, and references. The other group members would be notified that the final document has been put together and to be revised for any errors or for any incomplete parts.

This organized system our team has been following has not only made our weekly report accurate, but efficient and timely as well.

(b) Project Coordination and Progress Report

Our project consists of 12 main use cases. In case 1 we focus on downloading and setting up the app onto one's smartphone device. In case 2 we are implementing a way to access the app through WiFi. Case 3 allows the user to safely turn off the sensor in their household. Case 4 focuses on the signal. The main focus is to when a sensor picks up a change in the environment there will be a signal that will be picked up. Case 5 will give the user a list of companies to contact in case an emergency is found. Case 6 focuses on notification aspect of the app development. This is where the signal will be send from the sensor to the app to notify the user. Case 7 focuses on the app, where we plan to have to ability of the user to make an account on. Case 8 is the ability for the user to add new sensors in their home and the ability for them to register the new sensor into the app. Case 9 allows the user to check the status of the sensors to make sure that all the sensors are on or off. Case 10 allows the user to change their notification settings. They are able to turn off receiving messages. Use case 11, warns the user If there is a network error in the system, like wifi outage, a signal will be sent to the application to notify the user. The last user case enables the user to log out of the system in the application.

For our first demo we will focus on use cases: 2, 4, 6, 7, 9, 10, 11, and 12. Our main focus for the first demo is to have a working sensor that can connect to the app and send a notification to the app when the sensor detects a change in the environment. On the app development side we focus on being able to successfully create an account, and the ability to logout of the account. Our final task is to connect the sensor and the app all together into one system. Once this is complete we will be able to send information through the sensor and store them into a database. This will allow the sensor information to be send to the user and notify the user as well as to check the status of the sensors.

For our first demo we plan to work on most of these use cases. For our app development we have implemented and successfully made a login and logout feature. The enables the user to create and account and logout of their account, use cases 7 and 12. We are currently in progress to include more functions in our app. Currently the main objective is to have a working notification setting working for our app. In order to have that to work we need to focus on our sensor development as well. Our plan is to have use case 4 done. This is where the sensor is able to pick up a signal. Once that is done we can work on the ability to send a notification to the user through the app. Now we will focus on connecting the two together to have a functioning system. The next plan is to add extra features/functions to the app, such as the ability to add new sensors, change notification settings, and networking error notification. This way we will have a functioning sensor and app developed to showcase for our first demo.

(c) Plan of Work

For our successful project completion, we have been using a Product Roadmap.

| <div> <div>Sensor Design Mar 1, 2019</div> <div>AWS Connectivity Mar 9, 2019</div> <div>App Dev Mar 16, 2019</div> <div>DEMO 1 Mar 26, 2019</div> <div>Q2</div> <div>Grouping of Sensors Added Apr 6, 2019</div> <div>More App Functionali Apr 17, 2019</div> <div>Final Demo Ready Apr 22, 2019</div> </div> | | | | | | | |
|---|------------------|-------------------------|---------------------------|---------------------------|--|--------------------------------------|------------------|
| Product Design | | | | | | | |
| Sensor Design | AWS Connectivity | Application Development | System Linked/ Demo Ready | | | More Application Functionality Added | Final Demo Ready |
| Hazard Prevention Team | | | | | | | |
| | | | | Add Other Hazard Sensors | | | |
| Home Security Team | | | | | | | |
| | | | | Add Home Security Sensors | | | |

Our first major deadline is March 26, this is Demo 1. Our system we are developing incorporates 2 Groups, Hazard Prevention and Home Security. For the first Demo, we want each group to have 1 working sensor, be connected to the database through WiFi, and have a simple app created to have the status of the sensors shown on the main page. This status description will be real-time and the status will be instantaneously updated.

We also want to be able to send an app notification to the User. Our main idea is hazard prevention, so in case of an issue that has arised, we would like a notification to be sent.

Afterwards, we would like to include the remaining sensors, including a live camera, and added functionality to the app, including using Google API to map out nearby assist from a reported hazard. This is what we would like to show at our Demo 2.

(d)Breakdown of Responsibilities

We have 9 group members, so we have broken each of us into 2 teams; Hazard Prevention team and the Home Security team.

- Hazard Prevention: Shaan, Shivum, Andy, Nathan, Kyle
- Home Security: Harshil, Adarsh, Avi, Parth

This is our original plan, currently we have broken up our work based on interest. Shaan, Nathan, and Kyle have narrowed into App Development using Android Studio. Harshil, Adarsh, and Avi have narrowed into Sensor Design using NodeMCU. Parth, Shivum, and Andy have narrowed into Sensor Design using Raspberry Pi to implement the Raspberry Pi Camera.

We are using the Google Firebase to link all aspects of this project together so we are all in communication with each other. All groups working on each of these concepts are in charge of the integration.

Section 8: References

- Software Engineering Fall 2013 project - Voice Control Based Home Automation System
- Software Engineering Spring 2012 project - autoHome
- Lecture 9 - Object Oriented Design Basics
- <https://www.lucidchart.com/>
- <https://www.ece.rutgers.edu/~marsic/Teaching/SE/report2.html>
- https://en.wikipedia.org/wiki/Software_architecture#Examples_of_Architectural_Styles_.2F_Patterns
- <http://www.uml.org/>
- <http://softwaretestingfundamentals.com/integration-testing/>