Software Engineering - Spring 2019

Report 3

GlassHome

Group 12

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

Avi Patel, Nathan Silva, Parth Patel, Kyle Abed

April 14, 2019

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

14:332:351 Software Engineering Report 3
Department of Electrical and Computer Engineering
The State University of New Jersey,
Rutgers

Contribution Breakdown for Report 3:

Responsibility	Harshil Parekh	Adarsh Gogineni	Shivum Mehta	Shaan Parikh	Andy Guo	Avi Patel	Nathan Silva	Parth Patel	Kyle Abed
Project Management	11.11%	11.11%	11.11%	11.11 %	11.11%	11.11%	11.11%	11.11%	11.11%
Revisions	11.11%	11.11%	11.11%	11.11 %	11.11%	11.11%	11.11%	11.11%	11.11%

Table of Contents

Summary of Changes:

Throughout this semester, we have gone through many changes in implementation of this project. Notable changes can be seen in the Glossary of Terms, System Requirements, Functional Requirements Specification, Domain Analysis, Interaction Diagrams, Class Diagram and Interface Specification, System Architecture and System Design, User Interface Design and Implementation, and Design of Tests.

Key notable changes throughout the evolution of this project:

- Change from AWS database to Google Firebase
- Change in Sensor-intensive to App-intensive development in features
- Addition of Emergency Contact Number feature
- Addition of Lock Mode feature
- Addition of sensor design manuals
- Addition of Map feature
- Addition of window protection sensors (security)
- Addition of fire and carbon monoxide sensors (home hazard)
- Addition of Camera is under-development

Section 1: Customer Problem Statement:

Problem Statement:

Residential Family Home:

Home Monitoring System Team,

I currently live in a residential area in the suburbs with my wife, my three children (ages 16, 10, and 5), and my mother in law who is not very well coordinated due to old age. We also have 2 well-behaved dogs. With that being said, you can only imagine how hectic our home becomes on a daily basis trying to run around and get everything done. Between my wife and I going to work, the kids going to school, taking care of my mother in law, as well as a never-ending list that goes on and on, it is extremely difficult to ensure nothing goes wrong in the process. Like most parents, our main concern during all of this pandemonium is the safety and security of our family.

We are burdened with a lot of extra and unnecessary stress that stems from trying to remember if we did something important or not. Many little things can be easily overlooked due to the constant chaos going on. Little things such as remembering if we left the stove on, if all of the doors and windows are closed and locked, or if we shut off the lights become very difficult. Other things such as checking the basement for water leakage or flooding due to rain, or making sure the heat/AC is turned off before we leave seem minimal but are actually major safety concerns. A stove being left on could, in an extreme case, burn our entire house down. Leaving windows and doors unlocked increases the likelihood of a burglar successfully invading our home. On a less serious note, even the money being saved from ensuring the heat/AC is turned off is something we value tremendously. We are looking for a solution that will put our minds at ease and let us continue on with our day without having to worry about the security and safety of our family or home.

Ideally, for us, a system that can be implemented into our mobile devices would be the most beneficial. We would like some sort of infrastructure to be set-up within our home that passively checks for all of the aforementioned problems. My wife and I will then be able to have customized settings on our mobile application that alerts us of potential problems based on our desired preferences. For example, I can turn on "monitoring" when I leave the house (would be even better if the application recognizes I left automatically) and be notified if the stove is still on after 10 minutes. Knowing this, I could go back home (or call somebody who is there) to turn the stove off. This would help us keep our family safe from any possible home issues.

A reliable system of this nature would be invaluable to my family. Knowing I will be alerted anytime there is an issue will allow me to focus more on the things that really matter in life.

Corporate Lab:

I lead a research laboratory where safety is our number one priority. Part of my job is supervising all of the lab technicians to make sure they practice safe work habits and closely follow the rules. Unfortunately, I am only one person and the company employs many tab technicians. It is difficult to keep an eye out on everyone while simultaneously catching every single safety measure they might forget. To remedy this problem, I am in the market for a sort of security system to help me monitor everyone's work and keep our laboratory a safe workspace.

Some experiments our lab technicians carry out require heating up different liquids and chemicals. Believe it or not, I have caught technicians leaving burners on after using them. Besides the obvious fire hazard, this is extremely problematic considering the fact that technicians sometimes work with flammable substances. Another problem we have is with the storage of all these materials used in experiments. We have this huge storage facility where everything the technicians need can be found. On occasion, materials are mishandled by either our lab technicians or our suppliers and we'll have stuff leaking. This is a big issue because if some extremely reactive chemical spills into another reactive substance, then not only are the technicians in danger, but our equipment in storage can also be destroyed. While we're on the topic of our storage facility, another problem we have is unauthorized personnel getting access to storage. We have a few select technicians in charge of the storage facility. They keep it clean, organized, and retrieve materials that any other technicians request. The problem, however, is that the storage technicians often leave the door open or unlocked when they go to retrieve materials for other technicians. In the past, we’ve had impatient lab technicians go into storage to grab what they needed rather than waiting for a storage technician to help them. This is obviously a big issue as these technicians are breaking the rules and are entering a restricted area. What’s even worse is that they will move around chemicals, which messes up the organization system of the storage technicians. We are concerned not only about reinforcing the storage facility, but also our records room. We have someone to oversee the records room, but they’re always in and out of there retrieving and storing documents for technicians. The door leading into where our documents are stored ends up open and unlocked most of the time. It is a big concern of ours that someone might try to steal information from our laboratory as we have many competitors in our business.

This only scratches the surface of the problems we have in our laboratory. Ideally we are looking for a monitoring system that: monitors the use of the burners and notifies us if they has been left on for long, monitors our storage facility for any leaks/flooding and notifies us where it is occuring, monitors our doors to the storage and records rooms and notifies us if it is open or unlocked. These specific issues are of highest priority to us and we would invest a great amount into anything that would help us solve them.

School/ College:

I am the team director for undergraduate campus housing. My job is to make sure the living conditions and dorms are capable of handling students living necessities. Students come from different places and have so many unique hobbies and traits so we want them to feel welcomed and at home on campus. In every dorm we have showers, bathrooms, heating, and cooling. We also have safety features such as fire alarms and carbon monoxide alarms. To increase the safety of our dorms we have security locks that only open with certain verifications. As the director for campus housing, my team has been working to make sure every student’s life is as smooth and comforting as possible.

As we continue to add more features and safety precautions in every building, we have come up with an adversity. With the added features, our staff has several applications that they have to go back and forth from. One application for the fire alarms, another for the security system, and others for every parts of the student dorms. Our staff wants to monitor all the activities in one screen. Not to mention the students will find it much more comforting if they can have access and knowledge about their own building. On top of this, we would want to know what in the building is malfunctioning, has been broken, or is in use.

There is a certain company that has everything we are looking for to solve all these problems. A company is selling their home automation system product. With this product we want to know all of the safety precautions in each building. If the fire alarm is on, the security system is not locking people out or locking people in. There is no plumbing errors. We want to know everything about our dorms in one screen. Also we would want the company to make a notification if something does go wrong what is it and how to attack the problem. Either calling the fireman, police, or a plumber.

Restaurant:

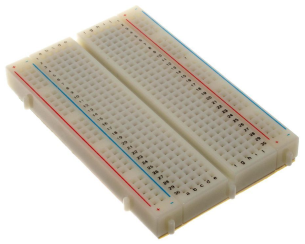
Hello Home Monitoring System team, I am a small restaurant owner who is interested in your product. As a small business owner I am always making decisions that I believe will benefit my business. I am also, however, the only one who keeps track of all the problems that happen around my restaurant. Some problems go unnoticed until it is too late. And as a small business owner, my main incentive is to increase profit while diminishing loss.

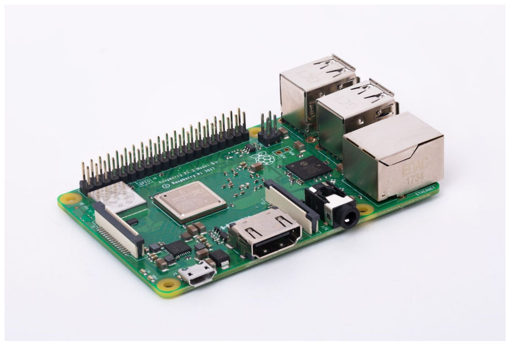
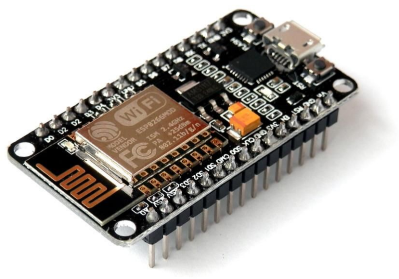
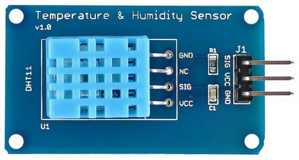
There are many problems I face daily and I believe your system will benefit me. One such problem is checking for water leakage. As an owner of a restaurant it is important to make sure that there is no water leakage near any of my food. Many times when we notice a water leakage in the freezer, it is already too late. This is because we only go into the freezer when the kitchen is out of a certain product, such as lettuce for example. By the time we discover a leakage in the fridge, our food products are already damaged. This leads to a great loss of money for us as well as a safety hazard for our customers. We want to use healthy ingredients in our products and also decrease food waste. We face another common problem during closing time. We always have to make sure that our stove is turned off. On busy nights, however, it is very easy to forget this simple task. I always find myself back at the restaurant after it is closed to ensure that everything is properly handled. This not only wastes my time, but causes a great deal of worry for me on a nightly basis. If I do not do this, this can easily lead to a fire and can greatly damage my restaurant.

Another great fear of every restaurant owner is the chance of a burglary. Every night there is a chance that someone could easily break into my restaurant and steal valuable items. With a system notifying me of any break ins, I would be able to quickly reach out and call the police department.

Having a system that can detect all of these issues and provide immediate feedback would be very nice to have. Instead of subscribing to multiple systems that only control individual aspects, I would be able to have one central system that monitors everything. Things such as home security and water leakage detection are not currently sold in one package, which means I have to purchase two different detection systems. With a centralized system like yours, I will be able to easily detect all types of problems in my restaurant without spending money or time on multiple products. I will get all my feedback in one device with a system that will notify me when there is a water leak, as well as if there is a break in. I will be able to fix numerous problems at once before significant damage is done. This will help me save money as well as keep my restaurant safe and secure.

Section 2: Glossary of Terms:

Application Programming Interface (API)	A set of functions and procedures allowing the creation of applications that access the features or data of an operating system, application, or other service
Arduino Uno Picture of Arduino Uno board	A microcontroller board equipped with sets of digital and analog I/O pins that may be interfaced to various expansion boards and and other circuits
Bread Board 	A board for making an experimental-model of an electric circuit
Circuit Board	A thin rigid board containing an electric circuit. A printed circuit.
Database	A collection of information that is organized so that it can be easily accessed, managed, and updated
Firebase Realtime Database	A cloud-hosted NoSQL database that lets you store and sync data between users in real time. This service is offered and managed by Google.

Internet of Things (IOT) 	A system of interrelated computing devices that transfer data over a network without requiring any human interaction
Mobile Application	A type of software application designed to run on a mobile device
Raspberry Pi  Picture of Raspberry Pi Model 3 B+	A low cost Linux and ARM-based computer on a small circuit board
nodeMCU 	A micro-controller unit capable of WiFi modularity and offers Analog and Digital GPIO pins
Sensor  Temperature & Humidity Sensor v1.0 DHT11 GND NC SIG VCC SIG VCC GND J1 U1 Picture of Arduino compatible temperature and humidity sensor.	A device to detect or measure a physical property and records, indicates, or responds to it
Server	A computer that serves information to other

	computers, called clients. These computers can connect to each other through a local area network (LAN) or a wide area network (WAN).
User Interface (UI)	The means by which the user and the computer system interact, in particular the use of input devices and software
WiFi Module	An independent self-contained unit capable of accessing WiFi network

Section 3: System Requirements:

We have extracted and analyzed the functional and nonfunctional system requirements from the Product Statements that our customers have given us. Using User Stories, we are able to make a clear and concise table to show exactly what our system aims to accomplish.

The most important task for any software to accomplish is to satisfy the requirements of the customer. Using what the customers have given us in the product statements, we were able to extract the major requirements for our specified system. These requirements however, are not in any order of importance to the customer. For us developers though, priorities in requirements matter. Hence, we need to prioritize all the requirements that were extracted and order them from highest to lowest priority in development. This will moreover help in the development of the software.

With prioritization in mind, we have created the table below. In the table there is a column for priority, which ranges from 1 - 5, with 1 being optional and 5 being absolutely mandatory.

Enumerated Functional Requirements:

ID	Priority	User Story
ST-1	5	As a user, I can check if the stove is on/off.
ST-2	5	As a user, I can check the status of my doors and windows if they are properly locked.
ST-3	5	As a user, I can check if there is a gas leak in my house.

ST-4	5	As a user, I can check whether or not my basement has flooded.
ST-5	1	As a user, I can look at reliable businesses near me to solve my problem.
ST-6	5	As a user, I will be alerted if a fire is detected in my home.
ST-7	1	As a user, I can turn off the monitoring system alerts.
ST-8	5	As a user, I can check to see if anyone has broken into my house.
ST-9	3	As a user, I can use this system on a cellular device.
ST-10	5	As a user, I can view a live camera feed of my home when anything is detected.

The priority of the functional system requirements has considerations of practicality and safety in them. Keeping these two principles in mind, we chose checking if the stove is on/off (ST-1), locked/unlocked doors and windows (ST-2), checking if the basement is flooded (ST-4), checking whenever anyone has entered the house (ST-9), and viewing a camera feed of dangerous/damaging occurrences (ST-6, ST-11) as the top priorities. Checking if the the doors are locked and if anyone has entered the user's home or business could also pose an important safety factor regarding potential thieves. We also included mobile integration for the system (ST-10) and being able to control the system remotely from any wifi destination (ST-8) as priorities three and four respectively.

The rest of the functional system requirements are identified as a lower priority, but are in no means unnecessary for the function of the system. Making sure the lights in the home or business are off (ST-3) may also be an important safety concern. Lastly, we marked off being able to turn off the monitoring system (ST-7) and looking at reliable businesses to solve the problem (ST-5). Being able to turn off the monitoring system allows for privacy and safety for users in case of any system failures that may result due to safety hazards. Additionally, the practicality of being able to identify reliable businesses should be a functional concern for our product.

Enumerated Nonfunctional Requirements:

ID	Priority	User Story
ST-11	2	As a user, I should understand how to use the system easily. The user interface should be easy to read and be usable for anyone.
ST-12	5	As a user, I should be able to check the status of my home at all times, so there cannot be any downtime for the application where my house is left

		vulnerable.
ST-13	4	As a user, I should be able to access the application from anywhere.
ST-14	4	As a user, I should be able to use this system on any Android device.
ST-15	3	As a user, I should be given quick and informative directions in case of an emergency.
ST-16	5	As a user, as soon as a problem is detected in my house, I should be alerted of it and be able to view a camera feed so I can address it as soon as possible.
ST-17	1	As a user, I should be able to contact the support team to troubleshoot any possible hardware/software errors.
ST-18	1	As a user, I should be able to easily install the sensors in my home.

As for the Non-Functional Requirements we broke them down to 4 categories: Usability, Reliability, Performance, and Supportability.

Usability: Usability is the ease of use and learnability of a device and object. A user should be able to access this application through (ST-14). Not only that but the application itself should be easy to learn and utilize.

Reliability: Reliability, or dependability, describes the ability of a system or component to function under stated conditions for a specified period of time. A user should be able to check the status of their home, restaurant, etc. whenever they want (ST-12). As such the user should also be able to access the application from any location as long as they have an Internet connection (ST-13).

Performance: Performance of a computer is essentially estimated in terms of efficiency, effectiveness and speed. If an emergency at a user's home occurs, the said user should be alerted immediately (ST-16). Furthermore, alongside the alert, the user should also be given quick and informative instructions (ST-15).

Supportability: Supportability refers to the ability of technical support personnel to install, configure, and monitor computer products. Technical support personnel should also be able to identify exceptions or faults, debug or isolate faults to root cause analysis, and provide hardware or software maintenance to solve a problem and restore the product into service. The user should also be able to install the sensors easily (ST-18). In case of any technical problems with the application or sensors, there should be a support team that they can contact (ST-17).

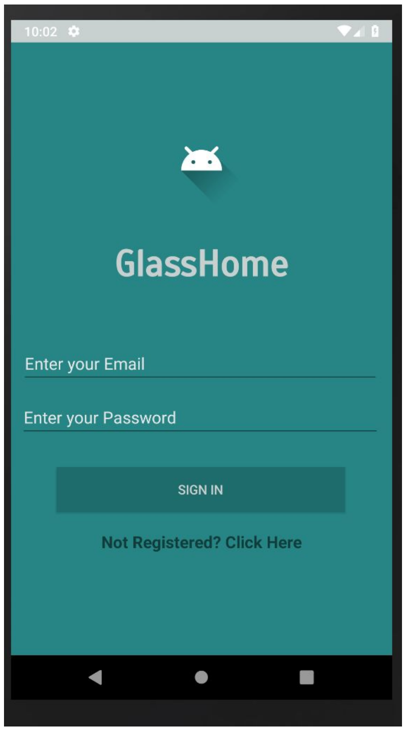
On-Screen Appearance Requirements:

ID	PW	Description
ST-19	5	The interface shall allow the user to input a username and password to securely log into their home server.
ST-20	5	The interface shall show users the status of different parts of the home by clicking different icons/buttons.
ST-21	5	Users should be able to click a button to turn on notifications that send alerts to their phones when something goes wrong.
ST-22	3	The interface shall allow a user to switch to an "Account" screen to view database information and the other users logged in.
ST-23	4	The interface shall allow a user to “Favorite” specific appliances to view when the application starts up.
ST-24	3	The interface shall allow user to switch to a "Support" screen to view local businesses using Google Maps.
ST-25	4	The user will be able to click a "Log Out" button to log out of their server.
ST-26	1	The interface shall allow a user to directly call a business from the support section when clicking a "Call" button.
ST-27	3	The interface shall allow a user to call emergency phone numbers.
ST-28	5	The application shall allow a user to view a live camera feed when motion is detected in the home.

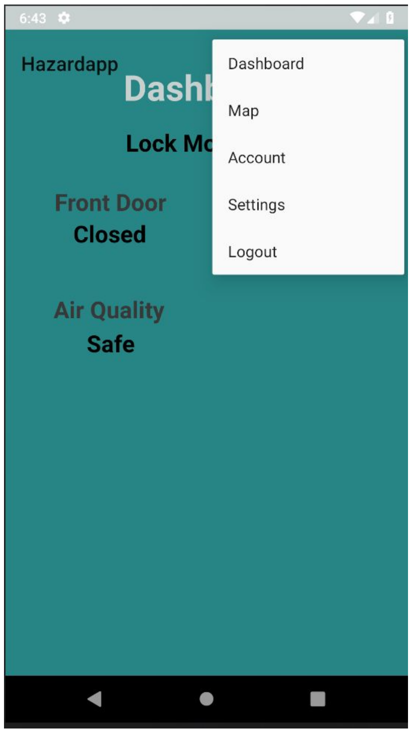
The priority of the On-Screen Appearance Requirements is to provide the best possible user experience. Because dependability and safety are crucial aspects of our system, it is important to provide an easy and quick user interface to navigate through the application.

To ensure that this application is safe and secure, a login screen (ST-19) will be the first graphic a
user will see on startup, similar to the one shown below. After this, the rest of the

application will be easy to navigate through the application using different tabs for each section, each showing important features ranging from ST-20 to ST-27, shown below. Each tab will take the user to the appropriate page in the application.



(ST-19)



(Navigating from ST-20 to ST-28)

Section 4: Functional Requirements Specification

Stakeholders:

1. **Internal Stakeholders**
    a. Developers : The people who have designed the system and decide how and what requirements to prioritize and fulfill.
2. **External Stakeholders**
    a. Home Owners: The customers who own the homes that have installed our product for security and protection. They value the reliability, function, cost performance, and quality of our system.
    b. Business Owners: The customers who own businesses like office workplaces that have our product installed for security and hazard prevention. They value the reliability, function, cost performance, and quality of our system.
    c. Restaurants: The customers who are the restaurant owners that installed our product into their workplace for everyday security protection. They value the reliability, function, cost performance, and quality of our system.
    d. Institutions: The customers who own an institution such as a school or college that have our system installed for security protection. They value the reliability, function, cost performance, and quality of our system.

Actors and Goals:

Actors	**Type**	**Goal**
Users	Initiating/Participating	To turn on/off the system, download, and use the application to check the status of all the appliances
Sensors	Initiating	To collect various signals and send it to the Firebase instance
Firebase Server	Initiating	To store data from sensors
Application	Initiating/Participating	To receive data from the Firebase instance and

		alert users about the status of their homes. To allow users to check the status of their homes.

Use Cases:

Casual Description:

Actor	Actor's Goal (What the actor intends to accomplish)	Use Case Name
User	To download the application on any Android Device	Download(UC-1**)**
User	To use the application with an Internet connection	Connect (UC-2)
User	To disarm the system	Disarm (UC-3)
Sensor	Send a signal when safety or security hazards are detected	Signal (UC-4)
App	To inform the user which support team to contact in case of an emergency	Contacts (UC -5)
App	To notify the user when there is a leak, stove is left on, and/or an intruder enters the home, company, etc.	Notify (UC-6)
User	To create new accounts to access the system	New Accounts (UC-7)
User	To add new sensors to the system	Add (UC-8)
User	To check the status of all the sensors of the system	Check (UC-9)
User	To control how often the notifications are sent	Control Alert (UC-10)
App	To notify the user if there is a network error or the signal of the network is weak	Network Error (UC-11)
User	To unlink user account from the phone application	Log Out (UC-12)

Use Case Diagram:

User Case 1: Download

Related Requirements: ST-9, ST-14
Initiating Actor:User
Actors Goal: To download the application from the Android store and be able to use it on any Android device
Participating Actor: Application
Precondition: User has an updated Android device with an internet connection
Postcondition: The application successfully downloads and connects to a server via Internet
Flow of Events for Main Success Scenario:
→ 1. User downloads the application from Android Store
→ 2. User makes an account on the application
← 3. User successfully connects

User Case 2: Connect

Related Requirements:ST-6, ST-12, ST-16
Initiating Actor: User
Actors Goal: To be able to access the application and receive notifications while away from home via cellular data or any WiFi.
Participating Actor: Server
Precondition: The user has cellular data enabled or is connected to some WiFi and has an account already made
Postcondition: The user will be able to check the status of the home and receive alerts
Flow of Events for Main Success Scenario:
← 1. The user connects to the Firebase Server
← 2. The Firebase Server sends data to the application via the Internet
→ 3. The application sends the data received by the Firebase server to the user as a notification

User Case 3: Disarm

Related Requirements: ST-7
Initiating Actor: User
Actors Goal: The ability to view various appliances in your home and safely shut down the

notification of the appliance through user's cellular device
Participating Actor: App
Precondition: The user, as well as the sensors, are connected to the application.
Postcondition: The user has deactivated certain sensors
Flow of Events for Main Success Scenario:
→ 1. The user opens the application and checks specific sensors
→ 2. User clicks on the sensor he/she wants to disable
← 3. The interface has a "disarm" option that the user clicks
→ 4. The user clicks "disarm" and receives a notification stating that the sensor is disabled

User Case 4: Signal

Related Requirements: ST-6, ST-12, ST-16
Initiating Actor: Sensor
Actors Goal: To send a notification to the user that informs them when there is a leak, when the stove is left on, and/or when an intruder enters the home, company, restaurant,etc.
Participating Actor: App, User, Server
Precondition: The sensors are continuously checking the status of the home
Postcondition: A signal is sent to the server, and the user receives a notification via Firebase Database.
Flow of Events for Main Success Scenario:
← 1. The sensors picks up a signal that there is a change in the system.
← 2. The signal is sent to the Firebase Server
← 3. The application receives the signal from the Firebase Server
← 4. The user is notified through the application

User Case 5: Contact

Related Requirements: ST-5, ST-15, ST-16, ST-17
Initiating Actor: App
Actors Goal: Once the app alerts the user about any possible issues, the app will provide the user with detailed steps on how to solve the issue.
Participating Actor: User
Precondition: A problem occurs and is detected by the sensor and the user is notified
Postcondition: The user is informed and will be able to act accordingly to solve the issue
Flow of Events for Main Success Scenario:
← 1. App notifies user on any possible issues(fire, water leak, gas leak, etc.)
→ 2. The notification is clicked and user is taken to the application
← 3. Depending on the problem, the application will display suggestions on steps to resolve the issue.

User Case 6: Notifications

Related Requirements: ST-5, ST-6, ST-8, ST-9, ST-10, ST-15, ST-16 **Initiating Actor:** Application **Actors Goal:** Notify the users about any signals that have been picked up by the sensors **Participating Actor:** User, Server, Sensor **Precondition:** The User is logged into the App on the Android device, Sensors are properly connected to the App, User is connected to any WiFi or data network and has notifications on **Postcondition:** Notification will have enough info for the User to be able to take appropriate action **Flow of Events for Main Success Scenario**: ← 1. Sensor picks up a signal ← 2. Signal is sent to the Firebase Server ← 3. Firebase Server sends this data to the App ← 4. App will send a push notification to the User

User Case 7: New Accounts
Related Requirements: ST-11, ST-13, ST-14 **Initiating Actor:** User **Actors Goal**: To create new accounts to access the system **Participating Actor:** Application **Precondition:** Having the app on an android phone **Postcondition:** Multiple accounts can log in to the application. **Flow of Events for Main Success Scenario:** ← 1. App displays the login interface. → 2. User selects ‘add a new account’ button. ← 3. User fills out information. ← 4. App sends a confirmation email. → 5. User confirms account and logs in to the app with the new account.

User Case 8: Add
Related Requirements: ST-18 **Initiating Actor:** User **Actors Goal:** To add sensors to the system. **Participating Actor:** Application, Sensors, System **Precondition:** Need an account and require the sensors to be properly installed physically beforehand. **Postcondition:** The sensor will be able to send alerts to the app. **Flow of Events for Main Success Scenario:** → 1. User logs in to the app. → 2. User selects to add a new sensor. ← 3. System looks for a new sensor signal. ← 4. Sensor connects to the system.

User Case 9: Check

Related Requirements: ST-1, ST-2, ST-3, ST-4, ST-8, ST-9, ST-10, ST-12, ST-13, ST-14
Initiating Actor: User
Actors Goal: To check the status of all the sensors in the system.
Participating Actor: App, Sensors, System
Precondition: User must be logged in the app and the sensors must be properly connected.
Postcondition: User will be shown the status of all the sensors through the app interface.
Flow of Events for Main Success Scenario:
→ 1. User logs into the app.
→ 2. User goes into the dashboard
← 3. The status of all appliances at home are displayed

User Case 10: Control Alert

Related Requirements: ST-6, ST-11, ST-13, ST-14
Initiating Actor: User
Actors Goal: To have control over the notifications that are sent and to control their regularity.
Participating Actor: Application
Precondition: App on Android phone, Connected system with other sensors in place
Postcondition: Full adaptive control of the notifications sent to the User
Flow of Events for Main Success Scenario:
← 1. User goes to Settings and clicks on the notifications tab
→ 2. User can view each sensor and their notification settings
→ 3. User can toggle on/off notifications for specific sensors that they do not want to view

User Case 11: Network Error

Related Requirements: ST-6, ST-12, ST-16, ST-17
Initiating Actor: App
Actors Goal: To be notified in case a network error or weak signal may occur.
Participating Actor: User
Precondition: App is downloaded and set up on Android phone, Notification system is properly set up
Postcondition: User is notified of a network error, and can take the appropriate steps to resolve the network connection
Flow of Events for Main Success Scenario:
→ 1. The signal weakens or disconnects
→ 2. The application detects a network error
→ 3. App sends notification to the User

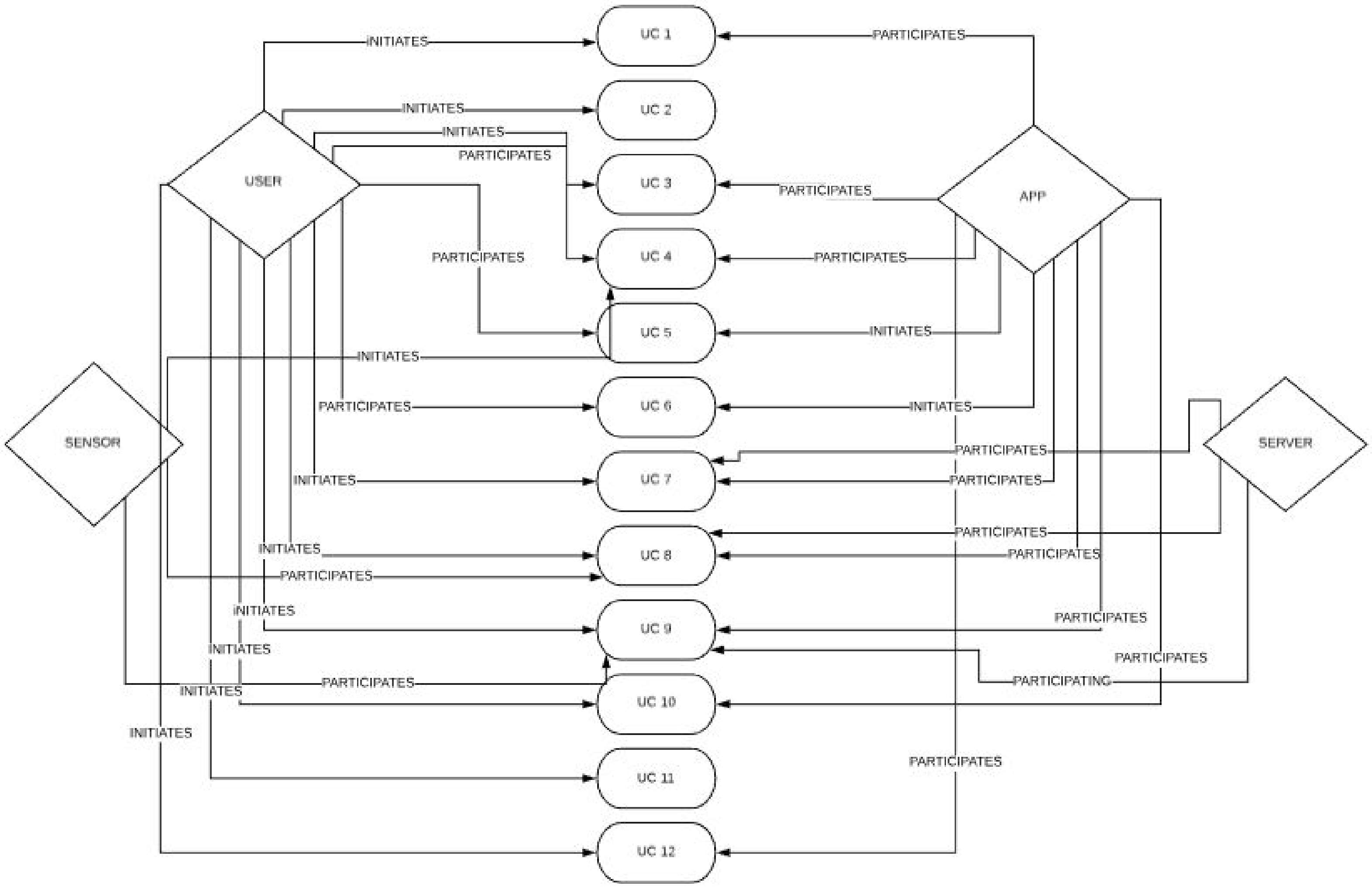
User Case 12: Log Out
Related Requirements: ST-11 **Initiating Actor**: User **Actors Goal**: To remove the user from the application **Participating Actor:** Application **Precondition:** The user downloads the software and is able to successfully log into the system **Postcondition:** The user is now logged out of the system and is unable to see his/her account details **Flow of Events for Main Success Scenario**: → 1. User goes into the application settings → 2. User presses the “Logout” button ← 3. App unlinks the user account from itself and returns to the “User Registration” screen

<u>Traceability Matrix:</u>

Req	**PW**	**UC 1**	**UC 2**	**UC 3**	**UC 4**	**UC 5**	**UC 6**	**UC 7**	**UC 8**	**UC 9**	**UC 10**	**UC 11**	**UC 12**
ST1	**5**									**X**			
ST2	**5**									**X**			
ST3	**5**									**X**			
ST4	**5**									**X**			
ST5	**1**					**X**	**X**						
ST6	**5**		**X**		**X**		**X**				**X**	**X**	
ST7	**1**			**X**									
ST8	**4**						**X**			**X**			
ST9	**5**	**X**					**X**			**X**			
ST 10	**3**						**X**			**X**			
ST 11	**2**							**X**			**X**		**X**
ST	**5**		**X**		**X**					**X**		**X**	

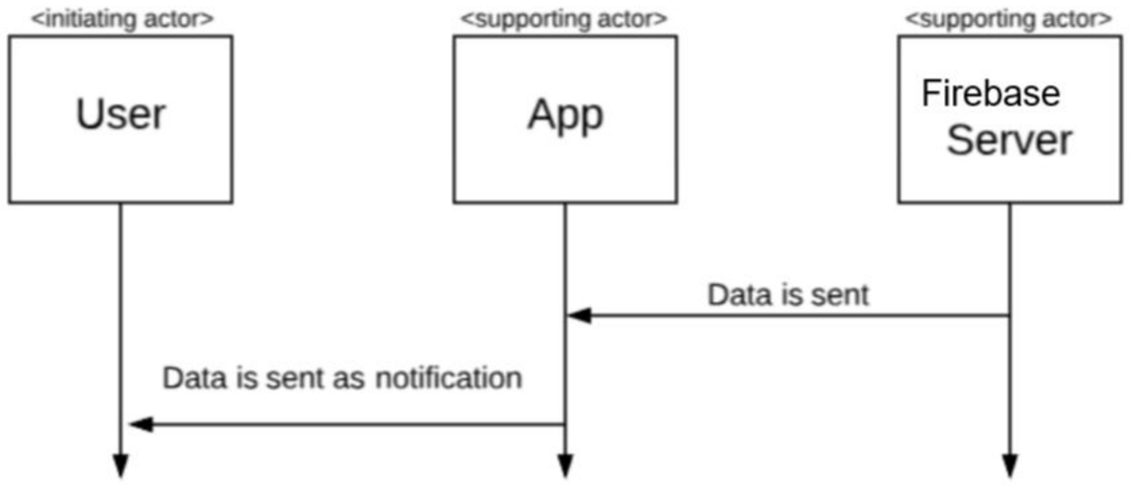
12													
ST 13	**4**							**X**		**X**	**X**		
ST14	**4**	**X**						**X**		**X**	**X**		
ST 15	**5**					**X**	**X**						
ST 16	**5**		**X**		**X**	**X**	**X**					**X**	
ST 17	**1**					**X**						**X**	
ST 18	**1**								**X**				
ST 19													
ST 20	**1**												
ST 21	**1**												
ST 24	**3**												
ST 25	**1**												**X**
ST 26	**5**												
ST 27	**2**												
Max PW		**5**	**5**	**1**	**5**	**5**	**5**	**4**	**1**	**5**	**5**	**5**	**1**
Total PW		**9**	**15**	**1**	**15**	**12**	**27**	**10**	**1**	**42**	**15**	**16**	**3**

<u>Fully-Dressed Description:</u>

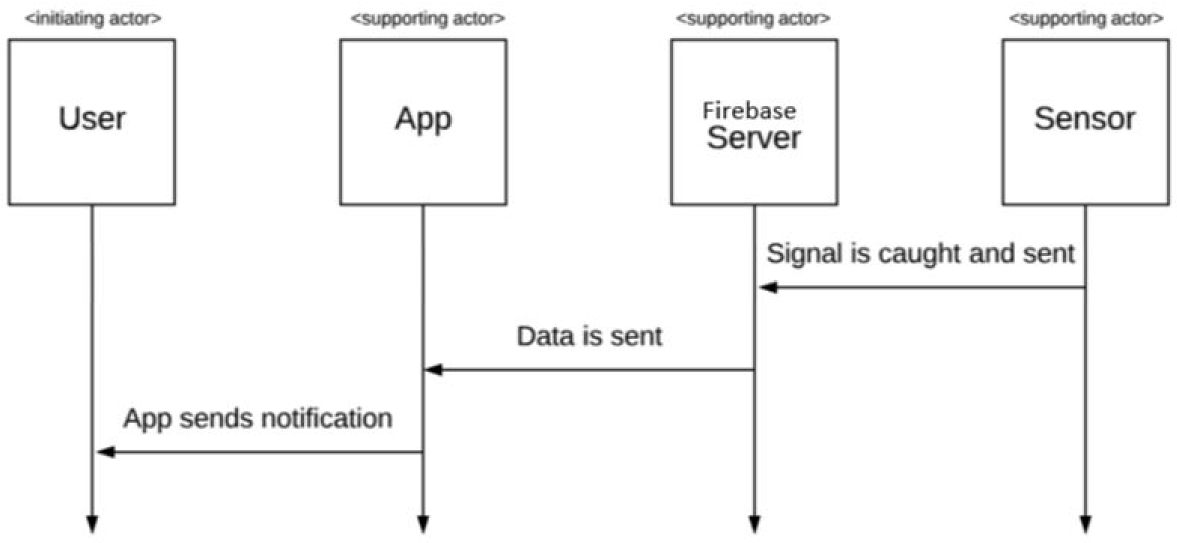


System Sequence Diagrams:

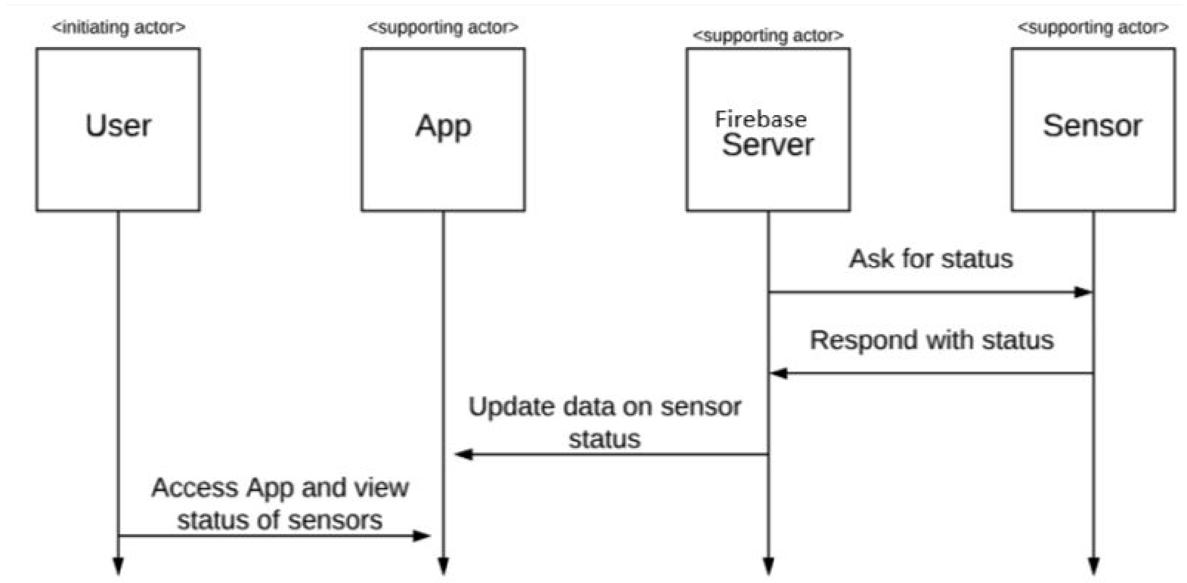
Connect sequence diagram UC-2:



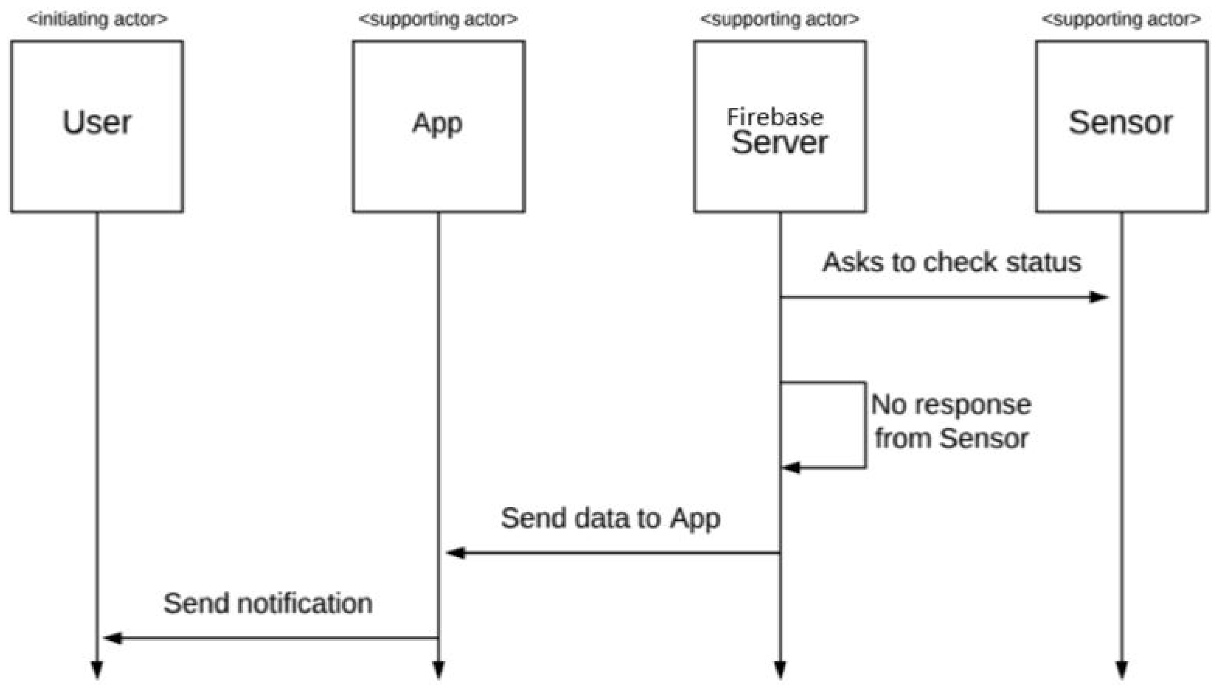
Signal sequence diagram UC-4:



Check sequence diagram UC-9:



Network Error sequence diagram UC-11:



<u>Section 5: User Effort Estimation:</u>

- Logging into account
    1. Navigation: 1 button tap, assuming application is already open, the “User Registration” screen will be the first thing the user sees.
        a. Tap “Already Registered? Sign in here.” button
    2. Data Entry: 3 button taps in addition to the button taps corresponding to the length of the email and password.
        a. Tap the “Enter your Email” text field
        b. Enter your email address
        c. Tap “Enter your Password” text field
        d. Enter your password
        e. Tap the “Login” button
- Creating a new account
    1. Navigation: 0 button taps, assuming application is already open, the “User Registration” screen will be the first thing the user sees.
    2. Data Entry: 3 button taps in addition to the button taps corresponding to the length of the email and password.
        a. Tap the “Enter your Email” text field
        b. Enter your email address
        c. Tap “Enter your Password” text field
        d. Enter your password
        e. Tap the “Register User” button

- Connecting/Syncing to the network
    1. Navigation: 1 button tap, assuming the “Your connected device” screen appears directly after logging in or registering.
        a. Tap “Network Settings”
    2. Action: 1 button tap, assuming the network being connected to is the same one the sensors are connected to.
        a. Tap “SYNC NOW”
- Disconnecting from the network
    1. Navigation: 1 button tap, assuming “Your connected device screen appears directly after logging in or registering.
        a. Tap “Network Settings”
    2. Action: 1 button tap
        a. Tap “DISCONNECT”
- Adding a new sensor
    1. Navigation: 0 button taps, assuming application is already open and the user is already logged in, the “Sensor List” screen will be the first thing the user sees.
    2. Action: 2 button taps to finish registering a new sensor.
        a. Tap the “+” button
        b. Tap desired sensor from list
- Disarming Appliance
    1. Navigation(1): 2 button taps, assuming application is already on “Sensor List” screen when the application opened and user logged in or registered
        a. Tap sensor that is associated with a specific appliance
    2. Action(1): 1 button tap, assuming the appliance is currently on
        a. Tap the “Lock mode” button (shuts down appliance)
    3. Navigation(2): 0 button taps, assuming a notification appears letting the user know an appliance is still on
    4. Action (2): 1 button tap
        a. Tap the “Lock mode” button (shuts down appliance notifications)
- Checking status of a registered sensor
    1. Navigation: 0 button taps, assuming application is already open and the user is already logged in, the “Sensor List” screen will be the first thing the user sees.
    2. Action: 1 button tap to open the “Details” screen of the desired sensor.
        a. Tap desired sensor from list
- Modifying the notification settings
    1. Navigation: 2 button taps, assuming application is already open and the user is already logged in, the “Sensor List” screen will be the first thing the user sees.
        a. Tap the “Settings” button
        b. Tap the “Notifications” button

    2. Action: 1 button tap to modify notification settings of a specific sensor.
        a. Tap to toggle notifications for any given sensor
- Logging out of account
    1. Navigation: 1 button tap, assuming application is already open and the user is already logged in, the “Sensor List” screen will be the first thing the user sees.
        a. Tap the “Settings” button
    2. Action: 1 button tap to disconnect user account from application.
        a. Tap the “Logout” button

Section 6: Domain Analysis:

Domain Model:

Concept Definitions

Responsibility Description	Type	Concept Name
RS1: Interacts with all of the subsystems of the home security system	D	Controller
RS2: Verifies the credentials of the User in order to get the User successfully logged into the app	D	IDChecker
RS3: User interacts with this and enters a UserID and Password in order to be logged in	K	IDEntry
RS4: User opens the main menu of the Status app to check each sensor status	K	StatusDisplay
RS5: Responsible for acquiring the status of the sensors by speaking to the sensors directly	D	SensorOperator
RS6: Obtains the status of each of the sensors in order to efficiently display the status to the User	D	StatusDisplayList
RS7: Alerts the user of a potential problem or security issue	K	Alert

received from the sensor		
RS8: Notifies the user of a home hazard such as a gas leak, a water leak, etc.	K	Notification
RS9: Connects the sensor controller to the application controller	D	Firebase

Association Definitions

Concept Pair	Association Name	Association Description
Controller ↔ IDChecker	Verifies identity	Verifies User credentials when logging into the app
IDChecker ↔ Controller	Passes check	Passes the success or failure back to the controller of identity verification
Controller ↔ StatusDisplayList	Obtain/Refresh from query	Obtain the status display list in order to efficiently relay the information to the user
StatusDisplayList ↔ SensorOperator	List query	Lists all the sensor status types
Firebase ↔ Controller	Connects	Firebase server connects the sensor controller to the application controller
SensorOperator	Request from sensor	Requests and obtains the status of the sensors from the sensors themselves

Attribute Definitions

Concept Name	Attribute Name	Attribute Description
Controller	numOfAttempts	Used to determine the number of attempts the user inputted as identity verification
Controller	listUpdate	Updates the lists on the

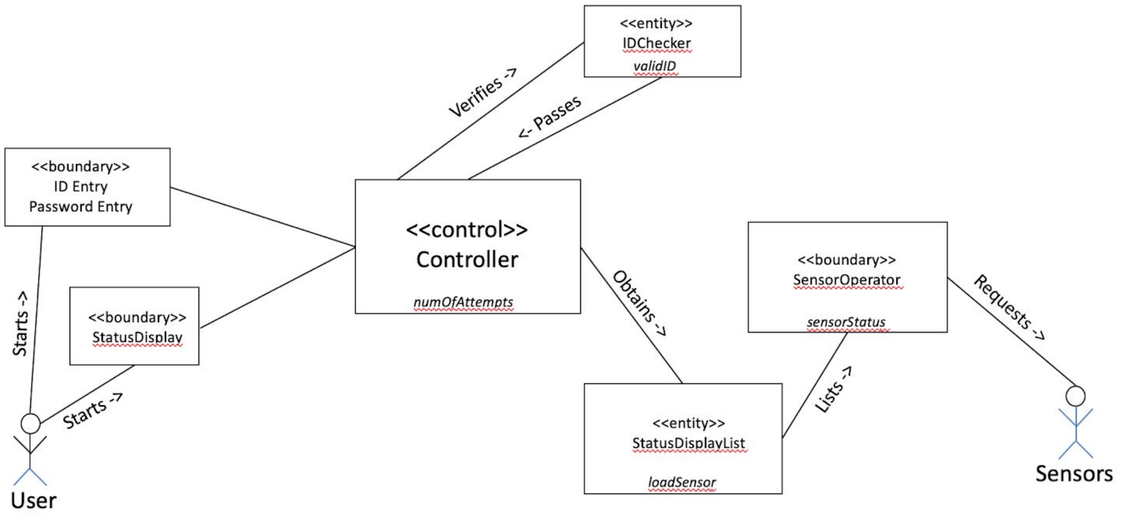
		sensor statuses
Controller	listRefresh	Periodically refreshes the sensor list with new statuses
Controller	receiveStatus	Used to receive an incoming status notification
Controller	sendStatus	Used to send the notification to the User
Controller	isAlert	Determines if the notification is an emergency alert
Controller	sensorCheck	Check the sensors for alers or notifications
IDChecker	validID	Matches the ID and password with an ID - password combination already present in the database
IDChecker	sensorRequest	User request to check status of sensors
Notification	sendNotify	Pushes notification to the User
App	displayStatus	Checks to see if the display is accurately showing sensor statuses
App	isNotify	Determines if the application needs to send out a notification or alert
App	alertStatus	Further determines the type of alert to best help the User
StatusDisplayList	loadSensor	Responsible for putting the sensor statuses acquired into list form
SensorOperator	sensorStatus/sensorConnection	Distinguishing sensor statuses as acquired by the

		sensor

Login and Check:

The login and check domain model of the Home Security and Safety System is shown as follows. The user is responsible for initiating one of two possible interactions: logging into the Status app, or opening the main menu of the Status app to check the sensor status. This then goes through a common controller to check the number of attempts that the user tried to login to the app with. It then verifies the credentials using an IDChecker entity and if it passes, it obtains the Status Display List on the dashboard of the Home Security application.

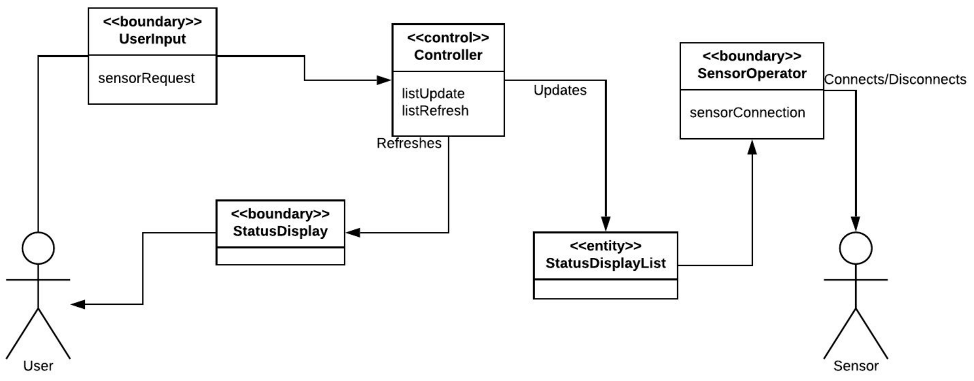
Periodically, the list much refresh in order to display an accurate reading of the sensor status. This is done by allowing the sensor operator to request an accurate reading from the sensors themselves. The attributes pertaining to these concepts are as follows: the IDchecker contains a validID parameter to check a credential match; the controller contains a numOfAttempts attribute to make sure the user does not exceed a certain number of ID-password attempts; the Status Display List contains a loadSensor attribute in order to efficiently load existing sensors; and the Sensor Operator contains a sensorStatus attribute to correctly obtain a sensor status for the application to list.



Add/Remove Sensor:

Assuming the user is logged into their account, this is how the domain model for modifying the sensor list works: First the user will indicate which sensor they want to remove/add. The Controller then updates the StausDisplayList by removing/adding the particular sensor. Next, the SensorOperator disconnects from/connects to the desired sensor. Finally, the Controller refreshes the StatusDisplay using the new StatusDisplayList.

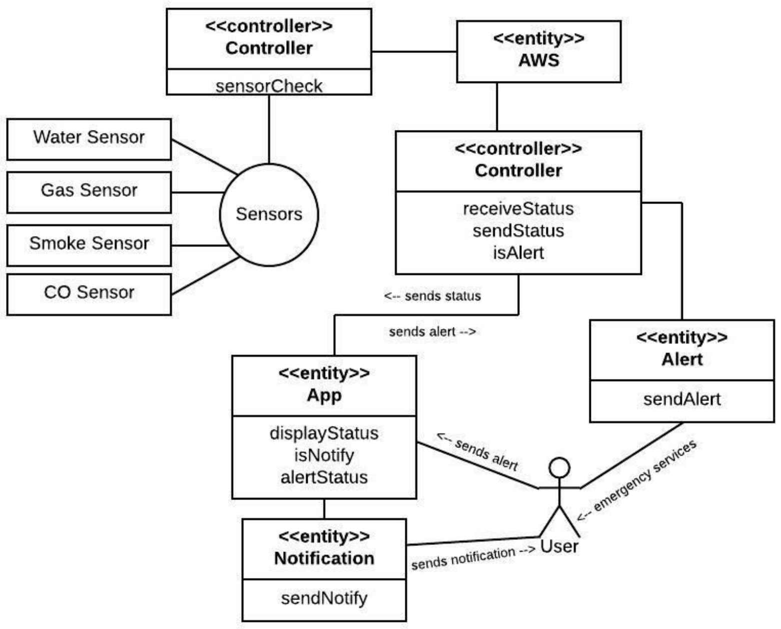
Regarding the attributes: the UserInput will have a sensorRequest attribute which indicates to the Controller what sensor they want to add or remove; the Controller will have a listUpdate attribute which will modify the StatusDisplayList when the user adds or removes a sensor, as well as a listRefresh attribute which will refresh the StatusDisplay after using the listUpdate attribute; the SensorOperator will have a sensorConnection attribute which will establish or destroy a connection to a particular sensor.



System Emergency Alert/Notification:

Assuming the user is logged into their account and connected to the various sensors through the network, the below diagram represents the domain model for emergency alert/notification of the system.

The four (4) sensors (water, gas, smoke, carbon monoxide) are all constantly checking for their respective issues. This data is then linked into Firebase which will be accessible through the applications controller. The controller manages the data send to and received from the user-facing application along with the alert system. The controller will send the status of the sensor to application which in turn determines if the user needs to be notified or not. If so, the application sends the user a notification. When a user receives a notification, they have the option of telling the application to alert emergency services (police, fire dept., etc). From there, the application will send this alert to the controller. The controller then triggers the "isAlert" function which contacts the appropriate service. Emergency services can then take action from there in helping the user.



System Operation Contracts:

Name:	Connect
Responsibilities:	This operation must connect the user to the Firebase server through the app.
Notes:	UC-2
Output:	A link is established between the App and the Firebase Server.
Pre-Conditions:	App already downloaded and installed on the Android device. Account already set-up.
Post-Conditions:	App retrieves the data from the Firebase server

Name:	Signal

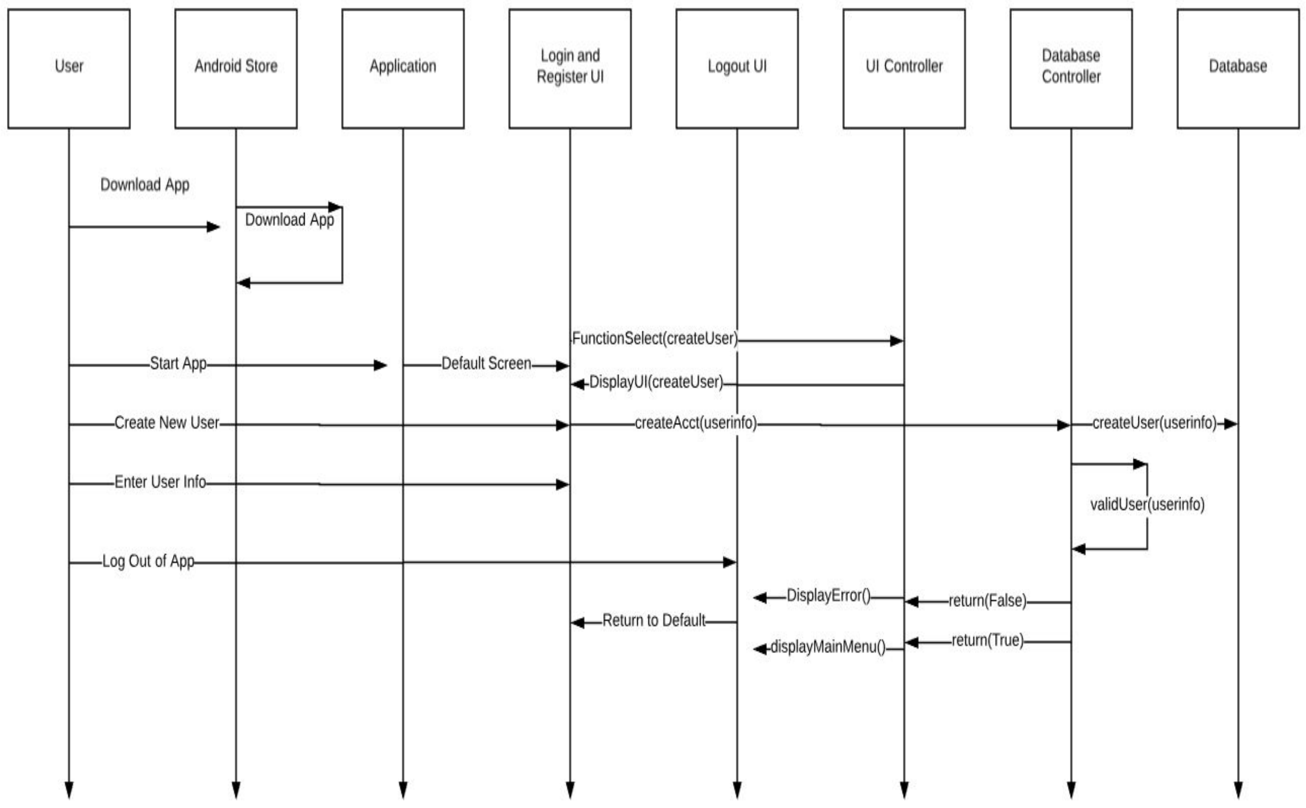
Responsibilities:	This operation must send a signal from the sensor to the user by using the Firebase server and the app.
Notes:	UC-4
Output:	The sensor is linked to the app by the Firebase. This link allows the transfer of data.
Pre-Conditions:	A sensor is installed and connected to the Firebase server
Post-Conditions:	Signal is captured and send from the sensor to the server

Name:	Check
Responsibilities:	Must update the current status of the sensors and display on the App for User to view.
Notes:	UC - 9
Output:	Firebase Server requests status update from the Sensor, and the Sensor will respond with an updated status.
Pre-Conditions:	Firebase Server and Sensor are already connected to each other.
Post-Conditions:	Firebase Server will share the updated status.

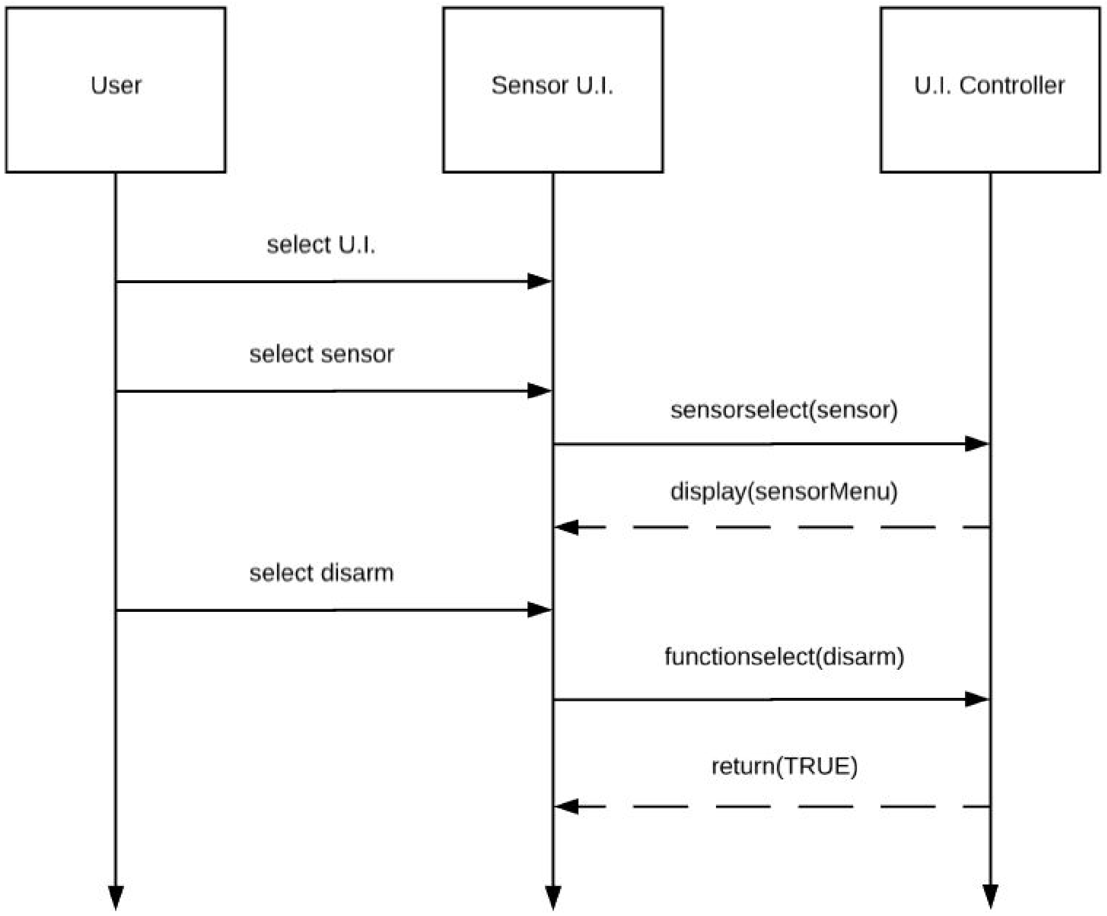
Name:	Network Error
Responsibilities:	Must update the user if the sensors are down due to network error/ outage.
Notes:	UC - 11
Output:	Firebase Server requests status update from the Sensor. After interval, no response recorded. Firebase Server describes this occurrence as network error.
Pre-Conditions:	Firebase Server and Sensor are already connected to each other.
Post-Conditions:	Firebase Server will keep checking with the Sensor during intervals for an updated status.

<u>Section 7: Interaction Diagrams:</u>

<u>User Case 1- Download:</u>

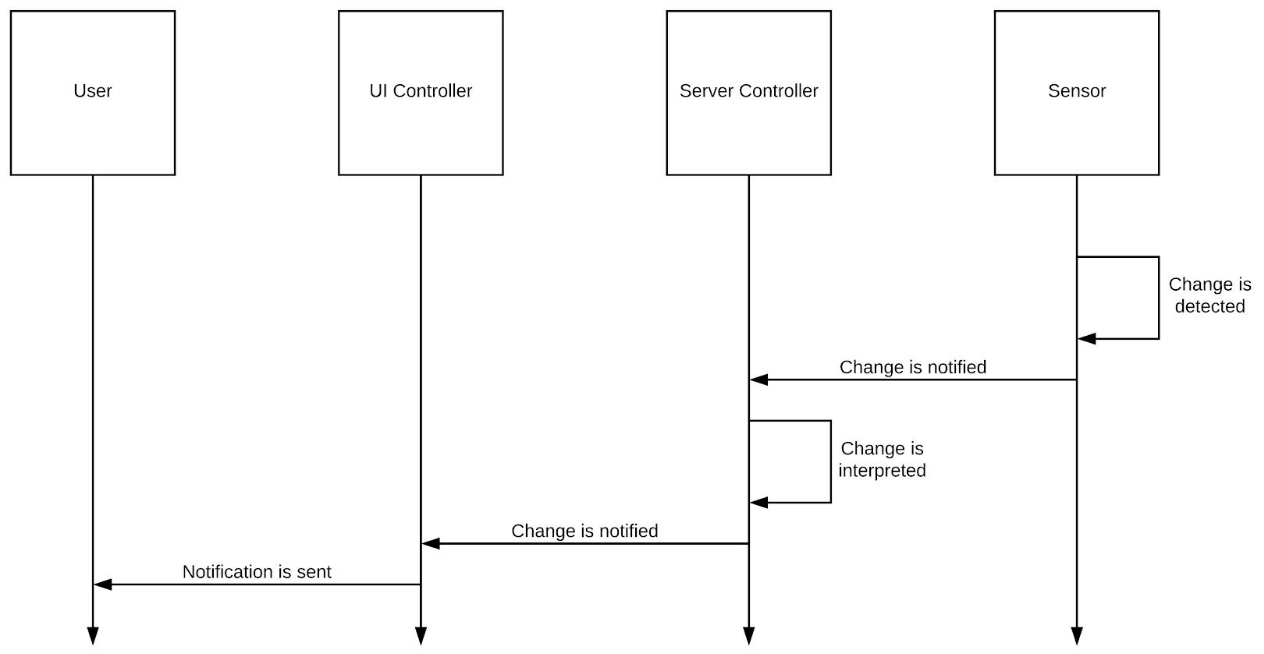


<u>User Case 3 - Disarm:</u>



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.

<u>User Case 4 - Signal:</u>

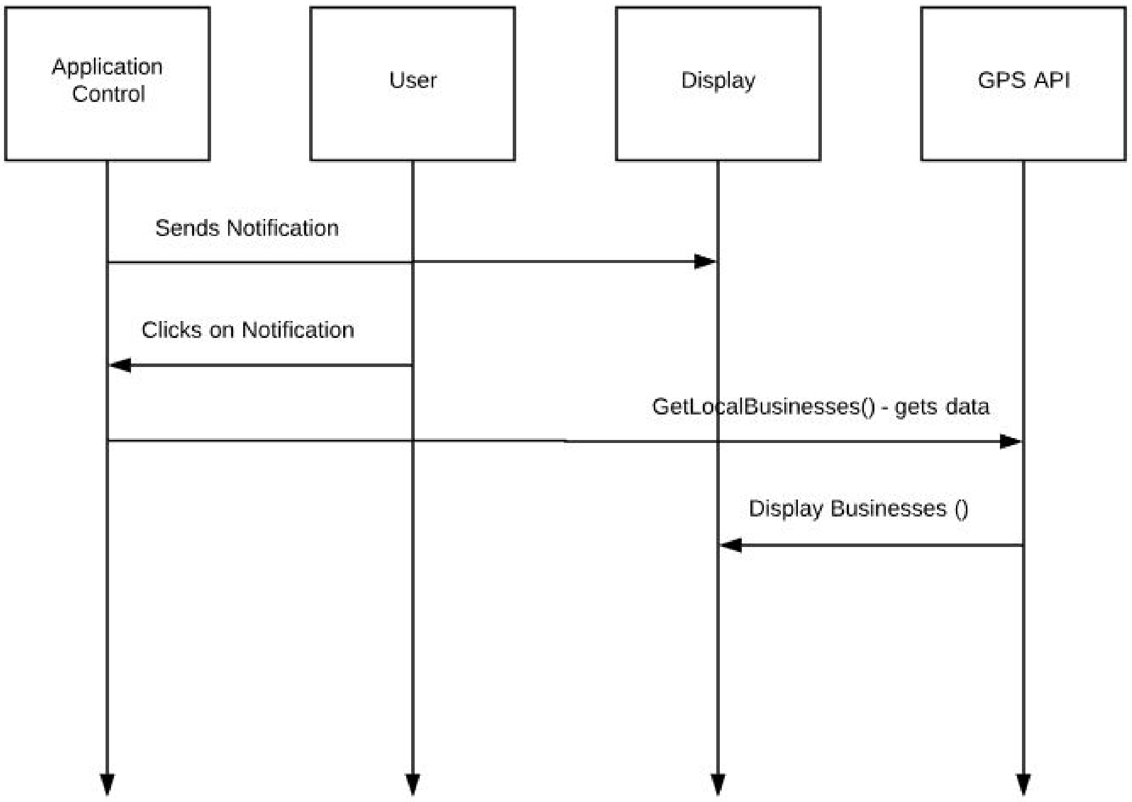


This user case allows the application to push a notification to the user once the sensor detects a signal (or a change).

The sensors will always be on, and periodically loop through the code given to check for signals in the household. These sensors are directly linked to the firebase database through WiFi. There are variables set up on the database that are linked to the same exact variables on the sensors. When the sensors notice a change or a signal, they will update the variable in the firebase to the correct value. For example if the door is opened, they will update "doorOpenStatus" to "true" which means that the door is opened. This will then be relayed to the application where the code in the app will recognize this change and send a notification to the user.

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 inpreted 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.

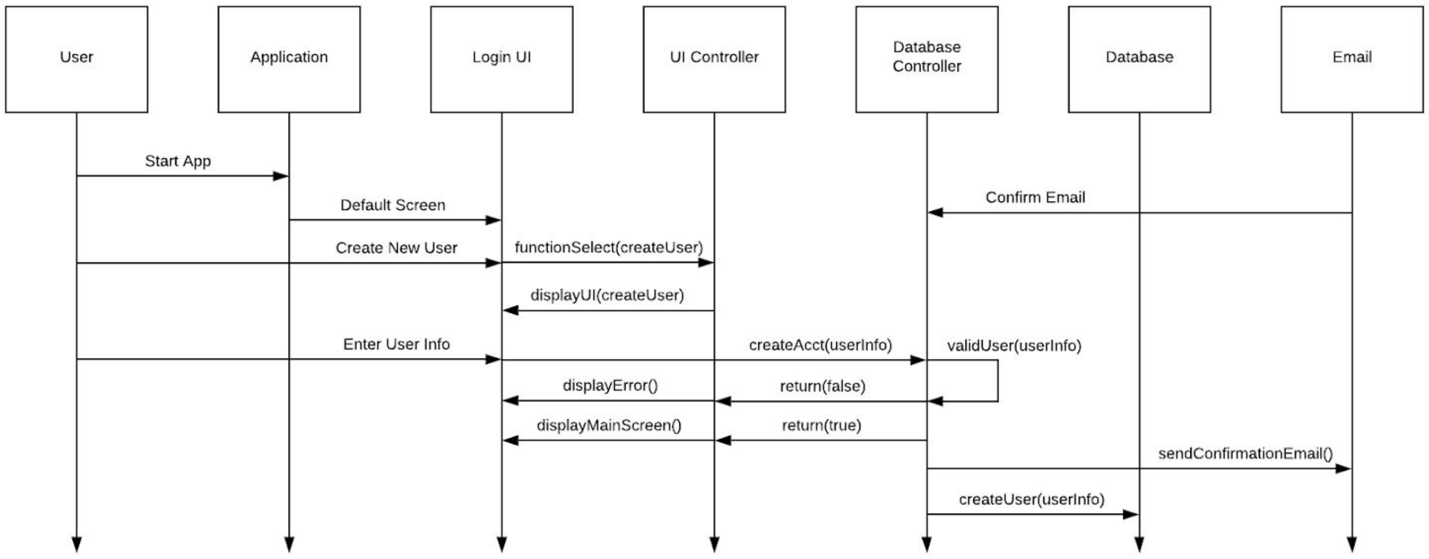
User Case 5 - Connect:



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.

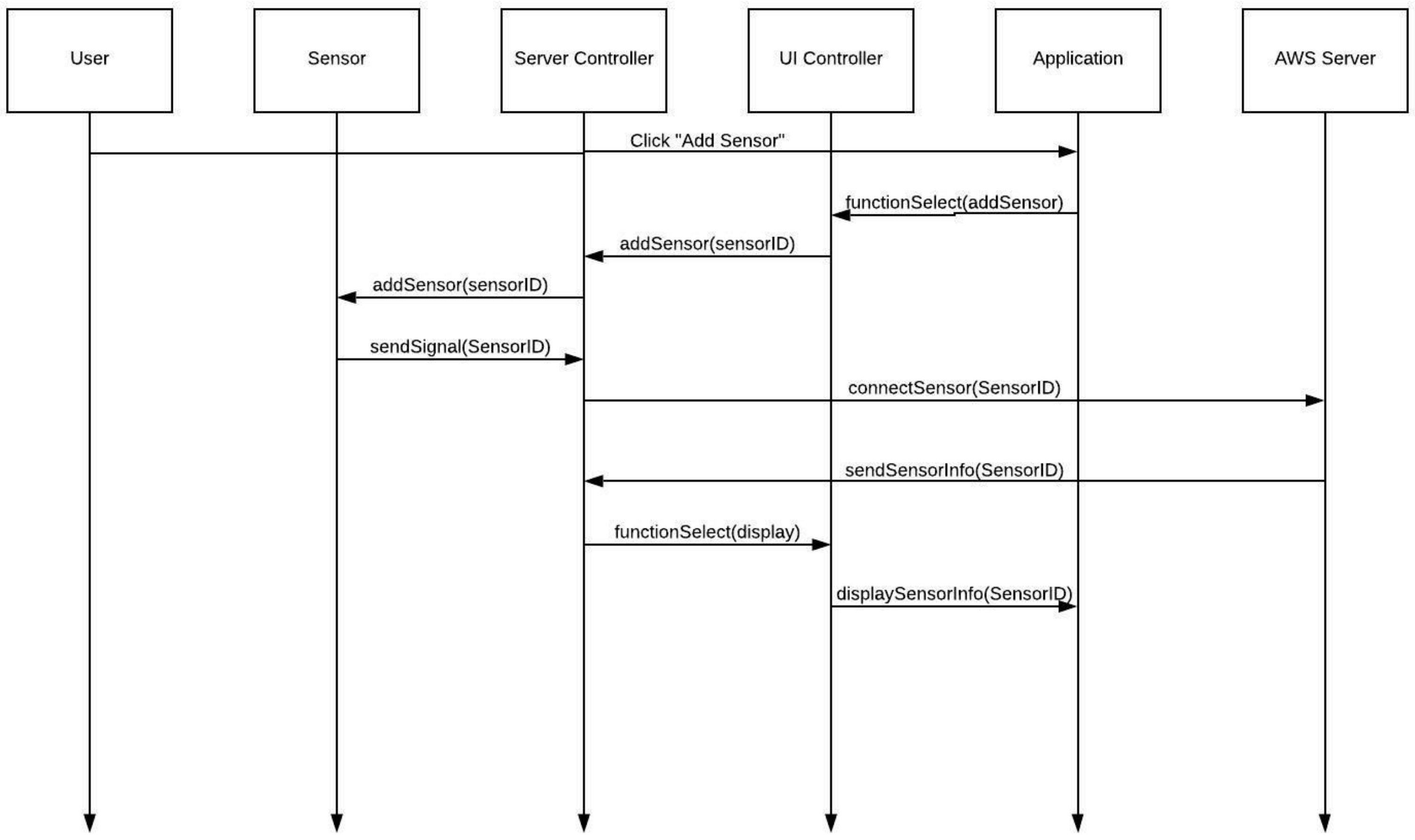
User Case 7 - New Accounts:

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.



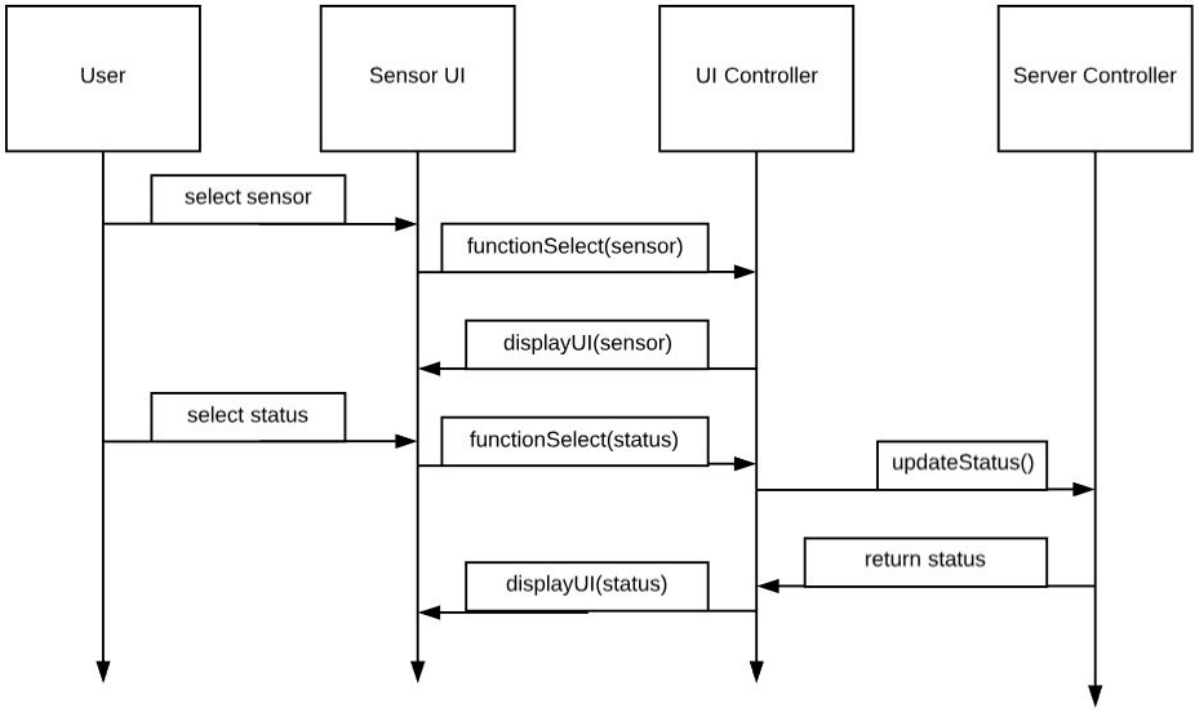
The user enters all the necessary information which the Login UI relays to the UI Controller. The information is brought to the Database Controller which makes sure the information provided by the user is valid. If it isn’t, the Database Controller returns a false boolean which will notify the user that 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 Firebase Server is needed to send and receive important Sensor data. All of the operations between the User Interface, Sensor, and Firebase 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.

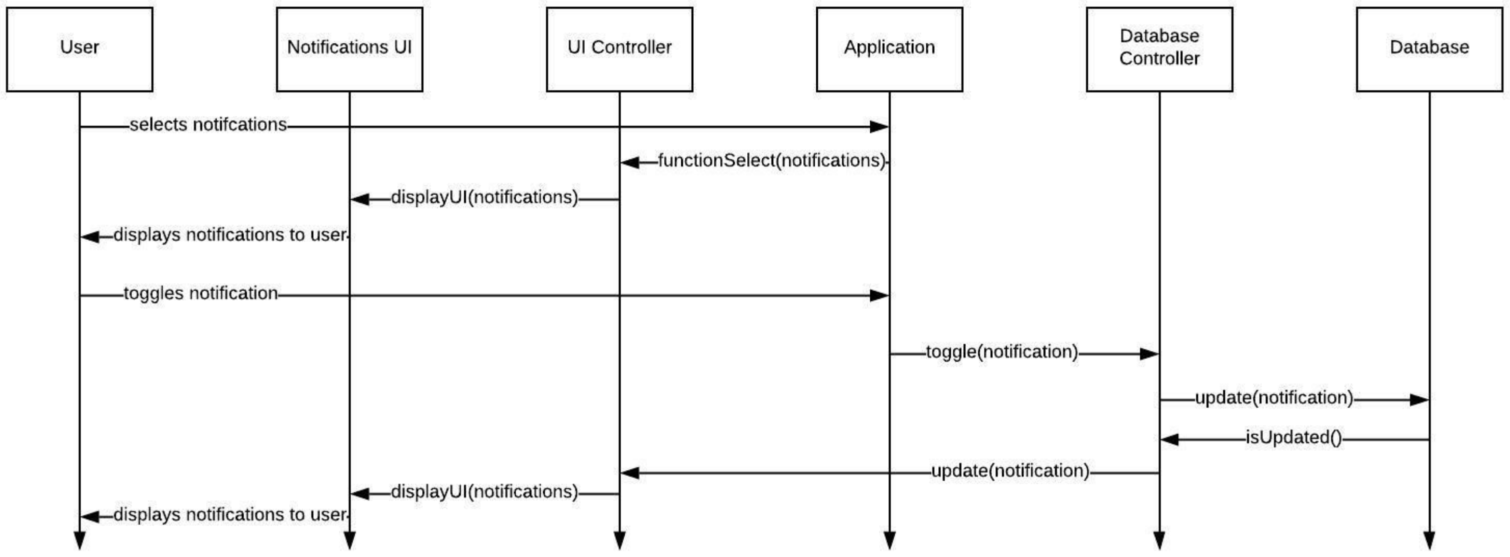
User Case 9 - Check:



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.

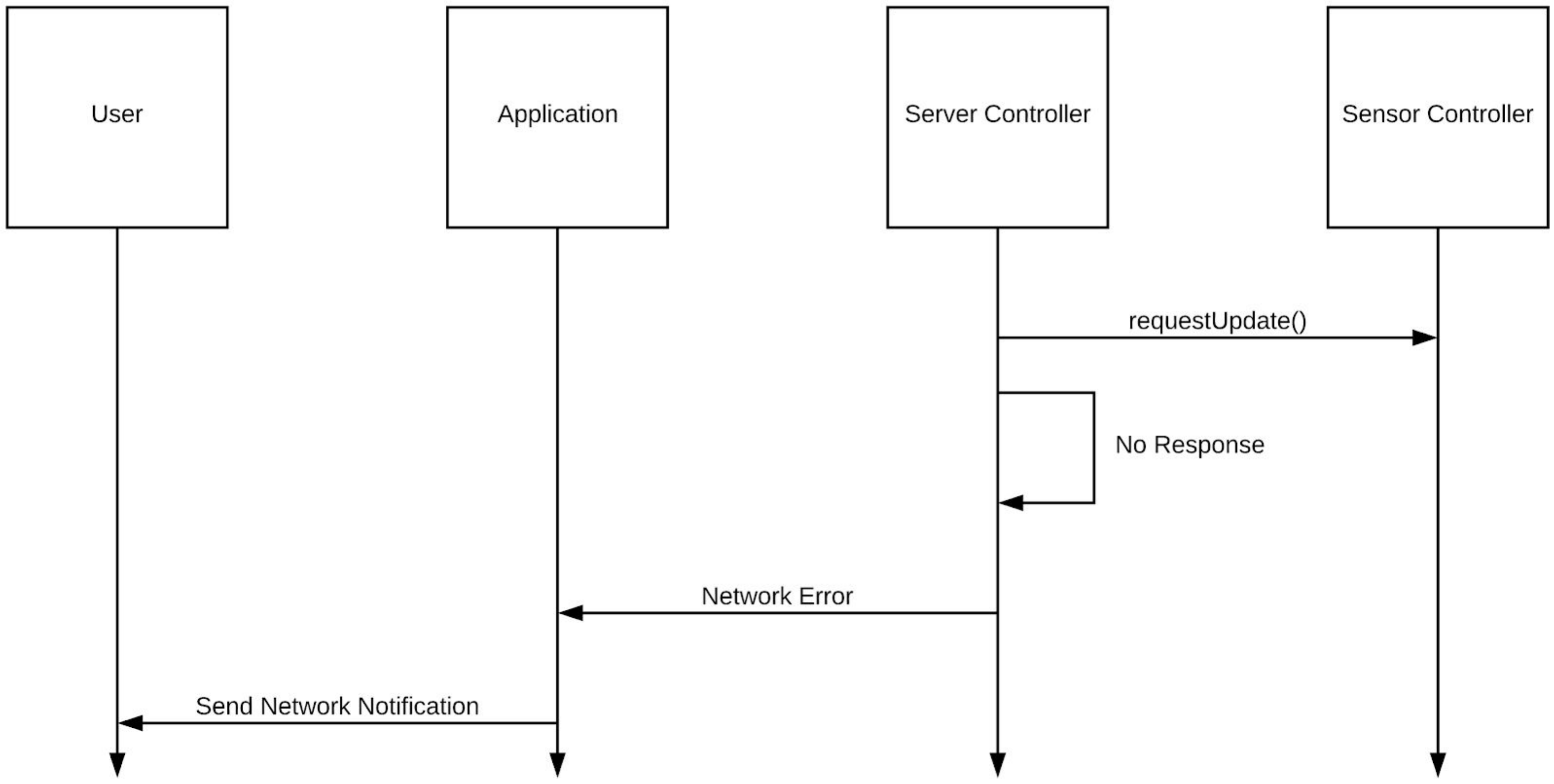
While creating the responsibilities, there could've been another UI for the settings. However, it was more efficient to included it in the sensor UI via the drop down menu.

User Case 10 - Control Alert:



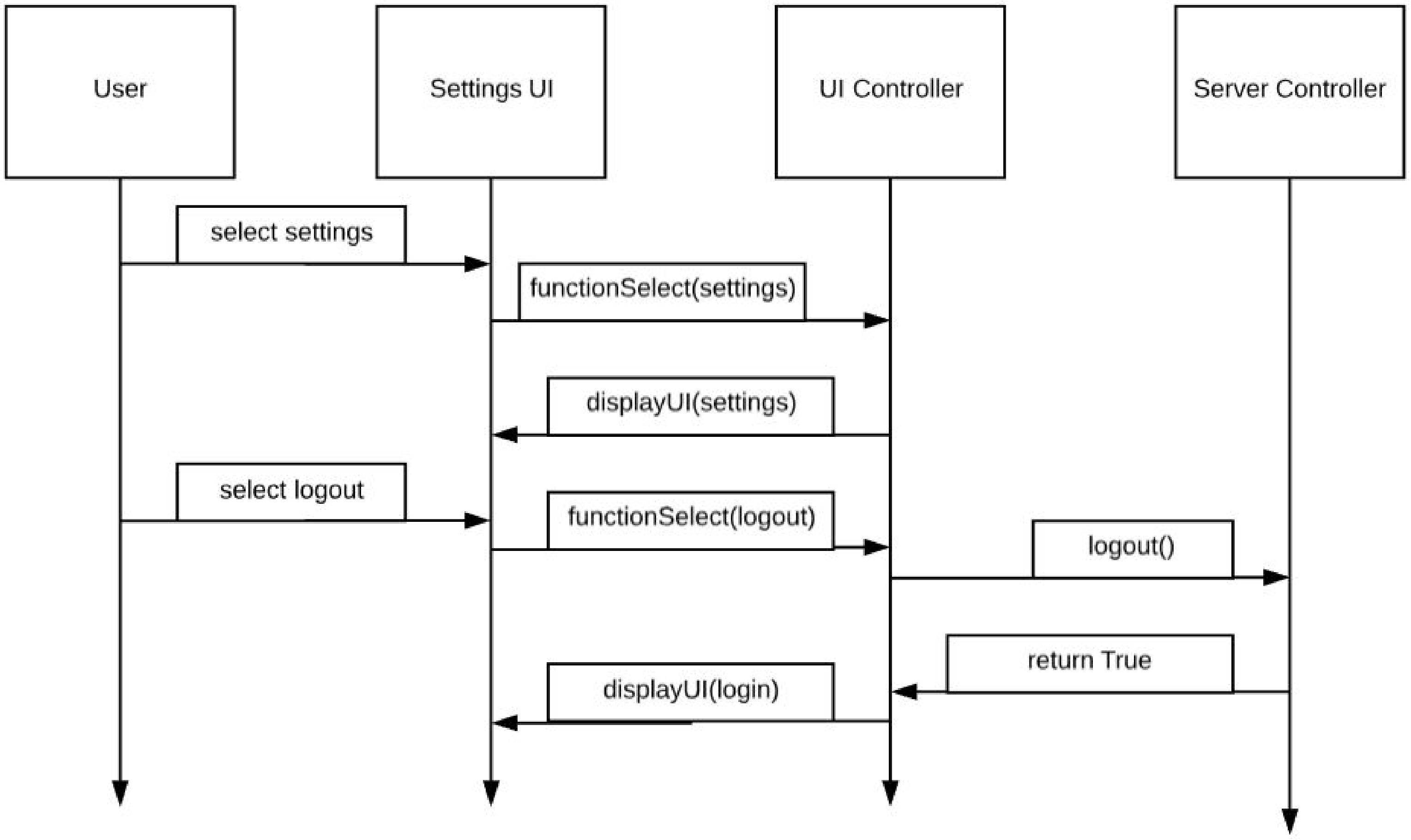
To update notifications setting, the user will first select “notifications” on the application. The application will this request to the UI Controller in which the UI Controller displays the current notification settings to the user (through the Notifications UI). Now the the user has the notification 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).

<u>User Case 11 - Network Error:</u>



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.

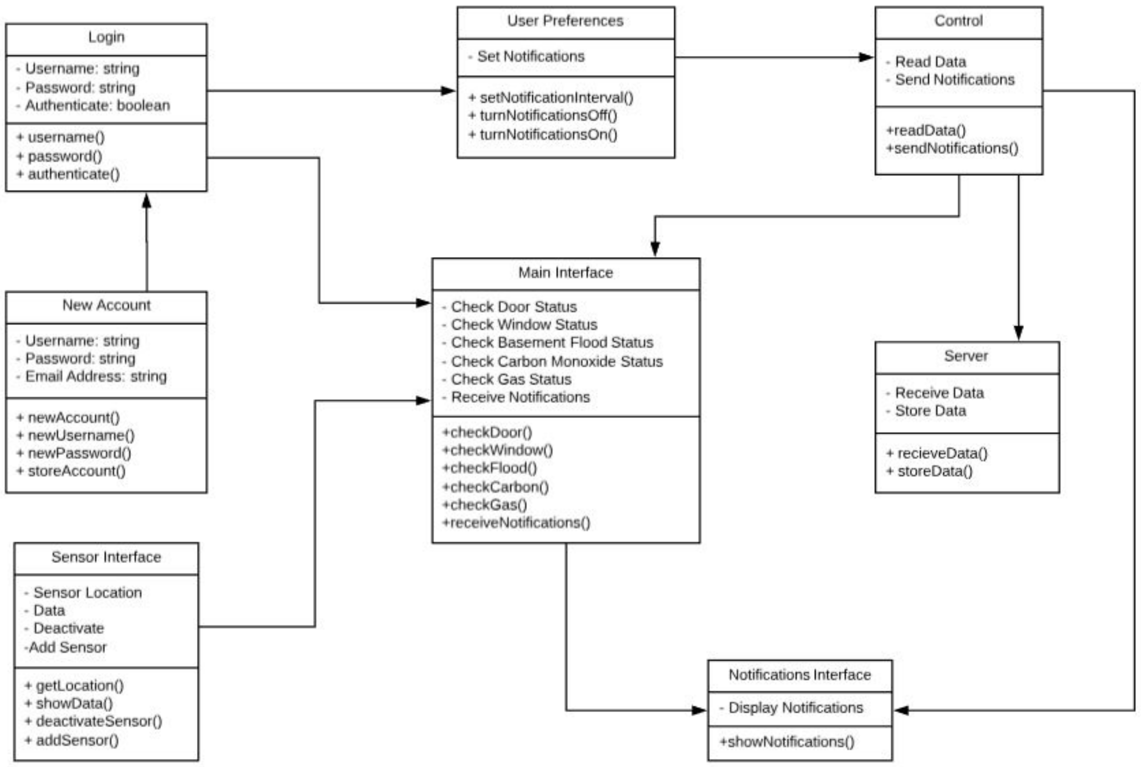
User Case 12 - Log Out:



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 8: Class Diagram and Interface Specification

a) Class Diagram

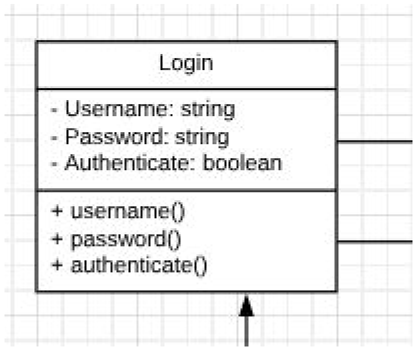


In this figure, we can break the classes down into three main User Interfaces: New User/Account, Sensor Interface, and Notification Interface. The class diagram shown illustrates these interfaces and highlights the necessary usage functions needed in order to get the home security system running. The 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 Firebase server. Then, the Firebase 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 Firebase 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

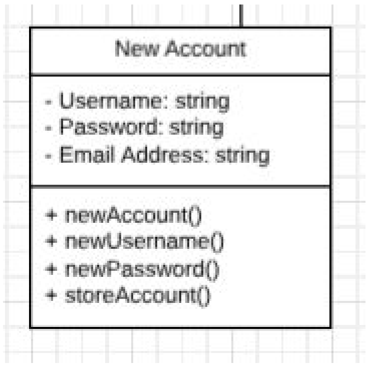
1)



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)

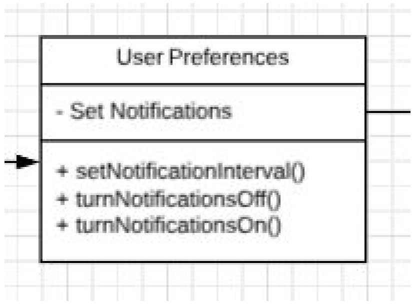


Username: string

Password: string
Email Address: string
newAccount: void
newUsername(): void
newPassword(): 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 was 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.

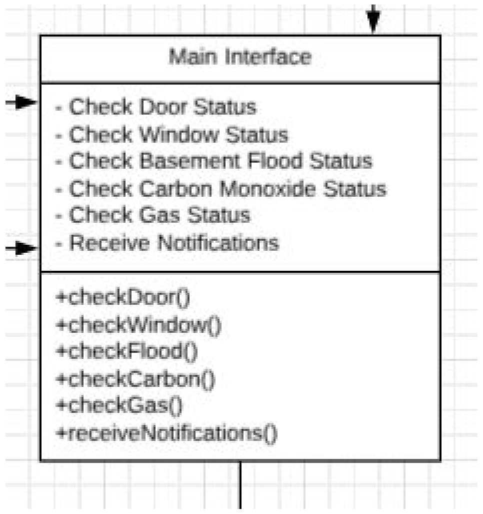
3)



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 as 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.

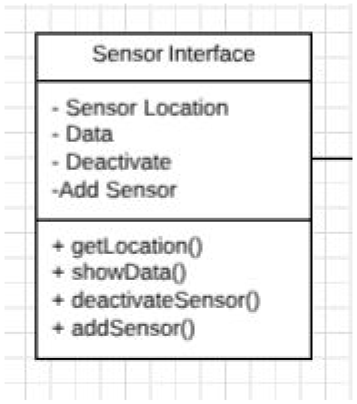
4)



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
receieveNotfications(): 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.

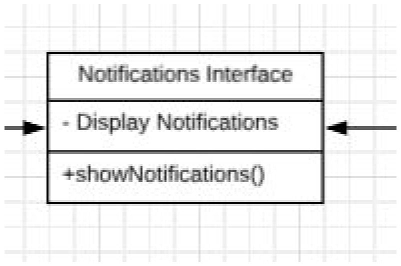
5)



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.

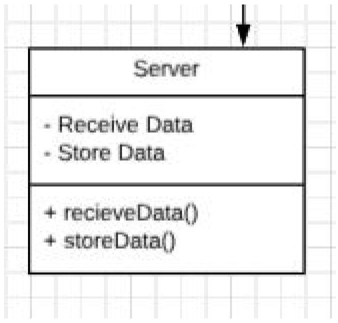
6)



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.

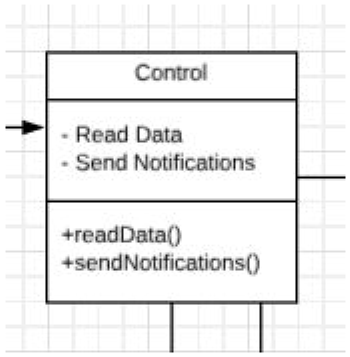
7)



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 receieveData() 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 Firebase 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 logins, 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.

d) Design Patterns

Three Types of Design Patterns:

- Creational Patterns
- Structural Patterns
- Behavioral Patterns

Creational Patterns are patterns that create objects rather than having the user create an objects directly.

Structural Patterns deal with class and object composition. They use inheritance to compile interfaces and define ways to compose objects to obtain new functionality.

Behavioral Patterns deal with the interaction and communication between objects.

Our project design is mainly behavioral. The whole premise of GlassHome is to lay sensors out around the house or building and allow them to communicate with the app to recognize and potential dangers, threats, break ins or any other emergencies. As a result there is an interaction between multiple objects resulting in a more behavioral pattern.

e) Object Constraint Language (OCL) Contracts

The classes we are using are: Login, User Preferences, Control, New Account, Sensor Interface, Notifications Interface, and Server. There are some invariants, preconditions, and postconditions which we should consider for each class. We broke this down into three main GUI's: New User/Account, Sensor Interface, and Notification Interface. We can break down the Invariants, Preconditions, and Postconditions for these three user interfaces first:

<u>New User/Account:</u>

Invariants:

- The user enters a valid email address and password String that is not already in the database.
- "Create Account" Button
- "Username" and "Password" Strings

Preconditions:

- The user does not have a string stored in the database for their account.

Postconditions:

- The user successfully enters a username and password and creates an account.
- The string that the user enters will get saved and sent to the server to be used in the database

<u>Sensor Interface:</u>

Invariants:

- Sensors are properly connected and powered on.
- There is an internet connection to access the server
- Wifi SSID and Password are coded into the Sensors

Preconditions:

- The user wants to view sensor information on their application.
- The user clicks a Sensor

Postconditions:

- The database collects the sensor information and sends it to the application
- Sensor information is displayed on the dashboard in a TextView format

<u>Notification Interface:</u>

Invariants:

- Notification channels are created in the Android Code
- Notification actions are coded for each notification

Preconditions:

- Notifications are allowed on the cell phone for GlassHome

Postconditions:

- Notification channels send notifications when data has changed

Classes:

Login:

Invariants:

- “Log in” Button created
- Username and Password Strings are not empty
- Firebase UID created for user

Preconditions:

- User’s information is stored in the server and database

Postconditions:

- The Username and Password Strings locate user’s UID and open their account

User Preferences:

Invariants:

- Notification Classes are created
- String is created for specific sensor
- Notification Switch button connected to database

Preconditions:

- Notification classes and channels are connected to the server
- Notifications are allowed on Android phone

Postconditions:

- The user can change settings
- String is sent to change the Boolean value to True/False based on user’s personal settings

New Account:

Invariants:

- User does not have a firebase UID created

Preconditions:

- User does not have account stored in Database

Postconditions:

- Username and Password Strings sent to database
- User’s UID stored in server for authentication

Sensor Interface:

Invariants:

- Wifi SSID and Password are coded into the Sensors

Preconditions:

- The user clicks a Sensor's TextView

Postconditions:

- Sensor information collected from database and displayed in a String

Notifications Interface:

Invariants:

- Notification channels are created in the Android Code
- Notification actions are coded for each notification

Preconditions:

- Notifications are allowed on the cell phone for GlassHome

Postconditions:

- Notification channels send notifications when data has changed

Section 9: 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 Firebase 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, Firebase 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 Firebase 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 occured.

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 Firebase 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.

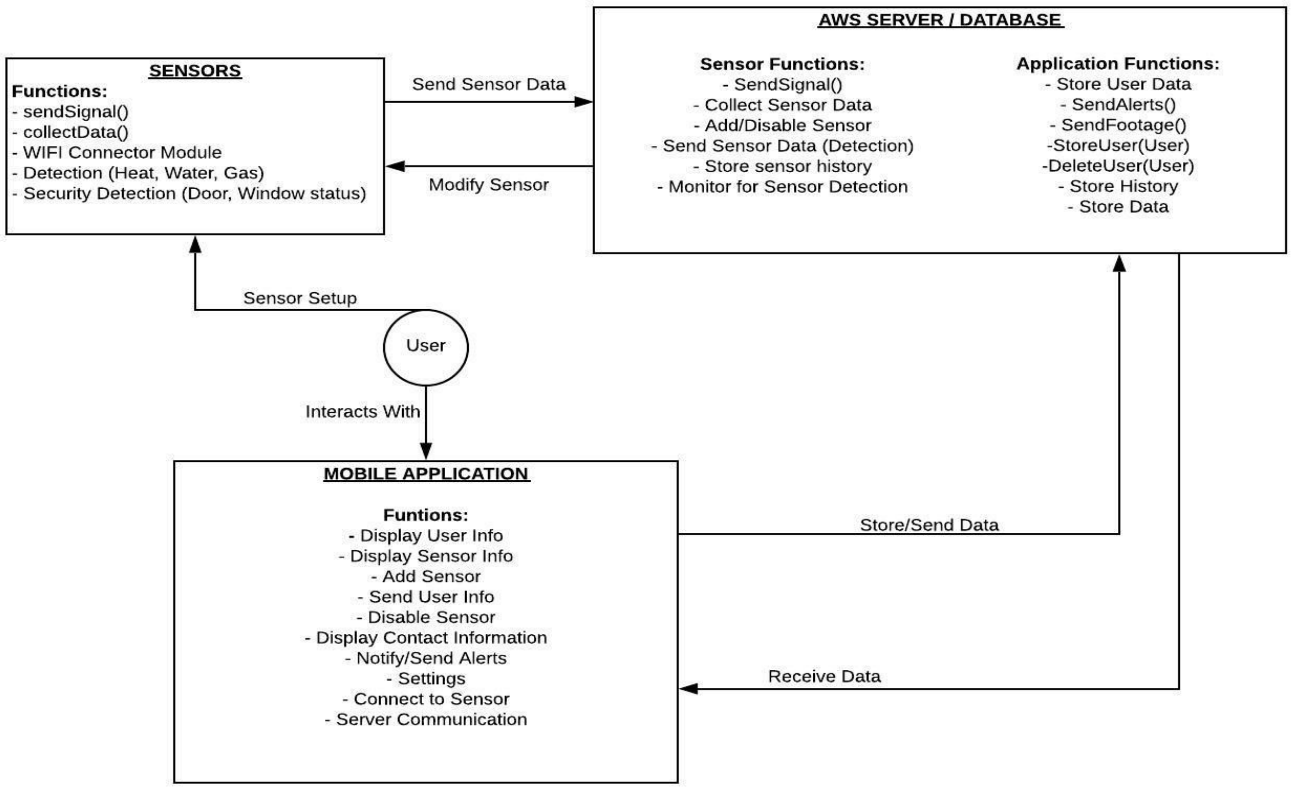
The Client and Server Architecture is what will allow proper communication between the user and the Sensor information as well as Application notifications. In our system, we want different users to be able to access similar information and feel connected to the sensors, which are connected to the Firebase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

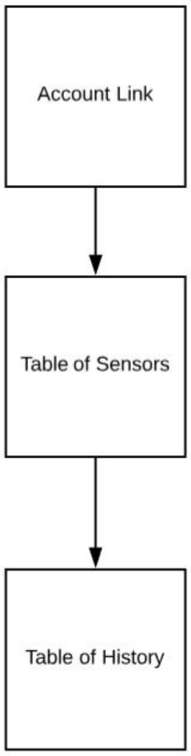
ID#	Sensor ID	Status	Report Status
1234	1111	online	Viewed
1234	1112	online	Not Viewed
1234	1113	online	Viewed

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:



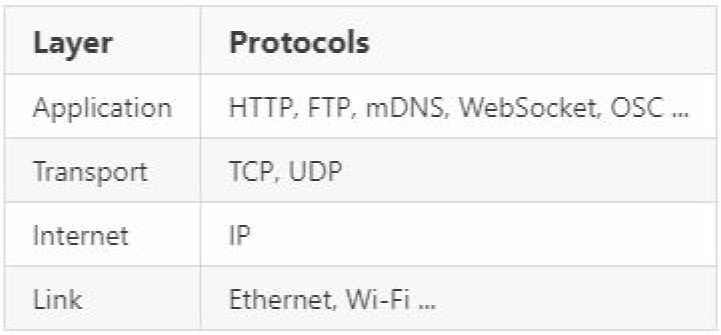
e) Network Protocol

Our system, GlassHome, runs on multiple machines working together coherently. This includes the sensors, the Firebase 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 Firebase 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 Firebase 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:

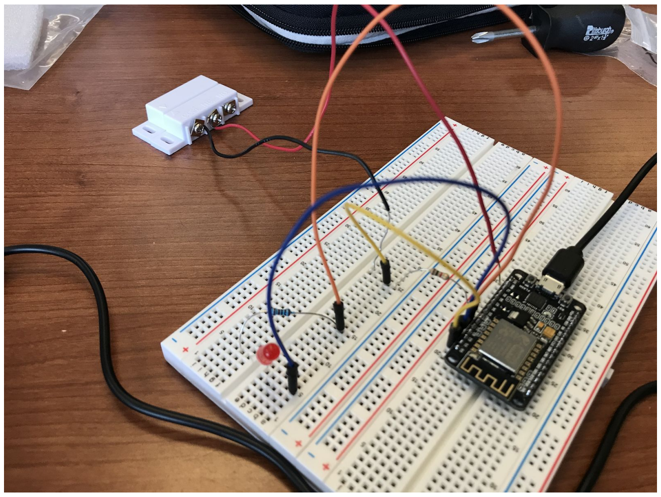
- Hazard Prevention : Carbon Monoxide Sensor, Water Sensor, Smoke Sensor, Gas Sensor
- Home Security: Motion Sensors coupled with Sound Sensors, Magnetic Read Sensors

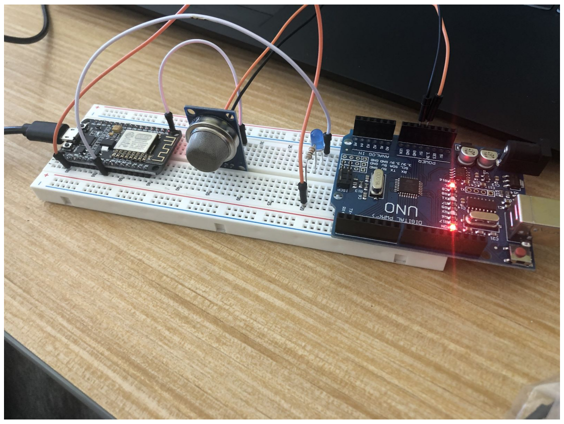
The sensors included will detect various signals around the house and send these signals to the real time firebase database using the Arduino and the NodeMCU WiFi module. The signals include, open/closing doors, gas leaks, floods, window breaks, as well as carbon monoxide leaks.

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.
Examples of Sensors:





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 10: Algorithms and Data Structures

a. Algorithms

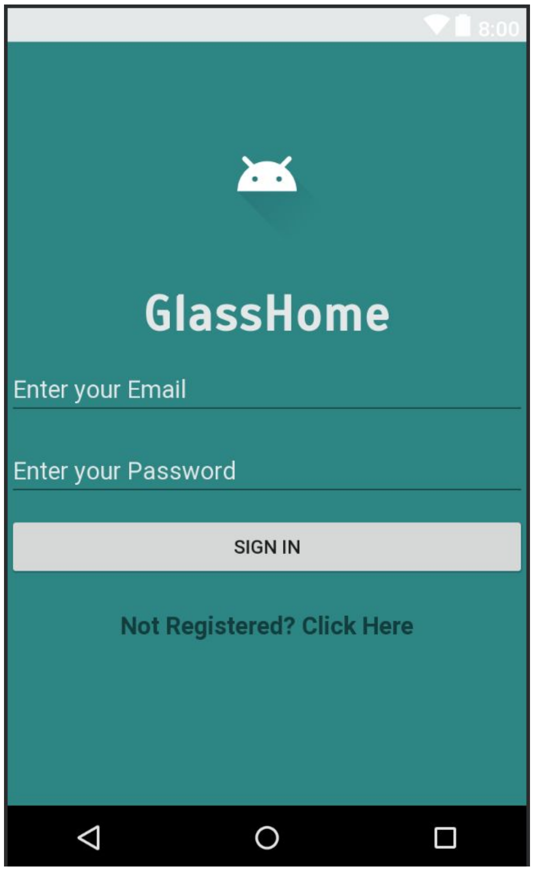
The current plan for our project will not make use of any specific algorithms. The only algorithms that *may* come into play later on in the project is searching (for logins and hash keys). In those situations, the complexity will be O(n).

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 off 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.

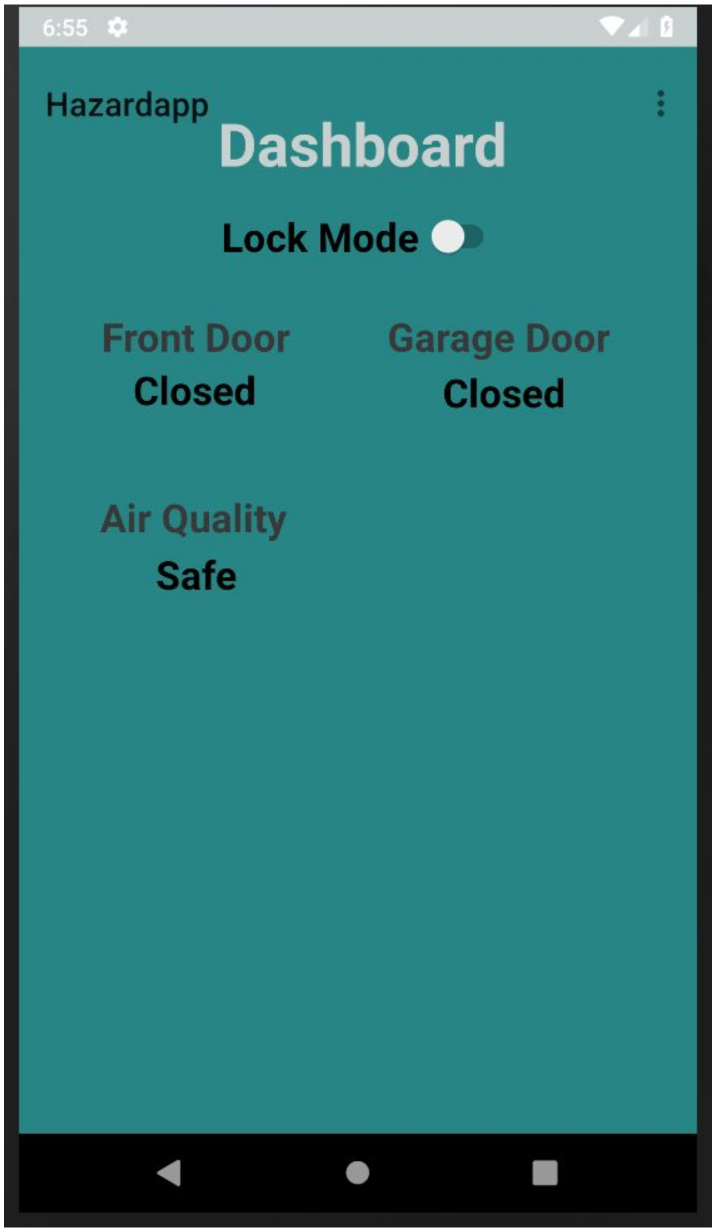
Our decision to work with Firebase was made because of the platform's compatibility with our ideas and flexibility. It made sense to use Firebase as it is a Google product and we are developing for Android devices; the two go hand in hand. Beyond this, Firebase provides us with not only the data structures 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 11: 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.

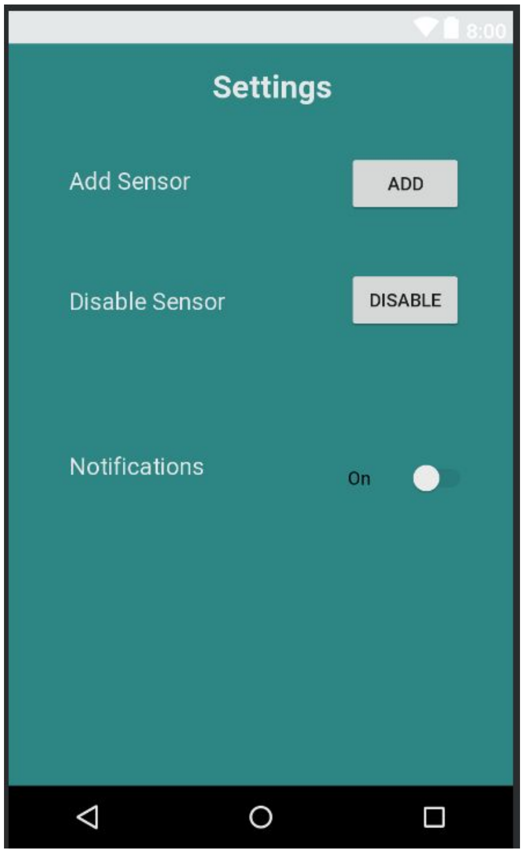


Use Cases: Check (Status), Log Out.

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.

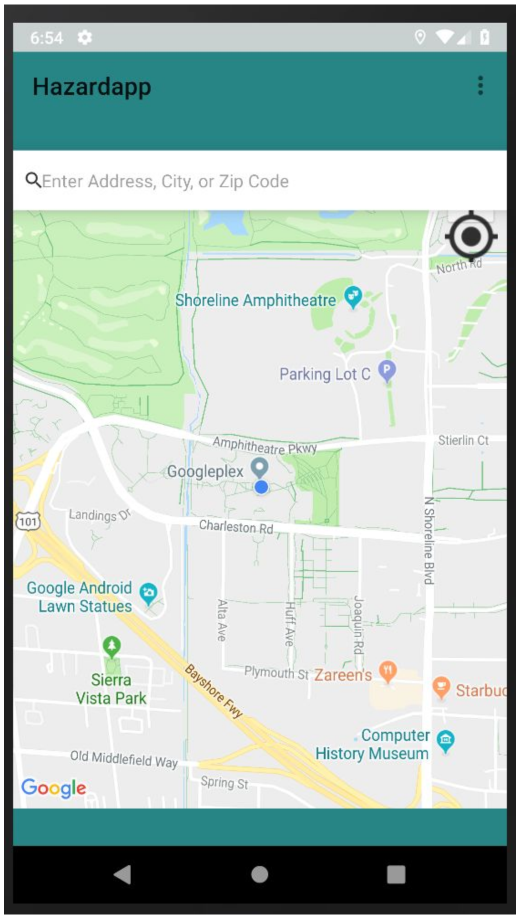
The user will also be able to log out by clicking on a tab on top of the application which is not visible, until clicked. This will give them an option to go to a different page or log out.



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 12: Design of Tests

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

Test Case Identifier: **Use Case Tested:** **Pass/Fail Criteria:**	TC - 2 UC - 2 (Signal) -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 Step 2: Check to see if the database receives and records this change	Sensor sends the signal to the database The database updates in real time to show the current signal

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

Test Case Identifier: **Use Case Tested:** **Pass/Fail Criteria:** **Input Data:**	TC - 4 UC - 4 (Notifications) -Test passes when an event is recorded by the sensor and a notification is sent to the user's phone Sensor Signal, Database Change
Test Procedure:	**Expected Result:**

Step 1: Create a triggering event Step 2: Check phone	Sensor will send a signal to the database to record a change The change in the database will trigger the application to send a notification to the user

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

Test Case Identifier: **Use Case Tested:** **Pass/Fail Criteria:** **Input Data:**	TC - 6 UC - 6 (Add) -Test passes if a new sensor is added and shows up on the dashboard of the application -Sensor signal
Test Procedure:	**Expected Result:**
Step 1: Add a new sensor to the account Step 2: Check sensor list to see if the sensor has been added	System saves the new sensor to the accounts database. Sensor is shown in the list of sensors under that account.

Test Case Identifier: **Use Case Tested:** **Pass/Fail Criteria:** **Input Data:**	TC - 7 UC - 7 (Check) -Test passes when user checks a certain area of the house and sees the real time status of each sensor in that particular area --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: **Use Case Tested:** **Pass/Fail Criteria:** **Input Data:**	TC - 8 UC - 8 (Network Error) -Test passes when app senses a loss in network connection and sends a notification -Lack of data from database
Test Procedure:	**Expected Result:**
Step 1: Disconnect sensors from WiFi Step 2: Disconnect app from WiFi	App recognizes a lack of data being sent from database of particular sensors and notifies user that those sensors are offline. App recognizes no connection to database and sends user a notification.

Test Case Identifier: **Use Case Tested:** **Pass/Fail Criteria:** **Input Data:**	TC - 9 UC - 9 (Log Out) -Test passes when user selects the log out button in settings and is brought to the log in UI -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 13: History of Work

- **Application Development**
  - We used Android Studio to build a fully functional application that includes log in functions, log out functions, a real time dashboard, lock mode, notifications and user settings, as well as a google maps API.
    - The application is linked to a google firebase database, which allows for storage of user information as well as sensor variables.
    - The application retrieves and validates these variables from the database to show on the dashboard.
    - The user settings allow for the user to create new users, and log out from the account as well as manage accounts
    - The notification settings allow for the user to change the frequency of the notifications, turn off/on notifications
    - The lock mode allows for the user to set a time period where notifications are automatically turned on.
    - The dashboard shows the real time status of multiple sensors around the house
    - The google maps API currently shows the location of the user

  Future Developments:
  - We are working on including multiple user accounts that each connect to separate sensors
  - We are working on the notifications being sent as soon as a sensor detects a signal
  - We are working on the map tracking the user's location and automatically turning the lock mode on when they leave a specific radius of the house
- **Sensor Development**
  - We used Arduino IDE to build and program our sensors. The main part that we had used for all our sensor designs was the nodeMCU chip. It provides WiFi capability and GPIO pins to be used with other Arduino sensor chips.
    - We were able to use the Magnetic Reed Sensors to signal if the Front Door or Garage Door are open or close
    - We were able to use an Air Quality sensor to detect if the gas levels within the home are not safe
    - Our biggest accomplishment with this is the task of uploading this to our Application in real-time by uploading the change in data to Google Firebase database

  Future Developments:

- We are adding Window protection with a motion detector and frequency sound detector
- We are adding Carbon Monoxide sensors and Fire Sensor
- We are adding a camera with a live feed to check the status of the house when the notification is sent or detection is noticed by the camera

Section 14: References:

- Software Engineering Fall 2013 project - Voice Control Based Home Automation System
- Software Engineering Spring 2012 project - autoHome
- https://www.lucidchart.com/
- https://app.productplan.com/CugVKlXI#
- https://www.romanpichler.com/blog/10-tips-creating-agile-product-roadmap/
- https://blog.asana.com/2018/08/product-roadmap-tips-templates/
- https://www.ece.rutgers.edu/~marsic/Teaching/SE/proposal.html
- https://www.ece.rutgers.edu/~marsic/Teaching/SE/report1.html
- http://www.cs.ucc.ie/~adrian/cs560/UML4%20SysSeqDiag%20Contracts.pdf
- https://www.androidcentral.com/sites/androidcentral.com/files/styles/xlarge/public/article_images/2016/02/fitbit-add-add-device.jpeg?itok=VdWTM11N
- https://lh3.googleusercontent.com/FIC7m61Z9oZFV0w6hO1NzfgrY-ZCF8fx-jpSU28lbKGC8AxCji-_NdySTV_P-TrIxc8=h310
- https://cdn57.androidauthority.net/wp-content/uploads/2015/09/Screenshot_2015-09-14-17-59-06.png
- https://en.wikipedia.org/wiki/FURPS
- https://app.productplan.com/CugVKlXI#
- https://www.romanpichler.com/blog/10-tips-creating-agile-product-roadmap/
- https://blog.asana.com/2018/08/product-roadmap-tips-templates/
- 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/

Project Management:

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.

Before Demo 1:

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.

Before Demo 1:

We then split up our team once more.

Home Security:

- Window Protection: Harshil, Adarsh

- Live Feed/ Camera: Kyle

Hazard Prevention:

- CO, Fire/ Heat: Nathan, Andy

App:

- Multiple Users: Shaan, PJ, Avi
- Notifications: Shivum, Shaan

With these mini group assignments, it was easier to work in pairs and concentrate on a smaller, more specific, task to be done for the next demo. Our weekly meetings helped everyone stay motivated and at a timely pace.