SMART LOCK™

FINAL REPORT

Software Engineering - S'19 - Group #10

Web App: **https://smart-lock-demo.netlify.com/login.html**

Repository: **https://github.com/software-engineering-s19-group10**

MEMBERS:

Ted Moseley (trm124), Michael Truong (mt842), Daniel Nguyen (dnn21), Eric Lin (ekl40), Mohit Khattar (mk1483), Mohammad Nadeem (mn535), Andrew Sengupta (ans165), Jasjit Janda (jsj87), Jeffrey Lu (jl2088), Amandip Kaler (ak1415)

Table of Contents

0.1 Individual Contribution Breakdown

All members contributed equally on the final report.

1 Customer Statement of Requirements

1.1 Problem Statement

Imagine that you are coming home from a long day at work. On the way back you went to the grocery store. You reach your home and are carrying a multitude of bags in your hand. As you are heading towards your door, you try to fumble into your pocket to retrieve your keys. Unfortunately, because of the bags, your keys suddenly fall to the ground and into a sewer. Now, you have to call the locksmith to open your door or somehow get your keys out of the sewer.

Unfortunately, the case described is a very common and possible scenario. What if all of this could be curtailed? A good solution to this is creating an authentication and home security management system for your front door. Such a system will allow a homeowner to easily enter and exit their residence easily and even allow friends to enter their home.

A large majority of home security systems today are both expensive and complicated. Two improvements to home security developed in recent years, namely keyless locks and home security cameras, are both sold separately, and individually pricey. These systems attempt to modernize home security by providing a product that allows homeowners to monitor and control their home remotely while also increasing home security to allow entrance into the home. We would like a cost-effective system that integrates the keyless lock and home security camera solutions to home security, and use them to simultaneously improve upon a consumer's experience.

In order to rectify the aforementioned problems, homeowners need an easy-to-use home security system, where a video camera is used to both monitor the house and authenticate users using facial recognition, condensing the functionality of both the keyless lock and home security camera systems. Such a system will be an improvement on current home security systems by allowing homeowners easy entry into their homes while also allowing many other features.

Conventional locks rely on physical keys. However, these can either be easily lost or easily replicated which is a cause for concern. Smart locks, on the other hand, are much more secure because they use other means of authentication. Most current smart locks do not necessarily have an option for unlocking using facial authentication (see Ring). Rather, they rely on apps or pass codes which can be broken into relatively easily. Facial authentication, on the other hand, allows for the user to easily unlock the door.

Then, there exists the need for security for a homeowner. Using the camera, a user can monitor the home remotely by viewing the video feed from the camera. The system can also monitor movements and faces using facial and motion detecting. Current home security systems either require a subscription or charge an exorbitant amount of money for cloud storage of video. Our proposal is less costly for the user while providing a secure, user friendly system.

For the purposes of this project, the current plan is to use a computer webcam for the video camera, and the lock will be assumed to already have been set up such that the lock defaults to being locked, and unlocks upon the sending of an unlock signal. Residents of the household will be able to enter their house through facial authentication. Records of all authentications will be able to be monitored through the use of a web application tied to the camera, with the option to be notified in the event of suspicious activity. A user will be notified of events through SMS and/or push notifications through a (mobile) browser which is the best option to communicate effectively with the user.

Users will be classified into three categories. The first category consists of homeowners. Users in this category will have the option to add or remove authenticated users and monitor the camera. Homeowners can monitor the camera through a live video feed. They will also be able to see a list of all the people that entered the home within a given timeframe in the past. All of this shall be available to homeowners through a web app. They can opt in for notifications on their browser or through SMS.

The second category will consists of residents of a home. Note that this category does not include homeowners. These users will be able to unlock the door for an indefinite amount of time, but do not have the ability to monitor or control the lock itself. That is, residents of a home do not have administrative privileges over the lock.

The third category of users consists of visitors. Visitors will be able to temporarily unlock the door for either a determinate amount of times or until a given date.

When a homeowner wants to allow authentication for temporary visitors, the homeowner can login to the web app and add the visitor using the web application. The homeowner can add a picture for facial authentication of the visitor. Or, the homeowner can create a temporary URL that they can send to the visitor. The homeowner can also set an expiration date for the authentication of the visitor.

If an unauthorized or unrecognized visitor approaches the door, a notification will be sent to the homeowner. The homeowner can view a live video and choose to report the stranger if they perform suspicious activity. The neighbors will be notified about the events that suspicious visitors show up so everyone will be aware of potential robbers and other threats in the neighborhood. This smart lock not only keeps the home safe but the community as well.

For a more secure neighborhood, homeowners need a new network called the Stranger Reporting Network. This network allows homeowners to report suspicious activity that they detect in their area. Other users in the neighborhood can then view a heatmap of suspicious activity in the neighborhood. This feature will be very useful in getting homeowners to be more vigilant to protect homes.

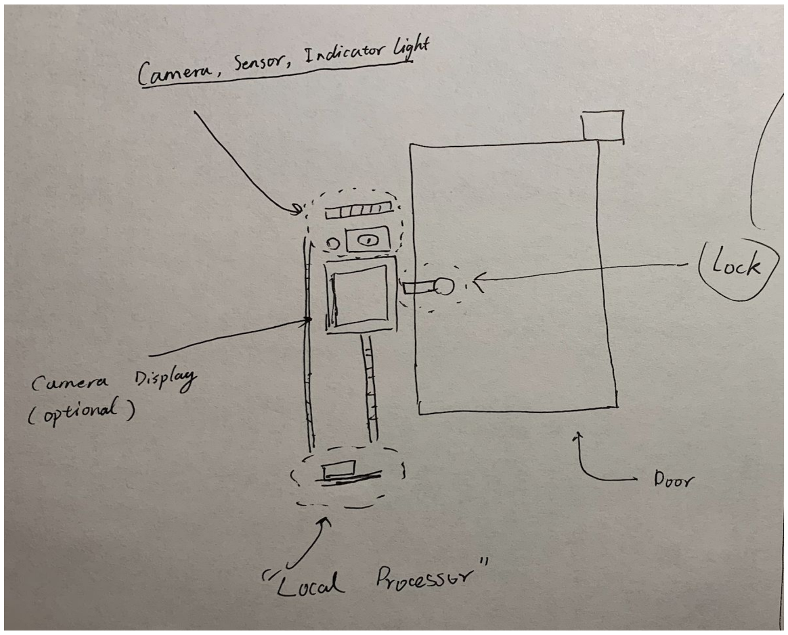
In order for the user to effectively use this product, the interface will be designed to be as user friendly as possible. The interface will revolve around a modern and simplistic design, in order to reduce the clutter that the user may become confused by. Different menus with categories such as statistics and options will be tucked away in a side menu, accessible with one click of a button.

This product is a much needed improvement upon existing home security systems and smart locks. Which provides the user friendly interface with features that are helpful to the homeowners such as SMS notifications, a live video feed, cloud storage (to a specified limit), visitor authentication, and the Stranger Reporting Network.

1.2 Project Overview

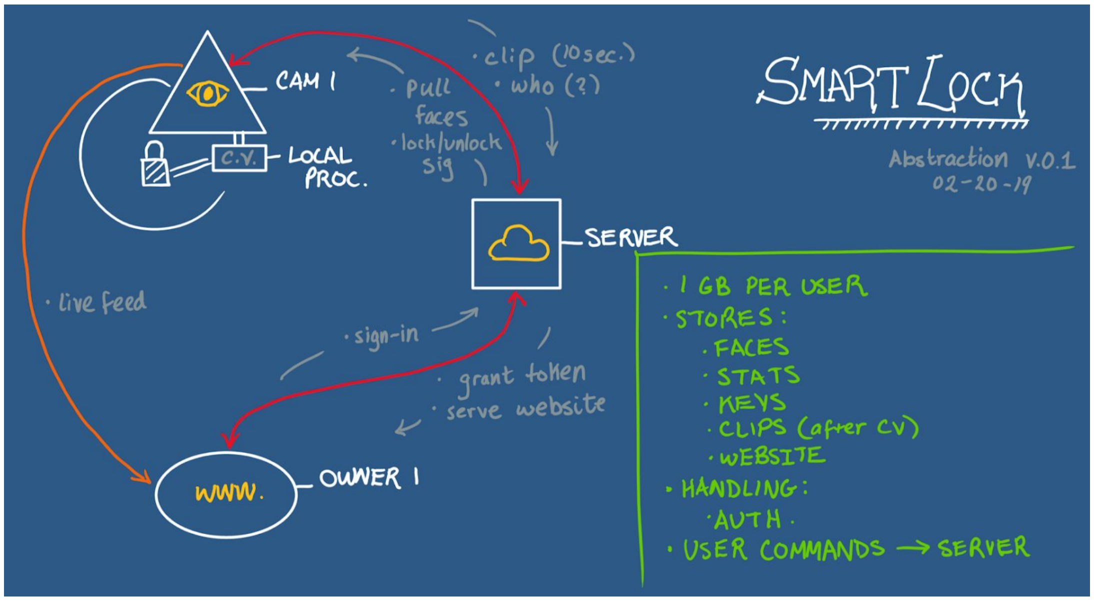
This project focus on the software behind vital home security aspects concerning the problem statement. We aim to implement facial recognition lock for better security and convenience to homeowners, as well as a web-server-based application that will allow homeowners to manage their lock device.

The "lock device" in this project is an abstraction, we use this term to refer to anything installed at the site (anything relating to the "smart lock" physically present in the house), it integrates the physical lock, the camera/sensor, and the local processor. Local Processor is where some of the most important aspect of the lock will take place: it will handle computer vision related processes and control the physical lock; for this project, we will use a raspberry pi for the local processor, which also handles implementation of the physical lock-and-door control. For the purpose of this project, we will not design a new lock mechanism, but use a DC motor to simulate the lock. Due to budgeting, we will choose from either web camera or raspberry pi camera, and the indicator light will be LED on breadboard:



Hand sketch of the "Lock Device"

The lock device connects to the server via home wifi, so homeowner can access their lock anytime and anywhere via a web application by logging in to their unique account. From the web app, homeowners can regulate authorized personnel, view data visualization and event log, and live stream camera footage. The web app is the primary user interface of the project where the most user intensive interactions will take place, the UI design of this report outlines important aspects of this web app. The server stores data of individual lock device and the information of the web app, it serves as a bridge of communication between the homeowner and the lock device when homeowner is away:



An Early Concept of the System

"Electricity is the blood of this system, power outage spells the end of the world" is true to a limited extent. Although this system is key-less, we can still implement a traditional lock system in case of power outage. Also, since the processor is a raspberry pi, although the primary power source is the utility, it still have an external battery for emergency (like power outage). As a result, we should be able to do local facial recognition using a backup power source to unlock the door. However, in events of power outage, there will be no connection to the internet thus limit the web app.

2 Glossary of Terms

- **Smart Lock:** The name of our project which focuses upon an intelligent locking system. The primary key for this lock is facial recognition, which will determine whether or not the individual can acquire access to the house.

- **Lock Device:** The physical locking mechanism implemented into the primary entrance to the house. This may be abstracted to fit different types of locks. The "device" integrates a lock, camera, sensor, and embedded computer.

- **Local Processor:** The Raspberry Pi will be the embedded computer that has primary control of the locking mechanism. It communicates with the server to update the event log and other useful data.

- **Web Application:** A client application presented over the web. This is primary platform on which the Smart Lock user interface is accessed by the Homeowner. Can be accessed easily on desktop and mobile web browsers. Sometimes referred to as "web app" for short.

- **Component:** A portion of the web app which handles a specific use case and exposes a set of functionality to the user.

- **Component View:** The frontend provided to the user by each component. The interface will include all details related to the components current state as well as a method for modifying settings.

- **Server:** The web app data, user information, event log etc. will be stored in the server, communicates with local processor, and serve the web app to clients.

- **Resident**: A person authorized to open the Lock via facial recognition.

- **Homeowner**: The primary user and owner of the house with the ability to log into the Web Application, and thus has the ability to add/remove authorized personnel to the system, change settings regarding the lock, and perform any other task allowed via the Web Application.

- **Temporary Visitor**: A person who is provided temporary access to the Lock via a Visitor URL linked to a Visitor Key

- **Visitor Key:** A string containing letters, numbers and hyphens which is typically accessed as a QR code for gaining temporary access to the home.

- **Visitor URL:** A hyperlink which when accessed presents the visitor with a QR code containing the associated Visitor Key. Sometimes called a "temporary URL"

- ❖ **Computer Vision**: The field relating to computers acquiring, processing and analyzing digital images in order to solve problems.

- ❖ **Computer Vision Module:** Essential role of the local processor that handles Computer Vision related processes, including but not limited to facial recognition, object detection.

- ❖ **Facial Recognition**: A segment of Computer Vision which uses biometric data to analyzes the face of the person.

- ❖ **Motion Detection:** A set of monitoring algorithms which detects when there is movement in front of the camera.

- ❖ **Object Sensor:** Hardware trigger of local facial recognition system.

- ❖ **Stranger Reporting Network:** An integrated network of strangers reported by the users and saved in the server's database. Frequently abbreviated as "SRN".

- ❖ **Event:** Something that happens around a lock. Examples of events can be: successful unlocks, detected movement, unrecognized visitors, temporary URL accesses, etc.

3 Systems Requirements

(note: 5 is the highest priority)

Req. Identifier	Priority	Requirement
REQ-1	5	Residents shall be able to unlock the door through facial recognition.
REQ-2	3	Residents shall be able to unlock the door using a physical keypad on the lock in the event that face recognition fails.
REQ-3	4	Owners shall be able to register other household members (Residents) to be able to unlock the door via facial recognition.
REQ-4	2	Owners should be able to create a temporary URL to allow a temporary visitor to unlock the door.
REQ-5	4	Owners shall be able to temporarily add people that can unlock the door.

REQ-6	4	Homeowners shall be able to subscribe for and receive notifications when events occur.
REQ-7	2	Owners should be able to report suspicious activity to the SRN.

3.1 Enumerated Nonfunctional Requirements

Req. Identifier	Priority	Requirement
REQ-8	4	The system should provide a desktop and mobile-friendly web application to view and manage aspects of the lock.
REQ-9	4	The web interface should be intuitive and user-friendly.
REQ-10	5	The web application should be quick and easy to access and log into.
REQ-11	3	Owners should be able to view an organized list of all events regarding the lock.
REQ-12	3	The system should be able to log all events related to the Lock, and be able to present this to the homeowner.
REQ-13	2	Residents may receive notifications via push notifications on a desktop.
REQ-14	1	Owners may receive notifications via push notifications on mobile.
REQ-15	5	The system should have security features to prevent unauthorized access into the Web Application and into the homeowner’s house. The web application should have an authentication page.

3.2 On-Screen Appearance Requirements

Req. Identifier	Priority	Requirement
REQ-16	5	The web app shall have a login page for homeowners to access and configure their Smart Lock.
REQ-17	4	The web app shall have the main page with a sidebar that links to other features of the web application.
REQ-18	2	The 'Stranger Reporting Network' page should provide homeowners with a map of suspicious activity in their area and provide a field to manually report suspicious activity.
REQ-19	3	The Live Feed view should allow users to see a live feed of the camera.
REQ-20	3	The Data Visualization view should contain clickable tabs that each provide a different graphic visualizing statistics generated by the lock.
REQ-21	4	The Visitors view should provide the Homeowner with an interface to generate a temporary URL to send to a visitor that gives one-time access into the home.
REQ-22	4	The Residents view should provide selections for uploading photos of Residents to access the home via facial recognition.

4 Functional Requirements Specifications

4.1 Stakeholders

- Owners
  - Each Owner should be able to efficiently and easily safeguard, monitor, and grant accessibility to their home. The homeowner should be able to achieve such tasks with minimal human intervention. Additionally, the homeowner should be able to report and convey the presence of potentially malicious actors without needing to be present at home.

- Residents
  - Other residents of the home should be able to easily unlock the lock device an unlimited amount of time through facial recognition. Residents should not be able to receive notification, watch live video feeds, and allow guests in the house; they have essentially the same goals as Owners, except they cannot view/change attributes of the lock.

- Visitors
  - Visitors are allowed temporary access to the house. This can be over a certain period of time or a limited amount of access. Visitors can be authorized through a temporary URL. The temporary URL will provide them with a QR code to scan on the lock device.

4.2 Actors and Goals

Actors	Goals
Owner (Initiating)	To safeguard and monitor their home against home intrusion in addition to having a secure and simple method of entering their own home.
Resident (Initiating)	To easily enter their own home without needing a key or other physical object.
Visitor (Initiating)	To easily enter the home of a homeowner using the smart lock if authorized.
Stranger (Initiating)	To perform suspicious activity or break into the home of the homeowner.
Camera (Participating)	Capturing and delivering video footage of the home entrance to the server. Used for motion detection and facial recognition as well.
Database (Participating)	SQL database storing information associated with households including but not limited to faces, authentication URLs, and logs of events.
Lock Device (Participating)	To keep the home secured by unlocking for only authorized individuals.
Server (Participating)	To manage the network between the devices, software, and database.
Scanner (Participating)	To temporarily unlock the door by scanning the QR code from the temporary URL.
Neighbors (Participating)	To have access to reported suspicious activity by the homeowner.

4.3 Use Cases

4.3.a Casual Description

UC-1: Unlock - Gives the homeowner and residents ability to unlock the door through facial recognition.
Derived from requirements REQ-1, REQ-2, REQ-3, REQ-5, REQ-6.

UC-2: Notifications - Allow homeowners to receive notifications about Events.
Derived from requirements REQ-6, REQ-11, REQ-12, REQ-13, REQ-14, REQ-16, REQ-17.

UC-3: Stranger Reporting - Allow homeowners to report suspicious activity in the neighborhood.
Derived from requirement REQ-7, REQ-11, REQ-12, REQ-16, REQ-18.

UC-4: Temporary Visitor Authentication - To allow visitors to the home in one(or more) time(s) using a URL or temporarily adding them to trusted faces.
Derived from requirement REQ-4, REQ-5, REQ-11, REQ-12, REQ-16, REQ-17, REQ-21, REQ-22.

UC-5: Add/Remove Visitors - To add or remove residents as visitors with permission to enter the house with facial recognition.
Derived from requirements REQ-1, REQ-2, REQ-3, REQ-4, REQ-21, REQ-22.

UC-6: Data Visualization - To allow homeowners to view data pertaining to who enters the house and when then enter.
Derived from requirement REQ-11, REQ-12, REQ-20.

UC-7: Live Video Feed - To obtain a live feed of the home entrance through the camera.
Derived from requirement REQ-19.

4.3.b Use Case Diagram

The following diagrams display and elaborate on the requirements, actors, conditions, and flow of events for each use case that will be implemented by the time of the final demo.

Use Case UC-1: Unlock (through face ID)	
Related Requirements	REQ-1, REQ-2, REQ-3, REQ-4, REQ-5, REQ-6
Initiating Actor	Any of: Homeowner, Residents, or authorized visitor
Actor's Goal	Homeowners and residents unlock the door through facial recognition.
Participating Actors	Camera, lock device, database, server
Preconditions	Facial Recognition Camera video upload to server Residents data stored in database
Postconditions	Door locks again after entering
Flow of Events: 1. Initiating actor approaches the door. 2. Camera sensor detects person approaching and signals the local processor to begin Object Detection and Facial Recognition. 3. The indicator light tells the person status of lock, and direct the person to look directly at the camera. 4. The facial recognition algorithm identifies the face as an authorized person. 5. System signals the lock device to unlock. Signals the timer to star auto-lock countdown 6. Person enters the door and shuts the door, door sensor signals the system when door is closed 7. System locks the door automatically	

Use Case UC-2: Notifications	
Related Requirements	REQ-6, REQ-11, REQ-12, REQ-13, REQ-14, REQ-16, REQ-17
Initiating Actor	Visitors, strangers, or other events

Actor’s Goal	Allow homeowners to receive notifications about events.
Participating Actors	Camera, server, and homeowners receiving the notification
Preconditions	An event occurs.
Postconditions	The owner is informed of the event.
Flow of Events 1. An event occurs such as someone coming up to the door which will be detected by motion detection. 2. This information is sent to the server. The server processes this information and checks if the user is signed up for notifications. 3. If so, the server then sends a notification (SMS or push, depending on user preference which they specify) to the user’s device. 4. The user sees the notification and is informed of the event.	

Use Case UC-3: Stranger Reporting	
Related Requirements	REQ-7, REQ-11, REQ-12, REQ-16, REQ-18
Initiating Actor	Stranger
Actor’s Goal	Allow homeowners to report suspicious activity in the neighborhood.
Participating Actors	Homeowner, server, and neighbors
Preconditions	Suspicious Activity occurs
Postconditions	Suspicious activity is now viewable by others in the neighborhood.
Flow of Events 1. A moving object arrives in view of the camera. 2. The system attempts to authenticate the moving object using facial detection and recognition but fails. The camera takes a picture and sends it to the homeowner. 3. Homeowner notified of unknown object in front of door through a picture.. 4. Homeowner looks at picture taken and feels like there is suspicious activity. 5. Homeowner reports it to the network. 6. A snapshot of the individual is saved by camera. 7. Now, anyone in the neighborhood can view this on the map.–	

Use Case UC-4: Temporary Visitor Authentication	
Related Requirements	REQ-4, REQ-5, REQ-11, REQ-12, REQ-16, REQ-17, REQ-21, REQ-22
Initiating Actor	Homeowner
Actor’s Goal	To allow visitors to the home in for one time using a URL
Participating Actors	Temporary visitor; Lock Device; Database
Preconditions	Temporary visitor must be a member of the allowed temporary visitor database.
Postconditions	Temporary visitor authenticated.
Flow of Events 1. Homeowner accesses the visitor authentication tab in Web App 2. Web App sends a “GET” request for all open Visitor Keys 3. Server records the name, creates a Visitor ID, Visitor Key and expiration time 4. Homeowner clicks “Copy”; Visitor URL is copied to clipboard; Homeowner sends the URL to the desired Visitor 5. Temporary visitor accesses URL via a web browser on a mobile device; Visitor is presented their Visitor Key encoded as a QR code 6. Temporary visitor shows QR code to the Lock’s integrated camera 7. Lock’s Embedded Computer sends a “GET” request to the Server to verify the Visitor Key 8. Use of the Visitor Key is recorded by the server; Visitor is authenticated and Lock unlocks	

Use Case UC-5: Add/Remove Residents	
Related Requirements	REQ-1, REQ-2, REQ-3, REQ-4, REQ-21, REQ-22
Initiating Actor	Homeowner
Actor’s Goal	Add/Remove residents with permission to enter the house with facial recognition.
Participating Actors	Residents
Preconditions	The ability to store a user with their own facial ID tied to them.
Postconditions	Resident is able to use facial recognition if they are authorized.

Flow of Events (Homeowner adds Resident):

1. A Resident wants to be registered for facial recognition.
2. Homeowner accesses the Authorized Residents tab in Web App
3. Homeowner enters a name for the Resident; Selects “Add Residents”
4. Web App sends a “POST” request containing the requested name
5. Server records the name and creates a Resident ID.
6. Server responds with the new Resident ID; Web App updates list of Residents.

Alt. Flow of Events: (Homeowner removes Residents)

1. The Homeowner wants to remove a Resident.
2. Homeowner accesses the Authorized Resident tab in Web App
3. Homeowner enters a clicks delete button next to Resident name.
4. Web App sends a “DELETE” request containing the requested Resident ID
5. Server removes the Resident and related data from the database
6. Server responds with the status of the removal (“success” or “invalid id”)

Use Case UC-6: Data Visualization	
Related Requirements	REQ-11, REQ-12, REQ-20
Initiating Actor	Homeowner
Actor’s Goal	Allow homeowners to view data pertaining to who enters the house.
Participating Actors	Residents, Visitors
Preconditions	Homeowner, Resident, or Visitor unlocks the door.
Postconditions	Name of Visitor, time entered, and data is viewable by the Homeowner.
Flow of Events: 1. Homeowner opens web app and signs in if not already signed in. 2. Homeowner navigates to the tab for looking at events. 3. Homeowner selects the time, date, or visitors. 4. App displays the desired data and homeowner can view easy-to-read logs, graphs, and charts on who entered the house and when.	

Use Case UC-7: Live Video Feed	
Related Requirements	REQ-19
Initiating Actor	Homeowner
Actor’s Goal	To obtain a live feed of the home entrance through the camera.
Participating Actors	Camera
Preconditions	Homeowner is logged into the website and camera is functioning.
Postconditions	None
Flow of Events: 1. Homeowner selects the live streaming tab on the web app. 2. Camera sends current video feed to the server. 3. Homeowners can view the video on web app.	

4.3.c Traceability Matrix

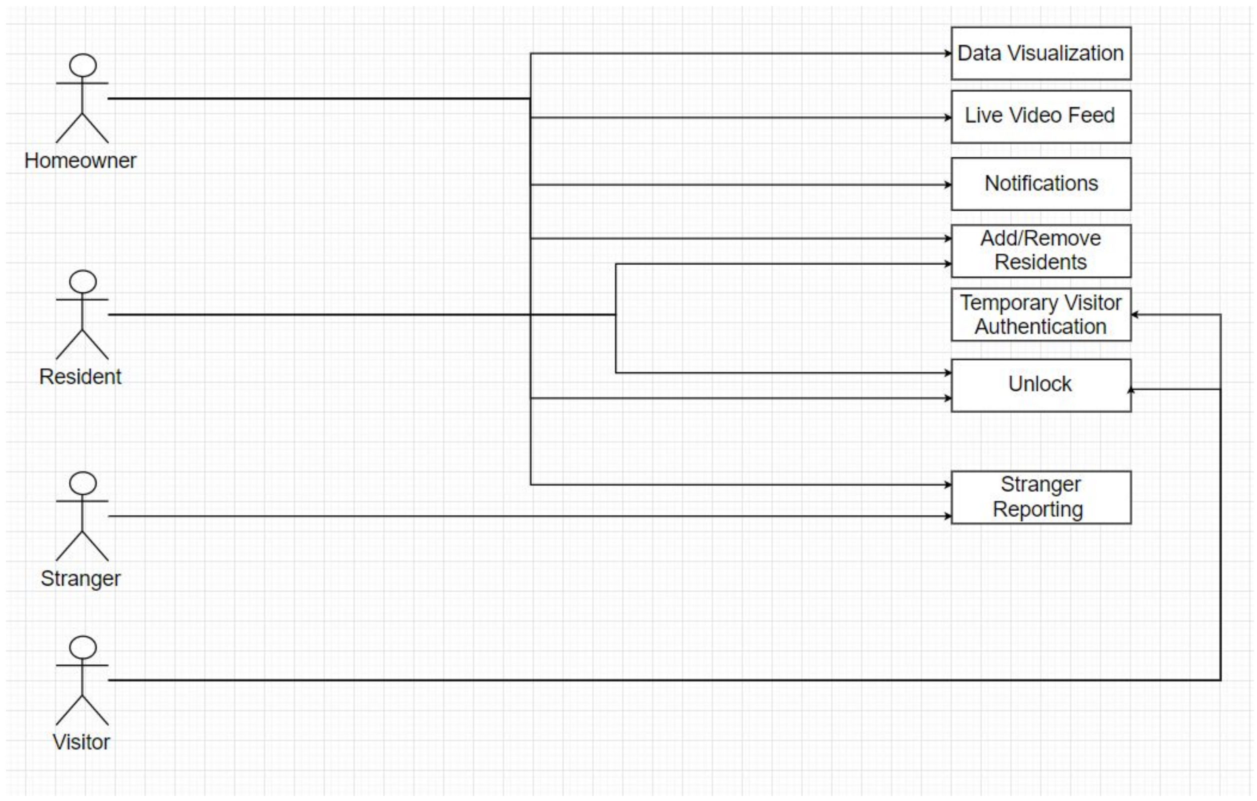
This matrix displays how the use cases are related to the system requirements. An "X" indicates that the requirement on the left is related to the corresponding use case.

REQ-#	PW	UC1	UC2	UC3	UC4	UC5	UC6	UC7
REQ-1	5	X				X		
REQ-2	3	X				X		
REQ-3	4	X				X		
REQ-4	2	X			X	X		
REQ-5	4	X			X			
REQ-6	4	X	X					
REQ-7	2			X				
REQ-8	4							
REQ-9	4							
REQ-10	5							
REQ-11	3		X	X	X		X	
REQ-12	3		X	X	X		X	
REQ-13	2		X					
REQ-14	1		X					
REQ-15	5							
REQ-16	5		X	X	X			
REQ-17	4		X		X			
REQ-18	3			X				
REQ-19	2							X
REQ-20	3						X	

REQ-21	4				X	X		
REQ-22	4				X	X		

4.3.d Fully-Dressed Descriptions

The following diagram shows every action that the initiating actor can take while running the use cases in the Use Case Diagram.

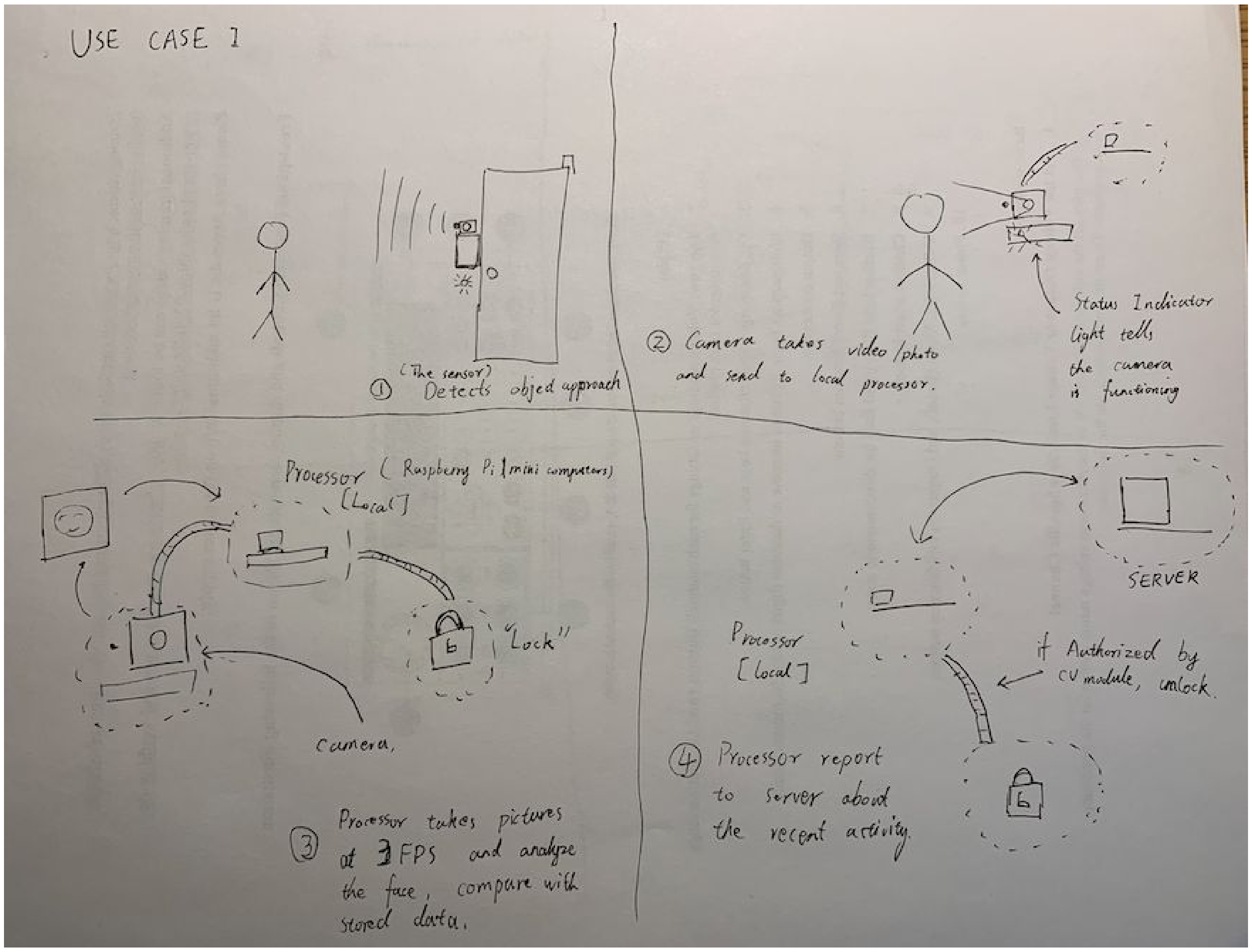


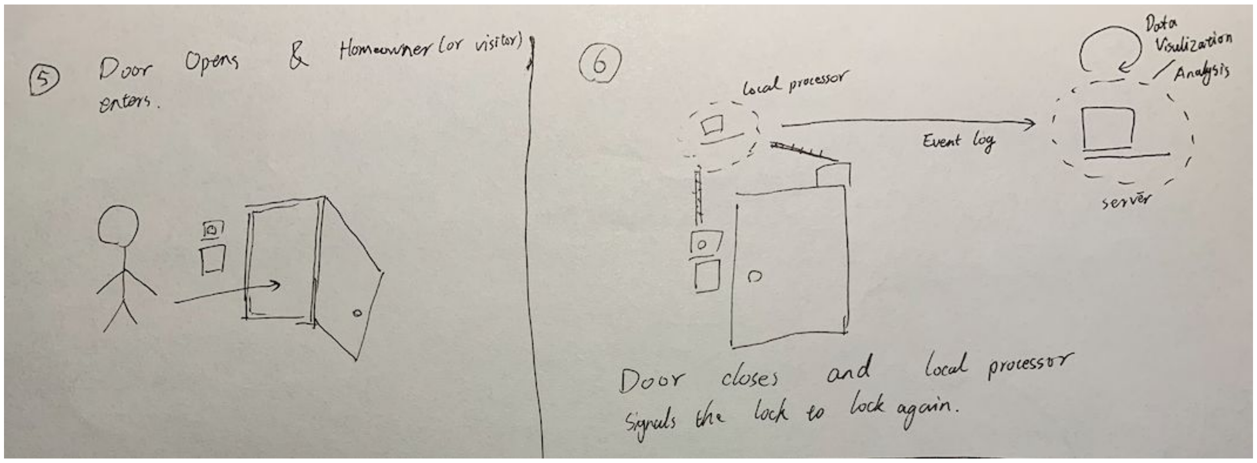
4.4 Systems Sequence Diagrams

The following diagrams display the sequence of events for use cases 1, 4, 6, 7, and 8, which are the most important use cases.

4.4.a Illustration and Sequence Diagram for UC-1

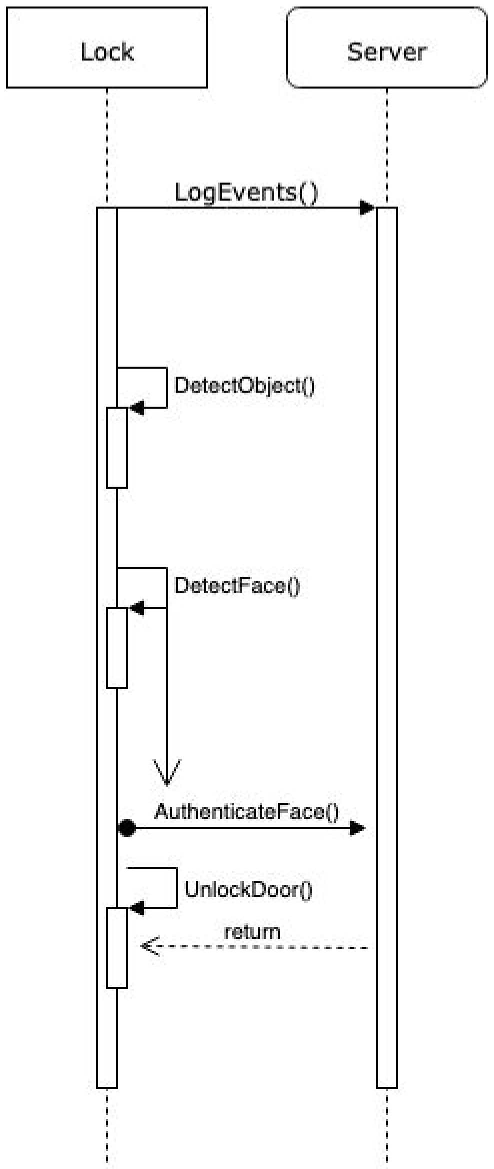
The following diagram displays the flow of events for UC - 1 (Unlock Through Face ID).


USE CASE 1
(The sensor)
① Detects object approach
② Camera takes video/photo and send to local processor.
Status Indicator light tells the camera is functioning
Processor (Raspberry Pi / mini computers) [Local]
"Lock"
Camera.
③ Processor takes pictures at 3 FPS and analyze the face, compare with stored data.
SERVER
Processor [Local]
if Authorized by CV module, unlock.
④ Processor report to server about the recent activity.

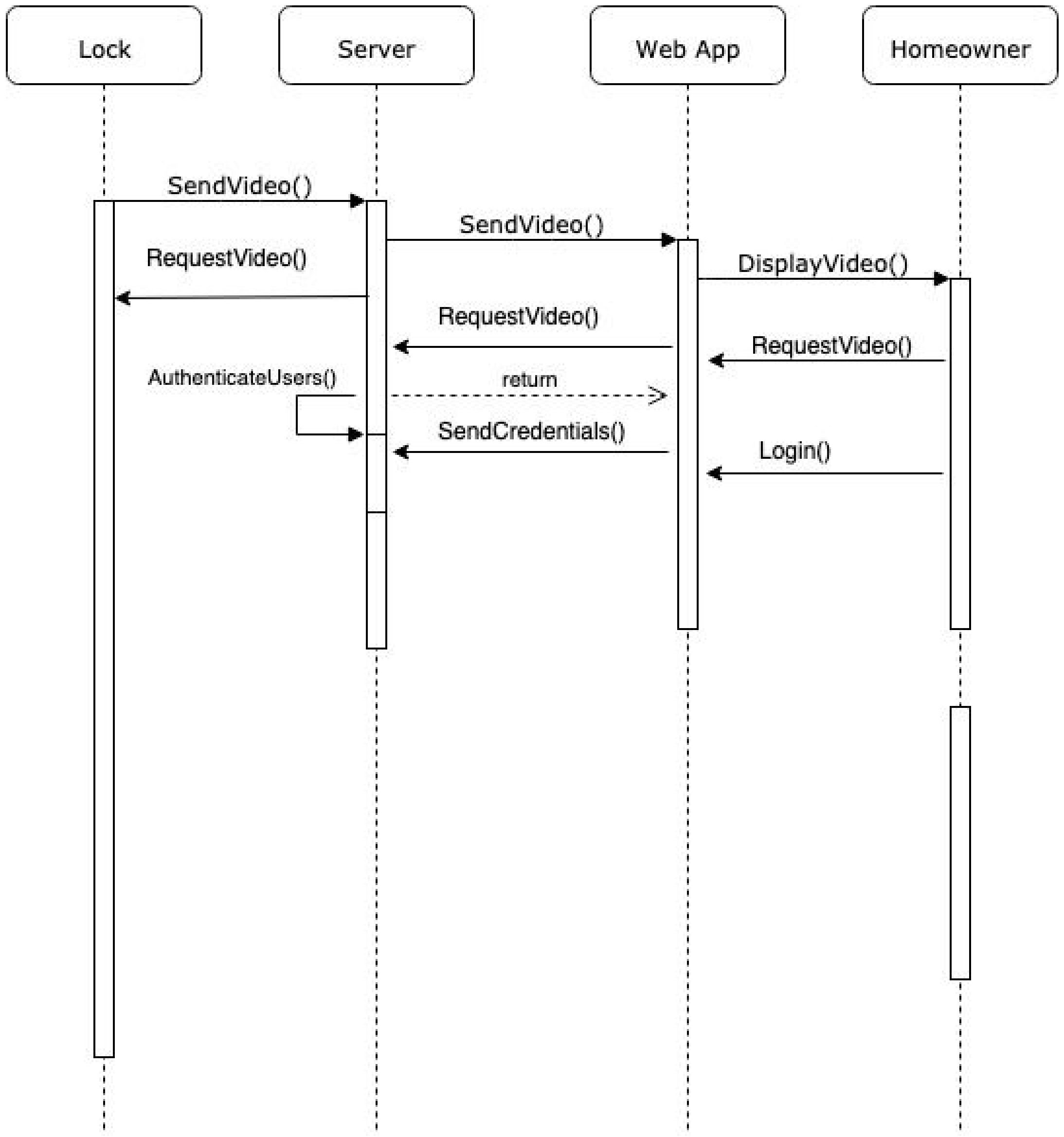


Door Opens & Homeowner (or visitor) enters.

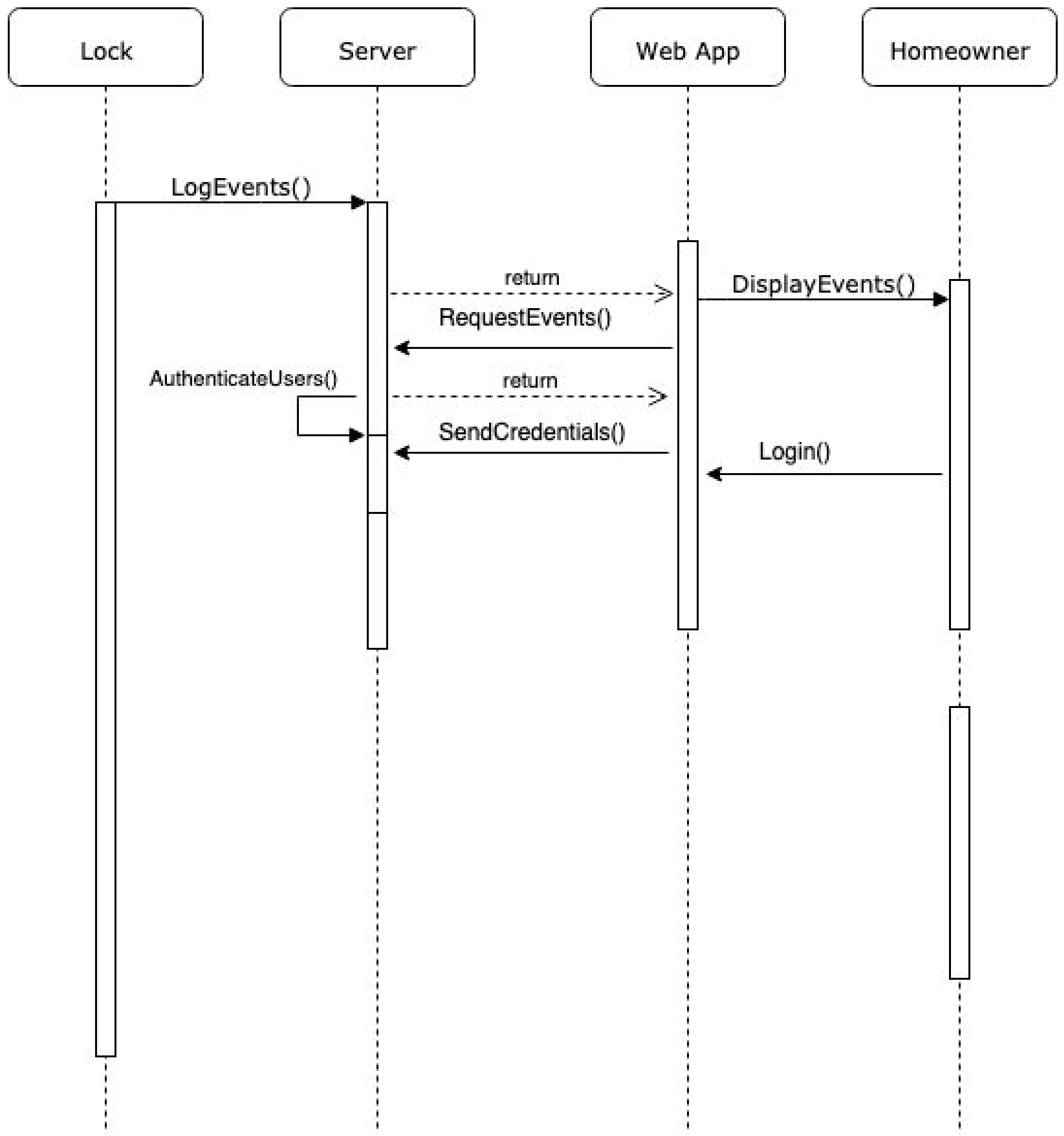
Door closes and local processor signals the lock to lock again.


Lock
Server
LogEvents()
DetectObject()
DetectFace()
AuthenticateFace()
UnlockDoor()
return

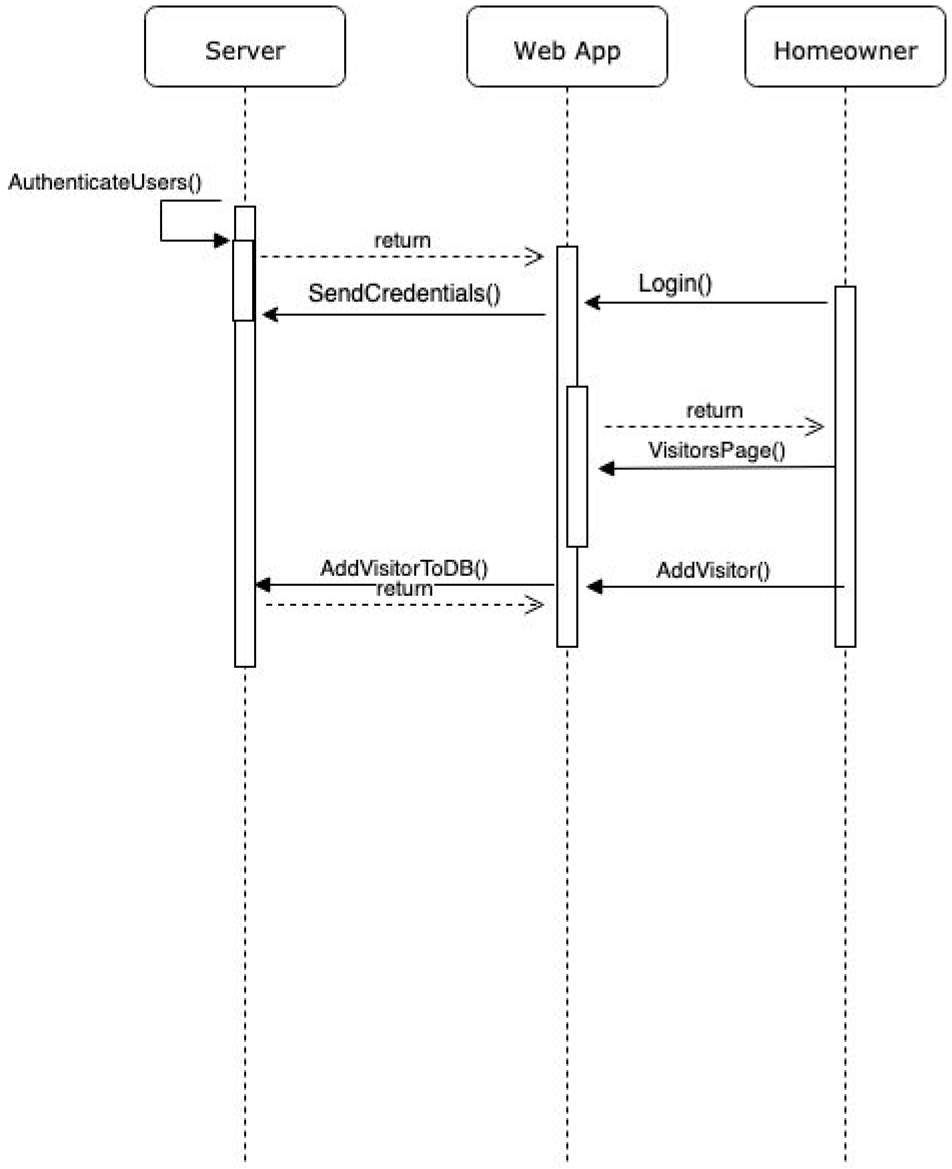
4.4.b Sequence Diagram for UC-7



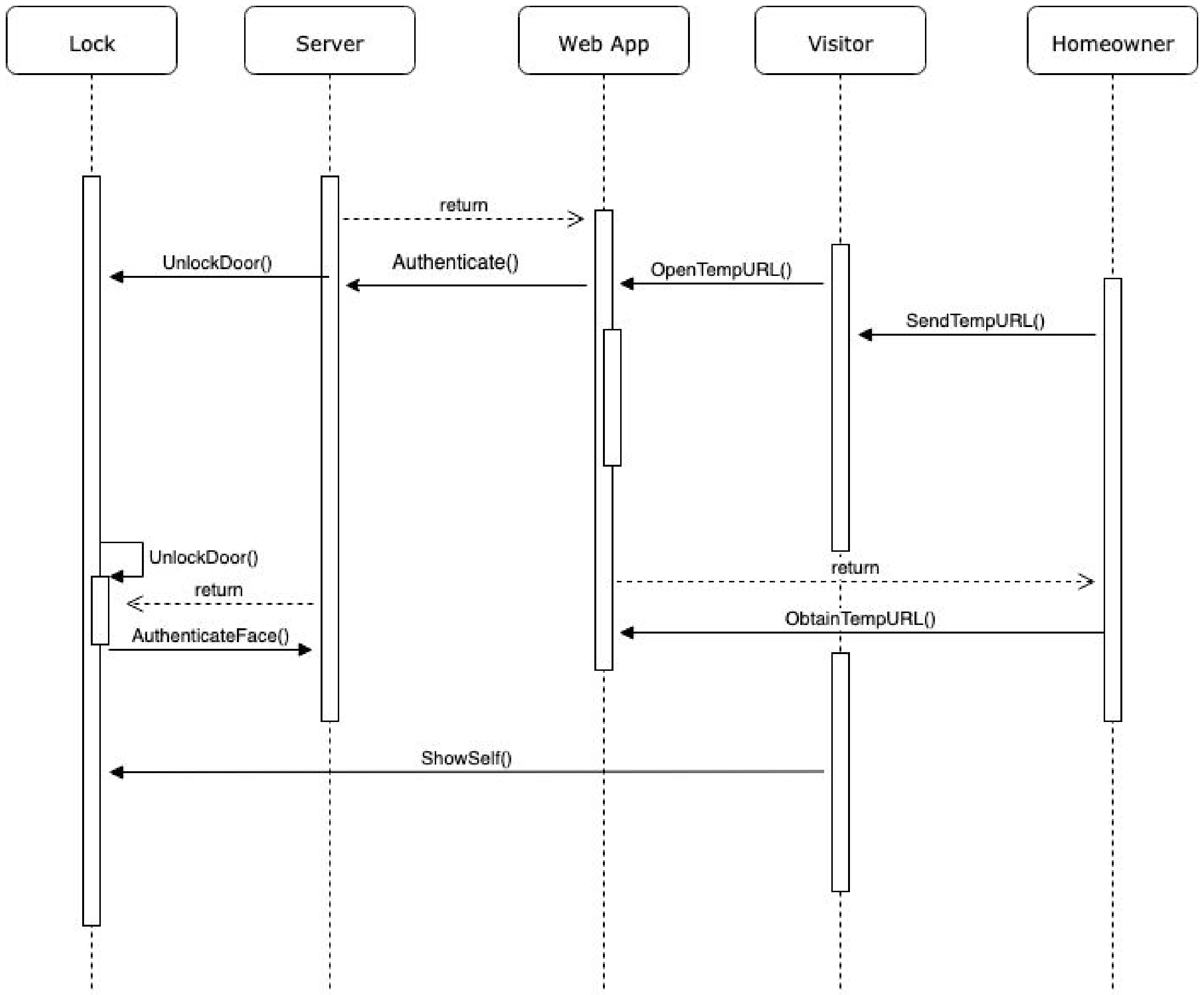
4.4.c Sequence Diagram for UC-6



4.4.d Sequence Diagram for UC-5



4.4.e Sequence Diagram for UC-4



5 User Interface Specification

While our goal with this project is to minimize user intervention as much as possible through automated intelligence, there will still be times when the user must interact with the system. To make this experience as painless as possible, we have deeply considered the user's needs for this aspect..

A major concern when designing mobile interfaces is reducing unintended input. For example, if the user happens to misclick a button they may get disoriented and have a *"what just happened?"* moment. These types of moments lead to a poor user experience and should be avoided at all costs.

To combat this problem, we have decided on a proactive approach. Each state of the interface will include large buttons typically spanning most of the screen's width for ease of access with either hand. However, we must balance the size of buttons with the amount of information provided on any one screen.

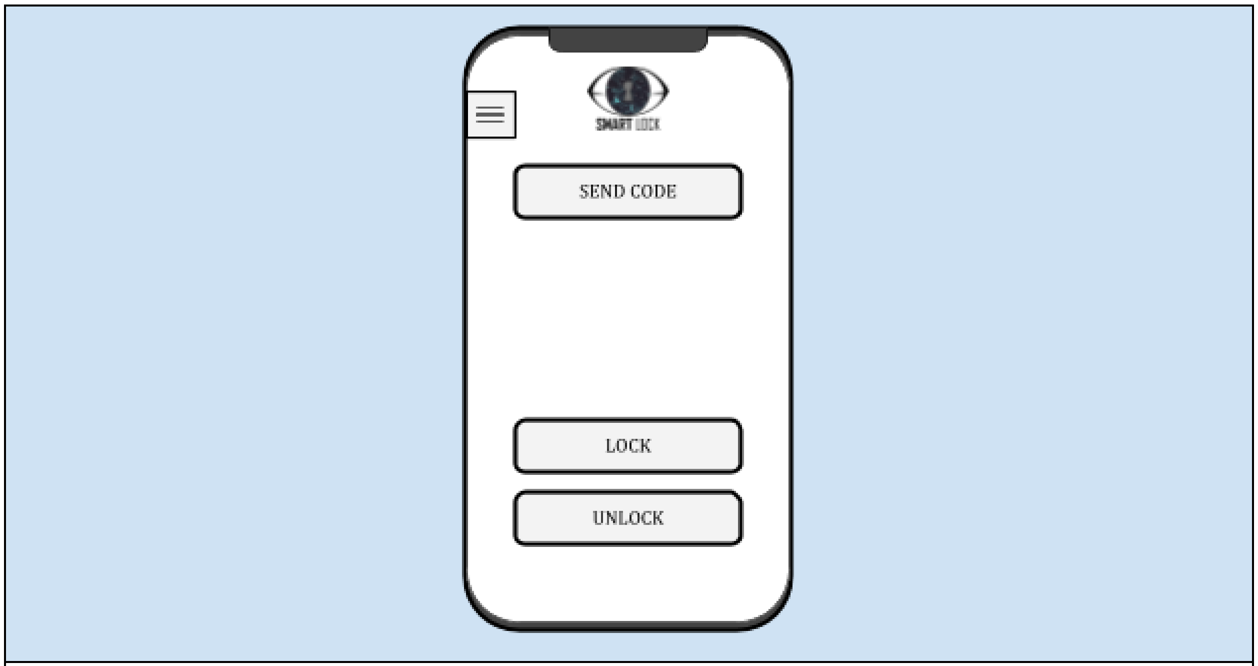


Figure 4 -- A simple web application view displayed on a smartphone. Notice the inclusion of wide buttons for ease of access on a mobile device.

Traversing different component views and their respective settings should be unified for all components and efficient. With this in mind, a collapsible sidebar will be implemented for the user to be able to navigate to the desired view with a maximum of two button presses.

5.1 Hand Drawn Designs

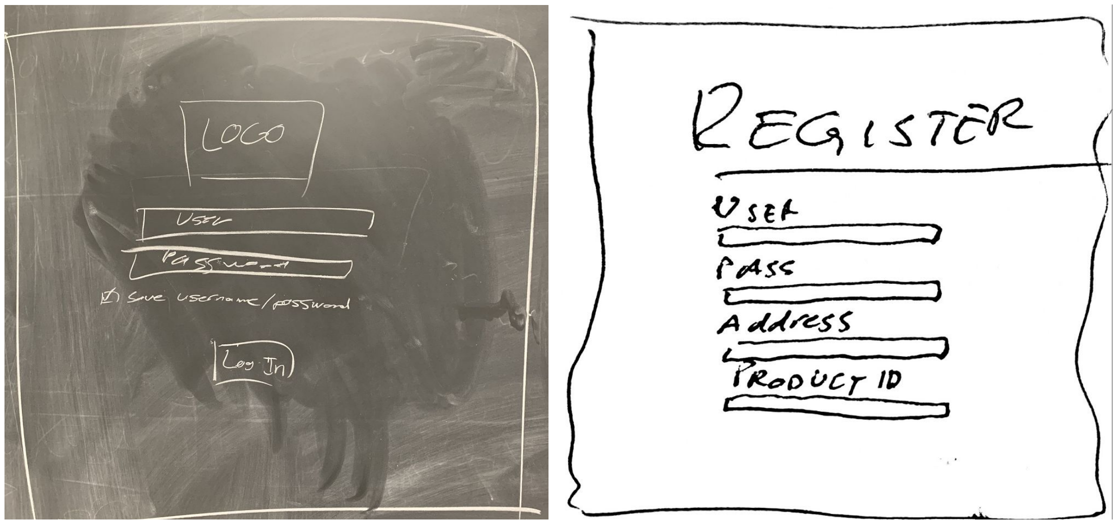


Figure 5.a.i -- Login & registration interfaces

In order to authenticate the owner must first enter their username and password. The layout consists of username and password fields followed by a login button. Although not depicted, the UI will also have a register button next to the login button that will allow for the user to register for an account by entering their username, password, street address, and Product ID. This will allow for the customer to link their smart lock with their account.

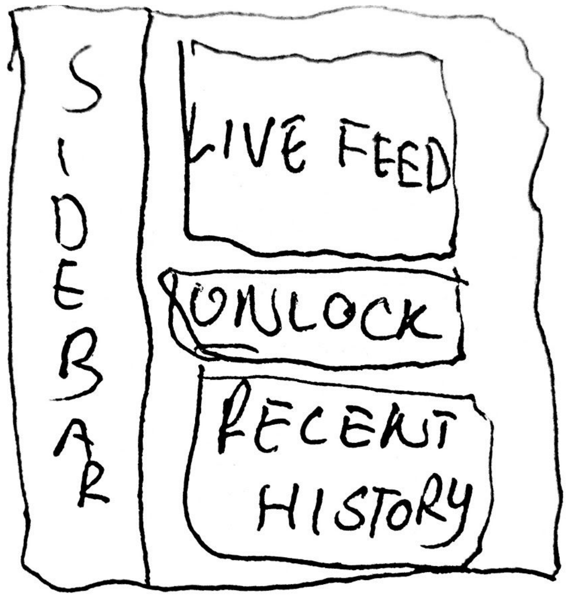
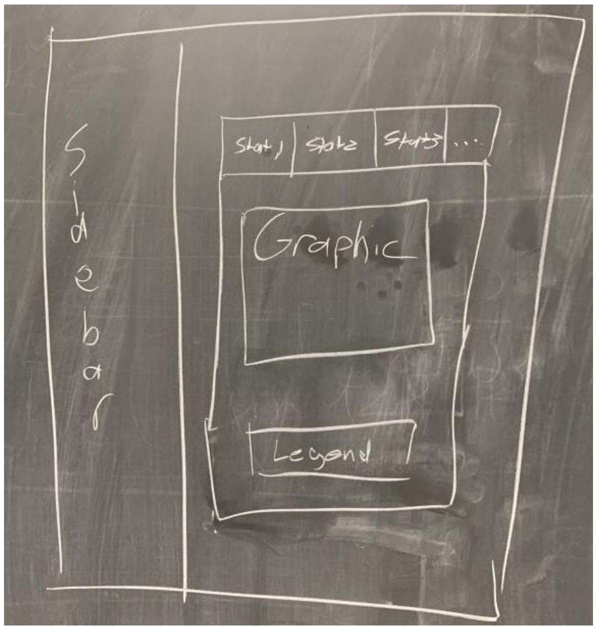


Figure 5.a.ii -- Main view that users will be presented upon logging in

Upon authentication as a homeowner, said user will be presented with the "Main" view. This view will attempt to present the homeowner with a summary of important information.

The main menu will consist of a sidebar displaying links to other important screens such as the log, settings, visitors / authentication, and statistics (data visualization). The home page will be very minimalistic. It will contain live feed of the lock (camera), an unlock button which will allow for the owner to manually unlock their door without having to use the facial authentication or URL authentication. Under the unlock button the user can preview some recent history (only the first few entries; the rest of the history will only be shown when the user clicks on "logs" [in the sidebar]).



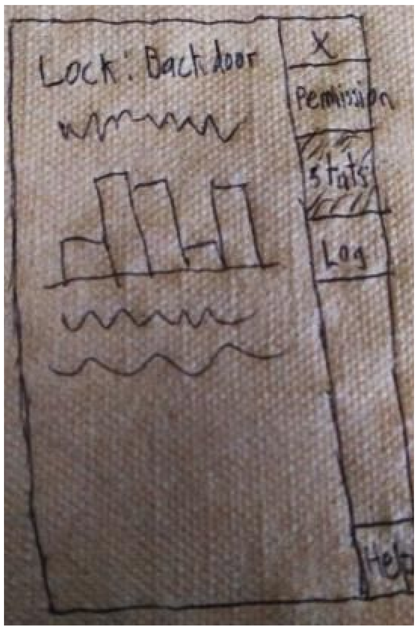


Figure 5.a.iii -- Data visualization interface with tabs and relevant graphics types under the "Stats" menu option

Data Visualization can be accessed using the menu item labeled "Stats." This page will contain graphics centered relative to the viewport. These graphics will include corresponding legends and other information along the bottom. Along the top, there will be tabs to other statistical information that would be important to visualize. The statistics displayed will include total number of visitors for each time of day, time of day visitors are most active, organized chronological history of when visitors are most active, etc.

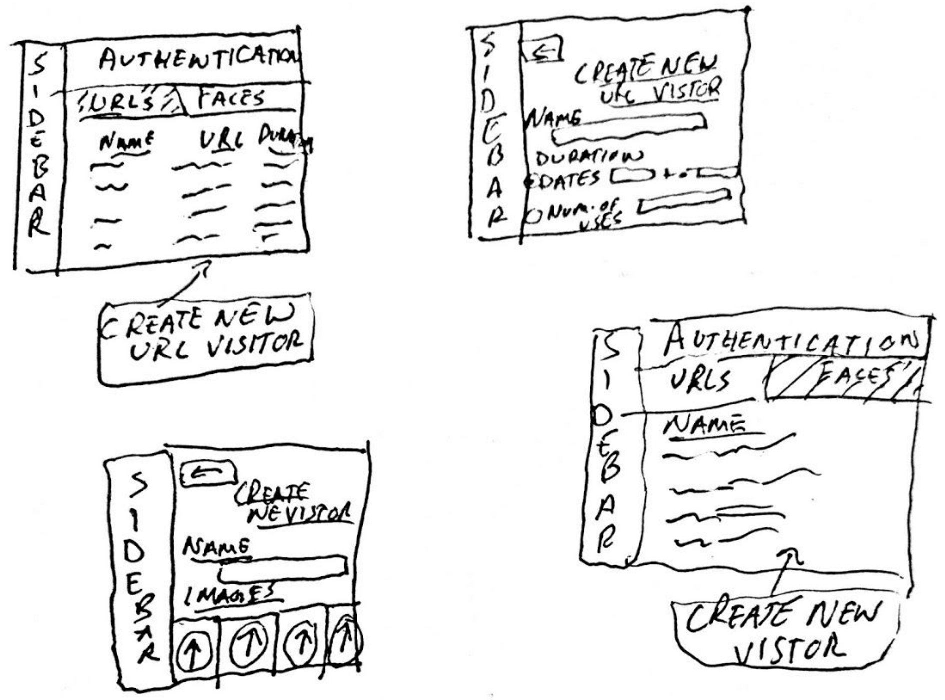


Figure 5.a.iv -- Various Automatic Visitor Authentication menus allowing the homeowner to view the status of visitors they have granted access to

Automatic Visitor Authentication can be accessed by the user by navigating to the "Permissions" tab on the sidebar. This page will host two tabs, URLs and Faces. URLs contains active visitors that were permitted by the URL system. Faces contains visitors that were permitted by inputting images. The user may also create new visitors using the button below the entries or delete entries as well. When creating new visitors using the visitor for with URLs the user needs to specify a name as well as duration (the duration can be specified in different ways including dates and number of uses). When creating a visitor by inputting faces, the user may specify name and then upload multiple images which the system may catalog for facial recognition. The user may also specify duration for facial recognition as well if so desired (not shown).

5.2 User Effort Estimation

Note that for each user effort estimation is based on the **maximum** total user effort.
For all variable text entries, assume ‘ λ ’ as a number of characters in the desired string. It is possible that λ are different lengths and will be differentiated by a subscript.

5.2.a Desktop

LOGIN: total 3 mouse clicks and $\lambda_1 + \lambda_2$ keystrokes (or 1 mouse click and $2 + \lambda_1 + \lambda_2$ keystrokes), as follows

1. ***Click*** “Username” box
    a. ***Press*** λ_1 keys.
2. ***Click*** “Password” box | ***Press*** “TAB”
    a. ***Press*** λ_2 keys.
3. ***Click*** “Sign-In” button | ***Press*** “ENTER”

NAVIGATION: total 2 mouse clicks, as follows

1. ***Click*** "Menu" button (top left)
2. ***Click*** on desired component view labeled by name

UNLOCK: total 3 mouse clicks, as follows

1. ***NAVIGATE*** to “Main Menu” (2 mouse clicks)
2. ***Click*** “Unlock” button

VISITOR AUTHENTICATION: total 6 mouse clicks and λ keystrokes, as follows

1. ***NAVIGATE*** to “Permissions” (2 mouse clicks)
2. ***Click*** “Create New” button
3. ***Click*** “Name” box
    a. ***Press*** λ keys.
4. ***Click*** “Image” button
    a. Upload Image *(Undefined keystrokes. Dependent on external directories)*
5. ***Click*** “Save”

5.2.b Mobile

LOGIN: total 3+ λ_1+ λ_2 gestures, as follows

1. ***Touch*** "Username" box
   a. ***Touch*** λ_1 keys.
2. ***Touch*** "Password" box
   a. ***Touch*** λ_2 keys.
3. ***Touch*** "Sign-In" button

NAVIGATION: total 2 gestures, as follows

1. ***Swipe*** from left edge of screen towards middle.
2. ***Touch*** on desired component view labeled by name.

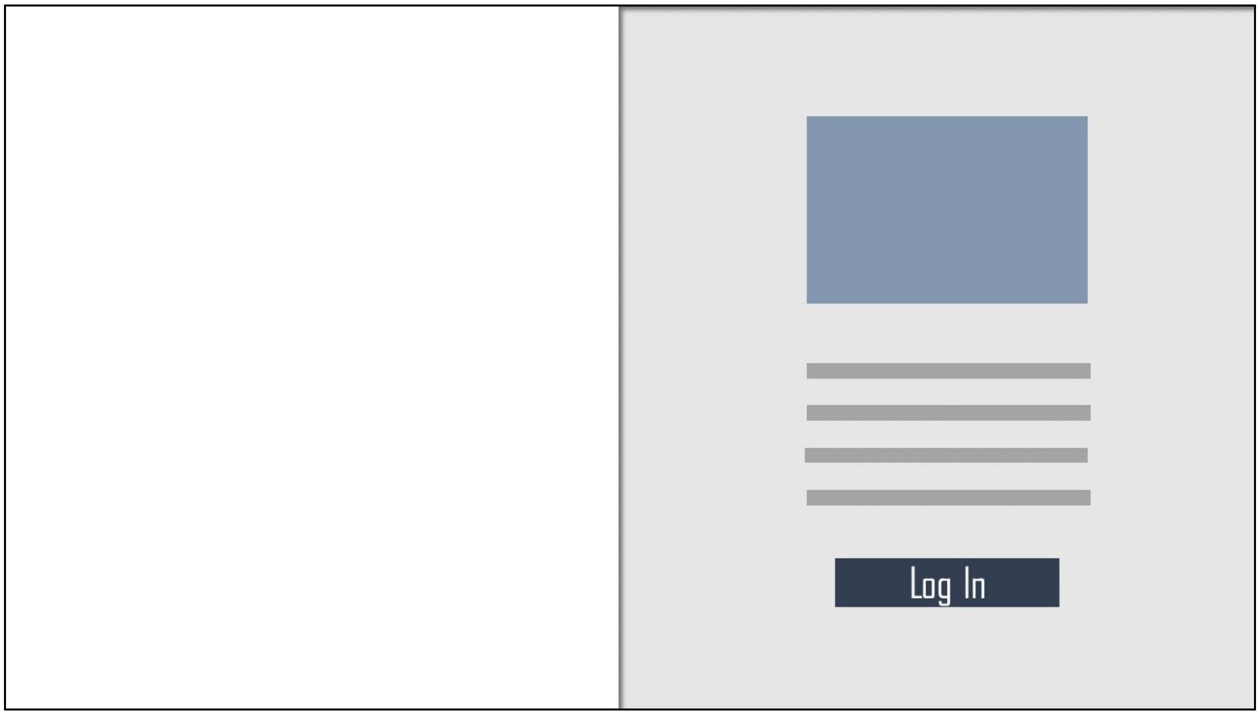
UNLOCK: total 3 gestures, as follows

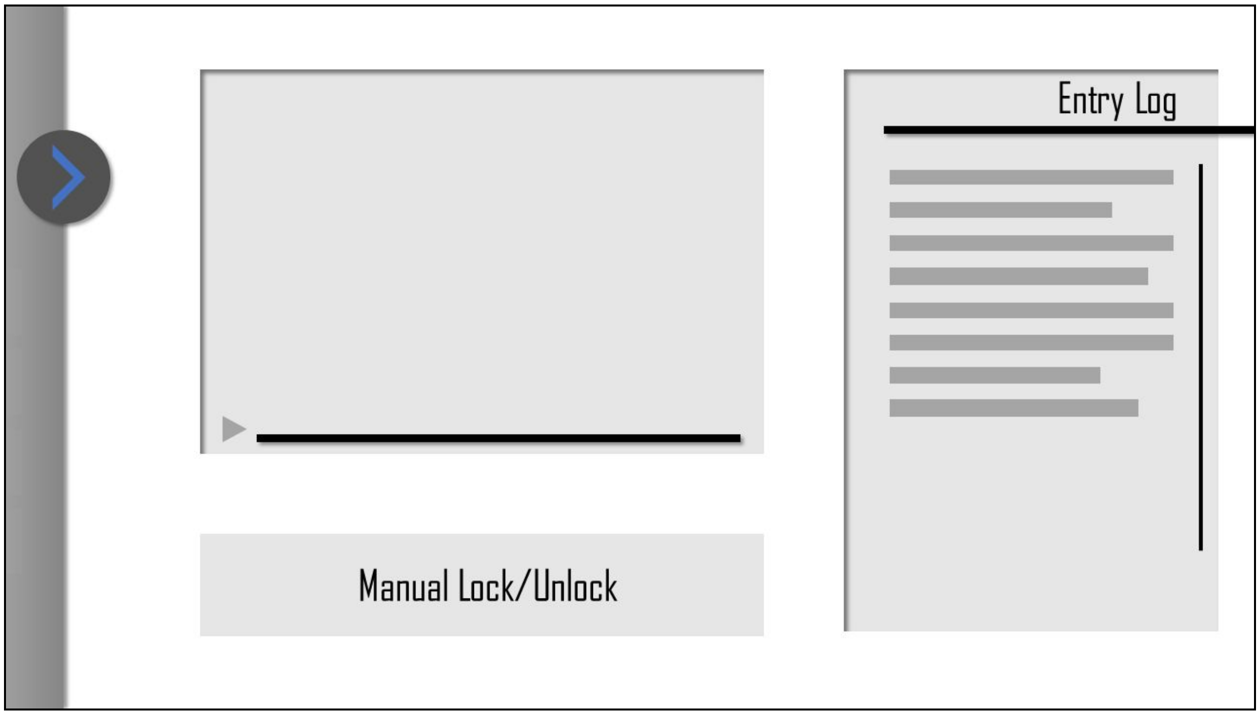
1. ***NAVIGATE*** to "Main Menu" (2 mouse clicks)
2. ***Touch*** "Unlock" button

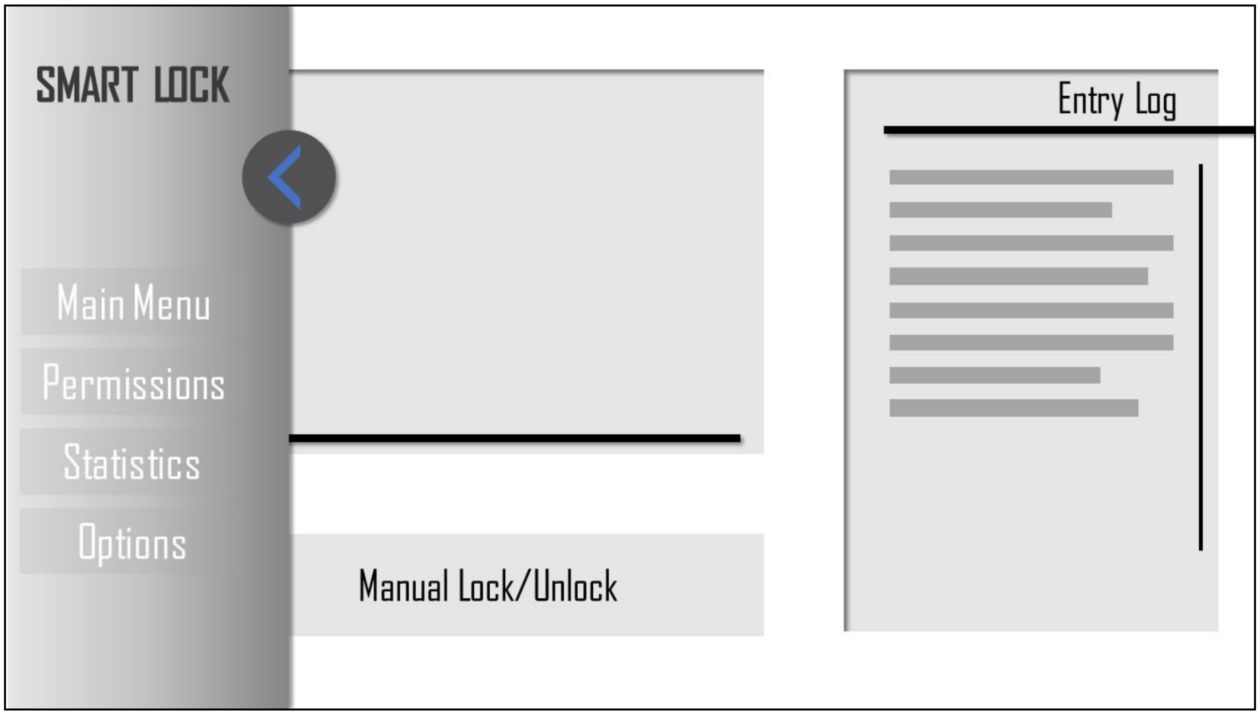
VISITOR AUTHENTICATION: total 6+ λ gestures, as follows

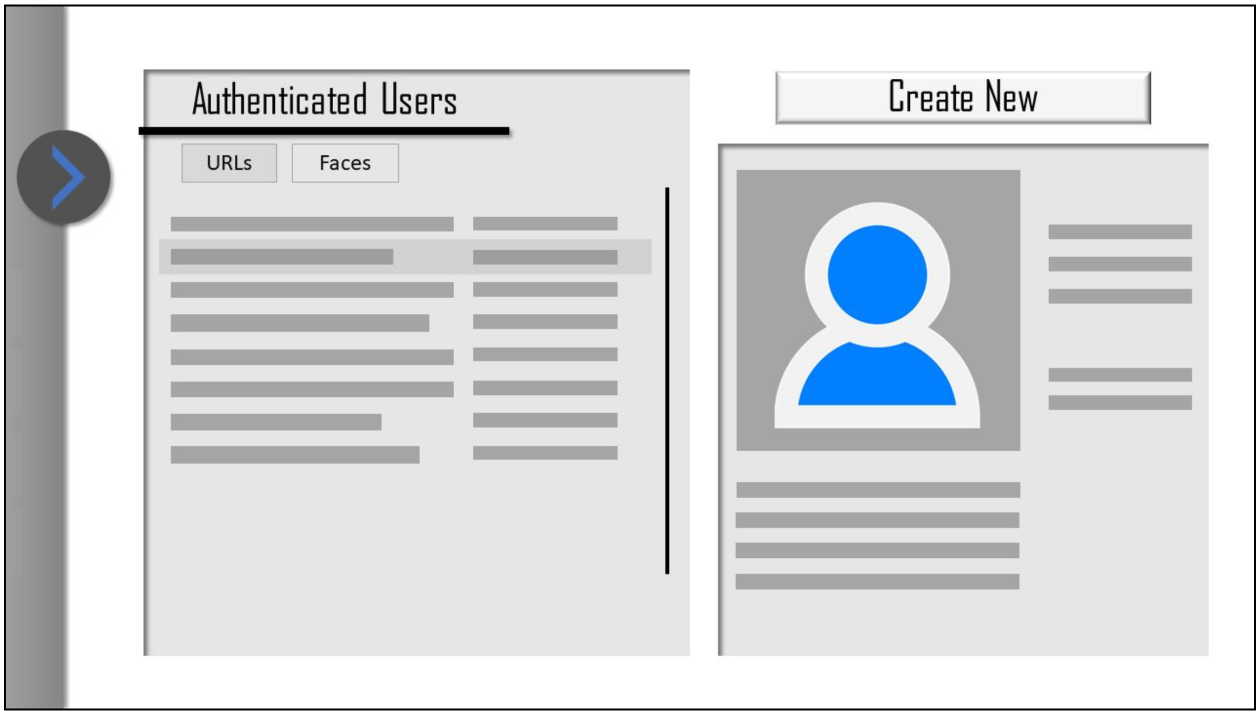
1. ***NAVIGATE*** to "Permissions" (2 mouse clicks)
2. ***Touch*** "Create New" button
3. ***Touch*** "Name" box
   a. ***Touch*** λ keys.
4. ***Touch*** "Image" button
   a. Upload Image *(Undefined keystrokes. Dependent on external directories)*
5. ***Touch*** "Save"

5.3 Conceptual UI Mockup (Extremely Basic)






SMART LOCK
Main Menu
Permissions
Statistics
Options
Manual Lock/Unlock
Entry Log


Authenticated Users
URLs
Faces
Create New

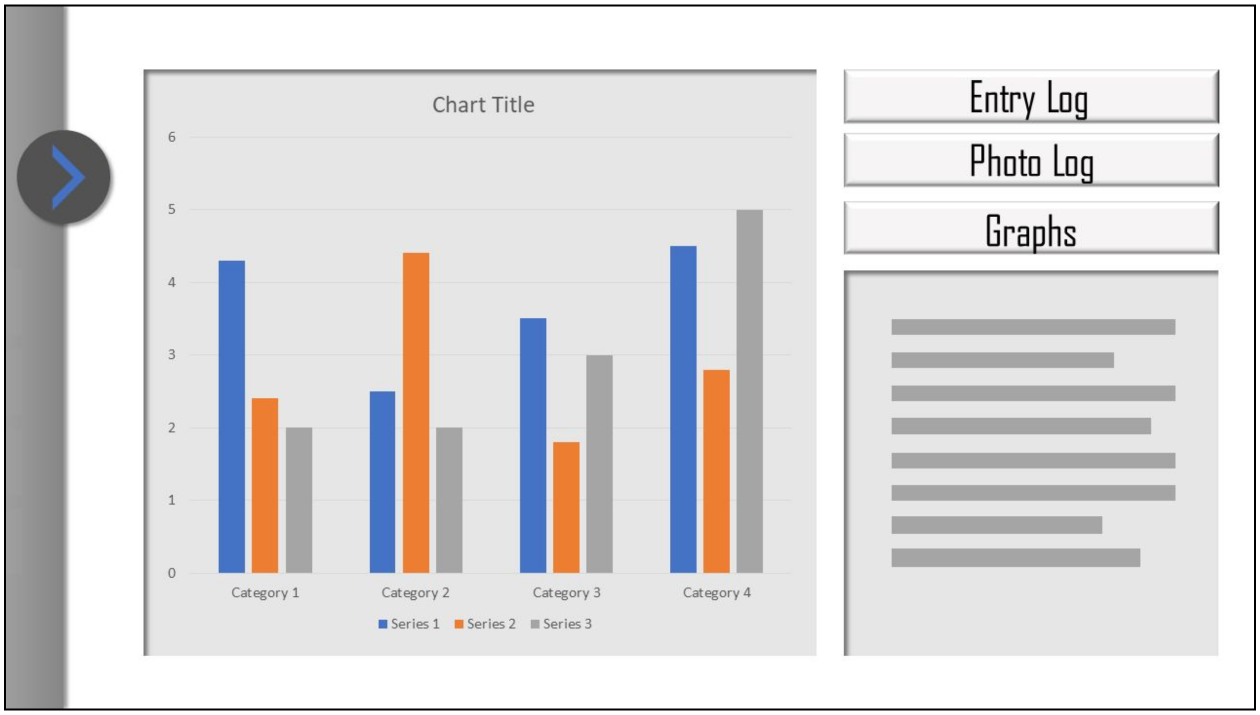


Chart Title
6
5
4
3
2
1
0
Category 1
Category 2
Category 3
Category 4
Series 1
Series 2
Series 3
Entry Log
Photo Log
Graphs

6 Domain Analysis

6.1 Domain Model

6.1.a Concept Definitions

Note: The "Type" column denotes the category of responsibility.
D = "doing", K = "knowing", N = "none"

Responsibility Description	Type	Concept Name	Related Use Cases
To route requests from the user interface to the appropriate handler to service the request.	D	Controller	ALL
To log information needed for use cases in persistent storage, and retrieve information as needed by the handlers for use cases.	K	CentralDatabase	ALL
To detect if an object has passed in front of a camera.	D	ObjectDetector	UC-1, UC-3, UC-4, UC-7
To detect if a face is being shown in front of the camera.	D	FaceDetector	UC-1, UC-3
To analyze the face that has been detected and see if the face is associated with any existing users who have access to the lock.	D	FaceRecognizer	UC-1, UC-3
To unlock the door upon successful authentication.	D	DoorUnlocker	UC-1, UC-4
To log events for a lock in the database.	D	EventLogger	UC-2
To send an SMS notification to a user's phone number upon an event occurring.	D	SMSNotifier	UC-2
To send a push notification to a user's device upon an event occurring.	D	PushNotifier	UC-2

To report an unidentified person trying to gain access to the house associated with a lock.	D	StrangerReporter	UC-3
To display the reportings of strangers within an area.	K	StrangerMap	UC-3
To assign a temporary URL containing a QR code that can unlock the lock exactly once per QR code.	D	TempAuthAssigner	UC-4
To detect if a temporary authentication code is being displayed in front of the camera.	D	TempAuthDetector	UC-4
To verify that the temporary authentication code being presented is in fact valid for one-time unlocking.	D	TempAuthVerifier	UC-4
To register allowed residents along with access permissions for a certain lock.	D	ResidentAdder	UC-5
To log images of a newly added resident's face for future use in facial recognition unlocking.	D	FaceScanner	UC-5
To display lock data to the user.	D	DataGenerator	UC-6
To fetch specific lock data based on specified fields.	D	DataFetcher	UC-6
To mirror the video feed from the camera wherever the user is viewing his/her lock information from.	D	LiveFeedViewer	UC-7
To delete a given resident from the list of allowed visitors for a particular lock.	D	ResidentDeletor	UC-5
To show relevant information about the specific use case to the user and allow for user input	K	WebInterface	UC-2, UC-4, UC-6, UC-7

related to the use case.			

6.1.b Association Definitions

Concept Pair	Association Description	Association Name
Controller ⟵⟶ CentralDatabase	Controller sends an arbitrary request for data to the database and receives the data it requests.	Database request
ObjectDetector ⟵⟶ FaceDetector	If an object is detected, determine if the object is a face.	Detect face
FaceDetector ⟵⟶ FaceRecognizer	If a face is detected, determine if it is recognized in the list of visitors.	Recognize face
FaceRecognizer ⟵⟶ CentralDatabase	Verify the facial detection with the permissions in the central database for the particular lock.	Verify face ID
FaceRecognizer ⟵⟶ DoorUnlocker	If the face is recognized, unlock the door.	Face ID unlock
EventLogger ⟵⟶ SMSNotifier	Upon an event being logged, notify the user via SMS.	SMS notify
EventLogger ⟵⟶ PushNotifier	Upon an event being logged, notify the use via push notification.	Push notify
FaceRecognizer ⟵⟶ StrangerReporter	If a face is not recognized, report the unrecognized face to the Stranger Detection Network (SRN)	Report stranger
StrangerReporter ⟵⟶ StrangerMap	Updates the stranger map with the location of the stranger and the time of the stranger’s visit.	Update stranger map
WebInterface ⟵⟶	Access the temporary	Access temp auth

TempAuthAssigner	authentication interface through the web app.	
ObjectDetector ⟵⟶ TempAuthDetector	If an object was detected, see if it is a temporary authentication code.	Detect temp auth
TempAuthDetector ⟵⟶ TempAuthVerifier	If a temporary authentication code was detected, see if it is valid to unlock the lock.	Verify temp auth
TempAuthVerifier ⟵⟶ CentralDatabase	Check the temporary authentication code with the database for permissions and extra verification	Confirm temp auth
TempAuthVerifier ⟵⟶ DoorUnlocker	Unlock the door after temporary authentication was confirmed.	Unlock temp auth
WebInterface ⟵⟶ ResidentAdder	Access the resident adder interface through the web app.	Access resident adder
ResidentAdder ⟵⟶ Controller	Adds a resident to the database with the given permissions.	Add resident
ResidentAdder ⟵⟶ FaceScanner	Registers a newly added visitor's face to the database, for facial recognition purposes.	Scan resident's face
FaceScanner ⟵⟶ Controller	Preps the database request to add a face image for a new resident.	Add a new face
WebInterface ⟵⟶ DataFetcher	Access the data visualization interface from the web app, in the form of a data fetcher query.	Access data visualization
DataFetcher ⟵⟶ Controller	Prep the data fetcher query for sending to the database.	Prep data query

DataFetcher ⟵⟶ DataGenerator	Update data graphs based on the results of the query from the data fetcher	Update data graphs
WebInterface ⟵⟶ LiveFeedViewer	Access the live feed streaming from the camera.	View live feed
WebInterface ⟵⟶ VisitorDeletor	Access the control panel that removes authentication permissions from select person.	Access auth control
VisitorDeletor ⟵⟶ Controller	Prep visitor delete request for sending to the database	Prep visitor delete
WebInterface ⟵⟶ SettingsUpdater	Access the settings update interface from the web app.	Access settings
SettingsUpdater ⟵⟶ Controller	Prep the updated settings for update in the database	Prep settings update
Controller ⟵⟶ WebInterface	Updates web interface based on results of a database query.	Update web app

6.1.c Attribute Definitions

Concept	Relevant Use Case(s)	Attribute	Attribute Description
SMSNotifier	UC-2	User phone number	Phone number which SMS notifications should be sent to.
StrangerReporter	UC-3	Time of visit	The time the stranger approached the house.
		Stranger image	A captured image of the stranger from the camera.
StrangerMap	UC-3	User location	Location which the map should be

			centered around.
		Map radius	Max radius around the current location to show.
		Heatmap opacity	Determines the opacity of the heat
		Stranger filter	Filters the heatmap by specific stranger sightings.
TempAuthVerifier	UC-4	Temp code	Temporary authentication code generated for one-time unlocking.
ResidentAdder	UC-5	Resident’s Name	Name of the resident to be added.
		Is resident allowed	General access allowance (boolean). If this is “false”, the resident is not deleted from the list of resident, but he/she is not allowed access to the lock until this field is set to “true” again.
		~~Start time~~	~~Earliest time on a given day where a resident can unlock the lock.~~
		~~End time~~	~~Latest time on a given day where a resident can unlock the lock.~~
DataGenerator	UC-6	Data sets	Sets of data to display.

		Data formats	List of formats which the data should be displayed in. Each item in the list corresponds to a data set.

6.1.d Traceability Matrix

Domain Concepts	**Use Case**	1	2	3	4	5	6	7
	PW	15	10	10	5	10	15	15
Controller		X	X	X	X	X	X	X
CentralDatabase		X	X	X	X	X	X	X
ObjectDetector		X		X	X			
FaceDetector		X		X				
FaceRecognizer		X		X				
DoorUnlocker		X			X			
EventLogger			X					
SMSNotifier			X					
PushNotifier			X					
StrangerReporter				X				
StrangerMap				X				
TempAuthAssigner					X			
TempAuthDetector					X			
TempAuthVerifier					X	X		
ResidentAdder					X			
FaceScanner						X		
DataGenerator							X	
DataFetcher							X	
LiveFeedViewer								X
VisitorDeletor								
WebInterface			X		X	X	X	X

6.1.e Diagrams

Due to the the large scope of the project, we have decided to split up a full representation of the problem domain into separate spaces. Each chart shown below is completely orthogonal to the others. That is, each chart can be treated as a "subsystem" for the total Smart Lock product. In general, all subsystems start from the same *boundary* of either the ObjectDetector or WebInterface concepts. From there, paths diverge depending on the goal of the use case.

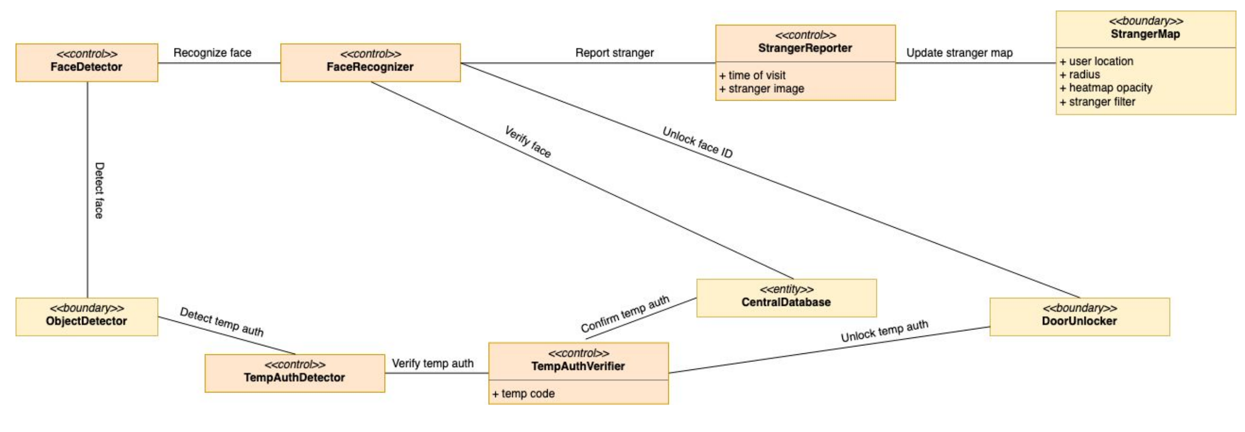


Figure 6.1.e.i:

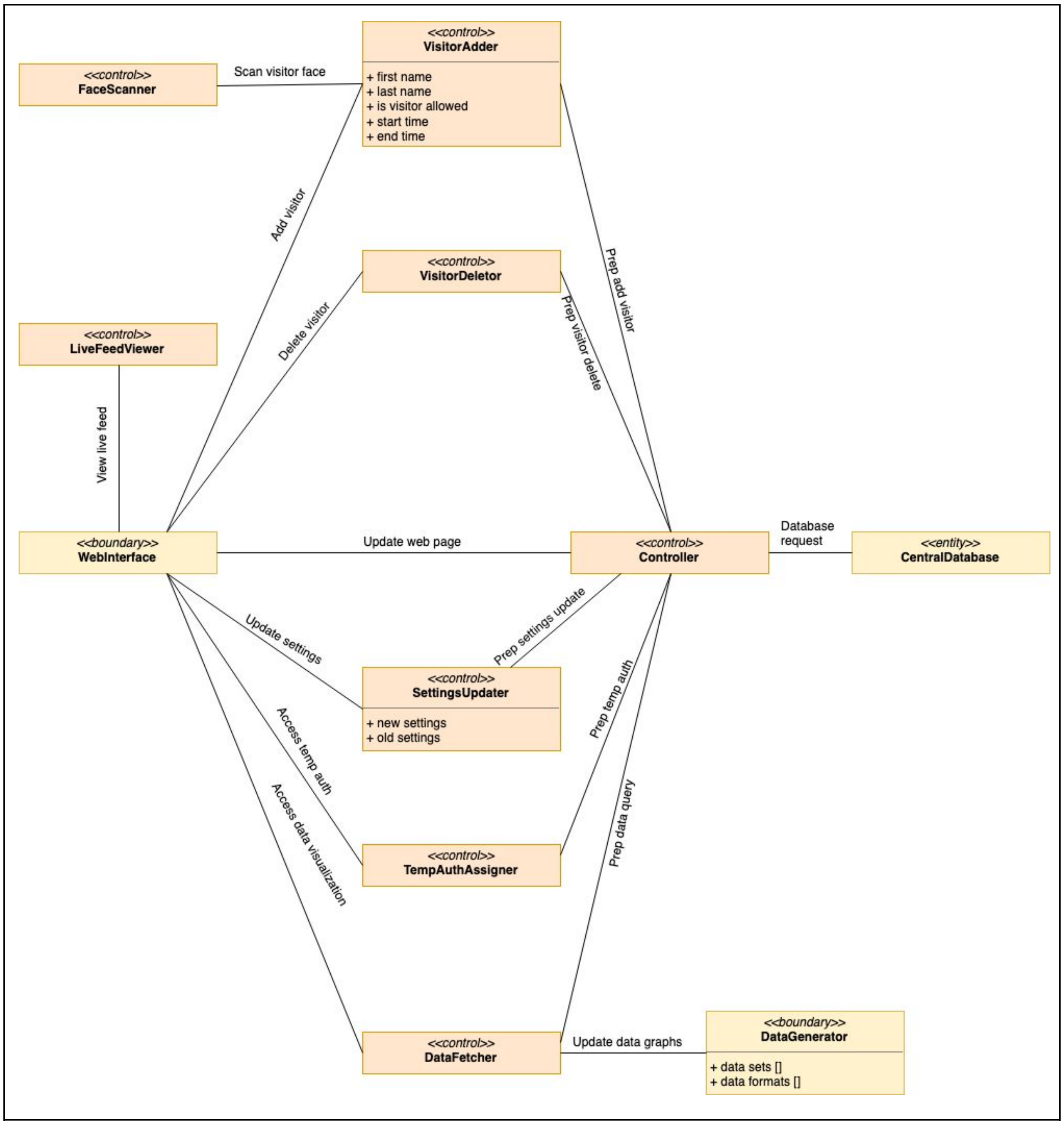


Figure 6.1.e.ii:

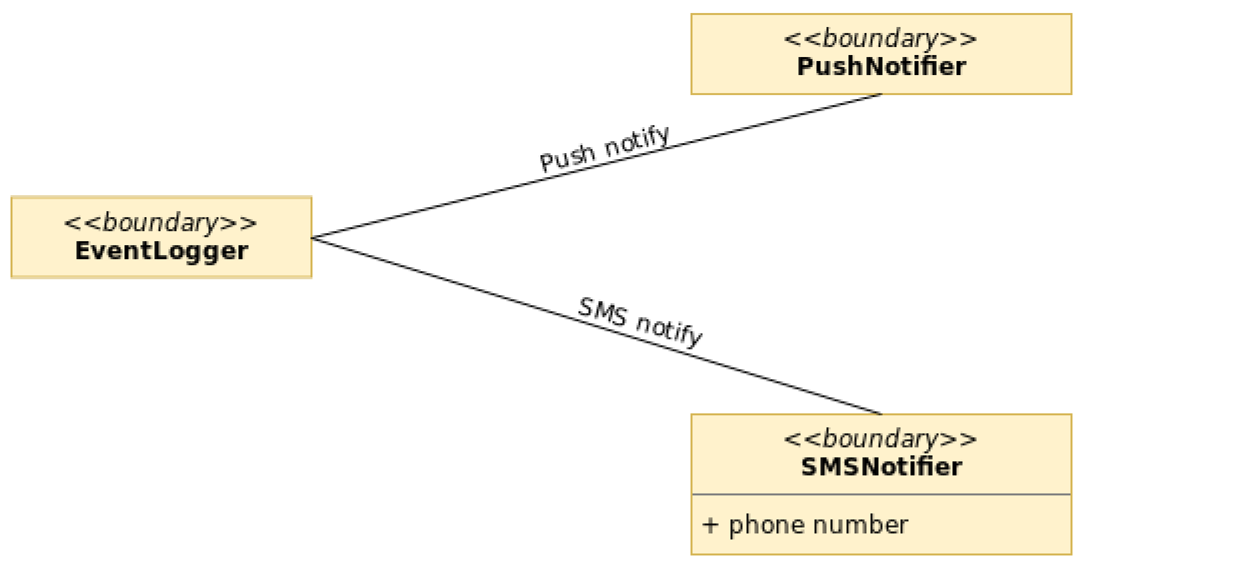


Figure 6.1.e.iii:

6.2 System Operation Contracts

Operation	Unlock [UC - 1]
Responsibility	Automatically authenticate visitor using facial recognition.
Output	Make autonomously make a judgement on whether to unlock and door or not, and notify user.
Preconditions	Visitor approaches entrance. FaceDetector detects face.
Postconditions	Notify Homeowner. If (visitor ∈ authenticated_users) Unlock Else Reject

Operation	Temporary Visitor Authentication [UC - 4]
Responsibility	Visitor is authenticated temporarily using facial recognition or a URL key.
Output	Make autonomously make a judgement on whether to unlock and door or not, and notify user.
Preconditions	Visitor approaches entrance. User holds up QR code.
Postconditions	Notify Homeowner. If (visitor_qr ∈ authenticated_tokens) Unlock; key_uses-- Else Reject

Operation	Add Visitors [UC - 6]
Responsibility	Add visitor's face to database of faces for facial recognition authentication.
Output	Add user's faces to database. Ready for CV.
Preconditions	Homeowner inputs / uploads approximately 5 images of visitor.
Postconditions	Preprocess images. Communicate images to CentralDatabase's repo of images.

7 Interaction Diagrams and Alternative Design

7.1 Interaction diagrams

7.1.a Interaction diagram for UC-1 - Unlock

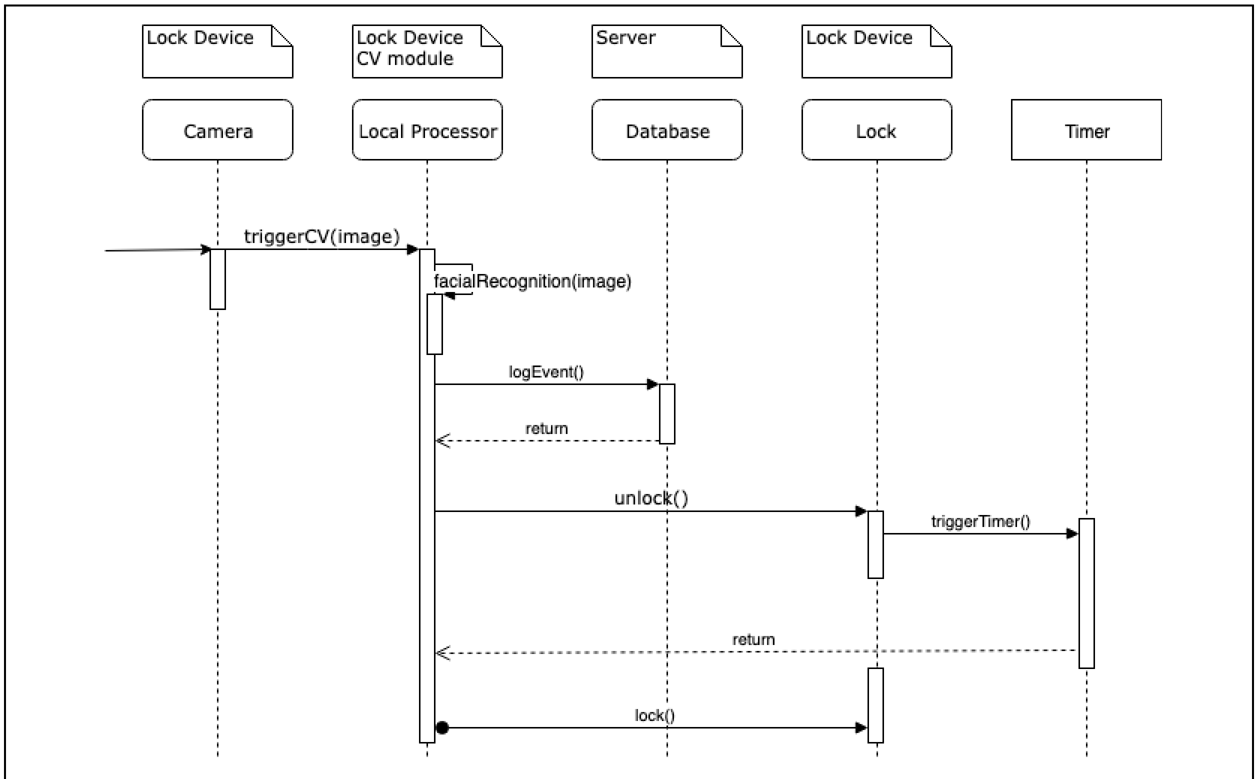


Figure 7.1.a - Interaction Diagram for Facial Recognition and Unlocking:

The approaching person triggers local processor’s CV module, then the Raspberry Pi will do the facial recognition authentication and log event to the server database. If the person is authorized to enter, the local processor will send unlock signal to the lock. When unlocked, a software timer will begin countdown, timer will notify local processor when the countdown is complete and local processor will lock the door again if the door is closed.

7.1.b Interaction diagram for UC-4 -Temporary Visitor Authentication

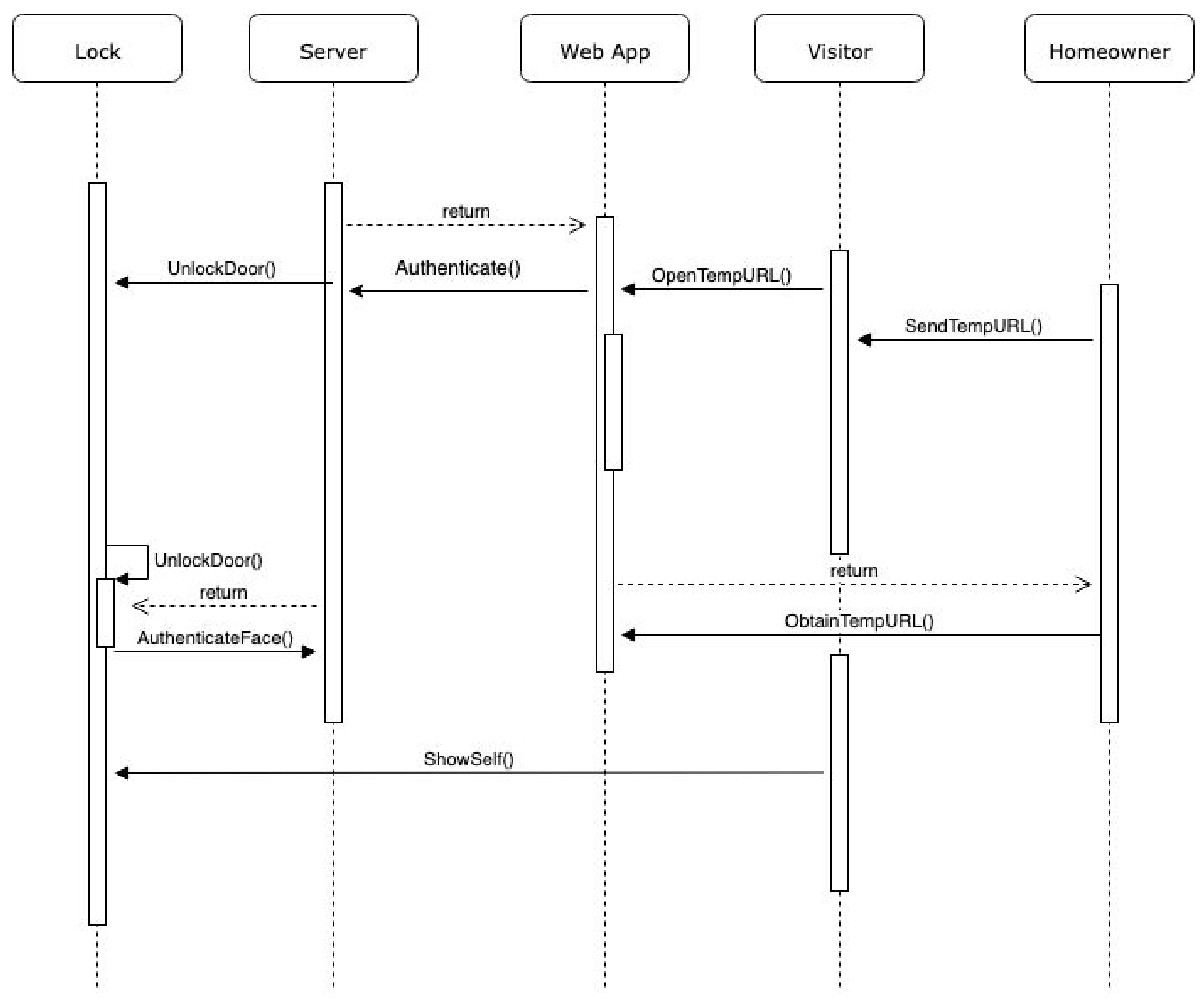


Figure 7.1.b - Interaction Diagram for a Visitor Unlocking the Door:

Visitor uses temporary URL created by the homeowner to enter a web app interface for temperor entrance. Web app pulls from server database to ensure the URL is active. The web app receives authentication and log event while the server notify the lock device to unlock. The rest of this use case proceed in the same manner as UC-1 after unlocking.

7.1.c Interaction diagram for UC-5 - Add/Remove Visitors

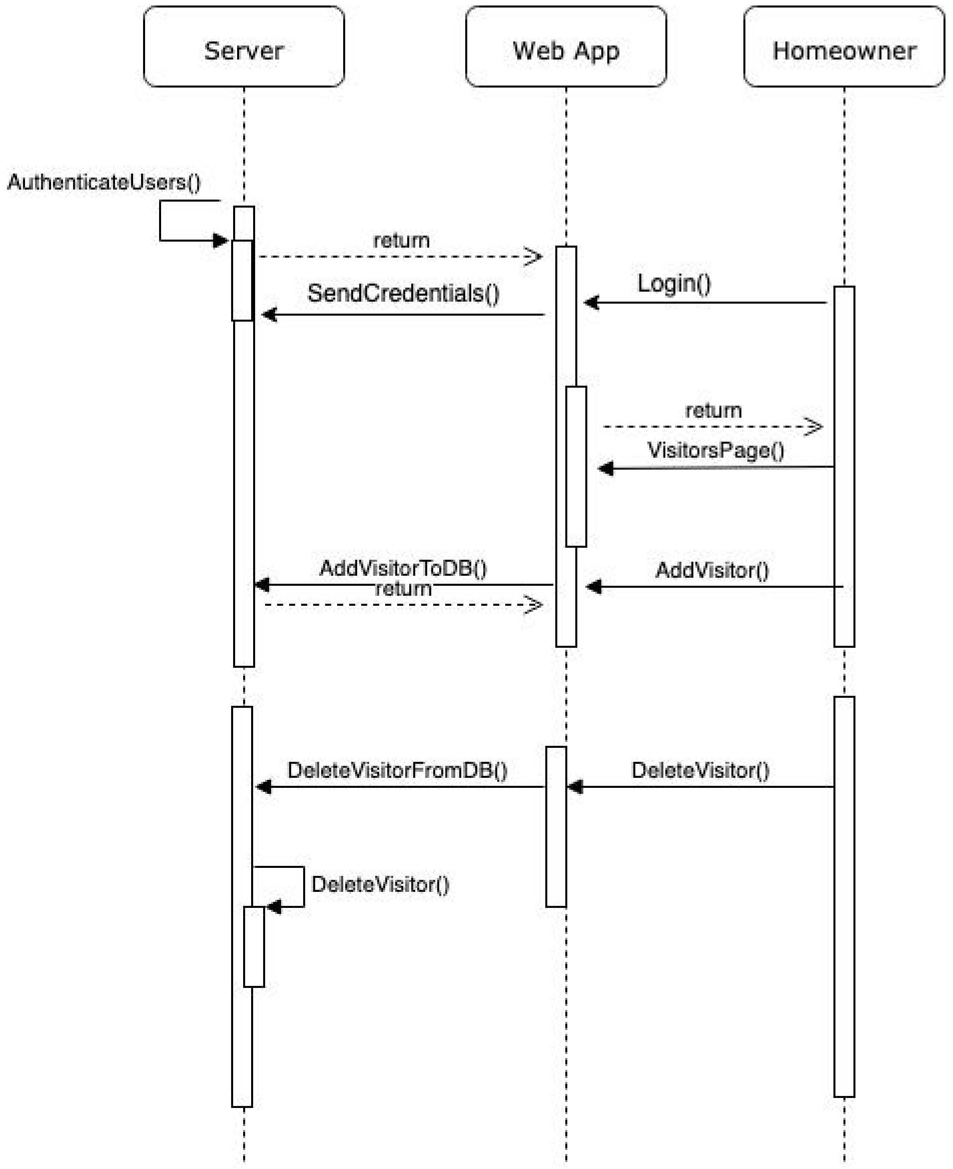


Figure 7.1.c - Interaction Diagram for Adding and Removing Visitors:

Homeowner with access to the web app can add visitor using visitor page. The Homeowner can upload or take a picture of the intended visitor and add the information to the server database, which will be pulled by the local processor for facial recognition authentication.

7.1.d Interaction diagram for UC-6 - Data Visualization

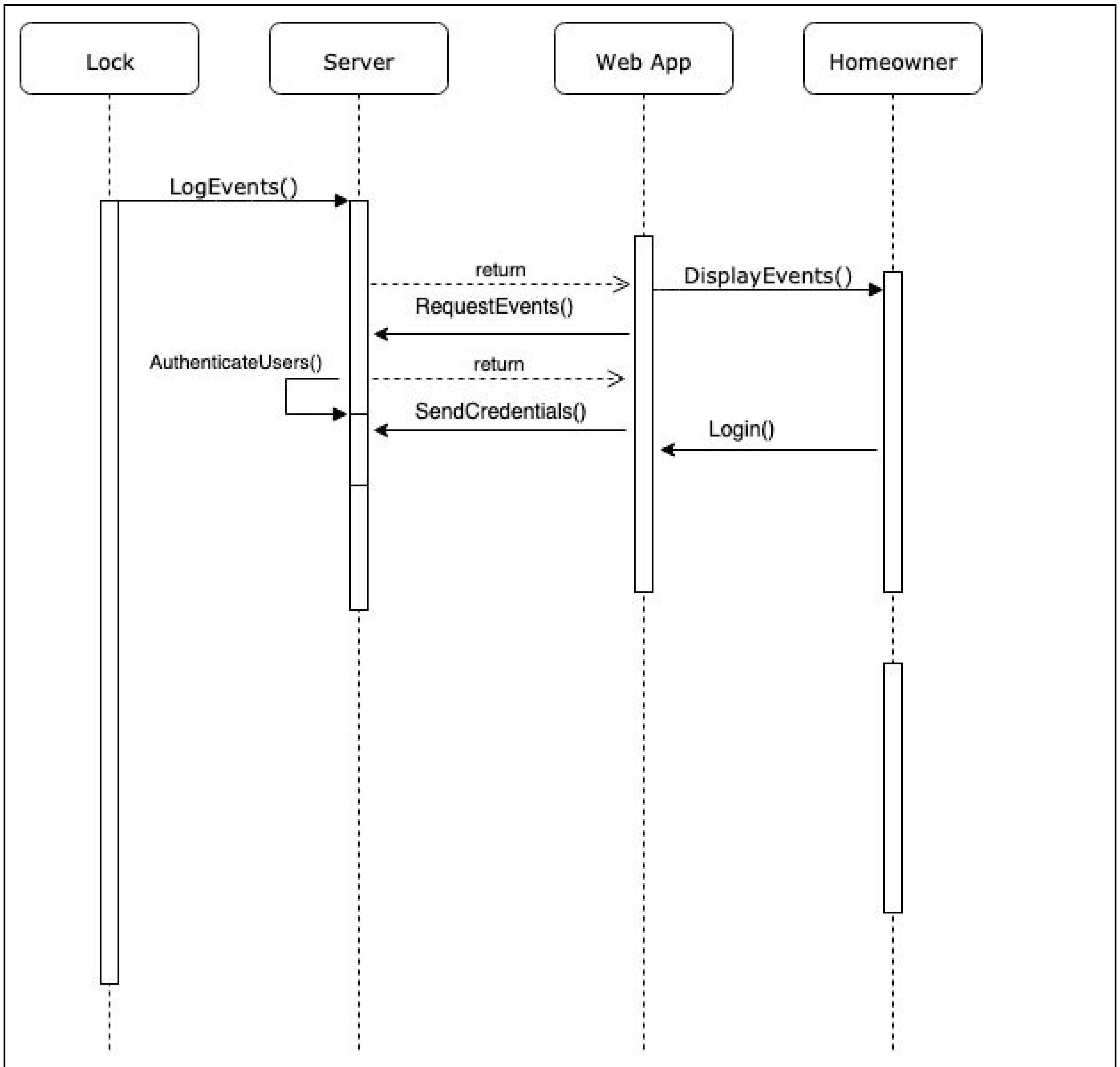


Figure 7.1.d - Interaction Diagram for Event Reporting and Visualization:

Home owner login to the Web App. The Web App request data from server database, and generate data visualization

7.1.e Interaction diagram for UC-7 - Live Video Feed

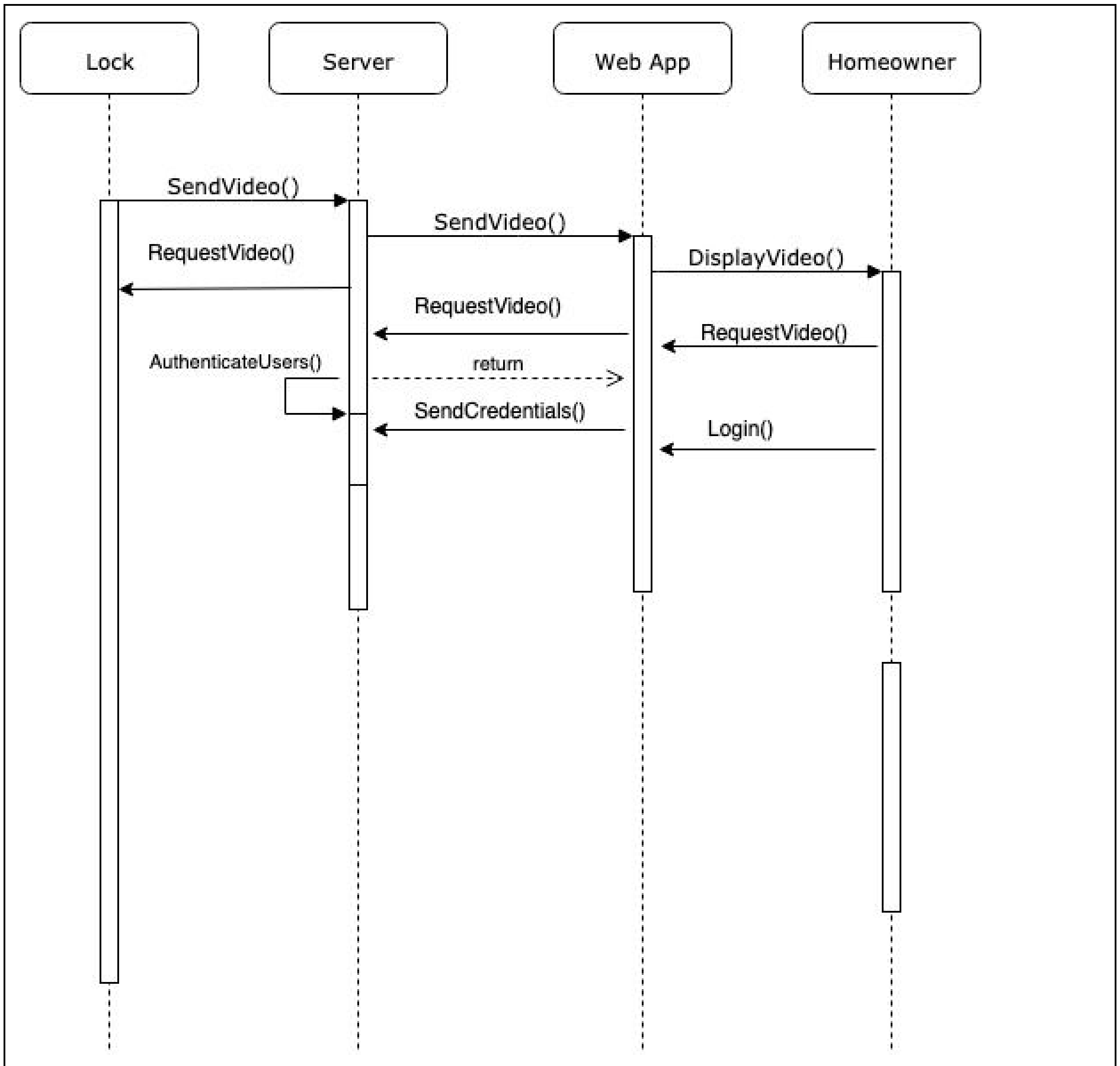


Figure 7.1.e - Interaction Diagram for Streaming Video:

The user is logged in to the web app and then chooses to view the live video feed. The web app requests the video feed from the server and then the server requests and receives the video from the Pi. The video is then sent to the web app and the homeowner can view it.

7.2 Alternative Design

During the initial planning phases it was suggested that the frontend be designed using popular frontend libraries. For starters, it was suggested that we use the React JavaScript library for a component-based frontend. However, it was decided against by members experience with frontend design who thought it to be too complicated for an academic project of this caliber. Plus, modern JavaScript provides us with many helpful interfaces for creating component-based front ends without the need for external libraries.

For the backend, first both Flask and Django were used. Django was being used for the main REST API while Flask was used for the SMS notification network and for the Stranger Reporting Network. Later, we found it to be the better design to merge the two backend systems together. We can allow the stranger report to be linked with a lock and the SMS notification system to be linked with a user with this design. Now, for the backend, we are using Django to implement the REST API.

We are using Flask currently for testing the visitor frontend since the visitor API is easier to implement in Flask for testing purposes.

While designing the facial recognition unlocking mechanism, we encountered hardware implementation problem, namely, the capability of computer. We would love to implement Convolutional Neural Network based face detector, however, Raspberry Pi, the local control computer is incapable to run CNN method. We ultimately decided to use the Raspberry Pi for facial detection for three major reasons: 1, it is necessary to have a computer to control the lock device locally, and Raspberry Pi computer can make breadboard prototype; 2, to have facial recognition happen on local processor reduces communication overhead; 3, Raspberry Pi computer, although unable to run more complicated CV method, is still a reliable facial recognition tool. The alternative to using local processor to do facial recognition is locating CV module in the server, which is ideally more powerful and capable to run CNN method. But due to advantages of Raspberry Pi in term of practicality, this design is abandoned.

8. Class Diagram and Interface Specification

8.1 Class Diagram

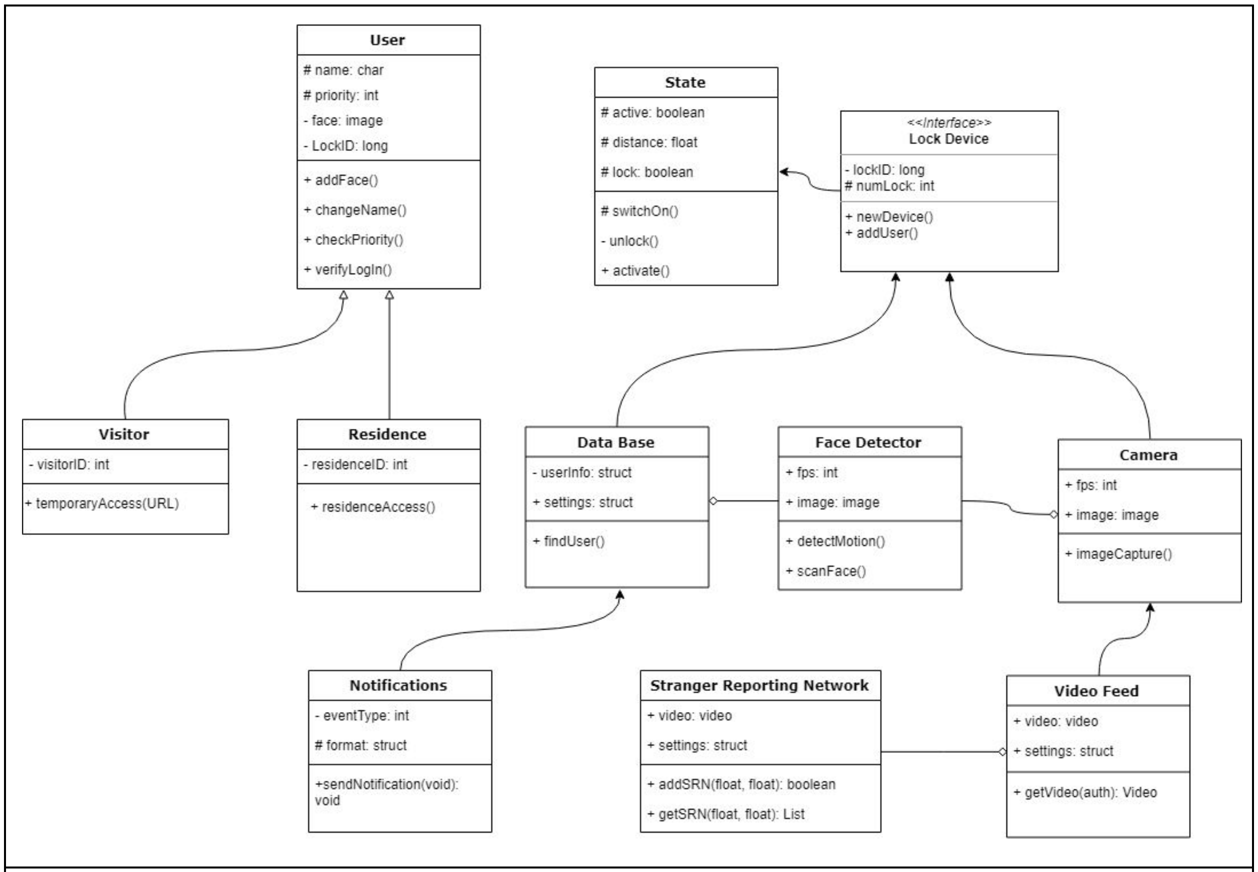


Figure 8.1.a: This UML class diagram shows the different subsystems of our product, their methods and parameters, and how they interact with each other.

8.2 Data Types and Operation Signatures

User:
A user is anyone with permission to enter the home whether it be a visitor or a resident.

Attribute	**Type**
name - Name associated with user	char
priority - How many entries allowed	int
face - the face of user stored as an image	image
LockID - ID associated with face to unlock	long

Method	**Type**
addFace() - storing user's face	void
changeName() - changing name if desired	void
checkPriority() - checking number of entries allowed	void
verifyLogin() - verifying if entry allowed	void

Visitor:
A visitor derives from user, and is allowed an entry to home by a temporary generated url.

Attribute	**Type**
visitorID - ID associated with visitor	int

Method	**Type**
temporaryAccess(URL) - allows acces to vistor to access home	void

Residents:
A resident derives from user class and is allowed access to home via facial recognition.

Attribute	**Type**

residenceID - ID associated with resident	int

Method	**Type**
residenceAccess() - checks if a person is a resident and has access to home	void

State:
State describes which condition the lock is currently in.

Attribute	**Type**
active - In process of changing states, unlocked currently	boolean
distance	float
lock - lock is locked	boolean

Method	**Type**
switchOn() - lock begins to change states	void
unlock() - unlocks lock	void
activate() - tells lock to perform action	void

Database:
Stores information and allows changes to information.

Attribute	**Type**
userinfo - information associated with users, i.e. names, faces	struct
settings - allows changes to userinfo	struct

Method	**Type**
findUser() - searches through user info for ID associated with name or face	void

Notifications:
Notification system to alert homeowner of an event.

Attribute	**Type**
eventType - event that would trigger a notification	int
format	struct

Method	**Type**
newNotification() - generates notification	void
sendNotification() - sends notification	void

Camera:
Camera connected to lock.

Attribute	**Type**
fps - frames per second	int
image - image being output each frame	image

Method	**Type**
imageCapture() - camera takes and stores image	void

Video Feed:
Video being input by camera.

Attribute	**Type**
video - video taken by camera after motion detected	video
settings - settings	struct

Method	**Type**
getVideo(auth) - outputs clips to database	Video

Stranger Reporting Network:
Storing videos of suspicious strangers to a database.

Attribute	**Type**
video - video feed from event with stranger	video
settings - settings	struct

Method	**Type**
addSRN(float, float)	boolean
getSRN(float, float)	List

Face Detector:
Searches for human faces.

Attribute	**Type**
fps - frames per second	int
image - image of face	image

Method	**Type**
detectMotion() - detects motion in front of camera	void
scanFace() - compares face to stored images of residents/visitors faces	void

Lock Device:
Device keeping door sealed from intruders.

Attribute	**Type**
lockID - ID associated with lock	long
numLock	int

Method	**Type**
newDevice()	void
addUser() - adds users as residents or visitors with permission to enter home	void

8.3 Traceability Matrix

UC-1: Unlock - Gives the homeowner and residents ability to unlock the door through facial recognition.
UC-2: Notifications - Allow homeowners to receive notifications about events.
UC-3: Stranger Reporting - Allow homeowners to report suspicious activity in the neighborhood.
UC-4: Temporary Visitor Authentication - To allow visitors to the home in one(or more) time(s) using a URL or temporarily adding them to trusted faces.
UC-5: Add/Remove Visitors - To add residents as visitors with permission to enter the house with facial recognition or remove these visitors.
UC-6: Data Visualization - To allow homeowners to view data pertaining to who enters the house and when then enter.
UC-7: Live Video Feed - To obtain a live feed of the home entrance through the camera.

	UC - 1	UC - 2	UC - 3	UC - 4	UC - 5	UC - 6	UC - 7
User	X						X
Log	X					X	
Notifications		X			X		
Video	X				X		X
Lock Device	X			X			X
Stranger Reporting Network			X			X	
QR Code Reader				X			
FaceDetector	X		X				

9 System Architecture and System Design

9.1 Architectural Styles

In general, we will use a **Model-View-Controller framework**. The frontend of the web application will be the view which is what the user interacts with. The frontend is responsible for sending HTTP requests to the backend. The model is our database which stores all of the data. The controller will be our backend which is the interface between our database and the frontend. Each of these will also incorporate other architectural and design styles as well.

The frontend will utilize a **Component-based** architecture for each of its features. These features will be separated into distinct sections/components so that each section addresses a different concern. Although they are separated, all the features will utilize a common interface.

Serving the web app will use a **Client-Server** architectural style. When the user's browser requests the web page, the HTML/CSS/JavaScript related to the page will be transmitted to the user's browser from the web server. This is how traditional web servers work, and we will be following the same model. This process will be greatly simplified provided the web app will be contained within a single page. Consequently, the web server will be able to cache the necessary files, allowing for very quick handling of requests.

All backend data will be contained within a **Central Repository**. The associated relational database will contain all data relevant to user accounts (names, owned locks) and locks (allowed visitors, user/visitor data, etc).

To access backend data, the frontend will use **Representational State Transfer (REST)** to communicate data. For example, when the visitors component is selected by the user, by default it will request a list of active temporary keys from the backend via an HTTP *GET* request. The backend will look up all temporary keys associated with the user's lock and transfer the data back to the client. All other frontend components will use a similar style to access data over the RESTful API.

The Lock device software uses an **Event-driven** architecture. As the name suggests, this system detects and reacts to events, specifically realizing an object approaching. In this structure, we have *event emitters (agents)* and *event consumers (sinks)*. Event emitters are responsible for detecting a person is nearby, collect visual information via a camera, and send the data.

Peer-to-peer communication will be used to transfer video streams of the camera directly to the homeowner's web app client. Peers make a portion of their resources, such as processing power, disk storage or network bandwidth, directly available to other network participants, without the need for central coordination by servers or stable hosts.

9.2 Identifying Subsystems

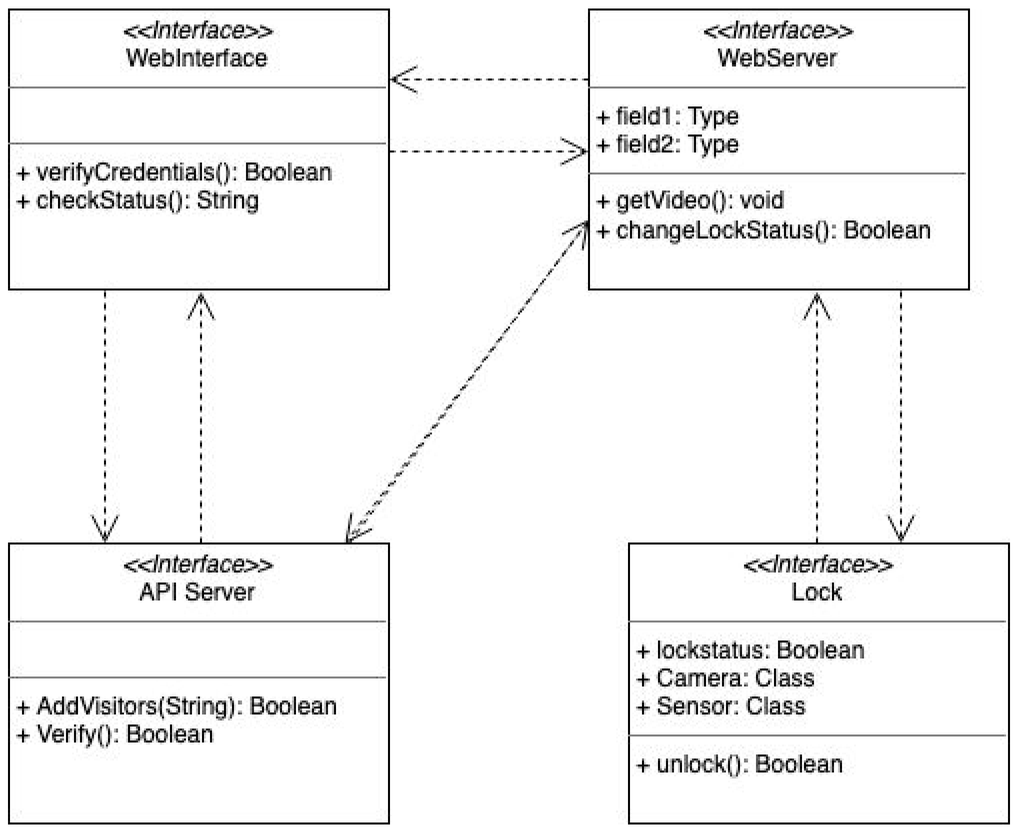


Figure 3.2.a: For our lock subsystem, we have the Raspberry Pi. The Raspberry Pi interfaces with the camera and sensor. These components make the lock. The lock interfaces with the web server only. Then, we have the API server which handles the API requests. The API server interfaces with the website and the web server. The website consists of the front-end user interface. The website interfaces with the API server and the web server. Finally, we have the web server. The web server interfaces with all other interfaces and can be thought of as a middleman.

9.3 Mapping Subsystems to Hardware

Subsystem: Lock

Lock will be the implemented as the lock Device, which consists of a local processor, a physical lock, and a camera (& sensor).

Subsystem: REST API

The REST API will run on the server. The API will interface between the database and the web app.

Subsystem: Web App

The Web App is hosted on the server, which we will host on a personal computer.

9.4 Persistent Data Storage

9.4.a Role of Persistent Storage

The need for persistent storage in our application is paramount to the success of the application as a whole. In our case, persistent data storage will be used to ensure that user experience is consistent on a user-by-user basis, which is a key part of our application. In addition to the fairly simple problem of storing user data, the persistent data storage system must also be able to store images and videos of visitors who approach the lock. This requires the storage system to interface directly with the lock subsystem to create new entries for visitors approaching the lock.

Aside from typical persistent storage applications, it is necessary from a security standpoint for our storage system to make sure individual user settings are separate from others to ensure the security of the lock. If a given user could access the information of other users, it would be detrimental as it would present a huge security vulnerability in our application. Therefore, we have the need to build an additional layer of authentication on top of our data storage system. This authentication system will restrict certain data requests to certain users to ensure that data physically cannot be accessed without breaking into a user's account

9.4.b Persistent Storage Strategy and Motivations

There are numerous types of persistent data storage systems. An abbreviated list of most commonly used types of persistent storage, along with an evaluation of their usefulness for our application, is shown below:

- *File based storage* is one of the most simple types of persistent storage. A file is opened and written to, and data is read from the file whenever requested. This system is useful for logging applications where a file needs to be appended to on an event. However, a file based system has the major drawback of only allowing "sequential" reading and writing access, which means that it is easier to read from the beginning of the file than from a random place in the file. This makes file based storage systems unsuitable for our application, as we would need to access data from users at seemingly random orders, which goes against the primary advantages of a file based system.
- *Relational databases* are one of the most popular types of persistent data storage today. They use SQL (Structured Query Language) to interface with a distributed network of files, in order to provide quick reads and writes to the database. One advantage of relational design is its following of ACID principles. ACID guarantees quick database operations that can be isolated from one another ("atomicity"), a

clean way to maintain database state while guarding from corruption ("consistency"), handling of concurrent database operations ("isolation"), and guarantees of future database state after changes ("durability"). These principles make relational databases a very reliable choice for our application. One drawback of relational databases is their need for highly structured data in the form of database tables, with clearly defined, unalterable field categories for each entry of the table.

- *Non-relational databases* act much in the same way as relational databases, with the primary difference being their ability to handle unstructured, flexible data. They also can guarantee even faster lookup speeds than relational databases due to their flexible nature. Unfortunately, their advantages in that category also lead to their main drawback: they do not guarantee the same level of reliability that relational databases do with their ACID principles.

Looking at these choices, it becomes clear that the storage system should be designed with a relational database. Because of our needs for reliability, relational databases cannot be beat. The drawback of being forced into structured data is not an issue for our application. However, this is not a concern for our application because all users will have essentially the same types of data associated with them, and it is easy to work around the limitations of relational databases to create more flexible designs.

9.4.c Database Schema

9.4.c.i Derivation of the Database Schema

The structure of the database was derived by considering the needs of each subsystem as seen in the domain model and architecture analysis.

A User table is surely needed to keep track of user attributes such as name, username/password, and phone number.

A Lock table is needed to manage the locking systems. Each lock will be owned by exactly one user, but a user can own multiple locks. Also associated with each lock will be the address of the home/building the lock is installed at.

In order to maintain permissions for users and locks, a separate table must be created to store access permissions for registered users as they relate to locks. Such permissions can include general unlock abilities and time ranges where users are allowed to unlock the lock. Note that even though permissions may be set, the unlocking ability is ultimately dependent on the ability of the lock and camera system to identify the user.

For the Stranger Reporting Network (SRN), a separate table is required to log stranger visits as they relate to locations on a map. The time of stranger visits must also be logged in this table, along with an image of the stranger. This forms a complete log of an unidentified user's approach to the lock system so that it can be used by the SRN to take appropriate action.

Along with all of these fields, separate tables for user images and videos are also required. The images and videos themselves can be encoded as binary data in the databases, and they will be associated with users. Each image or video can only have one user.

9.4.c.ii ER Diagram for the Database

Using the derivation presented above, it is simple to construct an ER diagram that will guide the organization of data for our application. The complete ER diagram is shown below:

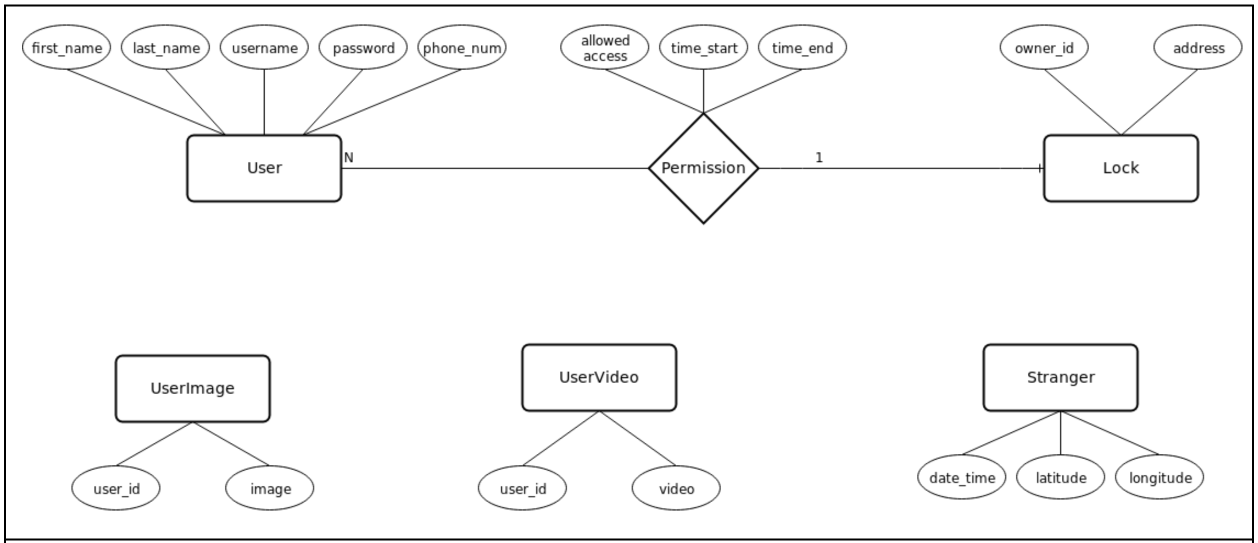


Figure 3.4.c.ii

Reading an ER Diagram:

- *Rounded rectangles* represent the "entities" of the system: essentially key components of the storage system.
- *Ovals* represent "attributes" of an entity or relation. These attributes are what the database will store in relation to the entities.

- *Diamonds* represent relations between two entities.
- *N or 1* on a line represents the degree of the relationship. If the mark is N, there can be more than one of the entity that is being connected in the relation. If the mark is 1, there can only be one entity in the relation for each entity on the other side of the relation.

9.5 Network Protocol

In order to orchestrate the communication between the lock, central server, and the web app, the project will use REST HTTP calls and use JSON for all communication. HTTP protocol will also be used for live streaming and the transmission of video between the lock, server & web app.

9.5.a HTTP

HTTP is an application-level protocol (meaning it connects parts of a large internet application together). HTTP is very reliable, using many areas of redundancy to achieve good error resilience. The basis of the HTTP framework is the use of URLs to identify resources (any data that is of interest to the application).

On the client side of an application, HTTP supports a series of "methods", which define how the data sent in an HTTP request should be acted on. The major types of methods are summarized below:

- GET - gets the resource at a URL that is passed in with the request.
- POST - creates a new resource at the URL passed in with the request, corresponding to the data that was also passed in.
- PATCH/PUT - alters the already existing resource at a passed in URL according to the data fields passed in.
- DELETE - deletes the resource at the URL passed in.

Data in HTML is typically represented in JSON form. JSON is a Javascript-like format for storing data that all web browsers support. JSON is easy to encode into the proper HTTP format, which makes it an attractive option for attaching data to HTTP requests.

HTTP responses are sent back when a server receives an HTTP request. The server tries to fulfill the request, and depending on the result, sends back a response code that gives the request sender an idea of the results of the request. The server can also attach JSON data to the response just as the client can to a request. This enables the client to receive results of operations (such as a GET request) from the server. Some common response codes are described below:

- 404 - resource was not found at the URL.
- 200 - operation succeeded with no errors.
- 401 - sender of the request is not authorized to access the URL.
- 503 - the resource at the URL exists but is unavailable for some reason.

By carefully managing HTTP requests and responses, it is possible to design a system in completely separate components, using HTTP requests to one another to communicate instead of traditional object-oriented programming communication (which is done through method calls and parameter passing). We design our application with this approach in mind. Because we are operating with essentially three different computers (the camera system, the web application, and the central repository), there needs to be a clearly defined way for communication between machines, which HTTP services perfectly.

9.5.b TCP/IP

TCP and IP operate at a lower level than HTTP, dictating how messages should be sent over a network. Specifically TCP deals with how to package and interpret data for transport between applications, and IP deals with where to send the data and how to direct it to its intended location. The details of TCP/IP are not relevant to the scope of our application, so we will not discuss them here. Instead, we will note that TCP and IP form the basis of the Internet, offering very high reliability and being very error resilient.

9.5.c Websockets

Websockets are an extension of the *socket API* used in many programming languages to Javascript, the language of the Web. To introduce websockets, it is first necessary to describe how sockets work in general. These same concepts are slightly modified when considering websockets, but the general ideas are the same.

Sockets provide a low level interface into the TCP/IP protocol. They allow direct transfer of data between two WiFi-connected computers. By creating sockets on two computers and linking them together, a line of communication is established, and each of the computers can send or receive messages from the other.

Websockets are built on top of regular sockets, and they allow websites to communicate over a single TCP connection. Websockets are typically used because of their low communication overhead and little setup resources, compared to HTTP. The most common uses of websockets are to stream video, which is exactly what we will be using them for. Websockets will form the core of our live streaming system.

9.6 Global Control Flow

9.6.a Execution Orderliness

Our lock exhibits an event driven system. Specifically, the CV module waits until it receives a notice form supersonic motion detector. After this, it goes into the state in which it will conduct facial recognition and authenticate. If a face was authenticated, the door unlocks; otherwise, the door stay locked. After the recognition is done, it exits facial recognition and waits until it receives another notice form supersonic motion detector.

For our server, we will receive HTTP requests for either adding data to the database or sending data. First, we will receive an HTTP request at a certain route for the server. If the request is a GET request, we will return the corresponding data from the database based on authentication of the user. If the request is a POST request, we will conduct a specific action. Examples of actions are adding data to a database or sending a SMS notification or push notification.

Our web app can be described as an event driven system. In the beginning, the homeowner (admin) has to login. In this sense, the app may be considered linear. Afterwards, the app behaves as event driven. The user can choose which page to view which will bring the user to a different page. The user can then perform certain actions on the respective page. These actions might be done in a linear fashion. For example, adding a visitor requires first choosing the option to add a visitor and then uploading a picture.

9.6.b Time dependency

A software timer is used during countdown when door is unlocked. If the door is unlocked but not opened for a certain time period (for example, 10 seconds), the door will be locked again.

9.6.c Concurrency

For our server, we are using Django as a framework. Django will handle all concurrency related issues so we do not have to account for multiple connections to our server. For our lock, we will use multithreading to send the video to the user and to the server. There are no major concurrency issues here since we are not modifying data. Rather, we are just reading data from the camera. Our web app is not multithreaded as well so there are no concurrency issues here.

For our main lock device, we used concurrency. We were running three separate processes in the background. We were running facial recognition/detection, QR code detection and the streaming server. Thus, we had to create a shared queue that was shared among the processes. The queue held the newest frames from the camera. As a result, we did not waste processing power just getting the same frames for all three processes.

9.7 Hardware Requirements

9.7.a Lock Device: (Lock + Local Processor + Camera & Sensor)

- Local processor: is a Raspberry Pi 3 computer, capable of facial recognition.
- Camera: must be compatible with Raspberry Pi 3.
- Electric Lock: the physical lock is implemented by a DC motor, which is controlled by the Raspberry Pi processor.

9.7.b Server

- Will run on a hosting service.
- Storage : > 5 GiB. This will give room for storage of frames. However, required size heavily depends on the amount of clips, and the amount of households currently registered.
- Bandwidth : 25 mbps upload and download. Very rough estimates.

10 Algorithms and Data Structures

10.1 Algorithms

In order to effectively stream the video from the lock to the user and the server, we will need to incorporate usage of algorithms. First, we will need to compress the video. Compression will reduce network bandwidth usage when streaming and will also allow less storage to be needed in order to store videos. For compression, we will use H.264 compression which is used for most video compression. After compression, we can send the video using sockets.

We will also incorporate algorithms for our Stranger Reporting Network. Specifically, when we send the data to the frontend for the frontend to display local reports, we do not want send all reports in our database. Rather, we would like to send only those reports which are in close proximity to the user in question. To do this, we make use of a mathematical formula known as the Haversine formula which will allow us to calculate the distance between two coordinate positions on the Earth. Using this, we can filter the data to only return reports within a certain radius of the user, effectively decreasing the load on the frontend.

For facial recognition on Raspberry Pi, we will use the Histogram of Oriented Gradient (HoG) based face detector method due to the fact Raspberry Pi computers are incapable of running convolutional neural network (CNN) based face detector.

HoG is a feature descriptor in image processing, the essential thought behind the histogram of oriented gradients descriptor is that local object appearance and shape within an image can be described by the distribution of intensity gradients or edge directions (wikipedia). We will extract these descriptors and train the machine with deep learning.

Deep Learning is a machine learning method that is constructed based on a greedy layer-by-layer architecture, it will disentangle each layer of network and improve performance . We employed the general purpose library in python -- Dlib toolkit to implement machine learning algorithms.

10.2 Data Structures

The different components of our Web App frontend will utilize a number of simple data structures. Provided that JavaScript allows for simple creation JSON objects (key-value pairs) we will be sure to take advantage of them. A major example being REST requests/responses.

Provided that our Web App frontend is designed around a RESTful backend framework, we must have a standard data structure for communicating between the frontend and backend. For this we have chosen to use JSON-encoded strings as our main data structure. These are simple string-encoded key-value pairs which are native to JavaScript.

When the frontend sends a request to the server, it will need to include a JSON payload with specifically named keys. In the figure below, the key *auth_code* is required for this type of request. If this key is not present or is not of the correct type, the server will return an error.

```
{
    auth_code: "8092acfb-ae28-4027-87c6-3f9a1dee5d72"
}
```

Figure 10.: Example of a request payload sent the frontend

Every message sent back from the server will contain at least two keys: a numerical *status* code and a *message* encoded as a string.

```
{
    status: 200,
    auth_code: "8092acfb-ae28-4027-87c6-3f9a1dee5d72",
    message: ""
}
```

Figure 10.: Example response following a successful request

However, an unsuccessful response will just include *status* and *message* values. This error message can then be shown to the user via the frontend.

```
{
    status: 404,
    message: "Invalid visitor ID requested"
}
```

Figure 10.: Example response following an unsuccessful request

11 User Interface Design and Implementation

One major improvement over the previous **mobile** interface design is that the menu allowing access between component views will no longer be a square menu button. We decided that having a grey bar along the side of the display spanning the entire height would be enough of an indicator. The user will swipe from the side of the screen where the bar is hidden to the opposite side, causing the menu to be displayed. This not only reduces screen clutter but also provides access to the menu at a similar or better speed. A swiping gesture can be much faster and more intuitive than pushing.[4] This change will affect navigation throughout the entire app.

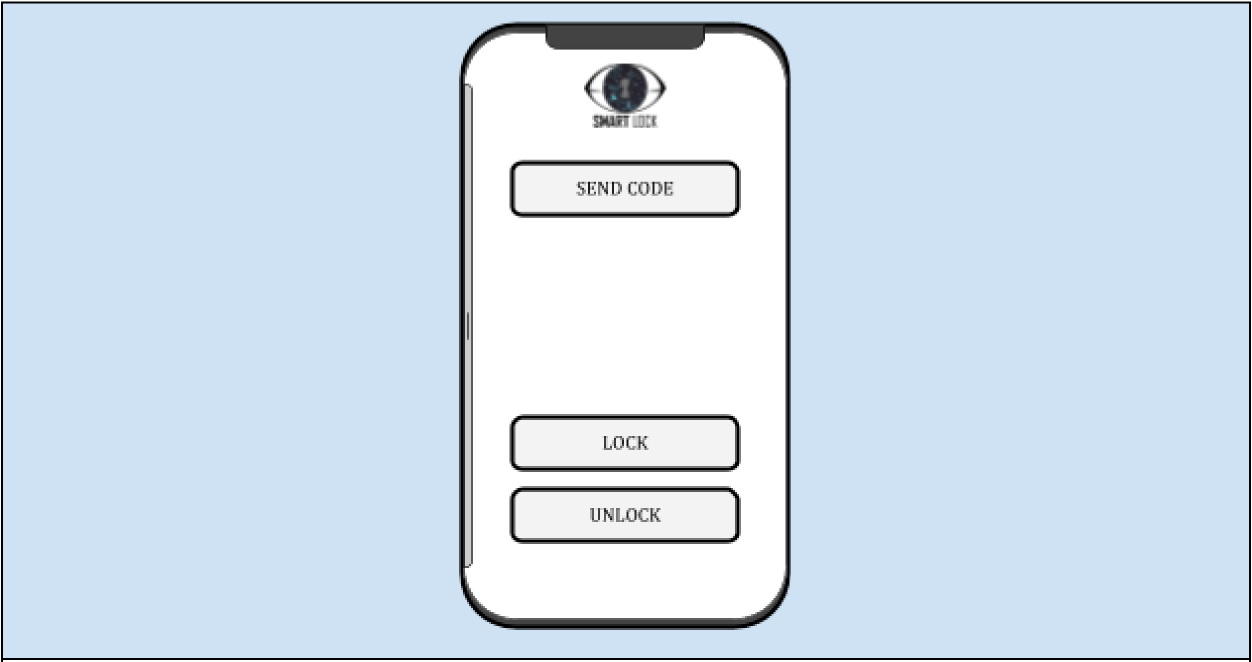


Figure 11.a -- The differently colored bar on the left side of the screen indicates that a menu is available for swiping.

Another improvement was the overhaul of the **desktop** main menu as well as the addition of a live stream component accessible via the side menu. Having the live camera feed play when the user opens the app could be problematic due to the amount of bandwidth needed. Therefore, we moved the live feed into its own component and redesigned the main page to only include important summary information and key controls.

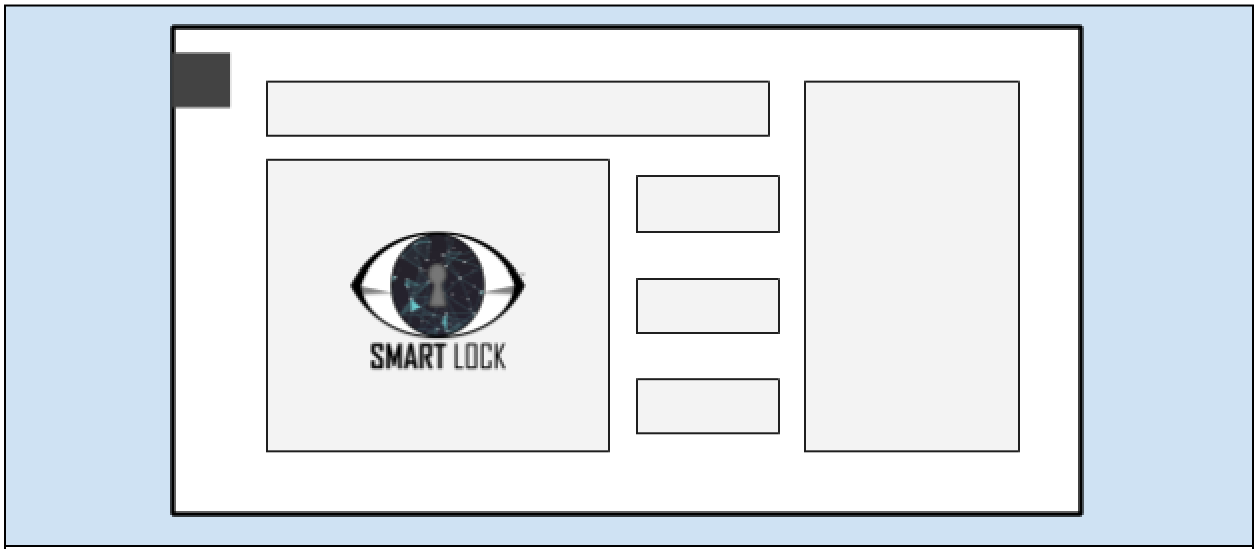


Figure 11.b -- Main menu without live feed

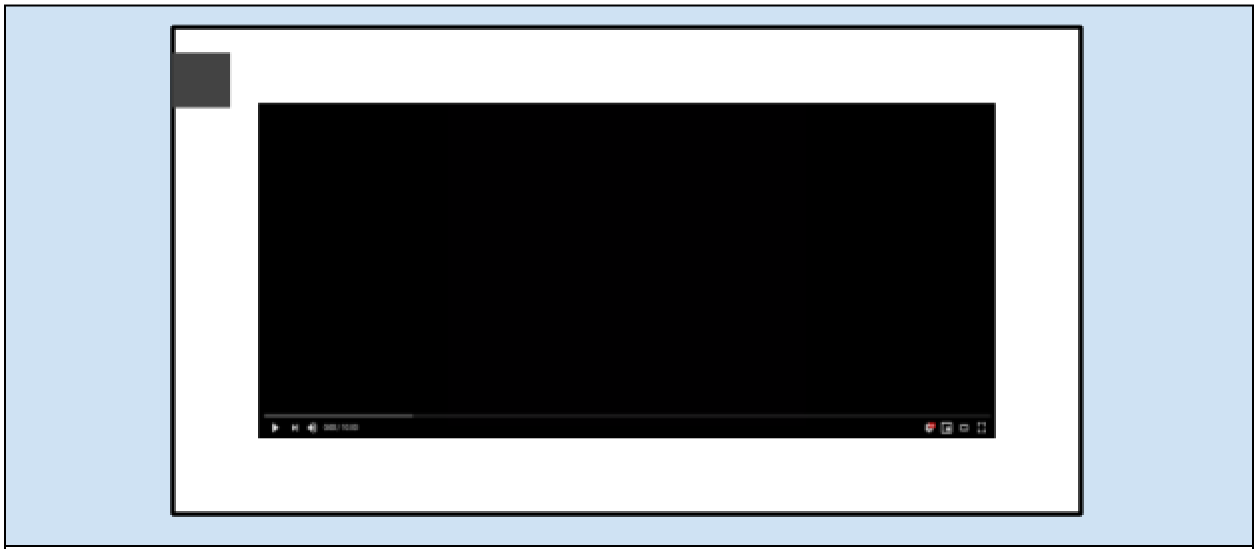


Figure 11.c -- Live feed view accessible through the side menu.

Final Implemented Frontend:

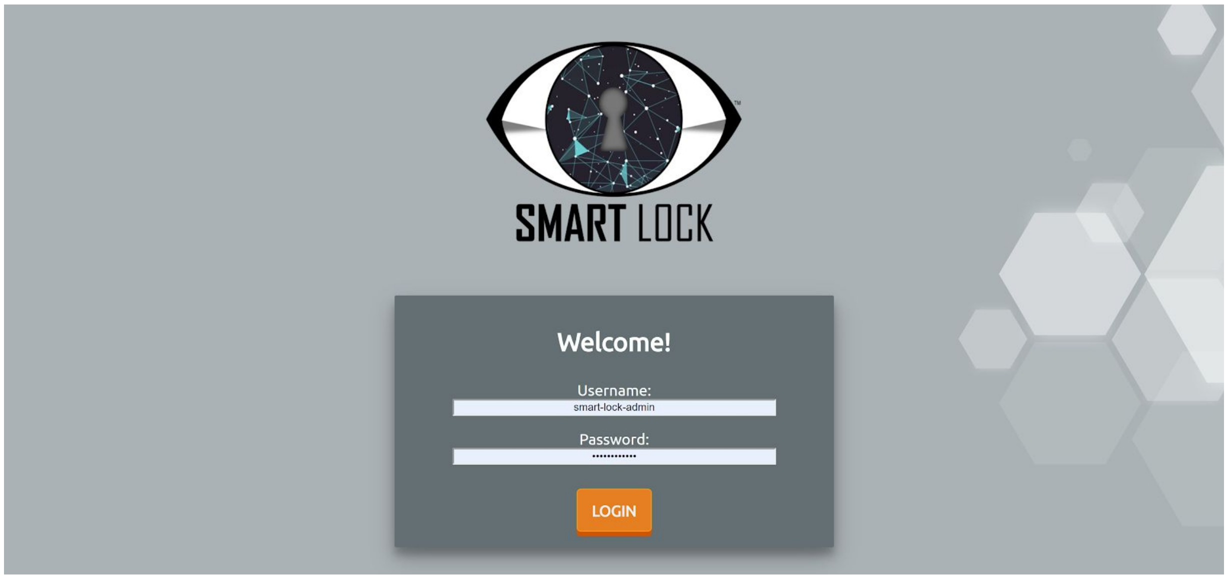
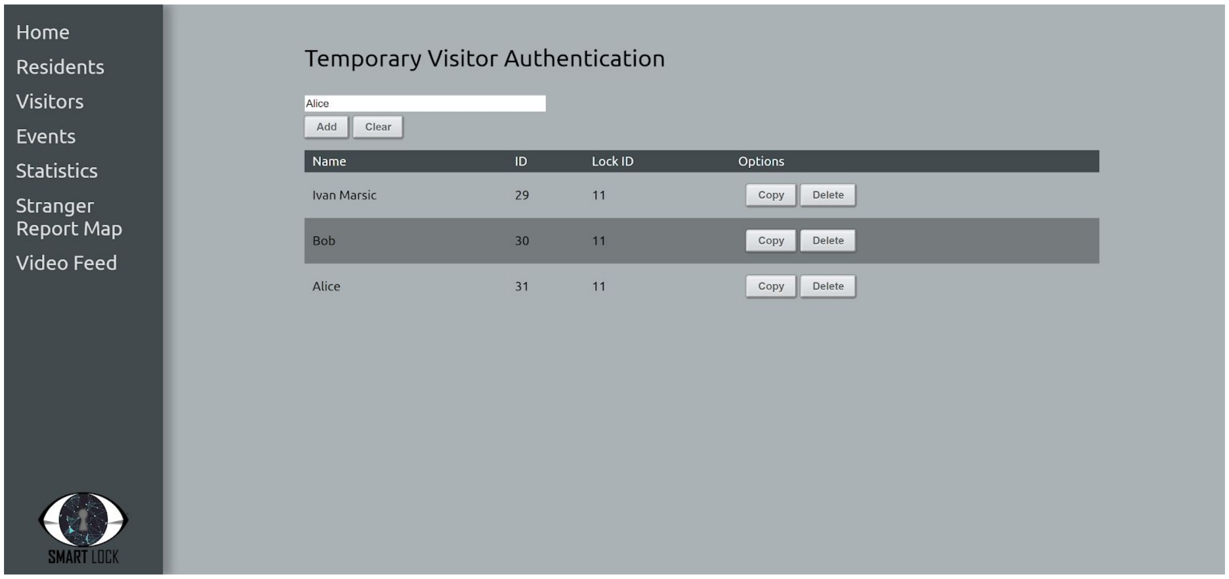


Figure 11.d -- Login Screen

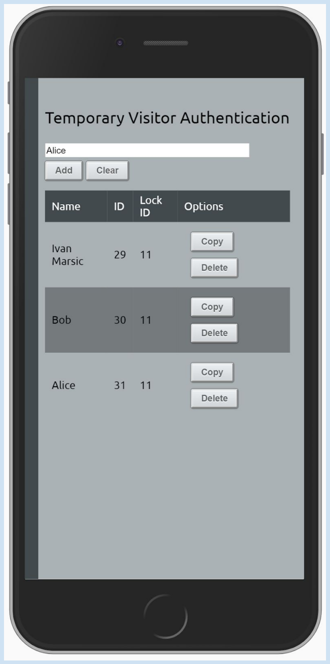


Temporary Visitor Authentication

Alice

Name	ID	Lock ID	Options
Ivan Marsic	29	11	Copy Delete
Bob	30	11	Copy Delete
Alice	31	11	Copy Delete

Figure 11.e -- Temporary Visitor Authentication Screen. Note the side menu.


Temporary Visitor Authentication

Alice

Add Clear

Name	ID	Lock ID	Options
Ivan Marsic	29	11	Copy Delete
Bob	30	11	Copy Delete
Alice	31	11	Copy Delete

Figure 11.f -- Mobile view. Note the collapsed side menu.

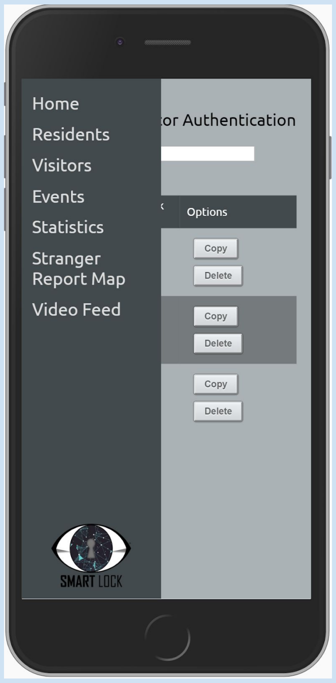


Figure 11.g -- Side menu pulled open.

12 Design of Tests

12.1 Test Design Methodology

Our application testing methodology consists of both unit tests (which test singular components to ensure basic functionality) and integration tests (which ensure components communicate properly and that large scale functions work).
Unit tests for the application are divided up by class. Detailed overviews of classes can be found in Section 2 of this report. Each class contains unit tests to evaluate the basic functionality of the component.
Integration tests for the application are divided up by use case. We feel that because use cases represent the normal flows and uses of our application, we should design our tests around the use cases to evaluate how our application will function in production.

12.2 Unit Tests (by Class)

User

- Test if user can set and modify all of its attributes (result is True or False). Each attribute will be tested separately.
- Test if the user can be successfully authenticated with certain credentials (result is True or False).

Visitor

- Test if the visitor can unlock door with temporary URL
- Test if the visitor can unlock door with their face
- Test if a removed visitor can still access the temporary URL and unlock the door with the URL
- Test if a removed visitor can still unlock the door with their face

Residence

- Test if residence ID can be set (result is True or False).
- Test if an already created Residence does not allow modification of the residence ID (result is True or False).

State

- Test if attributes can be created and modified (result is True or False). There will be one test case for each attribute.

Notifications

- Test if notification can be added (result is True or False).
- Test if notification is delivered to user's device (result is True or False). Note that the device can be either mobile or desktop, since the notification will be sent as a web notification.
- Test if the SMS notification sends when the user settings is set to receive them
- Test if the SMS notification doesn't send when the user doesn't want to receive them.

Camera

- Test if camera can detect an object with an object detection algorithm using various test images, compared to the actual presence of an object that we will manually input (result is True or False).

Video Feed

- Test if a clip from the video feed matches a video clip from the camera, within a certain level of quality (result must be within a certain quality threshold).
- Test if the video feed is live by setting a clock in front of the camera and testing the live video feed.

Stranger Reporting Network

- Test if a stranger can be added to the SRN (result is True or False).
- Test if the stranger map updates when a stranger is added.

Face Detector

- Test if a face can be detected on an image using the facial detection algorithm, and compare to the actual presence of a face in the image (result is True or False). We will need to test this on numerous images, while manually inputting data for whether a face is contained in the image.

Lock Device

- Test if a new device can be created (result is True or False).
- Test if an already created user can be assigned to a created lock device (result is True or False).

12.2 Integration Tests (by Use Case)

Use Case I: Unlock
Homeowner, Resident or authorized visitor approaches the door and the facial recognition algorithm identifies the face and opens the door. Testing can be done by sending an authorized Homeowner to the lock device camera and verifying that the system has been unlocked.

Use Case 2: Notifications
A face or QR code is detected, which sends an SMS or push notification to the Homeowner's device. Testing can be done by triggering one of these events and verifying that a notification has been sent to the Homeowner.

Use Case 3: Stranger Reporting
Unauthorized visitor or object approaches the door and facial authentication fails. The homeowner receives a notification and receives a picture of the face or object. Testing can be done by sending an unauthorized visitor or object to the testing device and verifying that the Homeowner receives a picture and notification.

UC-4: Temporary Visitor Authentication:
Homeowner creates a URL, which can be used by a temporary visitor to unlock the door. Testing for the generation of this URL can be done by querying the database for the URL. Additional testing includes sending a temporary visitor whose URL has been generated to the lock device camera and verifying that the system has been unlocked.

UC-5: Add/Remove Residents:
The homeowner can add and remove residents. We test if the resident's face has been added to the trusted database of users and if the resident can unlock the door with their face. We also test unlocking the door with the temporary URL. For removal, we test if the resident can still unlock the door with their face or with the URL.

UC-6: Data Visualization:
The Homeowner opens the web app and navigates to the tab for looking at events, where the Homeowner selects the time, date, or visitors, and can then view logs, graphs, and charts on who entered the house and when. Testing can be done via query of database and validating that the data is consistent with who unlocked the door and when.

UC-7: Live Video Feed:
A user logs in to the web app and navigates to the live video feed view. The web app starts to receive live video data and the user can view it. For testing, we want to verify that the video feed is in real time and is not showing previous frames.

13 History of Work, Current Status, and Future Work

13.1 History of Work

Implementation of separate components/features of the system.

- Temporary User Authentication
- SRN
- Facial Recognition
- QR Code Detection
- Data Visualization
- Event Feed
- SMS Notifications
- Live Video Feed

14. References

1. *Histogram of the oriented gradient for face recognition - TUP Journals & Magazine*. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6077989;https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients#Object_recognition. [Accessed: 2019].
2. *Google*. [Online]. Available: https://developers.google.com/maps/documentation/javascript/. [Accessed: 2019].
3. "ACID (computer science)," *Wikipedia*, 11-Apr-2019. [Online]. Available: https://en.wikipedia.org/wiki/ACID_(computer_science). [Accessed: 2019].
4. "App Design & Development Company | NYC," *App Partner*. [Online]. Available: https://www.apppartner.com/the-psychology-of-swiping-in-apps/. [Accessed: 2019].
5. "Cascading Style Sheets," *Wikipedia*, 07-Apr-2019. [Online]. Available: https://en.wikipedia.org/wiki/Cascading_Style_Sheets. [Accessed: 2019].

6. T. Christie, “Django REST Framework,” *Home - Django REST framework*. [Online]. Available: https://www.django-rest-framework.org/. [Accessed: 2019].
7. “Client–server model,” *Wikipedia*, 09-Apr-2019. [Online]. Available: https://en.wikipedia.org/wiki/Client–server_model. [Accessed: 2019].
8. “Communication APIs for SMS, Voice, Video and Authentication,” *Twilio*. [Online]. Available: http://twilio.com/. [Accessed: 2019].
9. “Component-based software engineering,” *Wikipedia*, 09-Mar-2019. [Online]. Available: https://en.wikipedia.org/wiki/Component-based_software_engineering. [Accessed: 2019].
10. “Deep learning,” *Wikipedia*, 10-Apr-2019. [Online]. Available: https://en.wikipedia.org/wiki/Deep_learning. [Accessed: 2019].
11. “Django,” *Django*. [Online]. Available: https://www.djangoproject.com/. [Accessed: 2019].
12. “Event-driven architecture,” *Wikipedia*, 26-Mar-2019. [Online]. Available: https://en.wikipedia.org/wiki/Event-driven_architecture. [Accessed: 2019].
13. “Haversine formula,” *Wikipedia*, 07-Apr-2019. [Online]. Available: https://en.wikipedia.org/wiki/Haversine_formula. [Accessed: 2019].
14. “HTML,” *Wikipedia*, 13-Mar-2019. [Online]. Available: https://en.wikipedia.org/wiki/HTML. [Accessed: 2019].
15. “JavaScript,” *Wikipedia*, 13-Apr-2019. [Online]. Available: https://en.wikipedia.org/wiki/JavaScript. [Accessed: 2019].
16. “MPEG-4 AVC,” *Wikipedia*, 31-Mar-2006. [Online]. Available: https://en.wikipedia.org/wiki/H.264/MPEG-4_AVC. [Accessed: 2019].

17. "OpenCV library," *OpenCV library*. [Online]. Available: https://opencv.org/. [Accessed: 2019].
18. "Peer-to-peer," *Wikipedia*, 25-Mar-2019. [Online]. Available: https://en.wikipedia.org/wiki/Peer-to-peer. [Accessed: 2019].
19. "Representational state transfer," *Wikipedia*, 10-Apr-2019. [Online]. Available: https://en.wikipedia.org/wiki/Representational_state_transfer. [Accessed: 2019].
20. "Representational state transfer," *Wikipedia*, 10-Apr-2019. [Online]. Available: https://en.wikipedia.org/wiki/Representational_state_transfer. [Accessed: 2019].
21. M. Sajjad, M. Nasir, F. U. M. Ullah, K. Muhammad, A. Kumar, and S. W. Baik, "Raspberry Pi Assisted Facial Expression Recognition Framework for Smart Security in Law-Enforcement Services,"*Information Sciences*, Jul. 2018.
22. S. O. N. Silva and S. Luciano, "A LINUX MICROKERNEL BASED ARCHITECTURE FOR OPENCV IN THE RASPBERRY PI DEVICE," *International Journal of Scientific Knowledge*, vol. 5, no. 2, Jun. 2014.
23. "The World's Most Advanced Open Source Relational Database," *PostgreSQL*. [Online]. Available: https://www.postgresql.org/. [Accessed: 2019].
24. "WebSocket," *Wikipedia*, 15-Mar-2019. [Online]. Available: https://en.wikipedia.org/wiki/WebSocket. [Accessed: 2019].
25. "Welcome," *Welcome | Flask (A Python Microframework)*. [Online]. Available: http://flask.pocoo.org/. [Accessed: 2019].
26. "Welcome to Python.org," *Python.org*. [Online]. Available: https://www.python.org/. [Accessed: 2019].

Large Deleted content

Use Case UC-5: Add Resident for Facial Recognition	
Related Requirements	REQ-1, REQ-3
Initiating Actor	Homeowner
Actor’s Goal	To grant a resident access (using facial recognition)
Participating Actors	Lock Device; Database
Preconditions	A set of (mugshot-like) images of the resident are available.
Postconditions	Lock device / system is equipped with latest images of the resident and the lock device is able to authenticate resident.
Flow of Events 1. Homeowner accessing authentication tab in web app UI; selects facial recognition authentication selection. Clicks on Create New Facial Recognition Authentication. 2. Homeowner adjusts duration (or amount of uses) parameters. 3. Homeowner uploads pictures (headshots) of the person. (Minimum of 8 or 9) 4. Database stores this information. 5. Resident is authenticated upon facial recognition.	

Note: We combined Use Case 5 so that it covers both Add and Delete residents (page 21).

Project Evolution

- We have decided to remove all use cases involving the packaging and courier system because we cannot expect all mail couriers to know how to use our lock device or attempt to use it. As a result, it can be a potential safety hazard if a stranger picks up a package left on the front door and gains access to the house by using the barcode on the package.

- We removed the settings and options menu because we felt that we should focus more on novel and interesting features that improve the product rather than creating a settings menu which is pretty standard.

- We changed the diagram on 4.3.d so that it is in UML format (page 25).
- We removed start time and end time for ResidentAdder because all residents should have 24/7 access to the lock, unless they are removed by the homeowner (page 48).
- We added frontend screenshots in order to see the differences and similarities between the conceptual design and the final implemented product. (page 86-87).