SMART LOCK

REPORT #2

Software Engineering - S'19 - Group #10

https://software-engineering-s19-group10.github.io/website/

MEMBERS:

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

DATE SUBMITTED: 03/17/2019

0. Individual Contribution Breakdown

(All members contributed equally in this report)

	Team Members and NetID									
	Janda, Jasjit	Kaler, Amandip	Khattar, Mohit	Lin, Eric	Lu, Jeffrey	Moseley, Ted	Nadeem, Mohammad	Nguyen, Daniel	Sengupta, Andrew	Truong, Michael
	jsj87	*ak1415*	*mk1483*	*ekl40*	*jl2088*	*trm124*	*mn535*	dnn21	*ans165*	*mt842*
1:										
Interaction Diagram					++		++			
Descriptions of Diagrams					++				++	
Alternative Designs					++	+	+			
2:										
Class Diagram		+		+	+					+++
Signatures	+	+							++	
Traceability Matrix		+		+	+			+		+
3:										
Architectural Styles						+++		+++		
Identifying Subsystems	++	++								
Mapping to Hardware				++	+++					
Persistent Data Storage			++++						+	
Network Protocol	++		+++	+	++					
Global Control Flow				+++	++		+++			

Hardware Requirement	+	+		+	+					+
4:										
Algorithms					+++		+++		+	
Data Structure						++		+	+	
5:										
User Interface						+++		+++	+	
6:										
Test Design			++		+		+	+	++	+
Project Management & References	10%	10%	10%	10%	10%	10%	10%	10%	10%	10%
Points	10	10	10	10	10	10	10	10	10	10

Table of Contents:

1. Interaction Diagrams and Alternative Design

1.1 Interaction diagrams

1.1.a Interaction diagram for UC-1

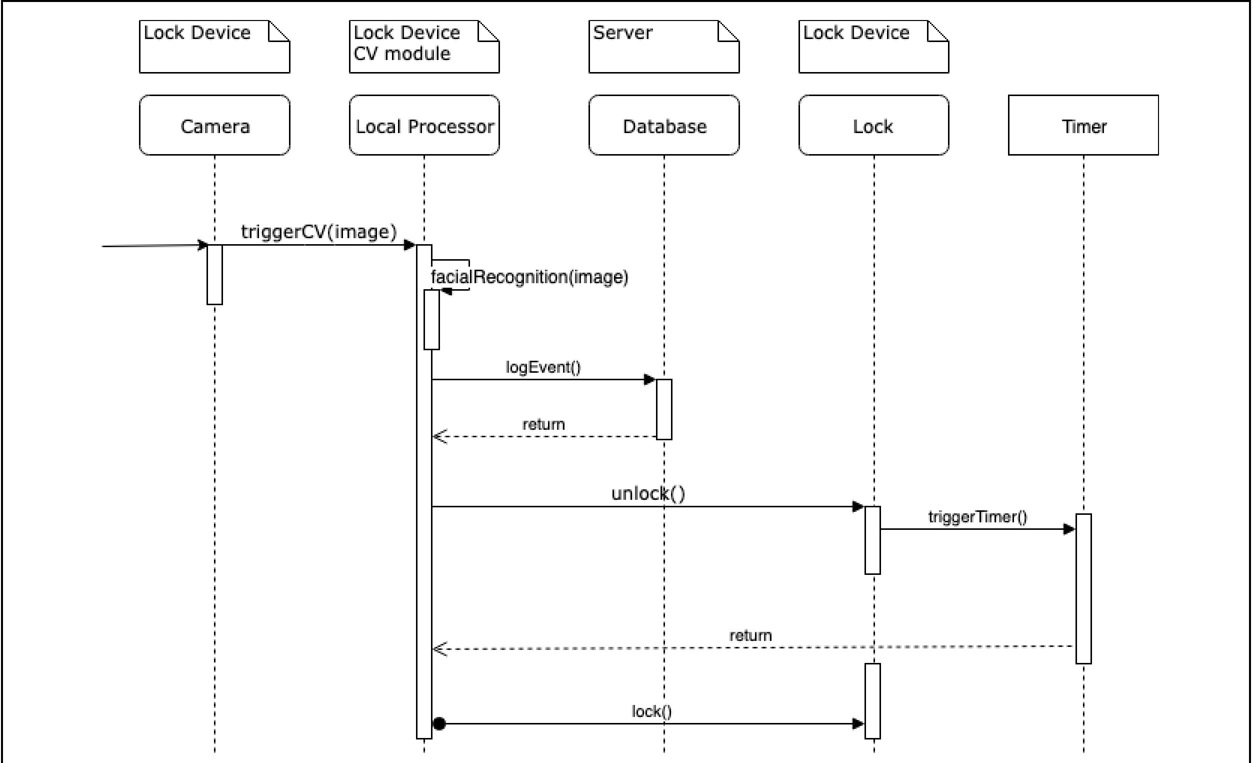

Figure 1.1.a - Interaction Diagram for Facial Recognition and Unlocking:

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.

1.1.b Interaction diagram for UC- 4

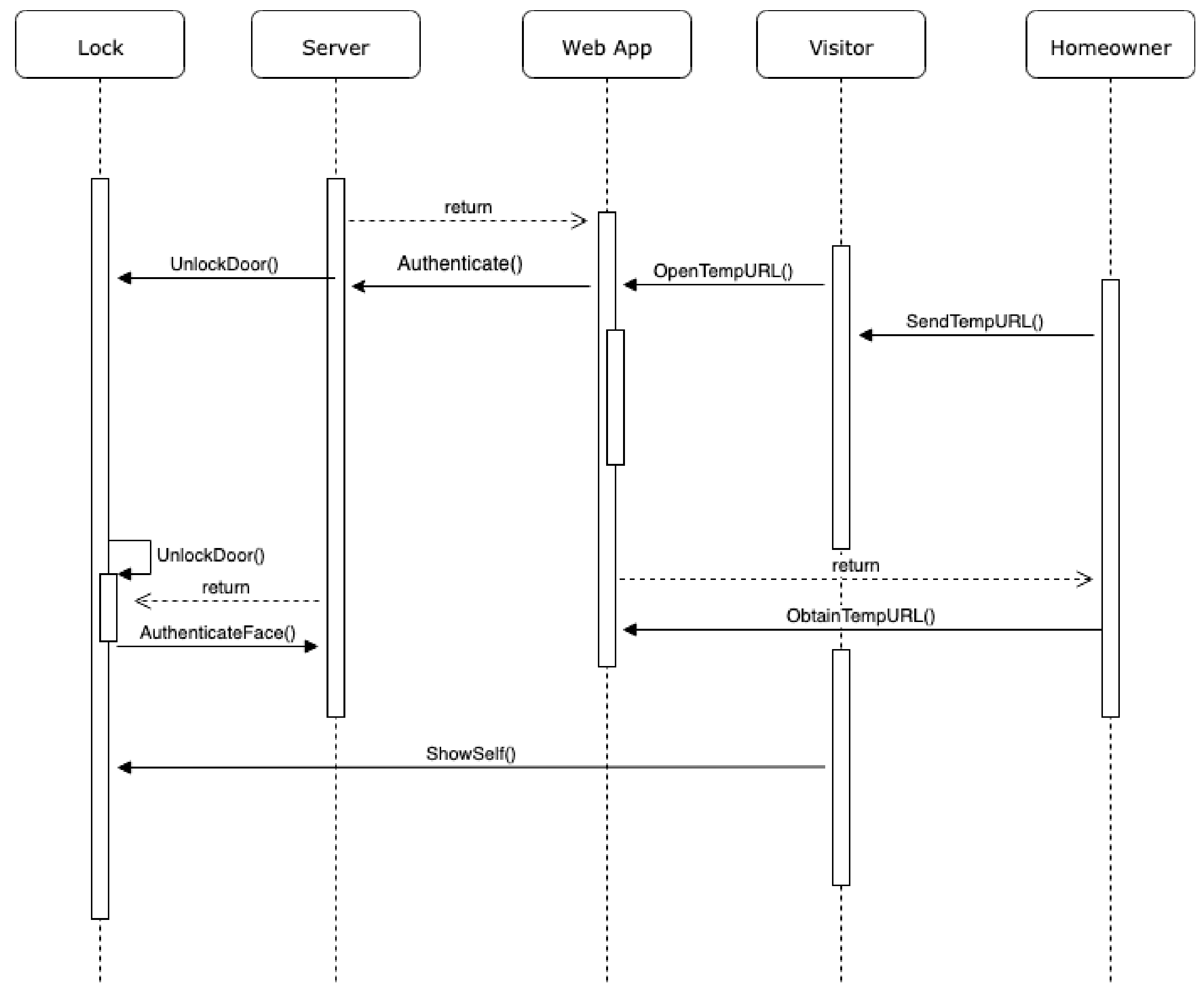

Figure 1.1.b - Interaction Diagram for a Visitor Unlocking the Door:

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.

1.1.c Interaction diagram for UC- 6

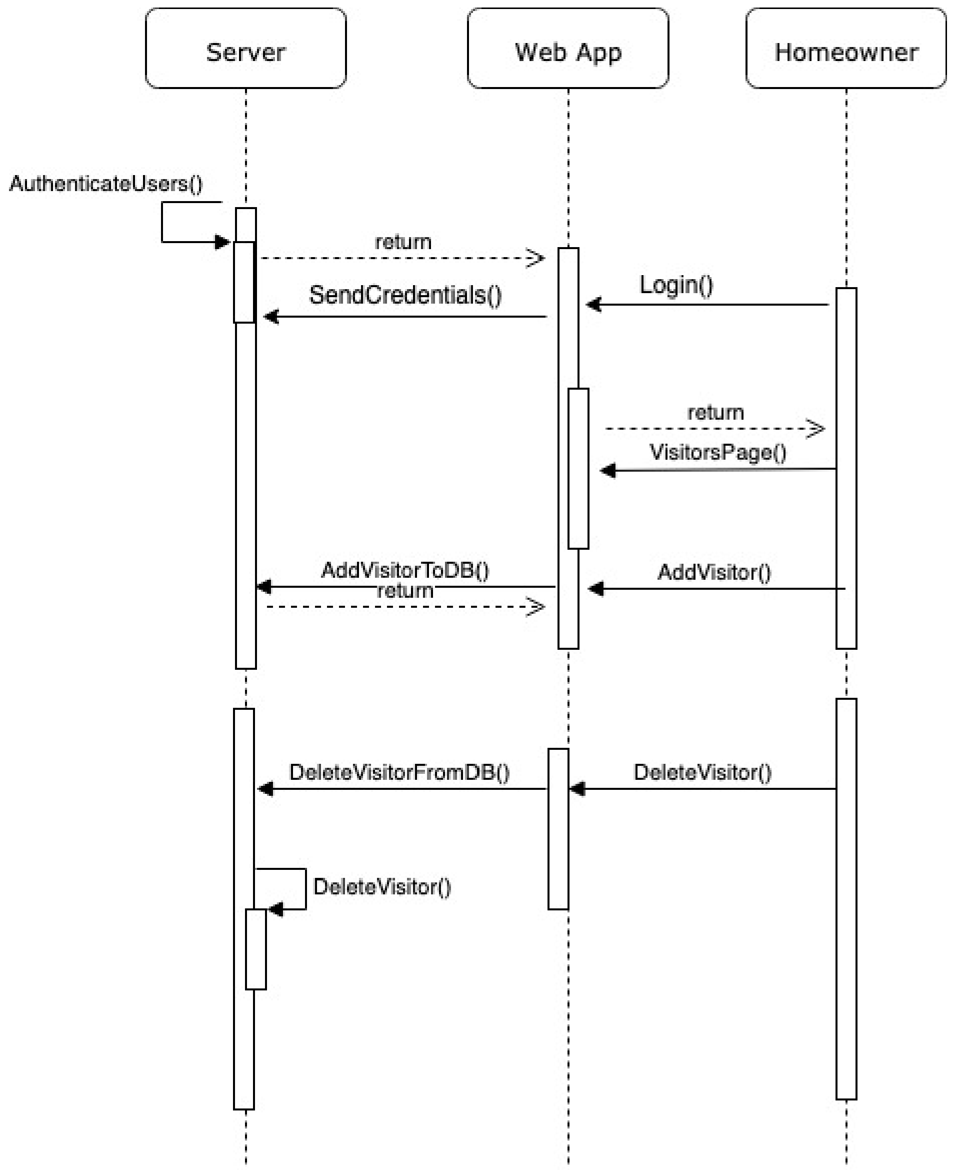

Figure 1.1.c - Interaction Diagram for Adding and Removing Visitors:

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.

1.1.d Interaction diagram for UC- 7

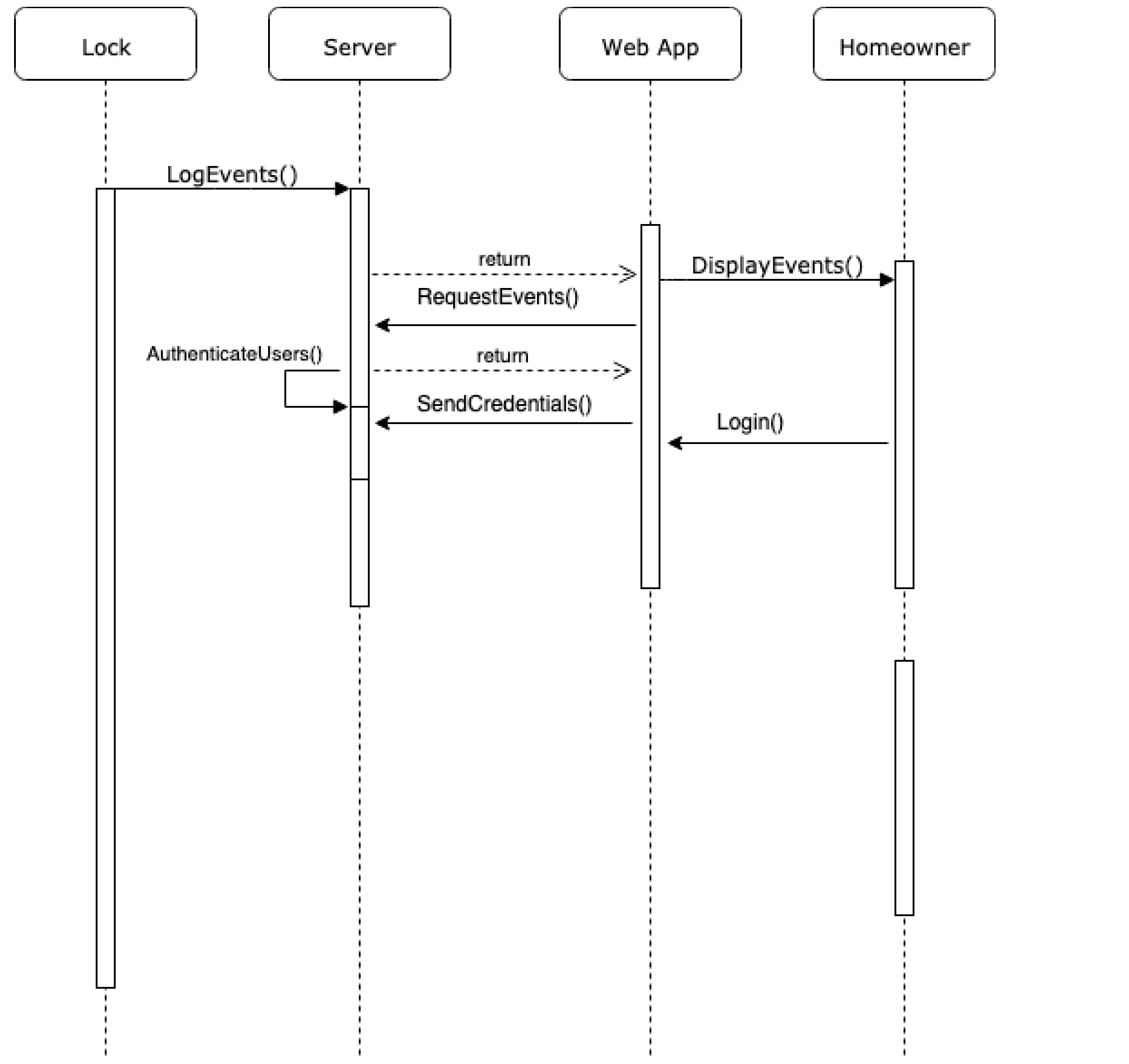

Figure 1.1.d - Interaction Diagram for Event Reporting and Visualization:

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

1.1e Interaction diagram for UC- 8

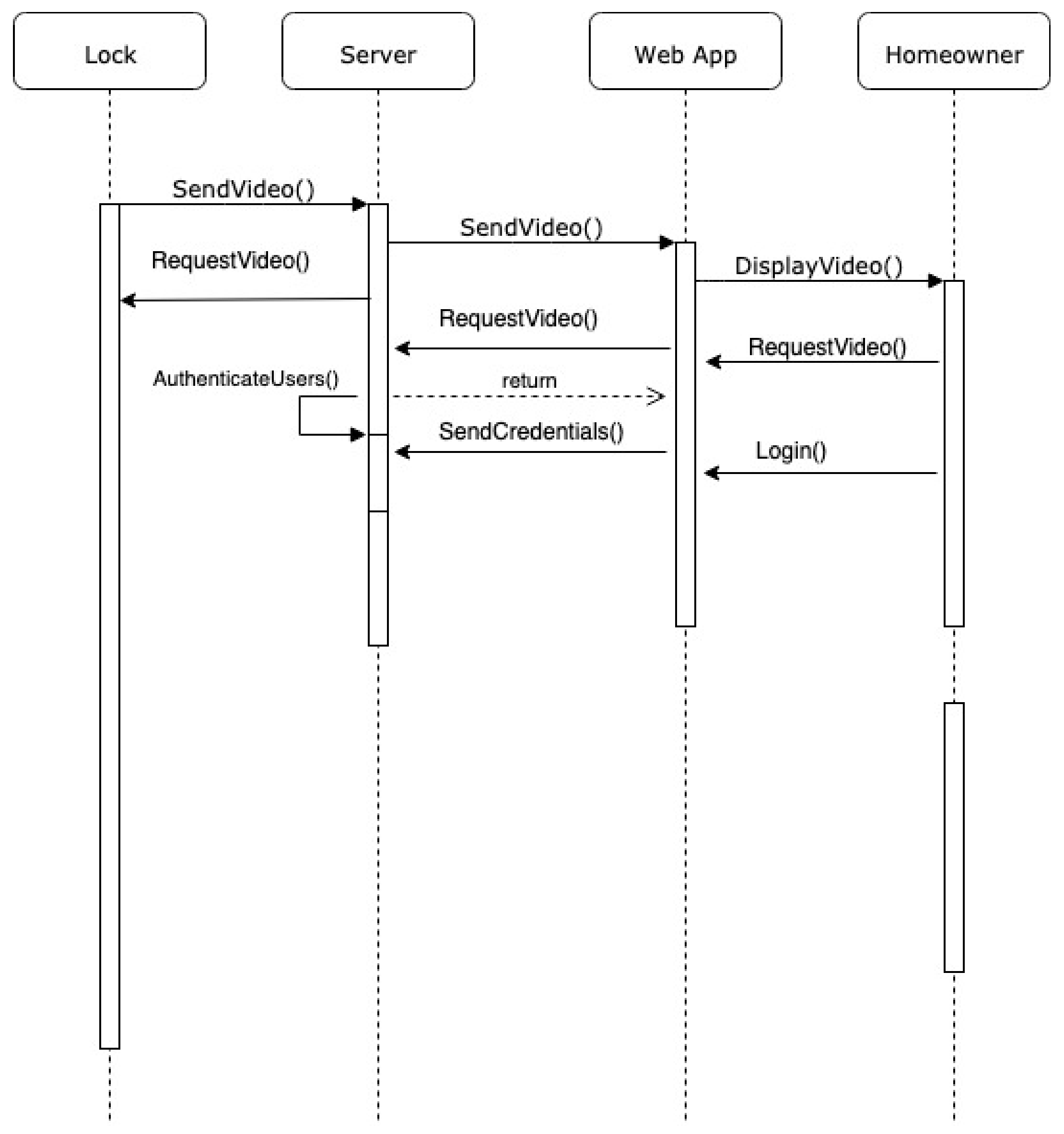

Figure 1.1.e - Interaction Diagram for Streaming Video:

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.

1.2 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.

2. Class Diagram and Interface Specification

2.1 Class Diagram

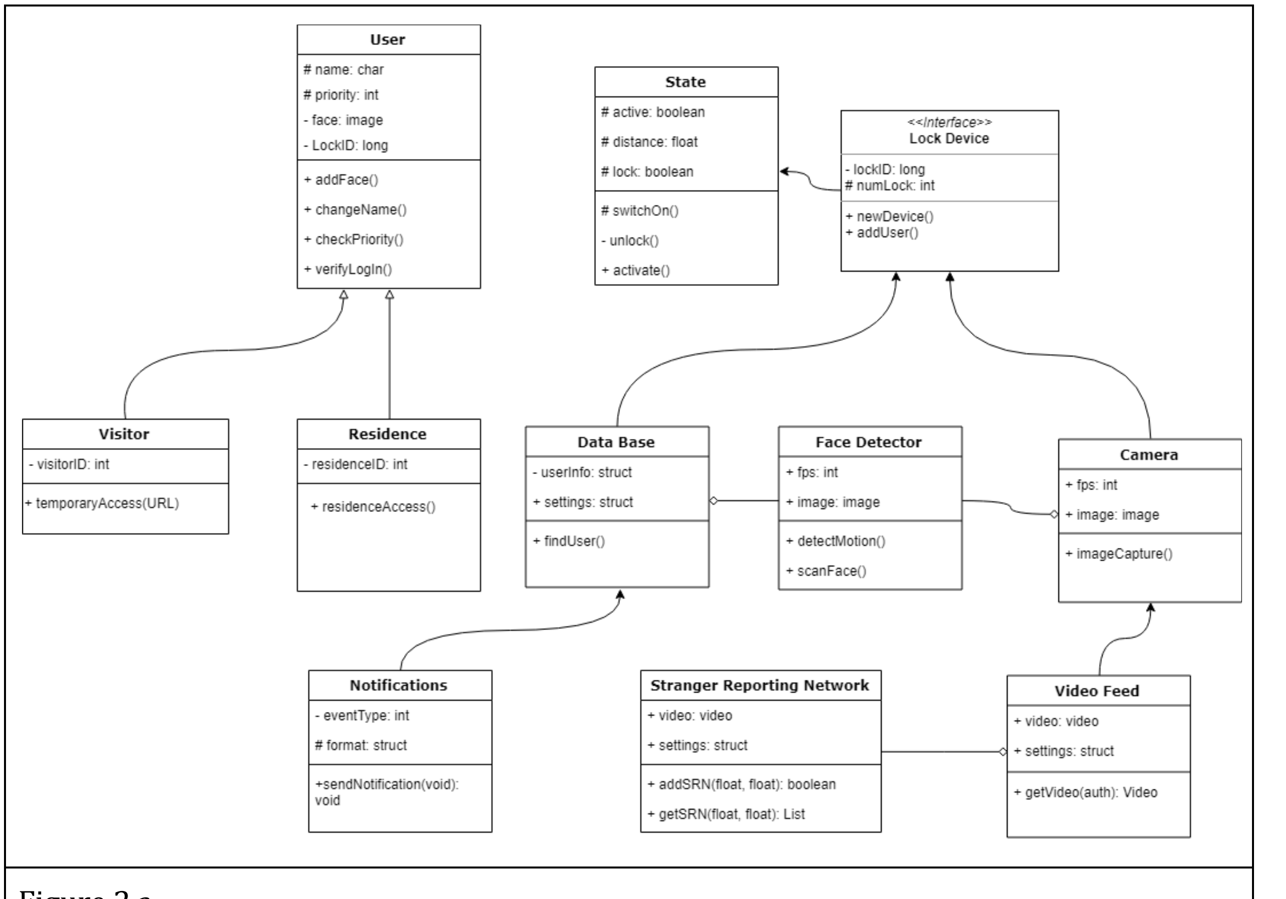

Figure 2.a

2.2 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

2.3 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: Package Delivery - To allow couriers to unlock the door so they may deposit packages.
UC-6: Add/Remove Visitors - To add residents as visitors with permission to enter the house with facial recognition or remove these visitors.
UC-7: Data Visualization - To allow homeowners to view data pertaining to who enters the house and when then enter.
UC-8: Live Video Feed - To obtain a live feed of the home entrance through the camera.
UC-9: Options/Settings - To allow homeowner to change his notification and lock settings.

	UC - 1	UC - 2	UC - 3	UC - 4	UC - 5	UC - 6	UC - 7	UC-8	UC-9
User	X							X	X
Log	X						X		
Notifications		X				X			
Video	X					X		X	
Lock Device	X			X	X			X	
Stranger Reporting Network			X				X		
Barcode Reader					X				
FaceDetector	X								

3. System Architecture and System Design

3.1 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.

3.2 Identifying Subsystems

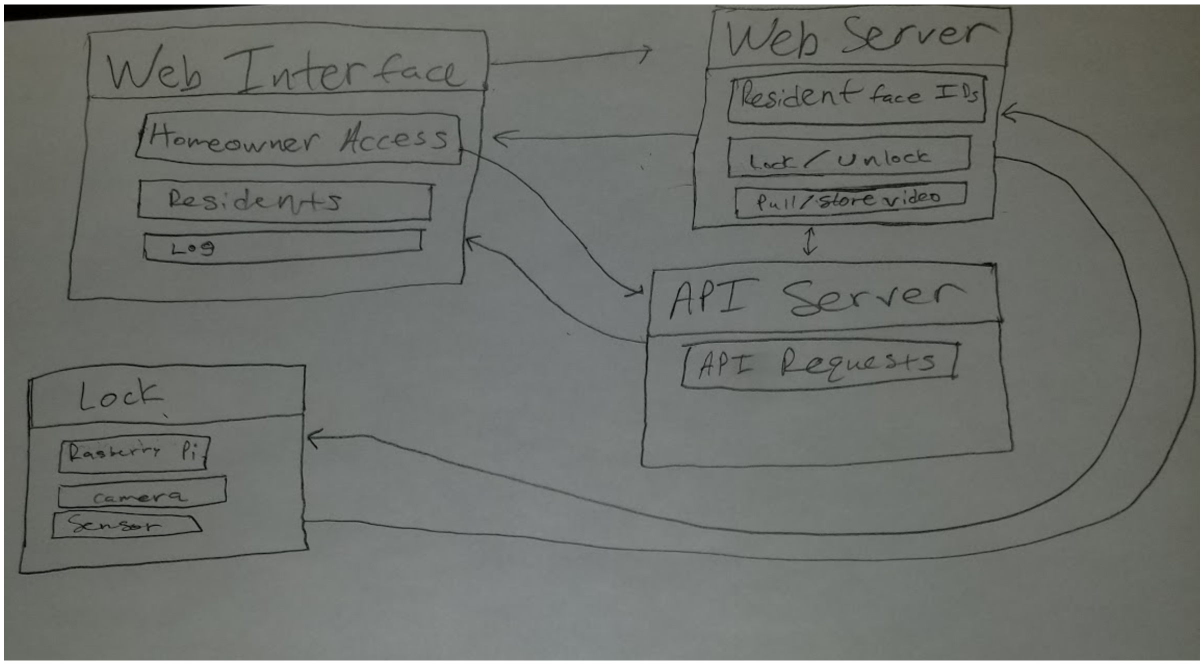

Figure 3.2a

The web interface is what the homeowner will be interacting with in order to monitor logs of who's entering the house and when, add and delete designated residents, and create "keys" for guests. All API requests to the backend will be sent to an API server, separate from the web server. The web server will request video from the camera which will be triggered by a sensor. The camera will also be used to detect resident face IDs, which will unlock the door if it matches. These will be ran by a local processor, being the "Raspberry Pi."

3.3 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.

3.4 Persistent Data Storage

3.4.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

3.4.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.

3.4.c Database Schema

3.4.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.

3.4.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:

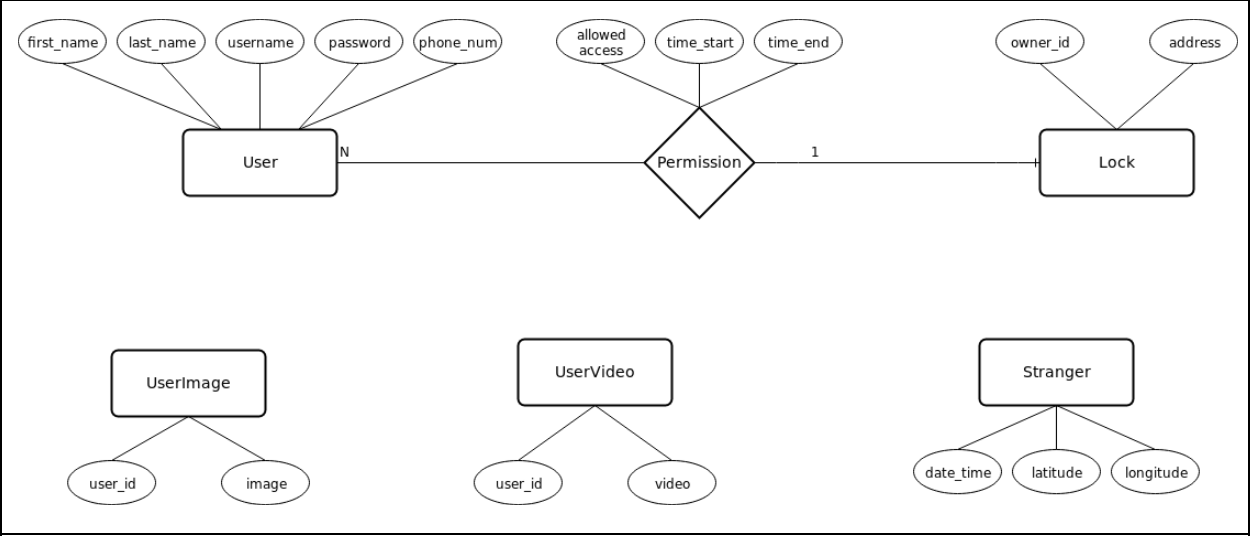

Figure 3.4.c.ii

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.

3.5 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.

3.5.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.

3.5.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.

3.5.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.

3.6 Global Control Flow

3.6.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.

3.6.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.

3.6.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.

3.7 Hardware Requirements

3.7.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.
- Sensor: Ultrasonic Distance Sensor (HC-SR04) for Raspberry Pi.
- Electric Lock: the physical lock is implemented by a DC motor, which is controlled by the Raspberry Pi processor.

3.7.b Server

- Will run on a laptop.
- Storage : > 5 GiB. This will give room for clip storage. 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.

4. Algorithms and Data Structures

4.1 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 .

4.2 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 of arrays and JSON objects (key-value pairs) we will be sure to take advantage of them. A few examples include the view menu and configuration options menu for each component.

Options in the view menu can be represented using a simple array of hash tables. As each component is loaded, the corresponding view menu option can be added. The array is an applicable data structure since it is an order list of items, much like a menu. The component will register itself with a JSON object containing it's name, an image to load as an icon, etc.

Alternatively, settings are a perfect use of a JSON object. Settings can viewed as a collection of name-value pairs. Within the frontend logic we will build a minimal framework which allows components to define settings values. This framework will include support for a variety of types (booleans are toggles, text boxes are strings, integers are sliders, etc.). The settings allowed by a component can be defined within a JSON object. Similar the current setting values can be shared with the Central Database and Smart Lock devices as JSON objects.

5. 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.[7] This change will affect navigation throughout the entire app.

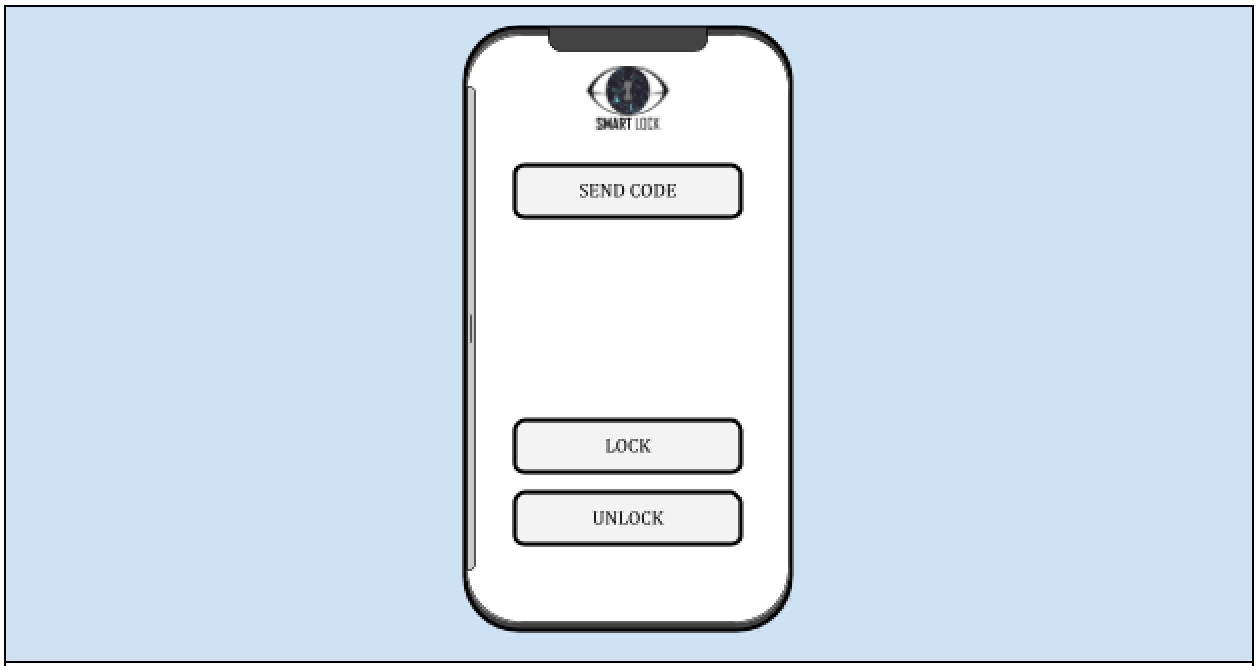

Figure 5.a -- The differently colored bar on the left side of the screen indicates that a menu is available for swiping.

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.

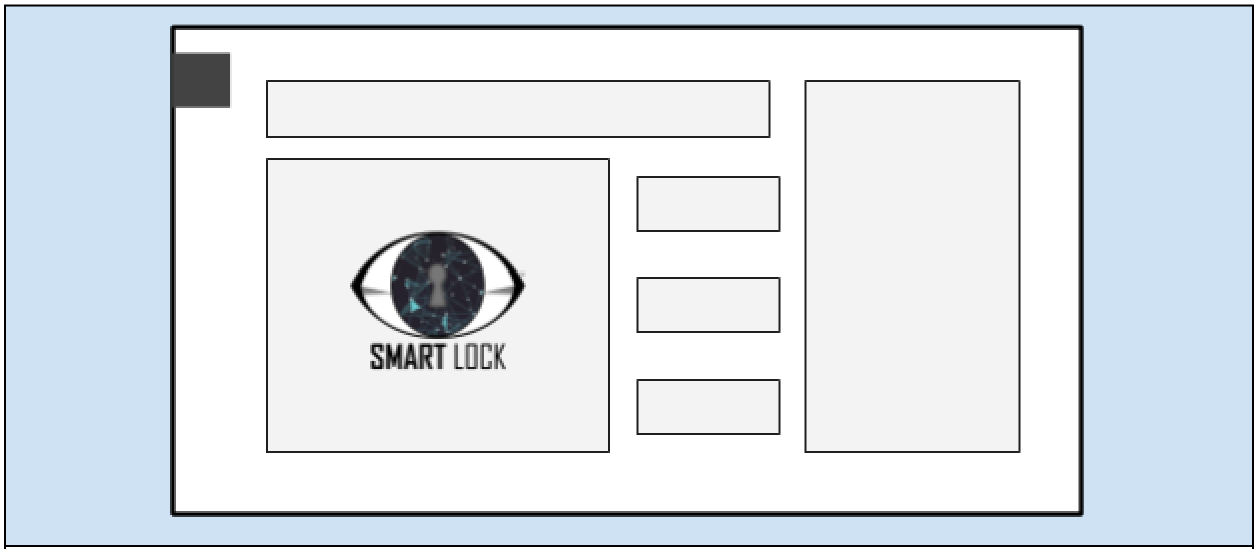

Figure 5.b -- Main menu without live feed

Figure 5.c -- Live feed view accessible through the side menu.

6. Design of Tests

6.1 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.

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

6.2 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 is detected or a package is delivered, 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: Package Delivery:
In order to test this, we can generate a fake tracking number and use a testing API that we built (instead of a postal company's API) in order to verify if the door unlock with the package barcode.

The user first enters the tracking number. The testing API will return the barcode based on the tracking number. Then, we print the barcode and put it in front of the camera and check if the door unlocks.

UC-6: Add/Remove Visitors:
The homeowner can add and remove visitors. We test if the visitor's face has been added to the trusted database of users and if the visitor can unlock the door with their face. We also test unlocking the door with the temporary URL. For removal, we test if the visitor can still unlock the door with their face or with the URL.

UC-7: 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-8: 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.

UC-9: Options/Settings:
A user logs into the web app and enables/disables settings as desired. Testing can be done via database query.

7. Project Management and Plan of Work

Group 1: (Ted Moseley, Michael Truong, Daniel Nguyen)

- Authentication subsystem for temporary access to the home for Visitors
    - Generate temporary URLs that Homeowner send to Visitors
    - Produce a web Front-end for generating QR codes that Visitors scan
    - Produce a web Front-end for Homeowner to get temporary URLs for Visitors
- Authentication Configuration (for temporary access URLs/QR codes)
    - Allows the Homeowner to configure a time-frame, number of uses, and/or expiration date.
- Smart Package Tracker and Barcode Authentication (*stretch goal*)
    - Allow Homeowner to enter data about incoming packages
    - Determine when a Courier may enter with a package based on tracking data
    - Produce a method for Couriers to scan into the home

Plan of work:
Authentication for mail carriers delivering packages into the home will be moved down in priority due to security concerns and package authentication problems.

Work on regular Visitor authentication system nearly complete.

Group 2A: (Mohit Khattar)

- Database Structuring (Shared Infrastructure) and communicating structure to other subteams.
- Assist group 2B with their responsibilities.

Plan of work:
A general structure for the database has been implemented along with authentication so only registered users can access data from the database. Future work will involve granting access to specific data for specific users and starting to help with Group 2B's goals.

Group 2B: (Eric Lin, Mohammad Nadeem, Andrew Sengupta)

- Data Visualization (be creative; using data on who and when [events log]; e.g. how many people visited this month? who visited this month?)
- Merge the Group 2B database and server with that of Group 2A.
- Create documentation for the REST API.

- Live feed communication between the server, embedded computer, and web client

Plan of work:
The database setup and notifications are completed. We currently still have to implement streaming of video which should be finished relatively quickly.

Group 3: (Jasjit Janda, Jeffrey Lu, Amandip Kaler)

- Motion Detection
- Facial Detection
- Facial Recognition
- Masked person Detection (must implement own interface/unlock signal to lock)
- Create backend and frontend for Auth Permissions (add active members)
- Add detail Authentication Configuration (for people recognized by face) (e.g. you may only want certain people to gain access for a certain period of time, certain amount of days)
- Lock Device

Plan of work:
We made final decision for the Lock Device first Demo: it will be a Raspberry Pi controlled lock. Facial Recognition will happen on the Raspberry Pi.

8. References

1. Component-Based Model: https://en.wikipedia.org/wiki/Component-based_software_engineering
2. Client-Server Model: https://en.wikipedia.org/wiki/Client%E2%80%93server_model
3. Central Repository:
4. Representational State Transfer (REST): https://en.wikipedia.org/wiki/Representational_state_transfer
5. Event-driven: https://en.wikipedia.org/wiki/Event-driven_architecture
6. Peer-to-peer: https://en.wikipedia.org/wiki/Peer-to-peer
7. Swiping in Apps: https://www.apppartner.com/the-psychology-of-swiping-in-apps/
8. ACID database principles: https://en.wikipedia.org/wiki/ACID_(computer_science)
9. OpenCV on Raspberry Pi https://www.researchgate.net/publication/302016369_A_LINUX_MICROKERNEL_BASED_ARCHITECTURE_FOR_OPENCV_IN_THE_RASPBERRY_PI_DEVICE;https://www.researchgate.net/publication/326454084_Raspberry_Pi_Assisted_Facial_Expression_Recognition_Framework_for_Smart_Security_in_Law-Enforcement_Services
10. HoG Face Detection: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=6077989;https://en.wikipedia.org/wiki/Histogram_of_oriented_gradients#Object_recognition
11. Deep learning: https://en.wikipedia.org/wiki/Deep_learning
12. Haversine Formula:https://en.wikipedia.org/wiki/Haversine_formula
13. H.264 Encoding: https://en.wikipedia.org/wiki/H.264/MPEG-4_AVC