



---

# AUTOHOME—REPORT 3

---

Group 15: Mahmoud Bachir (Leader), Aaron Christie, Smruthi Srikumar, Gunbir Singh, Brendan Li, Kyle Ross, Jimmy Wen, Karthikey Thapliyal



AUTOHOME

<https://github.com/mahmoudbachir/homeautomation>

<https://ruAutoHome.weebly.com/>

## Contents

Individual Contributions.....	2
CSR Narrative .....	2
Glossary of Terms.....	5
Functional Requirements Specification .....	6
User Effort Estimation.....	14
Interaction Diagrams .....	25
Class Diagram and User Specification .....	27
Design Patterns .....	28
OCL Contracts.....	29
Classes/Domain Traceability Matrix .....	35
System Architecture and Design .....	36
Algorithms and Data Structures.....	37
User Interface Design and Implementation.....	38
UI Mobile Design.....	41
Design of Tests .....	43
History of Work.....	60
References .....	60

## Individual Contributions

All group members contributed equally, everyone netted a total of 25/200 points, including the 2 main members handling project management.

## CSR Narrative

In the past few years I've found myself gradually more and more tired as I go through my day. I don't have a particularly stressful or physically laborious job, but rather I feel that my issue stems from the fact that there are several menial tasks that need to be completed around the house, several times over. Some examples of such tasks include changing the light settings around the house (turning it on/off, turning up/down the brightness, changing the color to create an ambience in the house), locking the doors and being sure that they are truly locked, and also, preparing an ambience and mood in the house before I arrive home (as my guests may arrive only moments after I do). Now stop, don't say anything, I know what you're thinking, you think I should hire a maid. Of course, that is certainly an option, however, that is not the best choice that I have. Hiring other people isn't the best course of action as they may be unreliable and untrustworthy, I couldn't see myself leaving them in my house without my supervision. I want my house to be secure and inviting a stranger into my house to take care of it simply does not make sense to me. Furthermore, it is also not good for my image in front of friends and coworkers, and they may look down upon hiring a maid. Instead of hiring a maid, I have a better idea.

The other day it occurred to me, 'Instead of hiring human workers to maintain and secure my house, I should have an electronic solution implemented!'. Machines are much more reliable and trustworthy than humans. Their behavior is programmed, and concurrently, they are predictable. A system can be devised where I can use my phone, laptop, or even my voice to give commands, and they will be executed these commands quickly and without question. Thereby, I want a system to automate my home, I will refer to it henceforth as AutoHome. The vision that I currently have for AutoHome contains 5 main classifications of features. These features include: remote control using a mobile application, light control, audible alerts, music control and voice-controlled activation of the system from within the house. In my opinion, the most important feature of AutoHome is the remote home control feature. This will be implemented with a mobile application. The application should have a secure internet connection to the home and the user should be able to connect to his/her house by simply logging in to a personal account. The account should only allow a predefined maximum number of people to log in at once, you can add more at any time if you'd like. This should serve a dual purpose

of preventing user conflicts and increasing account security (this way it is apparent when an unauthorized user accesses an account). The application should prevent a user from logging in without further verification of identification if the user fails to login after seven consecutive attempts. In the event that someone has attempted to access account more than seven times, then the application should also send an alert to the user's mobile device. Once the user has logged in, he or she should be able to monitor and control various connected devices around the house. The main device features that I currently envision for the user to use within the application include; viewing the status of each door lock around the house, the brightness of lights in each room, the colors of lights in rooms with RGB LED lights, the playing of music in each room, the volume of music in each room, and the connection status of the application to devices in the house via Bluetooth.

As an extra measure of security, buzzers should be installed next to each door. These buzzers will sound whenever a door is opened. The purpose of these buzzers is to alert any residents inside the house when another person enters or exits the house. Buzzers are installed next to each door so that residents can also have a sense of which door was opened. Additionally, this audible security measure is user-friendly to the visually impaired.

Another extremely appealing feature would be light control. First and foremost, I believe that users should be able to easily control the activation (on) and deactivation (off) of lights in each room of the house. You should be able to control the lights directly from the mobile/web application. This would easily and conveniently solve the issue of forgetting to turn off lights on the way out or being able to turn off my bedroom lights once I'm already in bed. Also, a potential use of this function could include remotely turning on lights when not in the house (giving impression that someone is home) to ward off potential burglars. Two more features regarding the automation of light include controlling brightness, and controlling the color of the bulb. While these features aren't necessarily related to home security, they are extremely versatile. The brightness and color of a light can create an ambience in the house and can influence the mood of people. These effects can be utilized in several situations such as functions (parties), meetings, studying, etc.

All of these devices should be able to be accessed from a app on my phone, or a website on my computer. I should be able to login, and view the status of every device. I should also be able to control the device remotely.

Those are the main features that I currently have in mind for a user to have access to. However, I understand that if multiple users are to be able to use this application, then users need to have profiles specifically for themselves and their homes, and they should have a username and password to log in to

these profiles. Thereby I think it is necessary that user information is stored in a secure database. The most important information is obviously the username and password, as this gives you access to someone's user account and thereby, control over their home. This will ensure that a user can only establish a secure connection to his/her specific home only. It is of the utmost importance that users are unable to gain unauthorized access to other homes.

Security is an important feature that is crucial to this electronic solution. I mentioned above that users should be alerted if there are multiple failed login attempts to an account (specifically 7 failed attempts) as well as being able to set a maximum number of users on a account. However, I think that there are additional safeguards that can be taken to combat this. I believe that there should be additional verification of user's identities, and in order to establish this, there must be verification questions (recovery questions) that a user must answer to gain access to an account. The system will have to store these questions and their answers in a database. The system should also record the time/date that these failed login attempts were made.

While I greatly value the convenience that an automated home system would give me, it is imperative that this system is secure. Nobody should have access to any account that they are not authorized to use, only me and other authorized users should be able login and use this system. It is my hope that AutoHome will be reliable and convenient, since it is my intention that it is utilized on a daily basis. Additionally, in the future I expect ownership of an automated home system to become a standard of living. The features previously described are applicable in a wide variety of situations, which gives the product a large target audience. I am excited to see the future of AutoHome and hope that it can be developed without any issues.

## Glossary of Terms

User - A person who intends to implement home automation in the home automation system.

User profiles – Profiles that store different settings for different users of the same system.

Database – Table of data storing user and device information

Home Automation – Automation of the home or household activities. This may include lighting, doors, speaker systems, etc.

Home Automation System – The system of devices that seamlessly connects all aspects of home automation in a form the user can easily understand and use.

RGB – Letters that represent red, green and blue. Each of these three colors usually holds a value between 0 and 255.

LED – An acronym for light-emitting diodes. Light emitting diodes are two-lead semiconductor light sources.

Mobile Device – A device meant for mobile use; tablets, smartphones, and other android-capable devices.

Application – The primary program the user will be interacting with to control various parts of the home automation system

Device Status – Status of the user's devices. Lets the user know which devices are off/on, or locked/unlocked

Mobile Application – Application used on a mobile device

Bluetooth – Wireless protocol enabling mobile device to interaction with the home automation system

Arduino – Open-source electronic prototyping platform enabling users to create interactive electronic objects.

Microcontroller – Remote controller that will allow the user to communicate with the devices from a remote location.

12V Relay-Relay – an electromagnetic switch which can be turn on and off by an applying the voltage across its contacts. In this project used a 12V 4-channel relay

Wi-Fi Chip – chip that interacts with the Arduino and allows the Arduino to connect to a wifi network, and effectively be able to utilize a server to send/receive commands

Relay driver – ULN2003 – Relay safely driven by ULN2003 IC. Protect microcontroller from relay kick back using integrated clamping diodes. Has 7 high current Darlington arrays each containing 7 open collector Darlington pairs with common emitters.

## Functional Requirements Specification

Identifier	Functional Requirements	Priority
REQ-1	The system shall be able to control various light fixtures around the house (on, off, dim)	5
REQ-2	The system shall be able to alert the user when a door opens or closes with a audible signal	5
REQ-5	The user should be able to control the system with their voice through the use of a key-phrase	1
REQ-7	The user should be able to easily add new items to be controlled with the system	2
REQ-8	The user should be able to create/delete user profiles and concurrently be able to login/logout of account	4

Table 1: Functional Requirements

Identifier	Non-Functional Requirements	Priority
REQ-3	The user should be able to monitor their system with their mobile phone through an application	4
REQ-6	The user will be able to save user profiles for different members in the household	1
REQ-10	The user should be alerted of suspicious logins and login attempts shown (account security)	1

Table 2: Non functional Requirements

Identifier	On-Screen Requirements	Priority
REQ-1	The application should allow activation of the voice control system	4
REQ-2	The application should provide audio/visual feedback when controlling a system	3

REQ-3	The application should have buttons to allow manual adjustment of various systems	5
REQ-4	The application should allow the user to see the status of the systems they have connected	5
REQ-5	The application should have a responsive and easy to use interface	2
REQ-6	The application should provide feedback on system alerts and status via notifications, when application is not running / running in background	3

Table 3: On screen Requirements

### **Use Cases**

<b>Use Case Name</b>	<b>REQ's</b>	<b>Actor</b>	<b>Actor's Goals</b>
Unlock (UC-1)	REQ-8	User	To login to personal account
Lock (UC-2)	REQ-8	User	To logout of personal account
Add-User (UC-3)	REQ-8	User	To create a new user account and allow access to home devices
Remove-User (UC-4)	REQ-8	User	To remove a user account and disable access to home devices
Access-History (UC-5)	REQ-10	Database	Stores all access attempts in log with login time/location (using ip address)
Lock-Monitor (UC-6)	REQ-2	Sensors	To monitor the door for possible intrusions
Add/Remove-User (UC-3, UC-4)	REQ-8	Database	To create/remove user profiles to database and subsequently enable/disable all user's home devices
Monitor-Device (UC-7)	REQ-3	User Web Interface	To monitor all devices statuses (light is on/off, music is paused/playing, door is locked/unlocked)



Monitor-Devices (UC-7)	REQ-3	User	To be able to monitor all device statuses from the mobile app, or the web app
Add-Device (UC-8)	REQ-7	User	To be able to add new devices (lights, speakers, etc.) easily
Add-Device (UC-8)	REQ-7	Database	To add new devices to database for each user
Device-Status (UC-9)	REQ-3	Database	To update the status of each device every interval of time
Read-Device-Status (UC-10)	REQ-3	User Web Interface	To take device status info from the database and display to the user in the appropriate section (status updates accordingly)
Light-Off (UC-11)	REQ-1	autoHome System	To turn off the light bulb
Light-On (UC-12)	REQ-1	autoHome System	To turn on the light bulb
Light-On/Off (UC-11, UC-12)	REQ-1	User	To be able to turn on/off the light bulb from app
Connect-Device (UC-15)	REQ-7	autoHome System	To be able to connect a new device to a user's autoHome (Bluetooth speaker, WiFi devices, etc.)
Add-Device (UC-16)	REQ-7	Database	To add a new device to a user's autoHome account

Table 4: Use Cases

## UC-1: Unlock

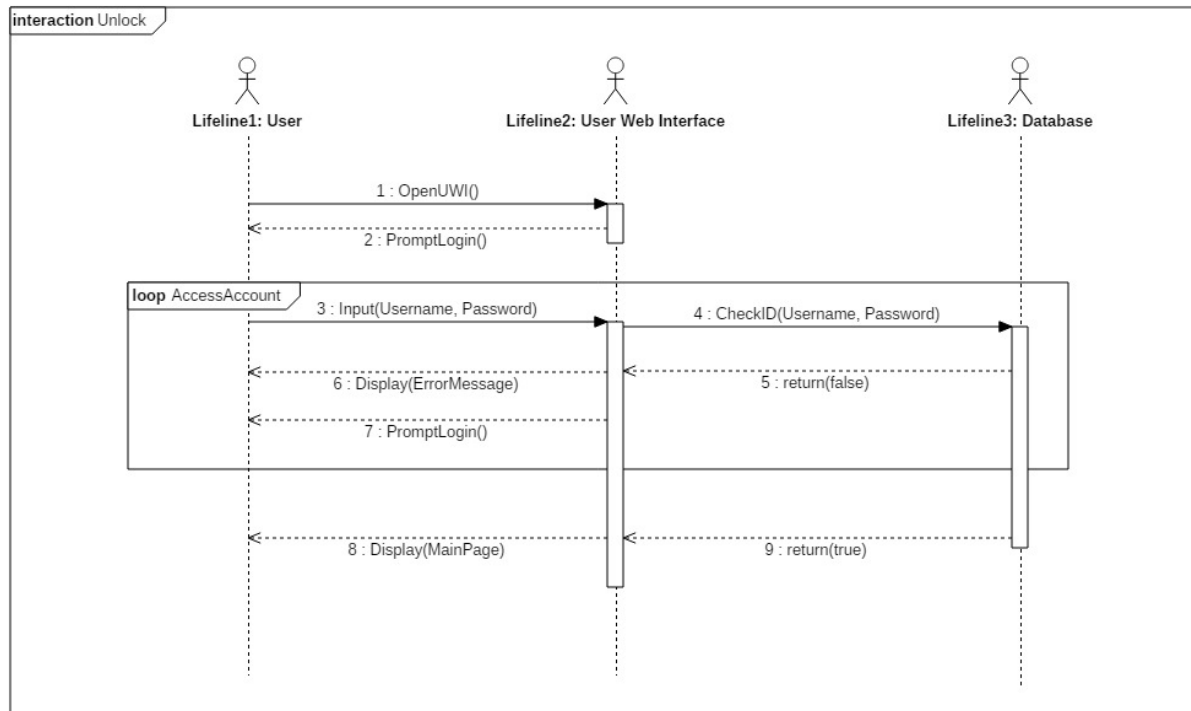


Figure 1: UC-1

The purpose of Use-case 1 is for the user to be able to log into his/her account where all their settings are saved. This design allows for multiple profiles for different people around the house. The use-case allows communication with the server and then the database which stores the user's login information, without the user ever having to interact with the database, increasing security. The general flow of the use case will be that the user will input their username and password into the UI and system will check if the credentials are correct. If they are, the interface will display the user's list of devices they can control. If the entered credentials are incorrect, the interface will display an error message and prompt the user to enter their credentials again. This error loop will continue until the user enters the correct credentials.

### UC-3 Add User

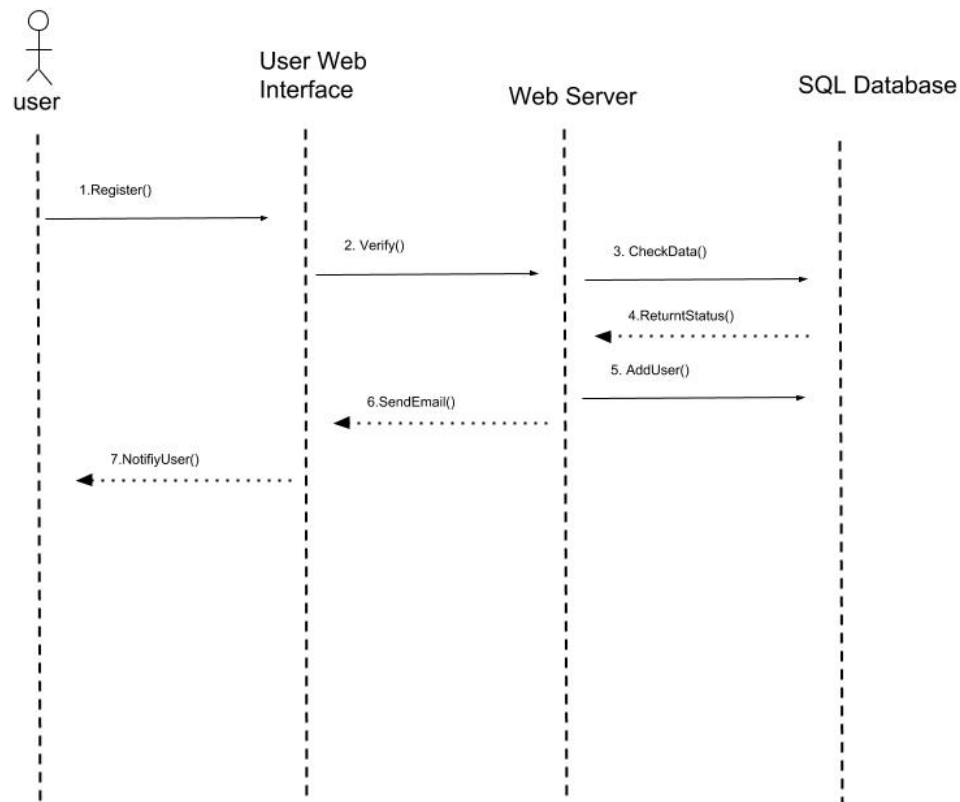


Figure 2: UC-3

The purpose of Use-Case 3 is to be able to add other users to the application. This will allow ease of access when there are multiple people living in the same house because they will all have equal control over the autohome devices. This is also a method to keep the app secure because every user will require a password to gain access to the application and the devices that are controlled by the autohome. This use-case allows a new user to create a username and password so they can easily log in to the app whenever they need to. After this is done, the user information is sent to and stored in the database. Any two people may not have the same username. After the information is successfully stored in the database, a success message is

returned to the user and a confirmation email is sent. The user is then prompted to click on the link sent in the email.

### UC-8 Add-Device

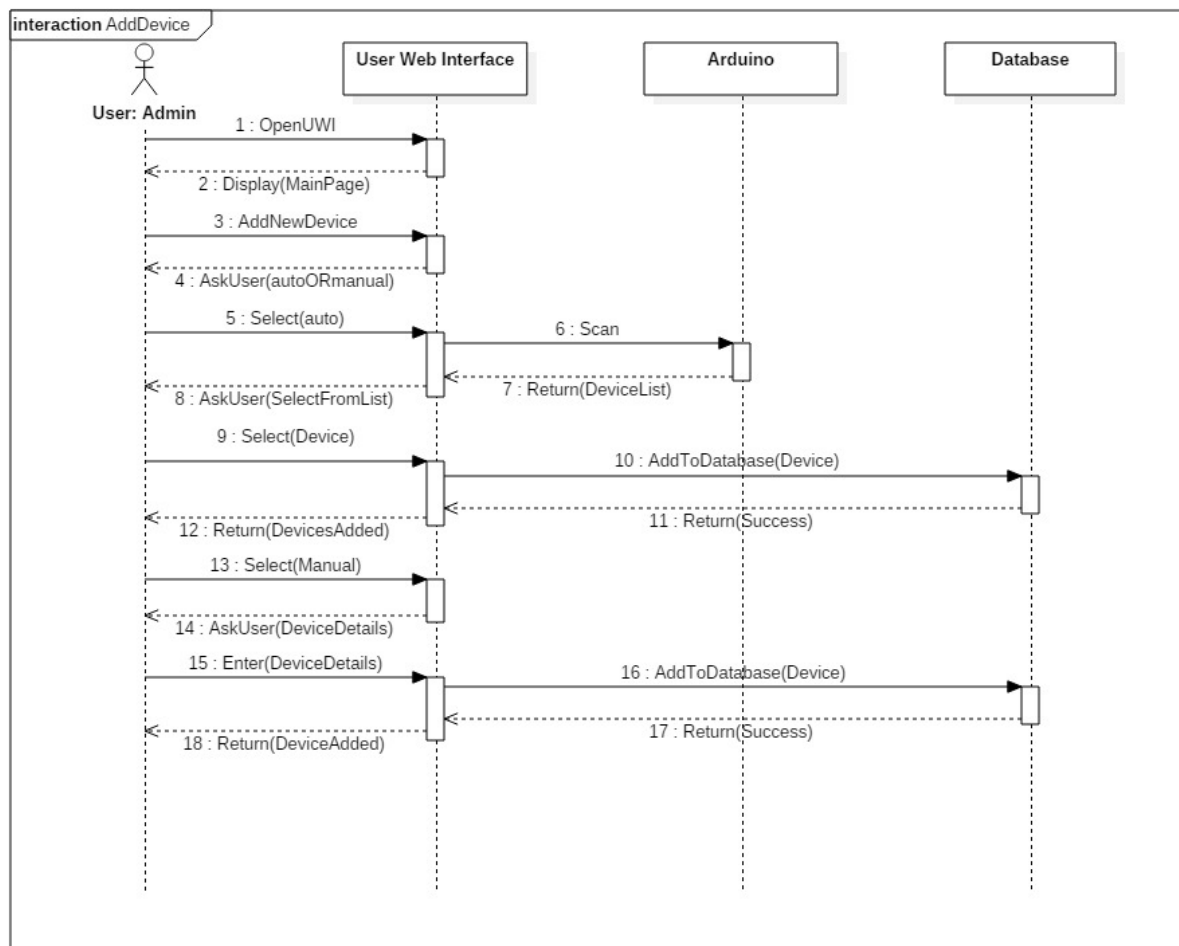


Figure 3: UC-8

The purpose of Use-Case 8 is to be able to add in another device from the list of available autoHome devices. This will allow all the nearby available autoHome devices to be managed by the user through the user web interface. The use case allows the admin to configure the devices he wish to add. The admin prompts the user web interface to add new device which returns with an option to add the device manually or automatically. The admin thus select the way it wants the device to configured and added. Choosing automatically makes the user web interface request the arduino to detect all the devices near it. Thus Arduino sends all possible connectable autoHome Devices to User Web Interface and then the User Web Interface displays all connectable

devices to admin. The admin then selects the device and then confirms it. The device is then displayed to the Web user Interface and the User Web Interface adds device to database. Choosing manually just makes the Web user interface stored the configuration directly to the database. Finally User Web Interface displays connected devices as well as actions for those devices.

#### UC-9 Device-Status

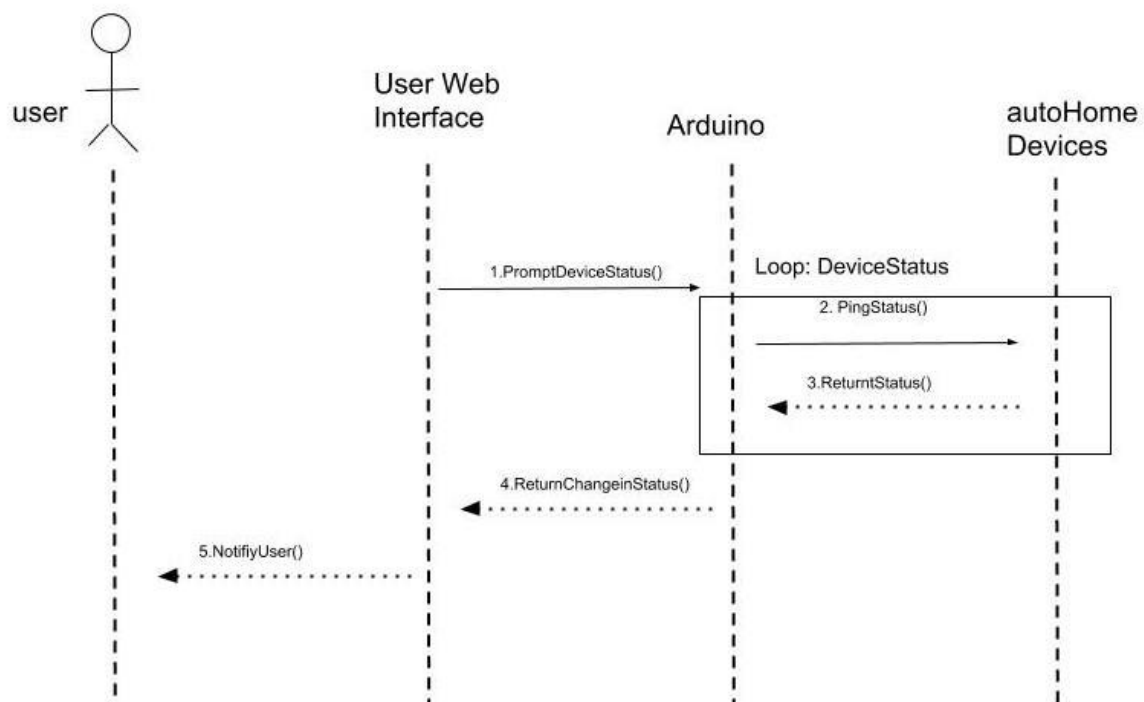


Figure 4: UC-9

The purpose of UC-9 is for the User to be able to view the status of all of his devices at any given time. This will allow the user to be in complete control whenever/wherever he is. The way this use case is designed to work is the the user will login to the app,and then he will select the page labeled “Manage Devices”. Once he picks that page, the web interface asks for the devices statuses, the Arduino will ask the devices for their statuses, and then this will be returned to the Web Interface. At this point the web interface refreshes with the updated status of each device. This page stays updated because it is constantly being updated after a certain interval of time.

## Traceability Matrix

REQ	PW	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	UC10	UC11	UC12	UC15	UC16
REQ2	5						x								
REQ3	4							x		x	x				
REQ5	1														
REQ6	1														
REQ7	2								x					x	x
REQ8	4	x	x	x	x										
REQ10	1					x									
Max	PW	4	4	4	4	1	5	4	2	4	4	5	5	2	2
Total	PW	4	4	4	4	1	5	4	2	4	4	5	5	2	2

Table 5: Traceability Matrix

## User Effort Estimation

Lights:

**Turn on:**

**Respective Use Case: UC12**

Data Entry: 1 keystrokes

Navigation: 2 clicks

Click "lights" button(1) → click "on" (1)

**Turn off**

**Respective Use Case: UC11**

Data Entry: 1 keystrokes

Navigation: 2 clicks

Click "lights" button(1) → click "off" (1)

Doors:

**Check Door Status**

Click "Doors" button (1) → "Door Status"(1)

2 clicks total

**Lock door**

Click "doors" button(1) → click "unlock" (1)

2 clicks total

**Unlock door**

Click "doors" button(1) → click "lock" (1)

2 clicks total

## **Main Screen Functions:**

### **Add new device**

#### **Respective Use Case: UC8**

Prerequisite: Log in (2 keystrokes and 3 clicks), physical device connection

Data Entry: 2 keystrokes

Navigation: 3 clicks

Click "Add new device" (1) → input device info → click "add"(1)

### **Log in**

#### **Respective Use Case: UC1**

Data Entry: 2 keystrokes

Navigation: 3 clicks

Click in "username" field and enters user ID

-Click in "password" field and enter password

-Click the arrow button to commence login

### **Log in (forgot password)**

#### **Respective Use Case: UC5**

Data Entry: 1 keystrokes

Navigation: 2 clicks

Enter email in "email" field (1) → click "forgot password" (1) → enter email again (1) → open mail app (1) → choose inbox(1) → open password reset email (1) → click reset link (1) → enter new password (1) → return to autohome app → Enter email in "email" field (1) → enter password in "password" field (1) → "sign in"(1)

### **Log out**



**Respective Use Case: UC2**

Data Entry: 2 keystrokes

Navigation: 1 clicks

Click “log out” at top corner(1)

**Device statuses****Respective Use Case: UC7**

Data Entry: 2 keystrokes

Navigation: 2 clicks

Click on any given device from home screen (1) → “device status” (1)

**Add User****Respective Use Case: UC3**

Data Entry: 3 keystrokes

Navigation: 4 clicks

“Settings” icon top right (1) → “devices” (1) → “add device” (1) → enter device information (min 1 click)

## Domain Analysis

### UC-1: Unlock Domain Model

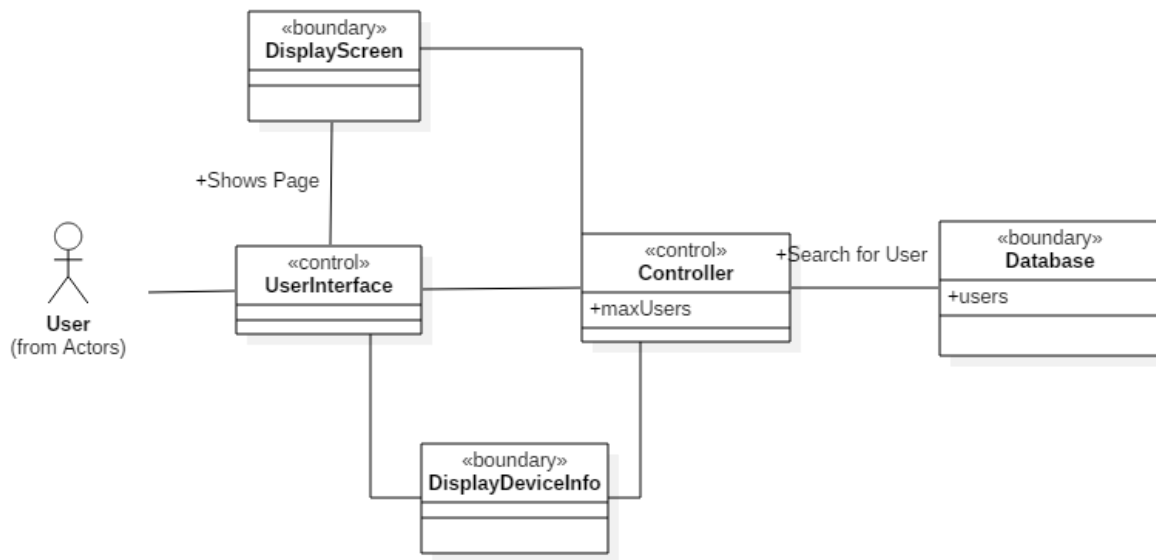


Figure 5: UC-1

Association:

Concept pair	Association description:	Association Name:
Interface page <-> DeviceControl	Sends request to add-device to DeviceControl. Then, if device is found, prompts user to connect device.	UserMachineInteraction

DeviceControl <-> DisplayScreen	Display the data received on the user screen	ShowPage
DeviceControl<-> >UserDatabase	Search for the user in the database	UserDetails
DeviceControl<-> >AddDatabaseDevice	DeviceControl sends device data to AddDatabaseDevice; prepares it to be uploaded into Database	Convey device data
AddDatabaseDevice <-> Database	Inputs device data into Database	DatabaseManagement
DeviceControl <-> DisplayDeviceInfo	DeviceControl conveys what type of page should be sent to user (success, confirmation, error, etc.)	Write page
DisplayDeviceInfo <-> Interface page	Shows page to user	UpdateStatus

Table 6: Associations

Attributes:

Concept	Attributes	Attribute Description
DisplayScreen	ShowPage	Display the login page to the user
DeviceControl	SearchUser	Search if the user is present in the database
Find User	MaxUsers	Determine if the database can handle another user

	SaveUser	Used to be saved as new user's username and password
	FindUser	Check the user login details for correctness
database	User Info	Used to store user information

Table 7: Attributes

UC-3: New User Domain Model

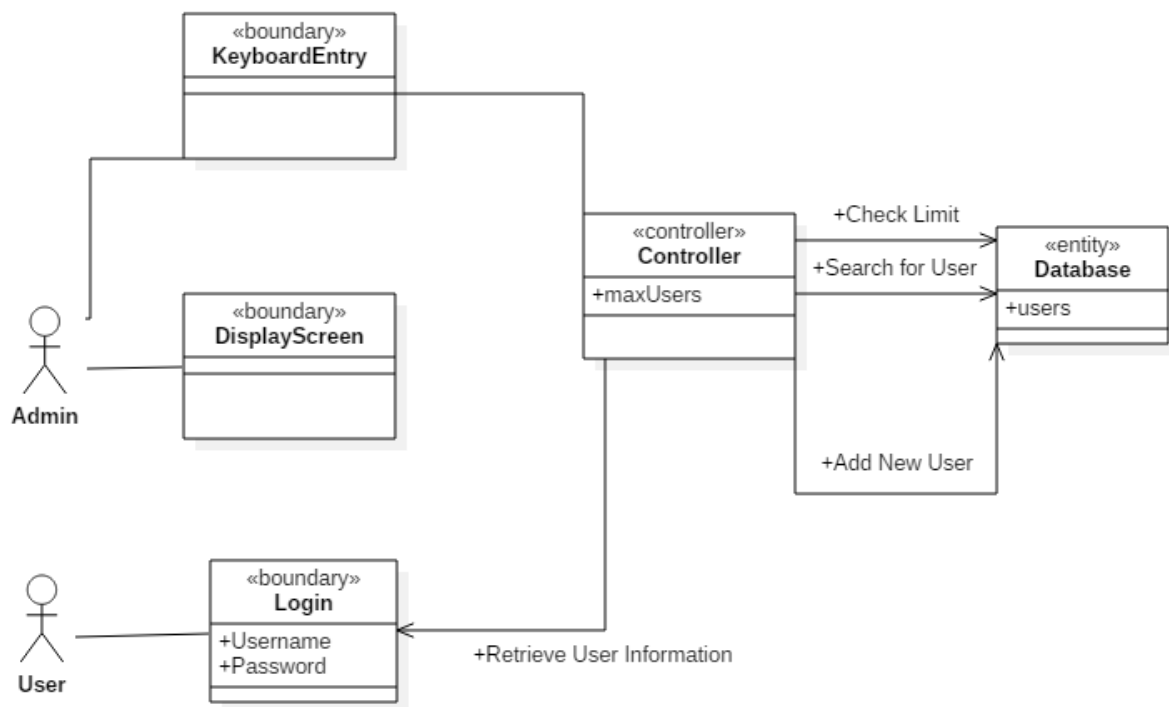


Figure 6: UC-3

Associations:

Concept Pair	Association description	Association name
InterfacePage <---> Device Controller	User sends request via the interface to the controller to create new user	Conveys request
deviceController <---> database Connection	Controller searches database to see if space available for new user, then continues	Searches

InterfacePage <---> Device Controller	User enters username and password	Provides data
deviceController <---> database Connection	Controller searches database for pre-existing user with same username, continues if no match found	search
deviceController <---> database Connection	Saves user information to database	Requests save

Table 8: Associations

Attributes:

Concept	Attributes	Attribute Description
Search Request	Max amount of users	Used to determine if spot is available for new user to be entered
	New user's ID	Used to be saved as new user's username and password
database	User Info	Used to store user information

Table 9: Attributes

### UC-8: Add-Device Domain Model

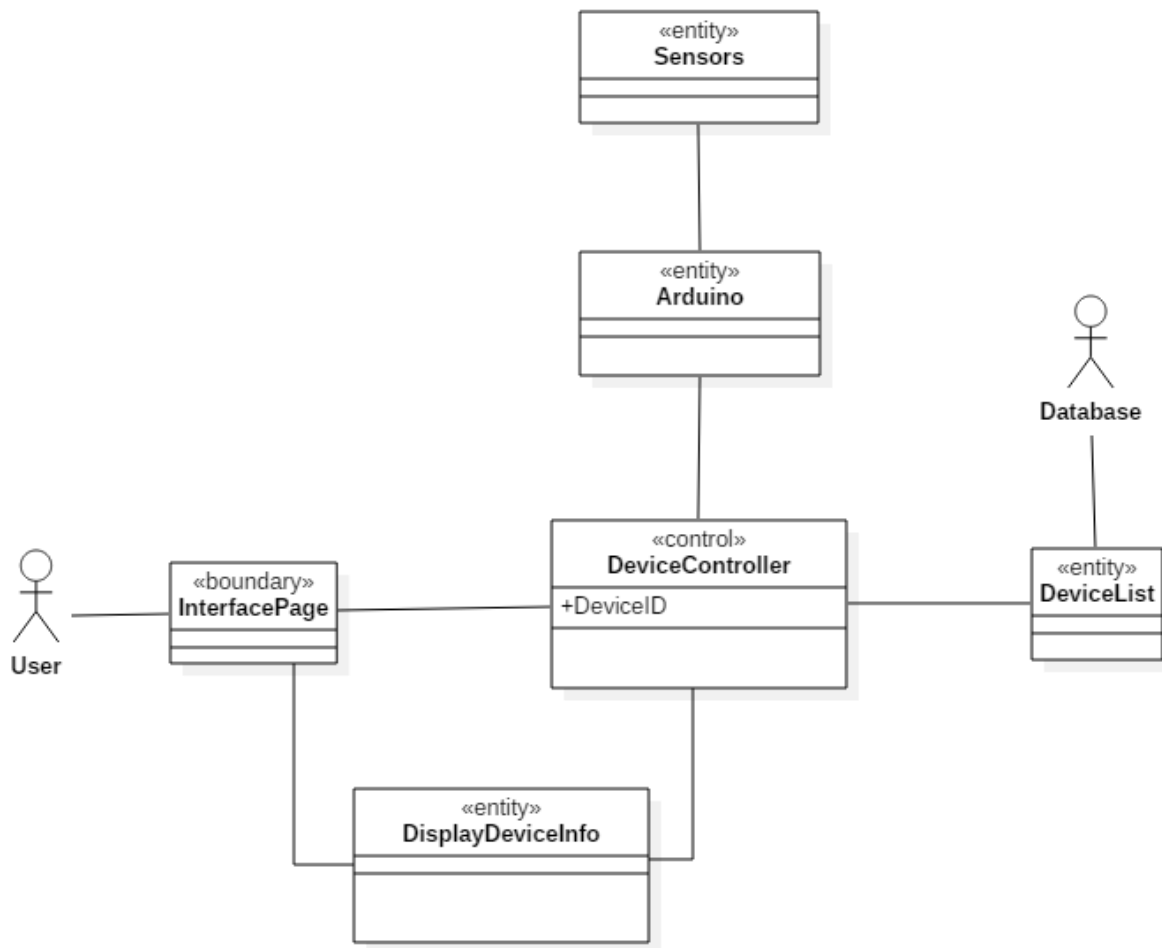


Figure 7: UC-8

#### Responsibilities:

Responsibility Description:	Type:	Concept name:
Application on user's device that connects to Arduino and sends device entry data to Database.	D	DeviceControl
User Web interface that receives data from user and displays data from DeviceControl.	D	DisplayDeviceInfo
Adds new device to user's profile in database.	D	AddDatabaseDevice
Shows possible actions user can take on the User Web Interface. Shows Successful connection after device data is added to Database.	D	Interface page

Sends data between DeviceControl to Device sensors.	D	Arduino
Sends Device Sensor data to DeviceControl	D	Device Sensors
Creates page for user based on DeviceControl's output.	D	DisplayDeviceInfo

Table 10: Responsibilities

Associations:

Concept pair	Association description:	Association Name:
Interface page <-> DeviceControl	Sends request to add-device to DeviceControl. Then, if device is found, prompts user to connect device.	UserMachineInteraction
DeviceControl <-> Arduino	DeviceControl prompts Arduino to search for devices and connect to them.	DeviceManagement
Arduino <-> Device Sensors	Arduino searches for active devices near it.	Arduino Search
Arduino <-> DeviceControl	Arduino gives device data to DeviceControl after search	Show devices
DeviceControl<->AddDatabaseDevice	DeviceControl sends device data to AddDatabaseDevice; prepares it to be uploaded into Database	Convey device data
AddDatabaseDevice <-> Database	Inputs device data into Database	DatabaseManagement
DeviceControl <-> DisplayDeviceInfo	DeviceControl conveys what type of page should be sent to user (success, confirmation, error, etc.)	Write page
DisplayDeviceInfo <-> Interface page	Shows page to user	UpdateStatus

Table 11: Associations

Attributes:

Concept:	Attributes:	Attribute Description
Arduino:	SearchDevices:  Connect Devices:	-Searches for connectable devices  -Connects device in range

DeviceControl:	ArdSearch:	-Commands Arduino to search for connectable devices.
	ArdConnect:	-Commands Arduino to connect to device in range.

Table 12: Attributes

UC-9: Add-Device Domain Model

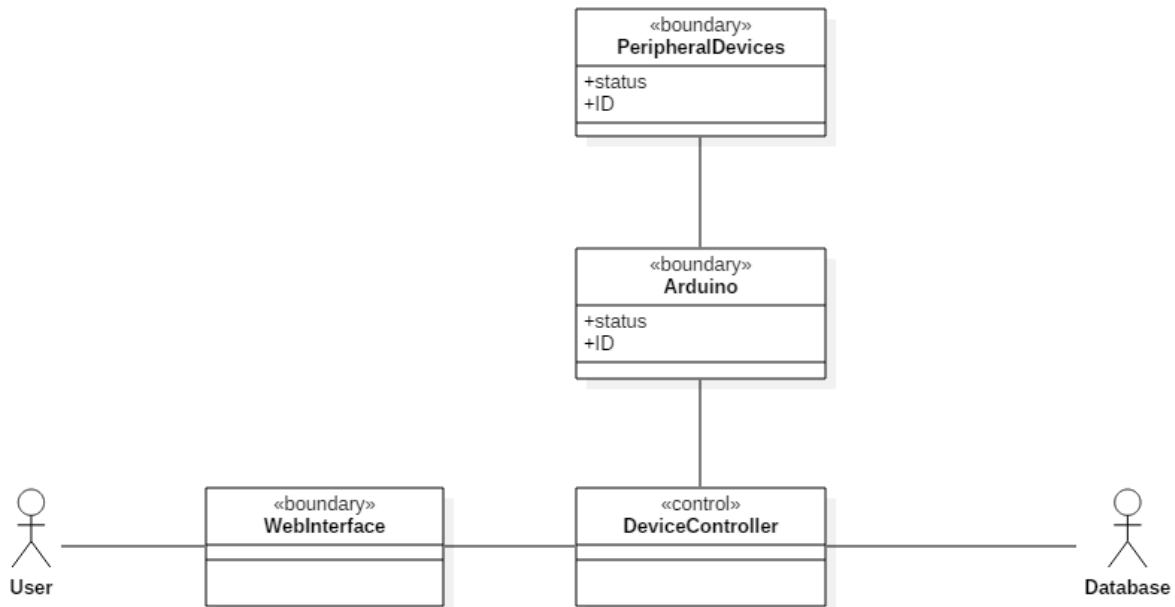


Figure 8: UC-9

Attributes:

Concept name:	Attributes	Responsibility Description:
DeviceControl	Controller	Application on user's device that connects to Arduino and sends device entry data to Database.
NotifyUser()	Web Interface	User Web interface that receives data from user and displays data from DeviceControl.
Interface page	Web Interface	Shows possible actions user can take on the User Web Interface. Shows the user which devices are currently connected and their current status
Arduino	CheckSensors() Controller	Sends data between DeviceControl to Device sensors.



Checksensors	Arduino	Arduino checks the signals sent from all the device sensors to notify the user of the device status
SendDeviceStatus	Autohome	Sends Device Sensor data to Autohome. Autohome device sensors send device status back to 24rduino so that the user eventually sees the current status of the devices
ArduinoSignal	Controller Arduino Web Interface	Web Interface acknowledges user's attempt to check device status and sends signal to the controller

Table 13: Attributes

Associations:

Concept pair	Association description:	Association Name:
Interface page <-> DeviceControl	Sends request to check signal to DeviceControl. Then, if device is found,shows user the current status of the device	UserMachineInteraction
DeviceControl <-> Arduino	DeviceControl prompts Arduino to search for devices and connect to them.	DeviceManagement
Arduino <-> Autohome	Arduino searches for active devices near it.	Arduino Search
Arduino <-> DeviceControl	Arduino gives device data to DeviceControl after search	Show devices
DeviceControl <-> NotifyUser	DeviceControl conveys the status of every device and sends any possible error messages that are encountered.	Write page
NotifyUser <-> Interface page	Shows page to user	UpdateStatus

Table 14: Associations

## Interaction Diagrams

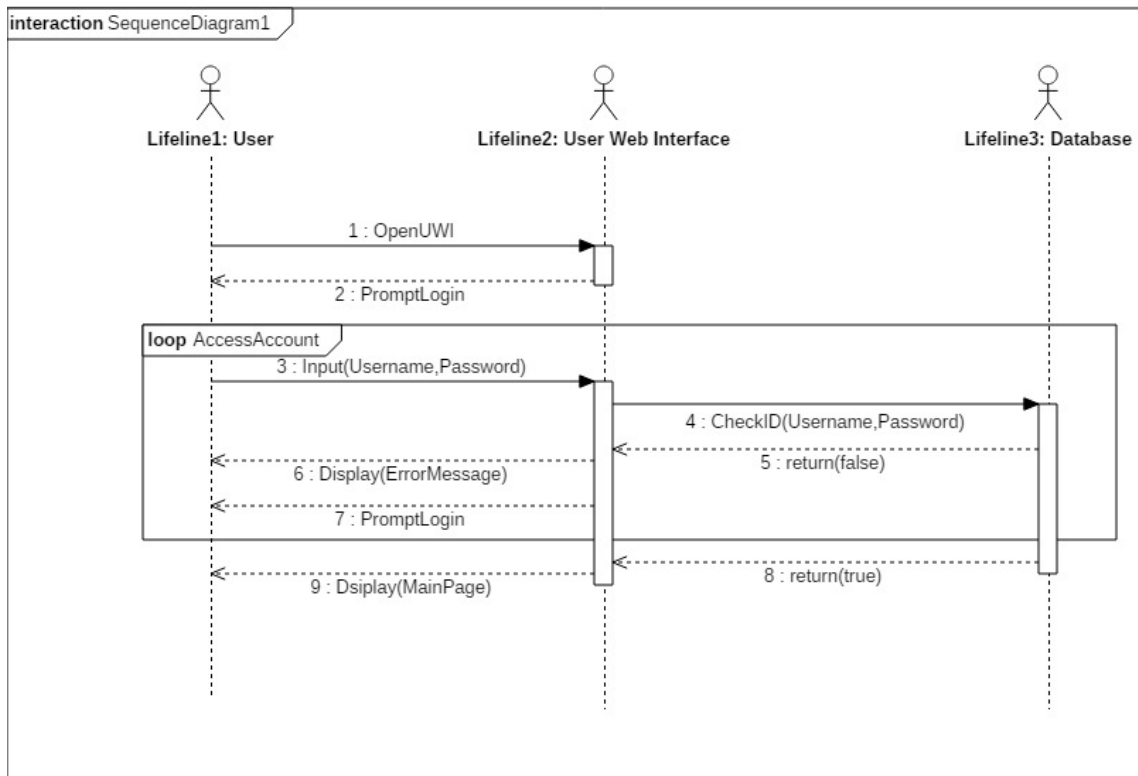


Figure 9: UC-1

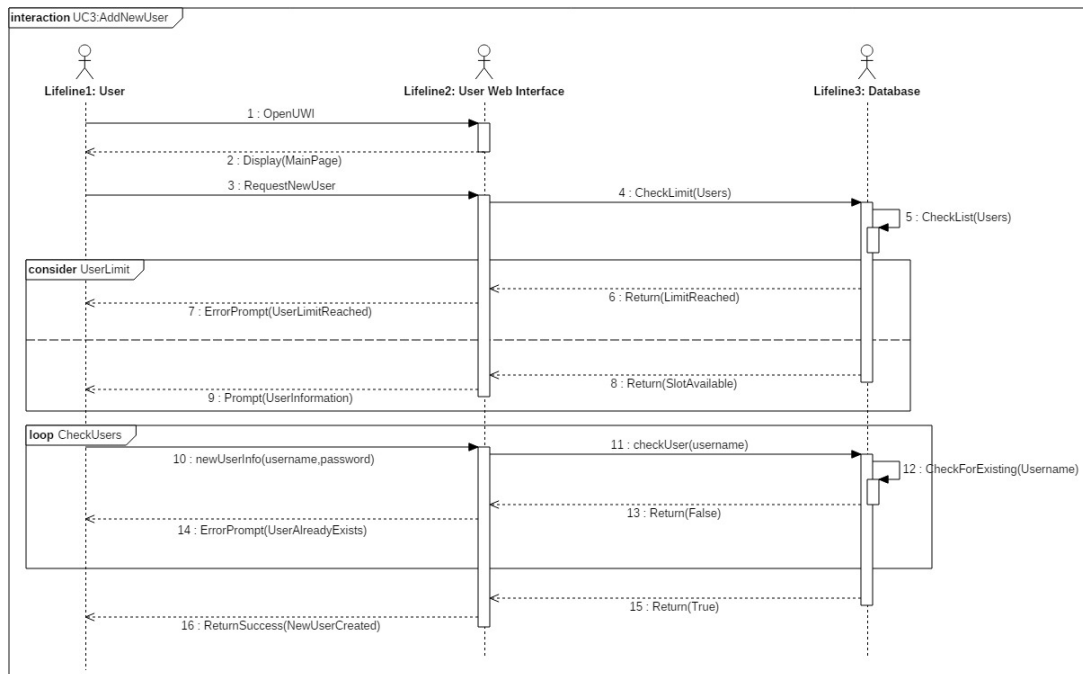


Figure 10: UC-3

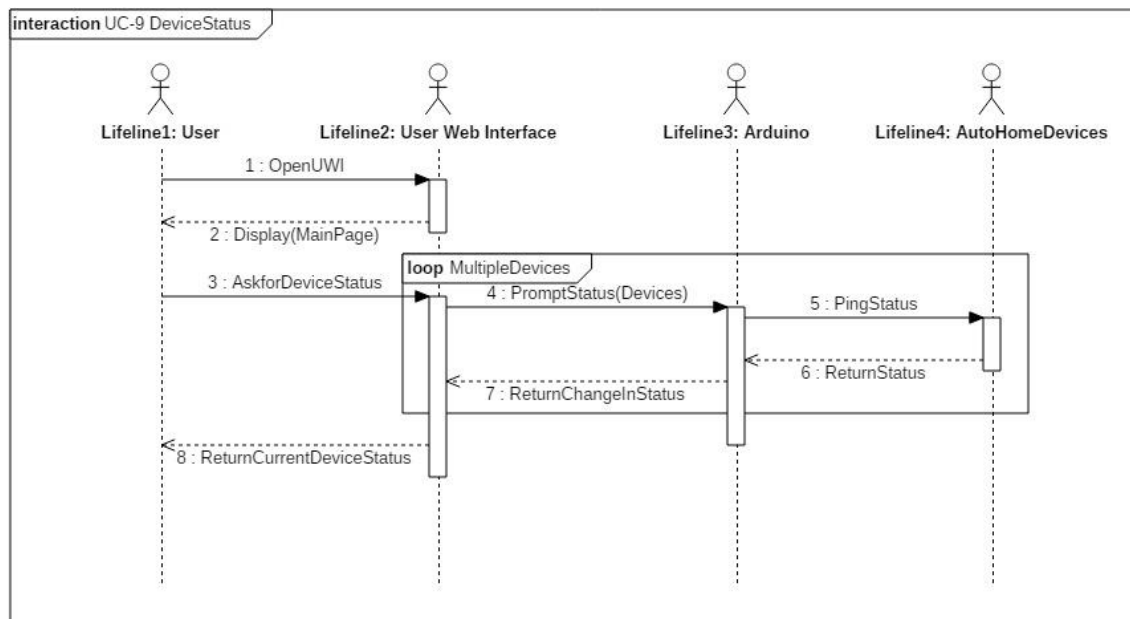


Figure 11: UC-9

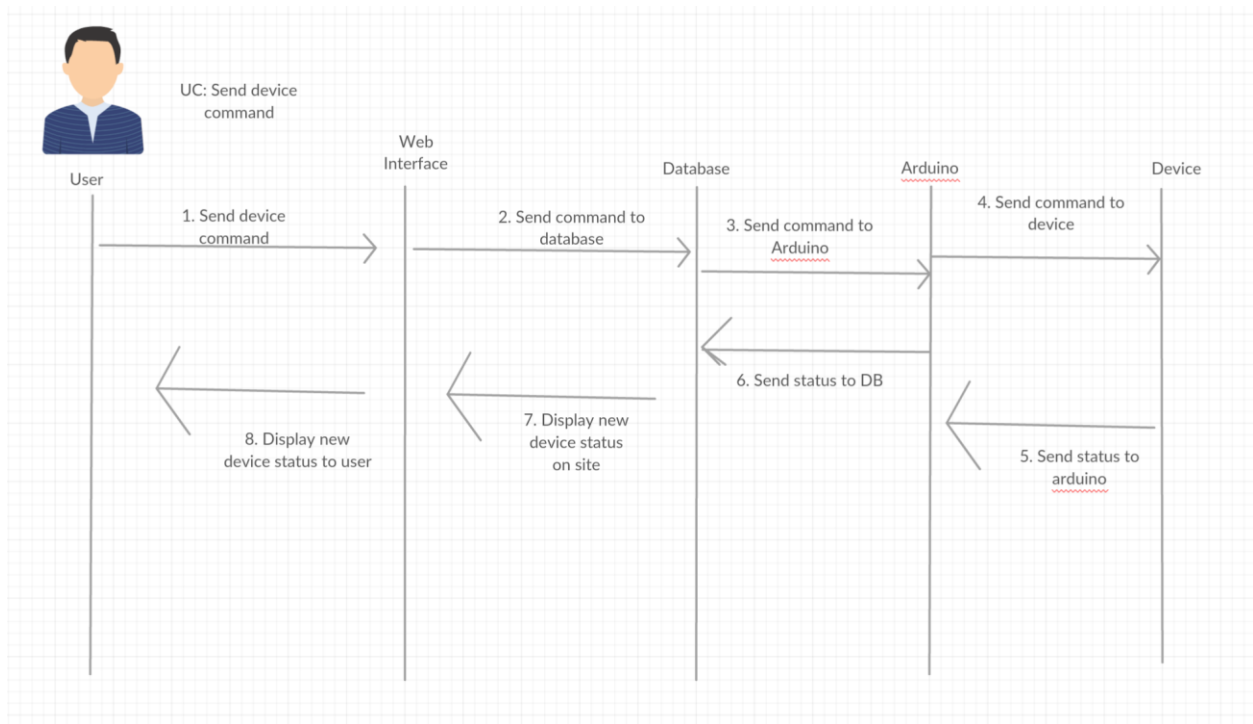


Figure 12: UC-Remote Device Command

## Class Diagram and User Specification

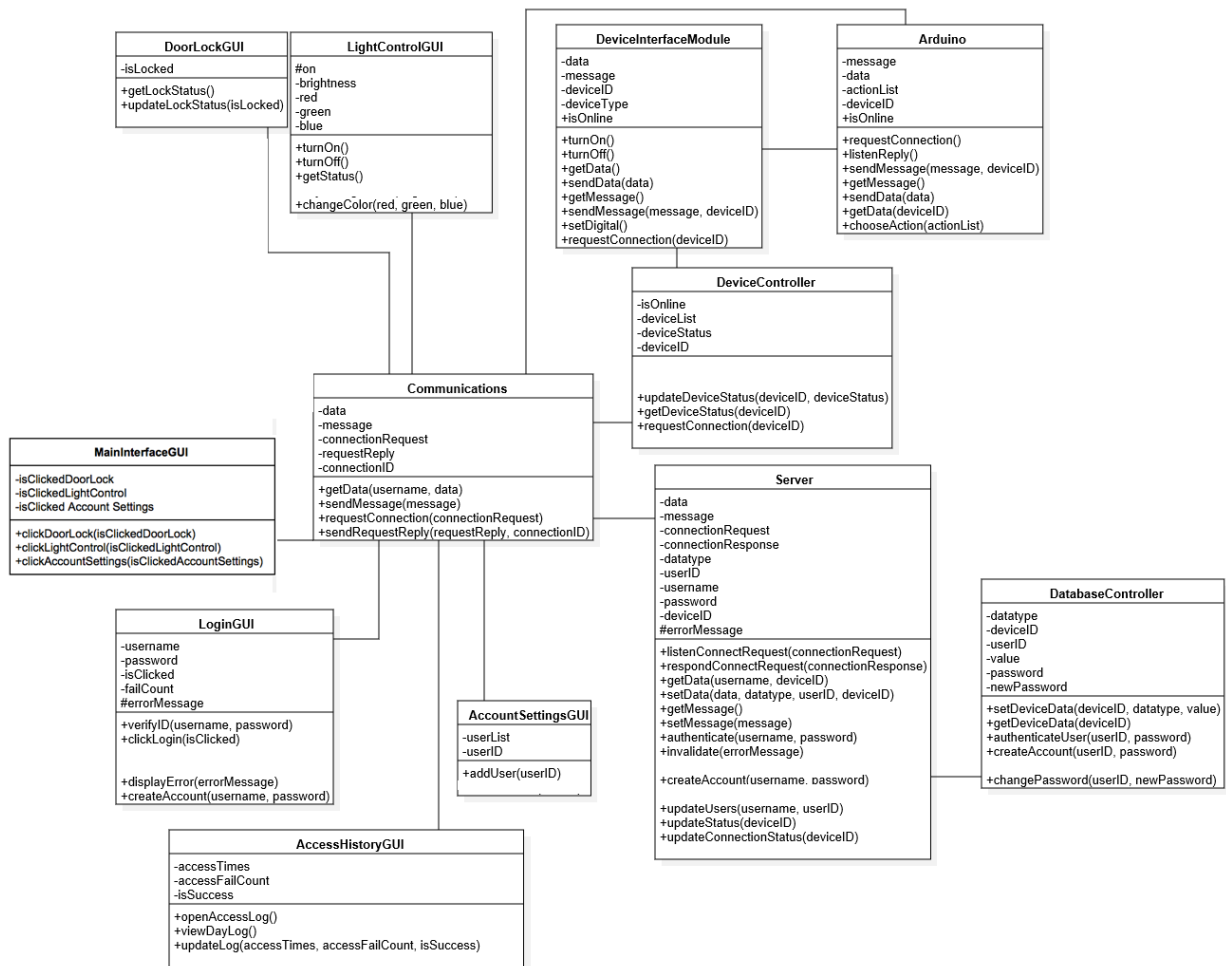


Figure 13: Class Diagram

There are a few things that we had to remove. These functions have to do with adding a device (can not support any more devices other than lights, and the door buzzer security system), deleting an account, access history, and thereby the corresponding warning that is sent to the user when his account is accessed with suspicious activity. The remainder is the same as before.

## Design Patterns

The Decorator Pattern is a good choice for AutoHome because AutoHome has various functions that aren't all dependent on each other. Decorator pattern allows one to separate essential functions from non-essential functions. In our case, the essential functions would be included within Communications, Server, and Database Controller. The “non-essential” functions in this case would be within AccountSettingsGUI, AccessHistoryGUI, DoorLockGUI, LightcontrolGUI, DeviceInterfaceModule, and DeviceController. Most of these functions do not necessarily rely on each other in order for them to work.

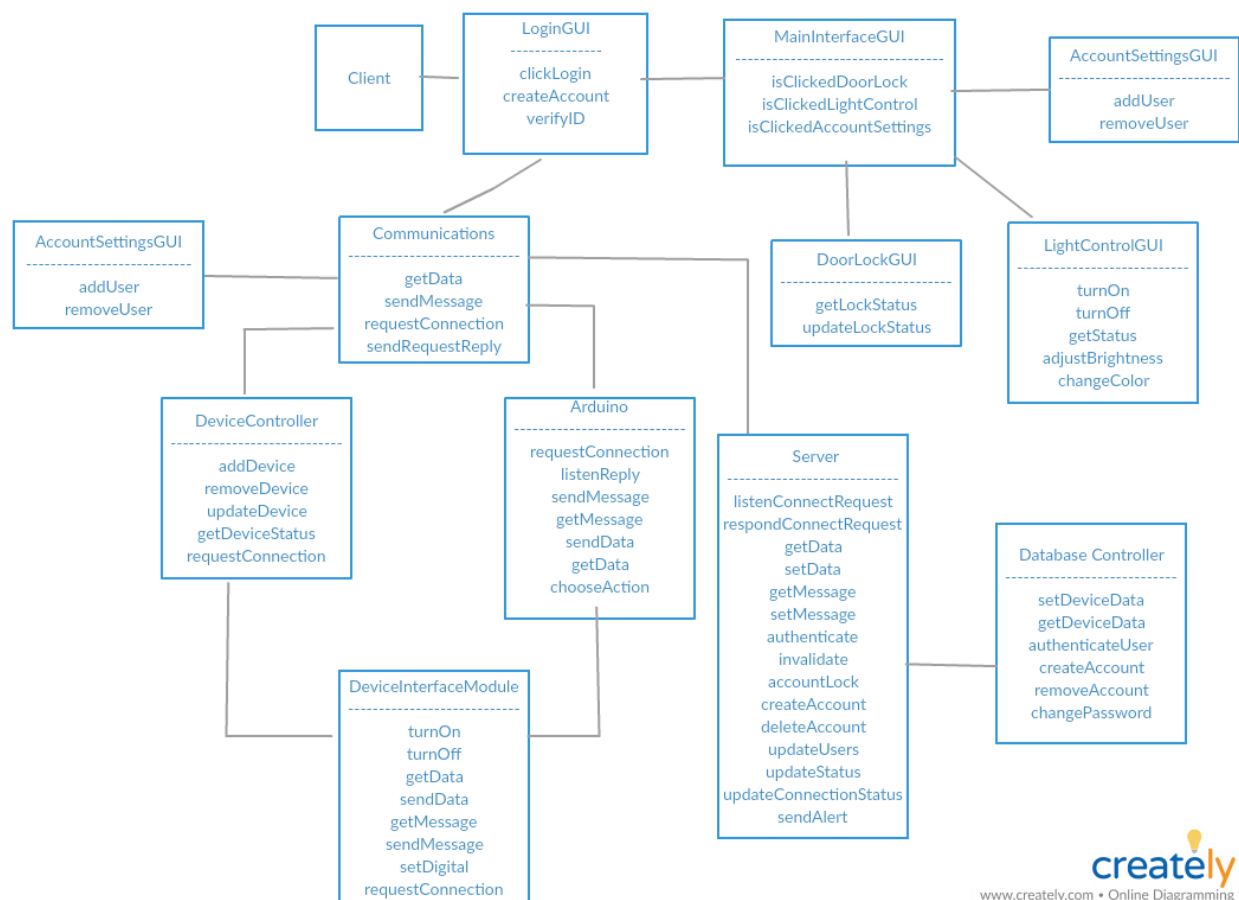


Figure 14: Pattern Design

## OCL Contracts

### Login GUI:

precondition:

access attempts and failed attempts should both be greater than or equal to 0

# of failed attempts should be less than or equal to number of access attempts

Invariant: username/password should be string

Postcondition: username/password combination should be valid

When creating an account, username should not be taken by any other user

New username & password should both be strings

### MainInterfaceGUI:

Preconditions:

Login GUI operations must complete before main screen is displayed

verifyID should run successfully

Invariant:

clickDoorLock

clickLightControl

clickAccountSettings

postconditions:

### Light Control GUI:

Preconditions:

MainInterfaceGUI

Invariants:

Switch (turnOn() and turnOff()) must be working with boolean values)

Post conditions:

### Door Lock GUI:

Preconditions:

MainInterfaceGUI

Invariants:

updateLockStatus should be working with boolean values

Post conditions:

### Communications:

Preconditions:

Message received from MainInterfaceGUI and server

Invariants:

No invariants because communications works with several different conditions and these can change every time

Post conditions:

request Reply is sent with valid connectionID

### **Server:**

Preconditions

message/instructions sent to the server from other parts in the system  
connectionRequest is received and

Invariants:

Valid connectionID, deviceID, and userID

Post conditions:

connectionResponse is sent back

### **Arduino:**

Preconditions:

requestConnection is sent from another device

Invariants:

isOnline must be true

Post conditions:

sendMessage is done

chooseAction is accomplished

### **AccessHistoryGUI:**

Precondition:

accessTimes must be greater than or equal to 0

accessFailCount must be greater than or equal to 0

accessFailCount must be less than or equal to accessTimes

Invariants:

accessTimes and accessFailCount should both be integers

Post conditions:

updateLog should update with accessTimes, accessFailCount and isSuccess with only int values

## System Operation Contracts

**Name:** Lock

**Responsibilities:** Log out and lock access to user's personal account

**Use Case(s):** UC-2

**Exceptions:** None

**Preconditions:**

- User is logged into the system

**Post conditions:**

- User is logged out of the system
  - Log out is logged in the system
- 

**Name:** Unlock

**Responsibilities:** Log the user into their personal account

**Use Case(s):** UC-1

**Exceptions:** None

**Preconditions:**

- User has account in the system

**Post conditions:**

- User is logged into the system
  - Log in is logged in the system
- 

**Name:** Add user

**Responsibilities:** Add a new user profile into the system

**Use Case(s):** UC-3

**Exceptions:** Only users with administrator privileges can do this

**Preconditions:**

- Username to be added is not already in the system
- Person adding user has administrator privileges

**Post conditions:**

- New user is added
- Addition is logged in the system



---

**Name:** Remove user

**Responsibilities:** Remove a user profile from the system

**Use Case(s):** UC-4

**Exceptions:** Only users with administrator privileges can do this

**Preconditions:**

- User to be removed is not logged in
- Person adding user has administrator privileges

**Post conditions:**

- User is removed
  - Removal is logged in the system
- 

**Name:** Access history

**Responsibilities:** Allow administrator is see status/history of devices connected to system and user history

**Use Case(s):** UC-5

**Exceptions:** Only users with administrator privileges can do this

**Preconditions:**

- User trying to access history has administrator privileges

**Post conditions:**

- History access is logged into the system
- 

**Name:** Lock Monitor

**Responsibilities:** Check the status of door lock

**Use Case(s):** UC-6

**Exceptions:** None

**Preconditions:**

- None

**Post conditions:**

- Access history is updated whenever the status of the sensor is changed
-

**Name:** Monitor Device

**Responsibilities:** Monitor the status of all devices connected to the system

**Use Case(s):** UC-7

**Exceptions:** None

**Preconditions:**

- User is logged into the system

**Post conditions:**

- None
- 

**Name:** Add device

**Responsibilities:** Add additional devices to monitor through the system

**Use Case(s):** UC-8/UC-16

**Exceptions:** None

**Preconditions:**

- None

**Post conditions:**

- System is updated with the new device
- 

**Name:** Device Status

**Responsibilities:** Periodically update the database with its status

**Use Case(s):** UC-9

**Exceptions:** None

**Preconditions:**

- Device has connection with the server

**Post conditions:**

- Device status is logged
- 

**Name:** Read device status

**Responsibilities:** Take database interface and display it to the user

**Use Case(s):** UC-10

**Exceptions:** None

**Preconditions:**

- Device has connection with the server

**Post conditions:**

- Device status is logged
- 

**Name:** Light off

**Responsibilities:** Turn the lightbulb off

**Use Case(s):** UC-11

**Exceptions:** None

**Preconditions:**

- Controller has connection with server

**Post conditions:**

- Device status is logged
- 

**Name:** Light on

**Responsibilities:** Turn the lightbulb on

**Use Case(s):** UC-12

**Exceptions:** None

**Preconditions:**

- Controller has connection with server

**Post conditions:**

- Device status is logged

## Classes/Domain Traceability Matrix

[illegible]

R19	x	x			x							
R20		x										
R21		x										
R22		x										
R23	x	x			x						x	
R24	x	x			x						x	
R25	x	x			x						x	x
R26												x
R27			x								x	
R28	x	x			x							
R29	x						x				x	
R30	x						x				x	

Table 15: Traceability Matrix

## System Architecture and Design

Because AutoHome has multiple functions and capabilities, each function is built upon different system architectures.

- The client-server model for our user interface and support for multiple users
- The database model for data collection and monitoring
- The event-driven model for triggering actions

The client-server model allows our system to have multiple users. The user's clients can all communicate with the central server. Each client will have a unique key that is used to authenticate itself with the server, granting them access to the client's hardware. Each client, the Arduino, will connect with the main server running on a Raspberry Pi. The client will be able to react to HTTPS GET requests to be able to control the connected devices.

The database model allows to AutoHome to collect and log data from the devices connected to the system. This data can be later access from a user account that has administrator privileges.

Via an SQL database, system is able to keep track of which systems have been turned on or off at what times.

The event-driven model applies to AutoHome's monitoring system. With an event-driven model, the specific actions that the sensors generate are not what trigger the system. The sensors first generate event notifications which are then sent to the server for processing. When the server receives an event notification, the server will process the notification, then take appropriate actions. This model corresponds to our door sensor, which sends an event to our server to log the action whenever the door opens or closes.

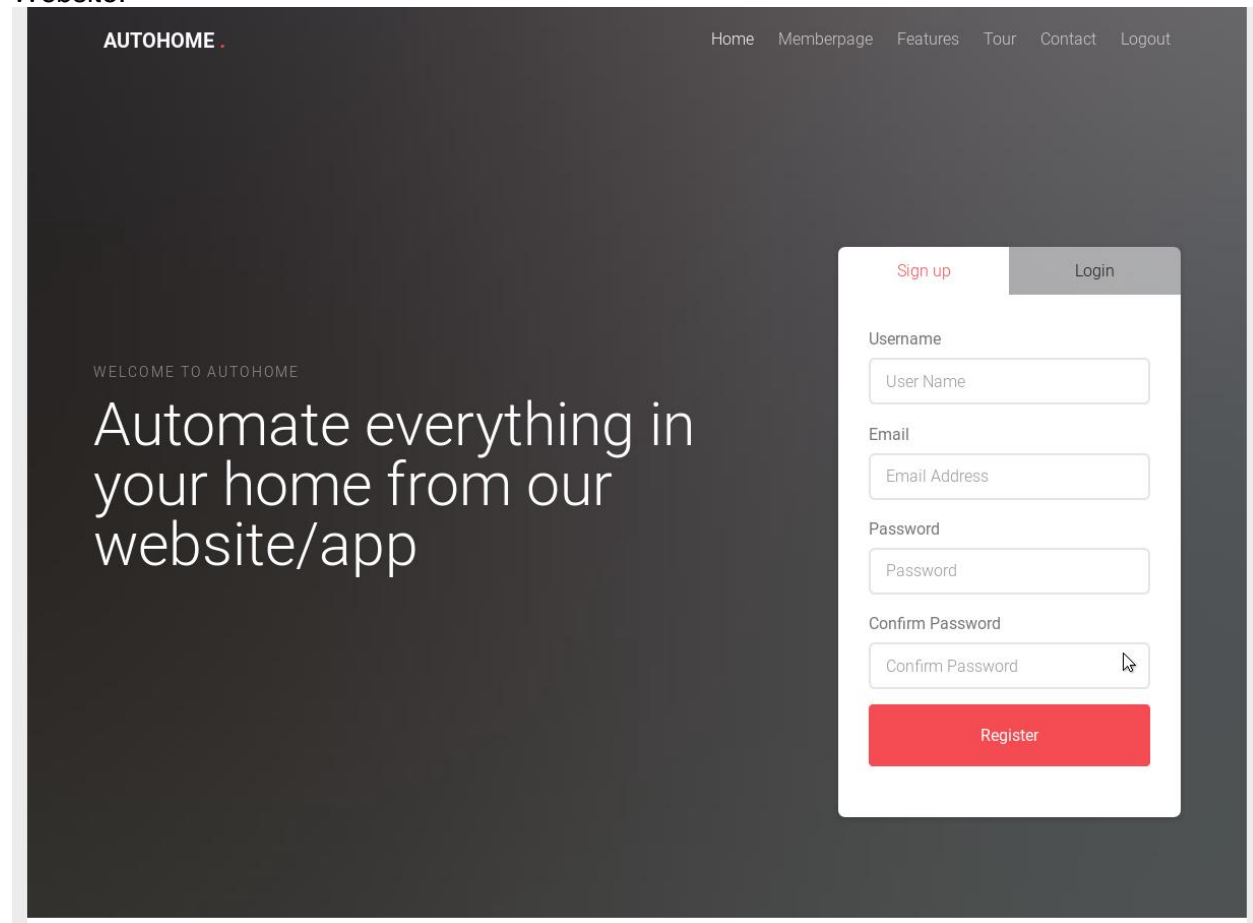
## Algorithms and Data Structures

Our software does not utilize any algorithms; however, we do utilize data structures. Since our entire project is centered around properly organizing user and device data and being able run scripts to properly utilize this data in a meaningful way, the way we organize our data is very important. We do not use any object-oriented data structures, however, since our project follows a client server model, we are also able to utilize a database located on the servers end. Thereby, we utilize MySQL in addition with PHP to be able to properly implement most of our features. Utilizing server-side scripting (PHP) in addition with SQL statements, one is able to build an entire web platform in which users can data pertaining to them (device data), and can even issue commands remotely to the device (achieved through server side scripting on both the web clients end, as well as on the Arduino's end).

## User Interface Design and Implementation

The webpage has a similar design to the app in that it is simplistic, yet user friendly and easy to use. The layout is similar with buttons that correspond to the buttons used in the App. Below is a mock up for our webpage. The webpage is designed to allow ease of access to user's devices.

Website:



The image is a mockup of a website for 'AUTOHOME'. The header is dark grey with the 'AUTOHOME' logo on the left and navigation links (Home, Memberpage, Features, Tour, Contact, Logout) on the right. The main content area has a dark background. On the left, it says 'WELCOME TO AUTOHOME' and 'Automate everything in your home from our website/app'. On the right, there is a white sign-up form. The form has two tabs: 'Sign up' (active) and 'Login'. It contains four input fields: 'Username' (with placeholder 'User Name'), 'Email' (with placeholder 'Email Address'), 'Password', and 'Confirm Password' (with placeholder 'Confirm Password'). A red 'Register' button is at the bottom of the form. A mouse cursor is hovering over the 'Confirm Password' field.

Figure 15: Home page

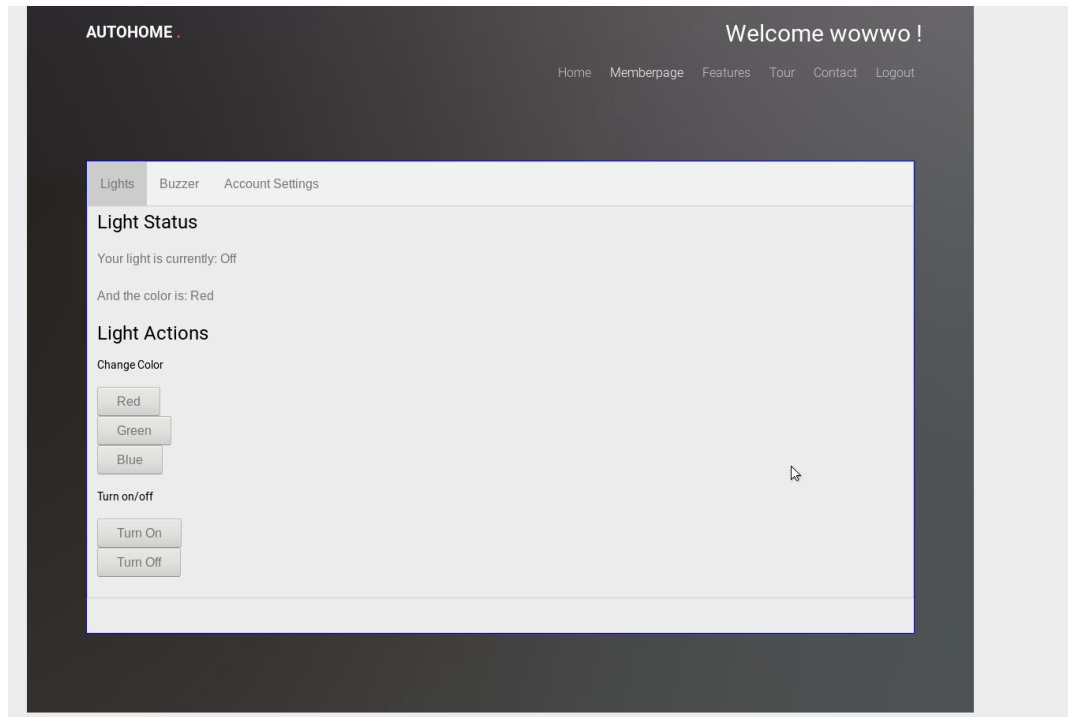


Figure 16: Light Control Page (membership page)

The members page allows a user to access the full functionality of our service through a website application. The website lets the user perform all tasks as outlined by use cases.

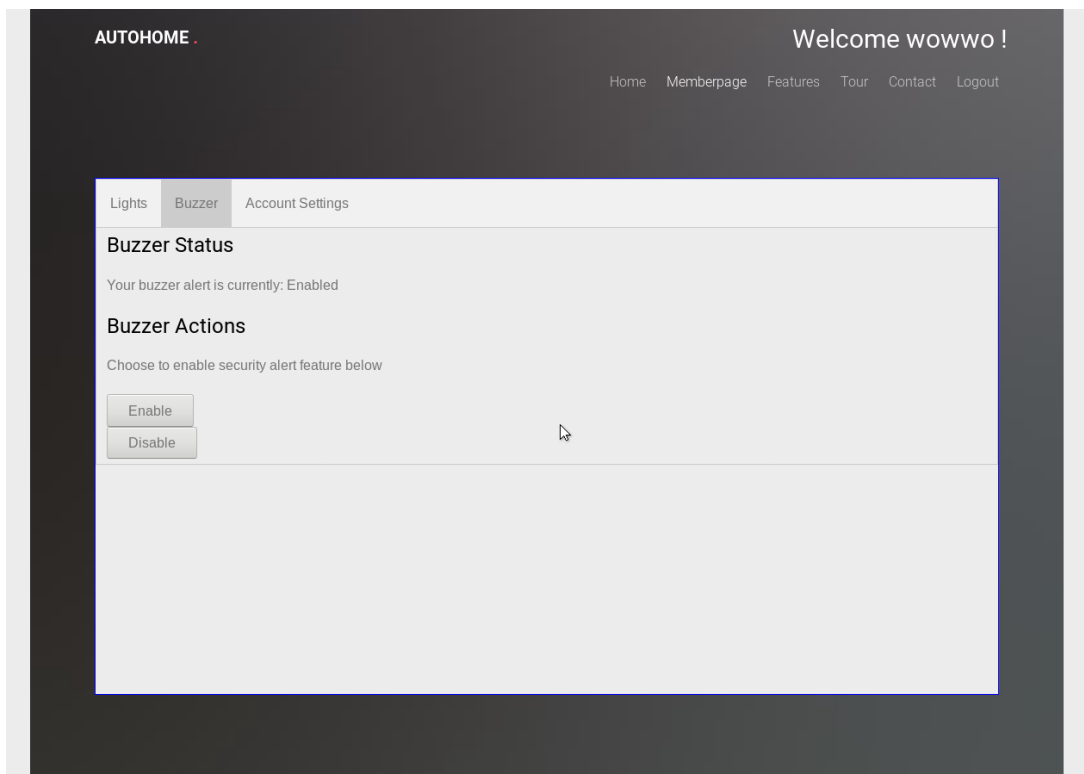


Figure 17: Buzzer Control Page (membership page)



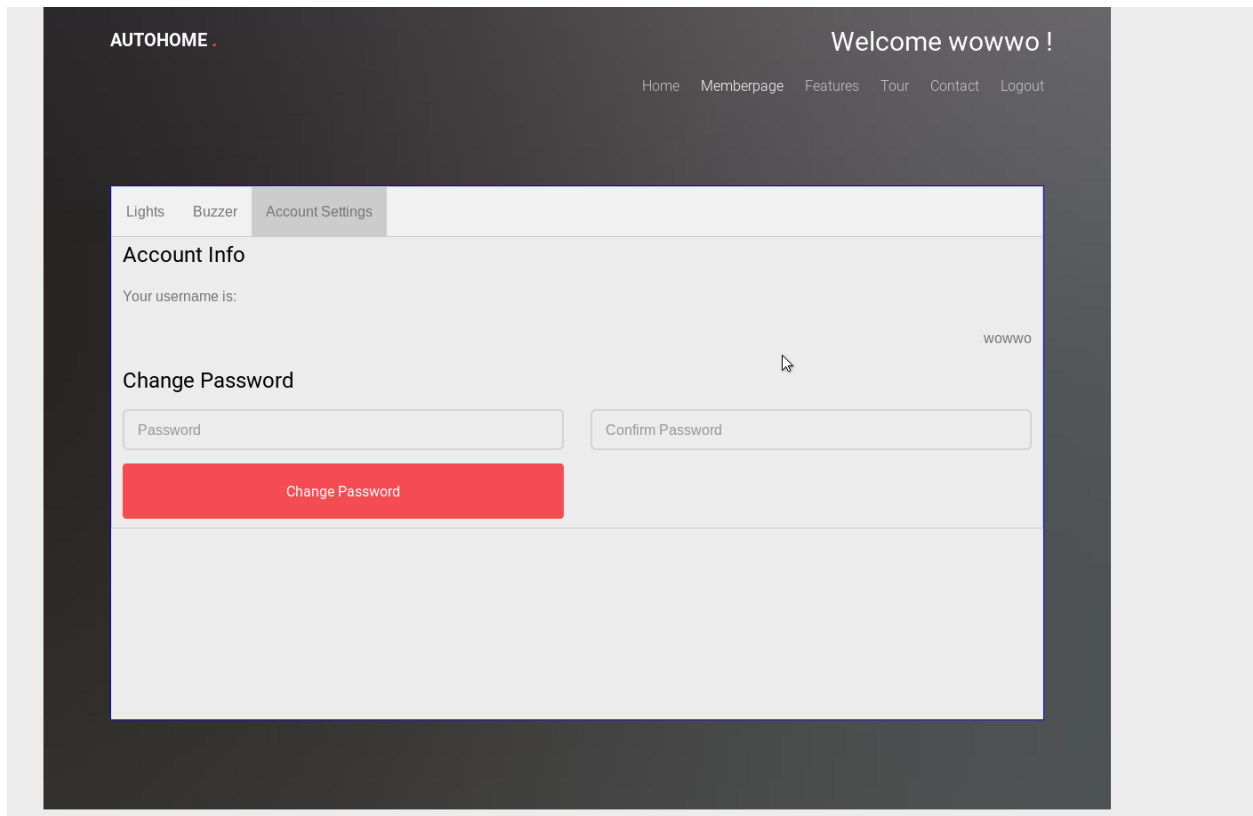


Figure 18: Account Settings Page (memberpage)

Once you login, you are automatically redirected to memberpage (you can only access if you're logged in). From here the user can properly view each of their devices statuses. They are also able to remotely control these devices and can see the updated status immediately afterwards. They can also change their password in the provided account settings tab.

## UI Mobile Design

Our initial mobile user interface design, although appealing, lacked ease-of-use and was not entirely intuitive. Below is a picture of our old design.

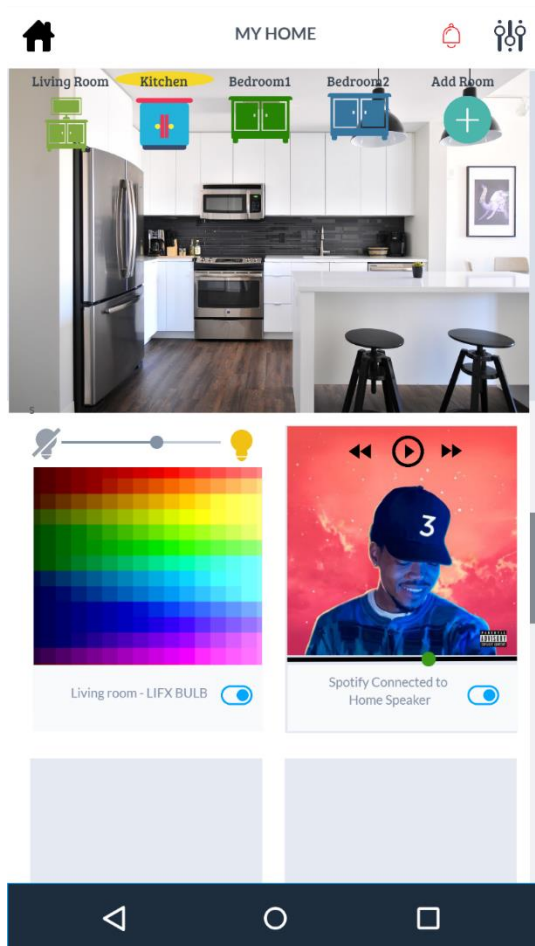


Figure 19: Old Mobile App UI

As you can tell from the new mockup of the home screen, it is greatly redone. Initially, our mockup was very flashy with lots of colors which may throw users off and confuse them with various different toggle bars and buttons on the page. The new design avoids all the flashy extra's which makes the interface a whole lot more user-friendly. The new design minimizes the amount of buttons on the screen which minimizes confusion.

In the new design, we still get all the same functionality as the previous design. Instead of having tabs for different rooms in the house, the new GUI home page looks very similar to our website. We decided that its better if the user gets a more unified experience and thereby we designed the mobile app to implement everything in the same exact manner as the website. This made the app easier to use, and fairly identical to the website so the user does not need to learn how to operate the web client AND operate the mobile app. Below is the new GUI of the mobile app.

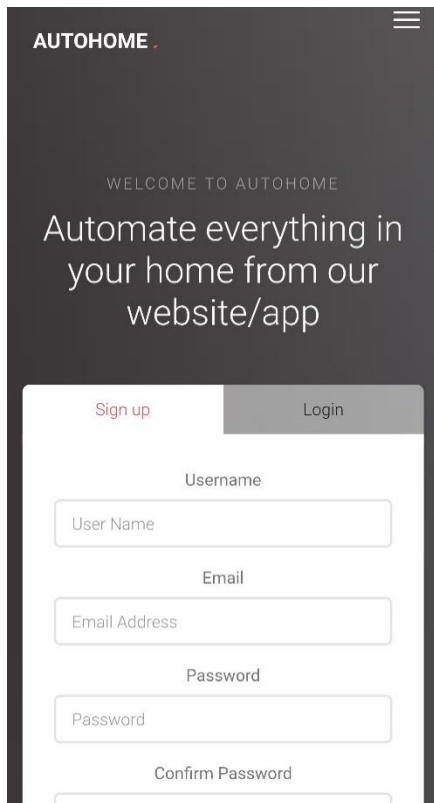


Figure 20: Main page

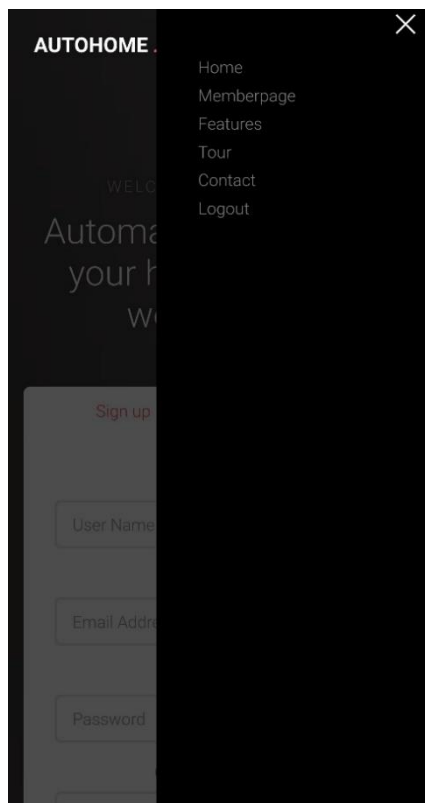


Figure 21: Navigation Bar

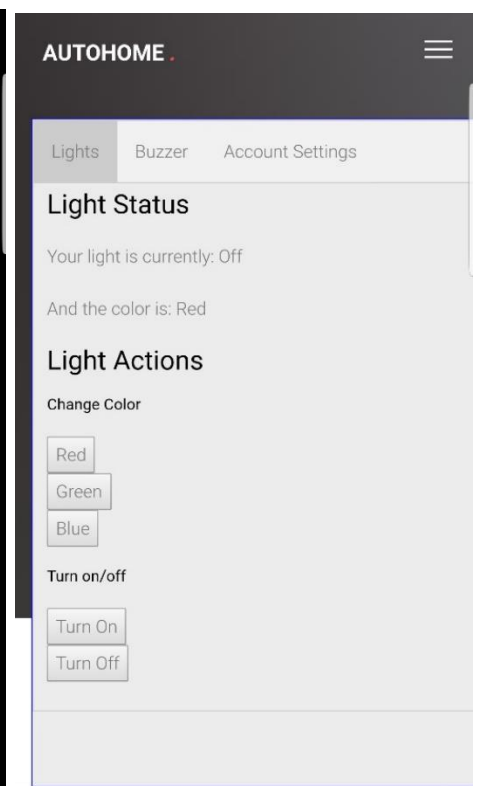


Figure 22: Device Control

## Design of Tests

The home automation system that we are building has a few central components with several supplementary features. The central software components include the code for the WiFi module, the user-web interface, the database, the server and the communication between all components of the system. These parts are essential because they are interdependent. The failure of any one of these features causes the whole system to fail. Supplementary features include light control, music control, monitoring of door locks, and system control using a mobile application. These features are supplementary because they can fail without affecting other parts of the home automation system. It is important to keep in mind that even though we call these features supplementary, this does not mean these features are unimportant.

Following this distinction, the way we design the acceptance tests for our home automation system follows two phases of “Bottom Up” integration testing. In the first phase, we perform unit tests on the central components. Once the system passes the unit tests, individual components are put into small groups and tested together. We continue combining groups until we test each central component as a whole and finally perform integration testing as we test the components together. After the central components have passed the integration test phase, we begin the second phase. This phase begins with unit tests on the supplementary features. Following the same procedure that was used to test the central components, these supplementary features are tested from the bottom upwards. After each supplementary feature has passed its own “Bottom Up” testing, it is tested with the system of central components in integration testing. Although it was previously stated that the second phase of testing starts after the first phase, we can perform the unit tests for both phases concurrently.

The section below shows plans for unit tests of each class shown in the class diagram. We start by isolating individual functions in a class for testing.

## Unit Tests

### DeviceController

#### updateDeviceStatus(deviceID, deviceStatus)

Success:

- The device controller sends a message to the server via communications. This message asks the database to update the status of the specific user’s device.

Failure:

- Some or all of the message sent is lost during communications.
- The message is sent and received, but the wrong user account is updated.
- The message is sent and received, but the wrong device is updated.
- The message is not sent.

### getDeviceStatus(deviceID)

#### Success:

- The device controller gets the status of a device from the Arduino via data from communications.

#### Failure:

- The device controller gets the correct device, but the wrong status.
- The device controller gets the status of the wrong device.
- The device controller gets a null status for a device that exists in the user's account.
- The device controller gets a status for a device that does not exist in the user's account.

### requestConnection(deviceID)

#### Success:

- The device controller receives a connection request from the Arduino via the device interface module. The controller then sends the request to server via communications and waits for a boolean reply. The function returns the boolean reply.

#### Failure:

- The controller receives a connection request, but does not send it.
- The controller sends a connection request when no request was received.
- The controller receives and sends a connection request to server, but never receives a reply and is stuck waiting.
- The reply is not boolean.

## **DeviceInterfaceModule**

### turnOn()

#### Success:

- The user presses a button on the device interface to turn on a specific device. The button changes appearance to indicate to the user its current status. The user's input is sent to the Arduino and the device corresponding to the button pressed turns on.

#### Failure:

- The user presses a button to turn on a specific device and the signal is not sent to the Arduino.

- The user presses a button to turn on a specific device and the signal is sent to the Arduino, but the device does not turn on.
- The user presses a button to turn on a specific device and the signal is sent to the Arduino, but the wrong device turns on.
- The button does not change appearance after being pressed.

#### turnOff()

##### Success:

- The user presses a button on the device interface to turn off a specific device. The button changes appearance to indicate to the user its current status. The user's input is sent to the Arduino and the device corresponding to the button pressed turns on.

##### Failure:

- The user presses a button to turn off a specific device and the signal is not sent to the Arduino.
- The user presses a button to turn off a specific device and the signal is sent to the Arduino, but the device does not turn off.
- The user presses a button to turn off a specific device and the signal is sent to the Arduino, but the wrong device turns off.
- The button does not change appearance after being pressed.

#### getData()

##### Success:

- The device interface module retrieves data from the Arduino.

##### Failure:

- The Arduino sends data, but the device interface module cannot or does not retrieve the data.
- The Arduino does not send data, but the device interface module retrieves data that is not null from the Arduino.
- The Arduino has data and the device interface module can retrieve data from the Arduino, but the data is corrupted.

#### sendData(data)

##### Success:

- The device interface module sends data to the Arduino.

Failure:

- The device interface module sends data, but the Arduino cannot or does not receive the data.
- The device interface module does not send data, but the Arduino receives data.
- The device interface module sends data and the Arduino receives data, but this data gets corrupted.

#### getMessage()

Success:

- The device interface module gets a message from a queue of messages sent by the Arduino. This message may be code, a signal or an error message.

Failure:

- The message retrieved is corrupted.
- The device interface module gets a message with content when the message queue is empty.
- The device interface module does not get a message when there are messages in the message queue.

#### sendMessage(message, deviceID)

Success:

- The device interface module sends a message to a queue of messages for the Arduino. This message should only be code.

Failure:

- The message sent is not code.
- The message is not queued.
- The message is not sent.

#### setDigital()

Success:

- The device interface module sets a digital value as a parameter for the Arduino. This digital value is used to change a device's output.

Failure:

- The device interface module does not pass the digital value to the Arduino.
- The device interface module passes the digital value to the Arduino, but the device's output does not change.
- The device interface module passes the digital value to the Arduino and the device's output changes, but not in the correct manner.
- The device interface module passes anything that is not a digital value to the Arduino.

#### requestConnection(deviceID)

##### Success:

- The device interface module passes a connection request from the Arduino to the device controller.

##### Failure:

- The request from the device interface module is not sent to the device controller even though a request was sent by the Arduino.
- The request is sent, but not received.
- The request is sent with the wrong device ID.
- The request is sent to the Arduino.
- A request is sent to the device controller when no request was sent from the Arduino.

## **AccountSettingsGUI**

#### addUser(userID)

##### \_\_Success:

- Input the userID to add a user to the home automation system. User is successfully added

##### Failure:

- userID is not recognized
- userID is not in string form
  - Both of these cases would not allow the method to proceed, and would lead to failure as a result
- userID is recognized but user is not showing up in the system

## **DoorLockGUI**

#### getLockStatus():

##### \_\_Success:



- Gets lock status from the physical lock via communications from Arduino

Failure:

- Gets incorrect lock status
- Fails to get lock status from arduino

updateLockStatus(isLocked):

\_\_Success:

- updates status after checking the isLocked value

Failure:

- LockStatus is updated to an incorrect value because of value isLocked
- isLocked is read incorrectly thus updateLockStatus is incorrect
- isLocked is not read, therefore lock status is not updated

## LightControlGUI

turnOn():

\_\_Success:

- User clicks button to turn light on, and light is turned on

Failure:

- User clicks button and light doesn't turn on
- User doesn't click button and light turns on

turnOff():

\_\_Success:

- User clicks button to turn light off, and light turns off

Failure:

- User clicks button and light stays on
- User doesn't click button and light turns off

getStatus(): [gets status of light]

\_\_Success:

- Gets the current status of light and correctly shows the status to the user
  - If light is on, the user is told that the light is on
  - If light is off, user is informed that light is off

Failure:

- Displays incorrect status of the light

adjustBrightness(brightness): uses brightness int to adjust to that specific brightness

\_\_Success:

- Takes int brightness and sets the brightness to that specified number.

Failure:

- Misreads int brightness and displays incorrect brightness
- Doesn't read int brightness and doesn't change brightness

changecolor(red, green, blue): change color based on intensity of each of the given variable values

\_\_Success:

- Reads values of variables red, green, and blue and adjusts color based on the intensity of each color

Failure:

- Fails to read or incorrectly reads one or more of the input colors, thus displaying an incorrect color
- Doesn't read one or more int and doesn't display the colors
- Displays color when the color values haven't been specified

## **MainInterfaceGUI**

clickDoorLock(isClickedDoorLock):

Success:

- DoorLock button clicked and signal was recognized → user is taken to door lock control

Failure:

- Button clicked but not read → user is not taken to door lock control
- Button not clicked and is incorrectly read → user is taken to door lock control as an error

clickLightControl(isClickedLightControl):

\_\_Success:

- LightControl button is clicked and signal is recognized → user taken to light control

Failure:

- Button clicked but not read → user is not taken to light control

- Button not clicked and is incorrectly read → user is taken to light control as an error

#### clickAccountSettings(isClickedAccountSettings):

##### Success:

- AccountSettings button is clicked and signal is recognized → user is taken to account settings

##### Failure:

- Button clicked but not read → user is taken to account settings as an error
- Button not clicked and is incorrectly read → user is not taken to account settings

## LoginGUI

#### verifyID(username, password)

##### Success:

- The user enters in his username and password. The LoginGUI passes this information to function where it verifies and matches every character to its corresponding table in the database, if both username and password matches, the User is able to login

##### Failure:

- The user enters in his username in password. The LoginGUI passes this information to the function where it verifies and matches every character to its corresponding table in the database, if just one character does not match, the user will not be able to login and it will be recorded as a fail attempt.

#### clickLogin(isClicked)

##### Success:

- The user clicks on the 'Login' button, the LoginGUI should recognize this, and upon verification of the ID, will allow the user to move to the next page

##### Failure:

- The user clicks on the 'Login' button, the LoginGUI should recognize this, if the ID is not verified successfully, the user will not be allowed to move and a fail count will be recorded
- The user clicks on the 'Login' button, the LoginGUI does not recognize this.

#### getFailCount()

Success:

- The LoginGUI will retrieve the number of failed login attempts from the database

Failure:

- The LoginGUI will attempt to retrieve the number of failed login attempts from the database, and the failcount is unable to be retrieved
- The LoginGUI cannot communicate with the database

#### setFailCount(FailCount)

Success:

- The LoginGUI will increase the fail count by one everytime the verifyID function fails

Failure:

- The LoginGUI will fail to increment the fail count by one even if the verifyID function failed.

#### displayError(errorMessage)

Success:

- The LoginGUI will cout/print a error message to the user depending on the type of failure

Failure:

- The LoginGUI will fail to print out a message to alert the user, and instead, nothing will be displayed

#### createAccount(username, password)

Success:

- The LoginGUI will send the new username and password to the database where it will be stored, then a message will be displayed to the user to relay that account creation was successful

Failure:

- The LoginGUI is unable to communicate to the database and is unable to store the new username and password

- The LoginGUI sends the new username and password to the database but the database is unable to store the new user account.

## Arduino

### requestConnection()

Success:

- Arduino requests connection to the main server, and accepts signals to control the connected systems after a successful connection.

Failure:

- Connection is unsuccessful due to server being offline.
- Connection is unsuccessful due to the Arduino not having internet connection

### listenReply()

Success:

- Arduino gets reply from server and successful acts upon it.

Failure:

- Arduino does not get a reply due to no connection to the internet
- Arduino does not get a reply due to the listening period timing out.

### sendMessage(message, deviceID)

Success:

- Arduino sends a command to the specified device and device performs the action

Failure:

- The message is invalid and the signal is meant for a different device
- The deviceID is invalid and is not a connected device

### getMessage()

Success:

- Arduino receives device's message

Failure:

- Device is offline and no message is received by the Arduino

#### sendData(data)

Success:

- Arduino sends device data to server

Failure:

- Arduino does not have internet connection and data is not sent to the server
- Server is not online and data is not able to receive data

#### getData(deviceID)

Success:

- Arduino gets device status and data from the specified deviceID

Failure:

- The specified deviceID is invalid and no data is received
- Specified device is not currently connected and no data is received

#### chooseAction(actionList)

Success:

- Arduino receives a valid action and sends it to the device

Failure:

- Inputted actionList is not valid and no action is sent to the device

## **Communications**

#### getData(username, data)

Success

- The device interface module retrieves data from the Arduino as requested by the logged in user

Failure

- The Arduino sends data, but the device interface module cannot or does not retrieve the data.
- The Arduino does not send data, but the device interface module retrieves data that is not null from the Arduino.

- The Arduino has data and the device interface module can retrieve data from the Arduino, but the data is corrupted.

#### sendMessage(message)

Success:

- The device interface module sends a message to a queue of messages for the Arduino. This message should only be code.

Failure:

- The message sent is not code.
- The message is not queued.
- The message is not sent.

#### requestConnection(connectionRequest)

Success:

- The device interface module passes a connection request from the Arduino to the device controller.

Failure:

- The request from the device interface module is not sent to the device controller even though a request was sent by the Arduino.
- The request is sent, but not received.
- The request is sent with the wrong device ID.
- The request is sent to the Arduino.
- A request is sent to the device controller when no request was sent from the Arduino.

#### sendRequestReply(requestReply, connectionID)

Success

- Reply is send upon successful completion of the request along with the connection ID requesting the reply.

Failure

- The request is sent, but not received.
- The request is sent with the wrong device ID.

## Server

### listenConnectRequest(connectionRequest)

Success:

- Server receives request from Arduino to establish connection

Failure:

- Server receives signal, but connection

### respondConnectRequest(connectionResponse)

Success

- After server receives message from arduino, the server sends "connectionResponse" message to the arduino
- Message successfully received by arduino.

Failure

- "connectionResponse" message sent by server but not received by arduino
- Connection unsuccessful.

### getData(username, deviceID)

Success

- Data is sent successfully to the arduino from the server for the given username and device ID

Failure

- Username is not found in database.
- Device ID is not found in the database.

### setData(data, datatype, userID, deviceID)

Success

- Data is sent successfully from the arduino to the server.
- Data is saved under the Device ID and userID

Failure



- Data is sent but not received by the server
- Data is being sent with wrong device ID or userID

#### getMessage()

##### Success

- Request sent and message successfully received.

##### Failure

- Request sent but message not received.

#### setMessage(message)

##### Success

- Message sent and set.

##### Failure

- Message sent but not successfully received.

#### authenticate(username, password)

##### Success

- Username matches with the password and user is confirmed.

##### Failure

- Username and password are not a match, authentication failed

#### invalidate(errorMessage)

##### Success

- Error message sent and received by device.
- User receives “username or password incorrect” error message

##### Failure

- Error message sent but not received by device.

#### accountLock(username)

##### Success

- Account with username specified status switched to locked

##### Failure

- Lock request sent but not locked
- Username specified does not exist

#### createAccount(username, password)

#### Success

- Request sent by the user and is received successfully by the server
- New user is created in the database with username as password as specified

#### Failure

- Request sent by the user but not received by the server
- Request sent by user but the username specified already exists in the server
- Request is sent by user with a valid username but password does not meet requirements.

#### updateStatus(deviceID)

##### Success

- Request sent to update status and perform an action
- Device with device ID receives the status update message

##### Failure

- Request sent to update status but message not received
- deviceID specified does not exist

#### sendAlert(userID)

##### Success

- Request sent to user for Alert
- User with userID receives the alert message

##### Failure

- Request sent but message not received by user specified
- userID specified does not exist in server

## **DatabaseController**

#### setDeviceData(deviceID, datatype, value)

##### Success:

- The DatabaseController is able to successfully set the data type for each individual device. For example, RGB values for lights range from 0-255 and have a datatype of int. The door buzzer would be a boolean datatype for locked/open, etc.

Failure:

- The DatabaseController is unable to set datatypes and their values for different devices
- The DatabaseController assigns incorrect datatypes and values for devices.

#### getDeviceData(deviceID)

Success:

- The DatabaseController is able to successfully retrieve the data values for each device (correct data types, etc.)

Failure:

- The DatabaseController is unable to retrieve the data values for a device
- The DatabaseController retrieves data values for a device, but is an incorrect value, or an incorrect datatype.

#### authenticateUser(userID, password)

Success:

- The LoginGUI sends over a login attempt with parameters for a user and password. First the DatabaseController will search for the username, if there is a match, it will then compare the password entered to the password stored in the database, if this is a match as well, then authentication succeeds.

Failure:

- The LoginGUI sends over a login attempt with parameters for a username and password. The DatabaseController searches for the username but is unable to find one that matches in the database
- The LoginGUI sends over a login attempt with parameters for a username and password. The DatabaseController searches for the username and there is a match, unfortunately the password doesn't match so authentication fails.

#### createAccount(userID, password)

Success:

- The LoginGUI sends over new user account data to be stored into the database, the DatabaseController takes the argument parameters and successfully inserts it into the database.

Failure:

- The LoginGUI sends over new user account data, but unfortunately the database is unable to insert it into the database
- The LoginGUI is unable to communicate with DatabaseController in order to send over new user account data

changePassword(userID, newPassword)

Success:

- After the user is authenticated, the LoginGUI sends over a request to change the password. The DatabaseController searches for the username, if the username is found, then the password is successfully updated.

Failure:

- The user is not authenticated.
- The user is authenticated but the DatabaseController is unable to find the username in the database
- The user is authenticated and the DatabaseController is able to find the username, however it is unable to update the password.

## History of Work

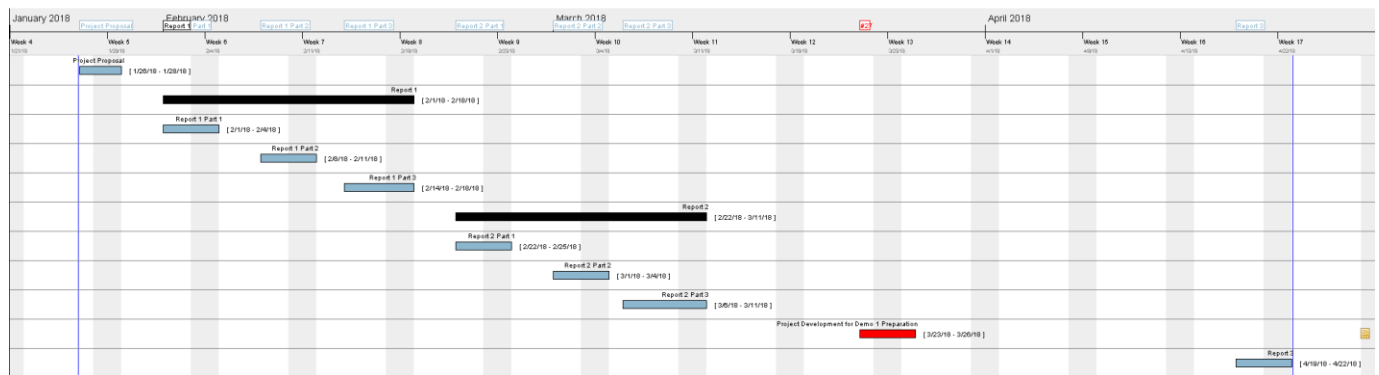


Figure 23: Gantt Chat (history of work)

Compared with our milestone deadlines that we set for ourselves in report 1 and report 2, we did not strictly adhere to the schedule because it was difficult to coordinate among 8 different students, especially considering we all had other difficult coursework to manage. This being said, our development efforts were not as consistent as we would have liked them to be, but we were able to come together as a group when it was truly necessary to implement our software successfully. In these last few weeks, there are only a few more things that need to be done, including polishing the mobile app and the web client, as well as fixing a couple server end problems.

### Key Accomplishments:

- Created Web Client/Mobile Application and connected it with our Database
- Wrote script commanding Arduino to turn on light bulb
- Wrote script commanding Arduino to change bulb color
- Wrote script in which Arduino would alert the user if a door is opened
- Created and host our own server to store website and database files on Raspberry Pi
- Created web platform to manage and control devices with server-side scripting (PHP + MySQL)
- Implemented rudimentary home security system with door sensor monitor
- Increased user security by methods of email verification, security questions, server security

## References

- [1] Home automation - <https://www.vectorsecurity.com/UserFiles/File/PDF/blog/Smart-Home-Interactive.pdf>
- [2] Arduino microcontroller - <https://www.arduino.cc/>, <https://www.elegoo.com/download/>, <https://www.linuxjournal.com/content/learning-program-arduino>
- [3] Version control system - <https://github.com/>
- [4] Software Engineering book - [http://eceweb1.rutgers.edu/~marsic/books/SE/book-SE\\_marsic.pdf](http://eceweb1.rutgers.edu/~marsic/books/SE/book-SE_marsic.pdf)
- [5] Android development - <https://medium.com/google-developers/developing-for-android-introduction-5345b451567c>, [http://www.ebookfrenzy.com/pdf\\_previews/AndroidStudioEssentialsPreview.pdf](http://www.ebookfrenzy.com/pdf_previews/AndroidStudioEssentialsPreview.pdf)
- [6] Smart home sensors - <https://www.smarthomeusa.com/plan-your-own-smart-home-system/>, <https://www.networkworld.com/article/2925722/security0/home-security-demystified-how-to-build-a-smart-diy-system.html>