جامعة الملك فهد للبترول والمعادن
King Fahd University of Petroleum & Minerals

College of Computer Sciences and Engineering
Computer Engineering Department

COE 485: Senior Design Project
Term 171

Final Report

Smart Campus: Face Recognition

by
Hussam Altaleb

for
Dr. Muhamad Elrabaa

25 December 2017

CONTENTS

I. PROBLEM STATEMENT

Students go through class attendance check every class. With large classes, a considerable portion of students' time is wasted on taking attendance. For example, a freshman engineering student attending 5 lectures a day, spends 30 minutes listening to students' names being checked. Furthermore, attendance checking in the traditional way where a lecturer goes through a list of names is not fully secure, since some students tend to pretend to be someone else causing unauthentic class attendance records. Moreover, traditional access control for laboratories and offices, where a user enters a 4-digit number on a keypad, placed on a door, is not fully secure. Security and time are valuable resources for a campus, and their maintenance through modern technology is a step into smart campus.

II. POSSIBLE APPROACHES

Time saving and authentication in a smart campus can be improved through many means. For example, smart card readers can be used to provide authentication. However, the use of smart card can be inconvenient and non-secure. Using modern biometric identification methods, campus activities can be more productive, secure and smart. Examples of biometric technologies are: fingerprint, Iris recognition and facial recognition. We chose facial recognition because it is not as invasive as fingerprint approach and more hygienic. Moreover, an Iris recognition approach is costly, requiring special technologies, while facial recognition can be achieved using simple cameras.

III. REQUIREMENTS

- Detect present faces, at all times when turned on, and send them to the system.
- Recognize a person from set of people.
- Ability to add new functions and services.
- 0% false positive person identification to enhance security.
- Maximum of 10% false negative person identification for convenience.
- User friendly interface.

IV. SYSTEM SPECIFICATIONS

To achieve the mentioned requirements, the following specifications and features have to be implemented:

- Software that handles streams from different cameras that will be used to capture face images. OpenCV –more on it later– provides an effective and convenient way to handle video streams.
- A camera for each system node, nodes that are used for service dispensing and are expected to handle one user at a time do not have high expectation for camera resolution or high image quality. This is because users are expected to be close to the cameras, thus, an accurate face image can be obtained even with low quality camera. However, a camera used to register users trainings when they are added by the system admin, is expected to have higher quality, but resolution is not of the most importance. This is because images that are used for training have to be, as much as possible, representative of actual face features. Cameras used for identification of many users at a time have to have higher resolution, that is because users are expected to be relatively further away from the camera, thus, higher resolution is needed to register more details, consequently, generating more accurate identifications. In order to capture an adequate face image, that can be detected and recognized, for a person 4 meters away from the camera, assuming the area of the picture the camera takes of a plane that is 4 meters away, is 4*4 squared meters, the area of the face is 10*10 squared centimeters, and the minimum required pixels for the face image is 80*80 pixels –this is a safe assumption for current face detectors and recognizers– we conclude that a 10.24 megapixels camera is adequate. That is, roughly, a camera with a resolution of 10 megapixels, is required for detecting and recognizing people that are at most 4 meters away.
- To accommodate for multiple simultaneous identifications that could be necessary at rush hours, the system should provide enough processing power to typically identify tens to hundreds of identifications per minute, i.e. few identifications per second. The OpenFace recognizer –mentioned later– can generate face features in about 200 ms on a single core on an Intel Core i7 CPU. Thus, around 250 face identifications per minute or more if the process is parallelized, can be achieved using an Intel Core i7 or an equivalent CPU. If less identification are required, a lower CPU may satisfy requirements.

V. USE CASES AND ACTIVITY

Figure 1 shows the system use cases, actors, and interactions between them.

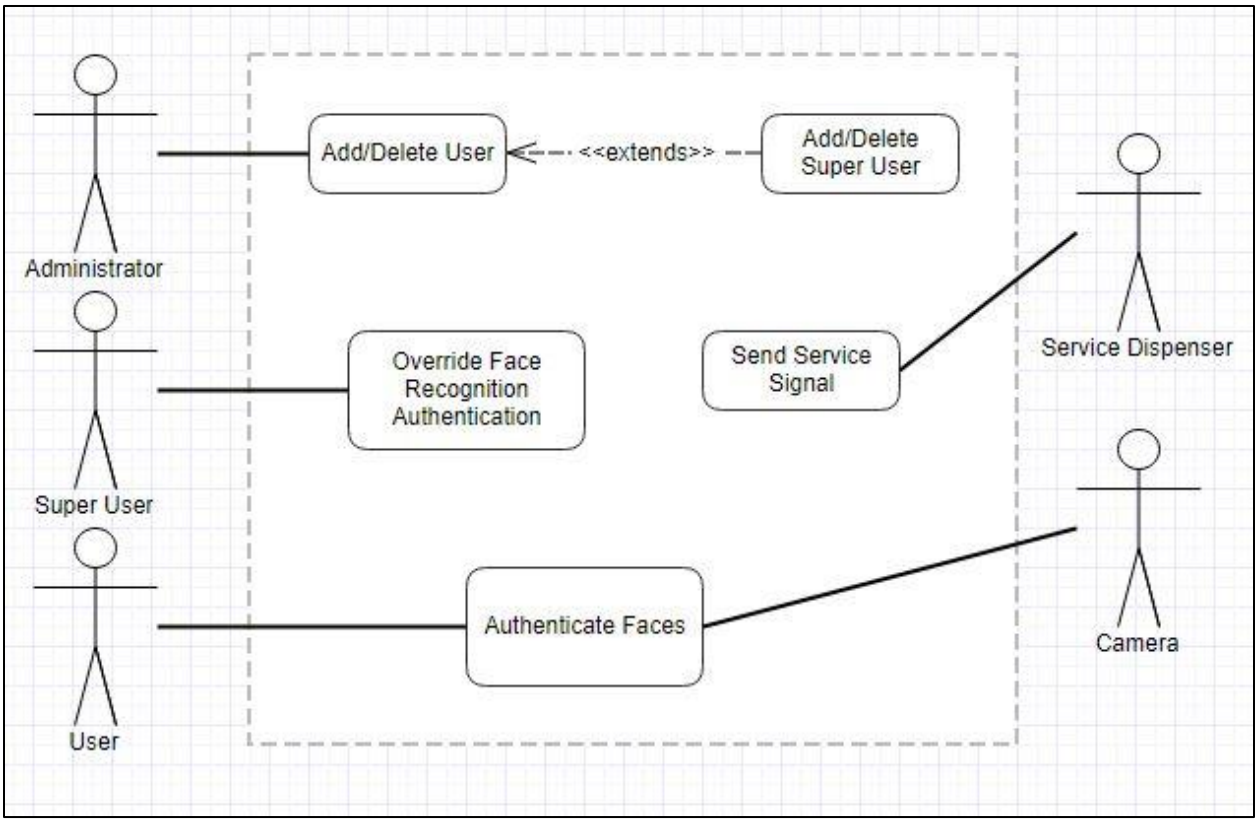

Figure 1 Use Cases and Actors of the System.

A. Override Authentication.

Primary Actor(s): Super-user

Description: A super-user login into the provided program and check in or out a user or a visitor. The door will open as he check someone in or out and the database record will change. Figure 2 Show an activity diagram for Override Authentication.

Pre-conditions: A super-user notice an error in the auto-authentication system or simply the system could not recognize someone.

Post-conditions: The records change in the database.

Assumptions: None.

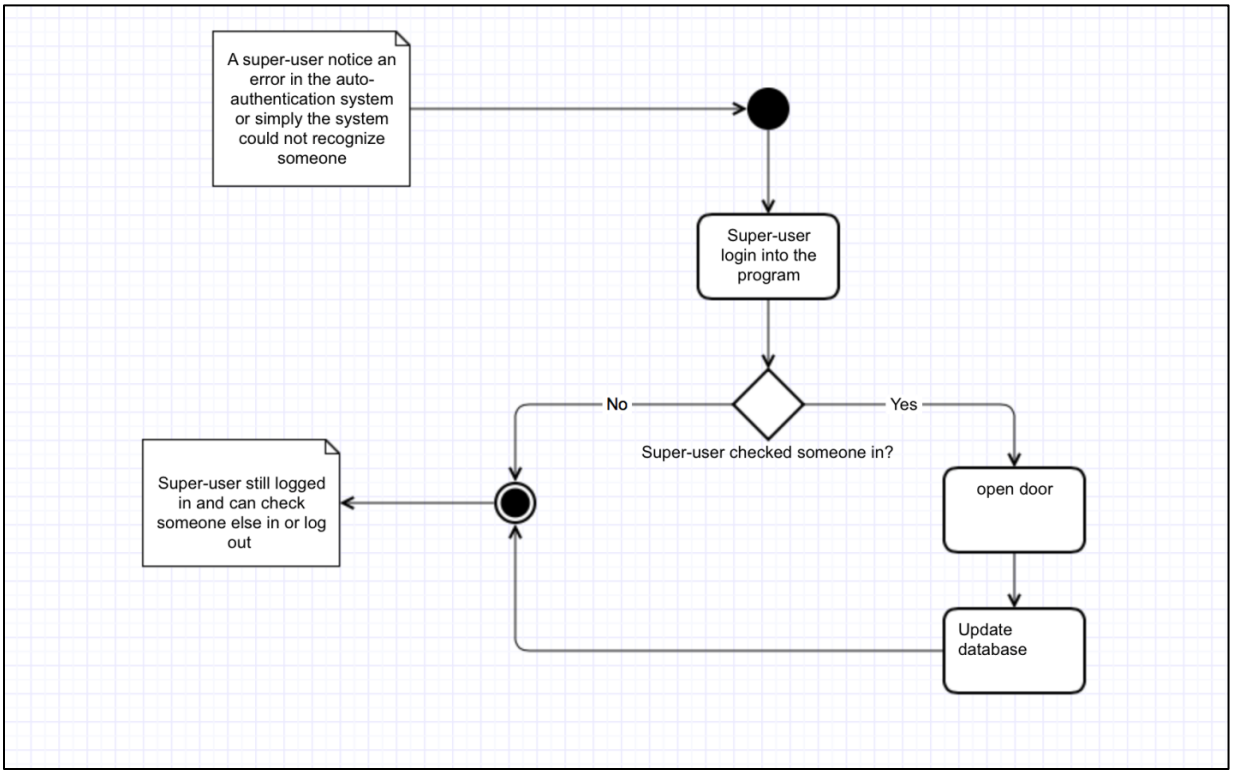

Figure 2 Override Authentication activity diagram.

B. Authentication

Primary Actor(s): Camera and user.

Description: A user passes by the camera. Then, the system identifies and recognizes this person. Based on the training database and the conditions set before, the system will decide to either open the door or not. Finally, it will update the database records. Figure 3 Show an activity diagram for Authentication.

Pre-conditions: A user passes by the camera.

Post-conditions: open the door and change in the database records.

Assumptions: None.

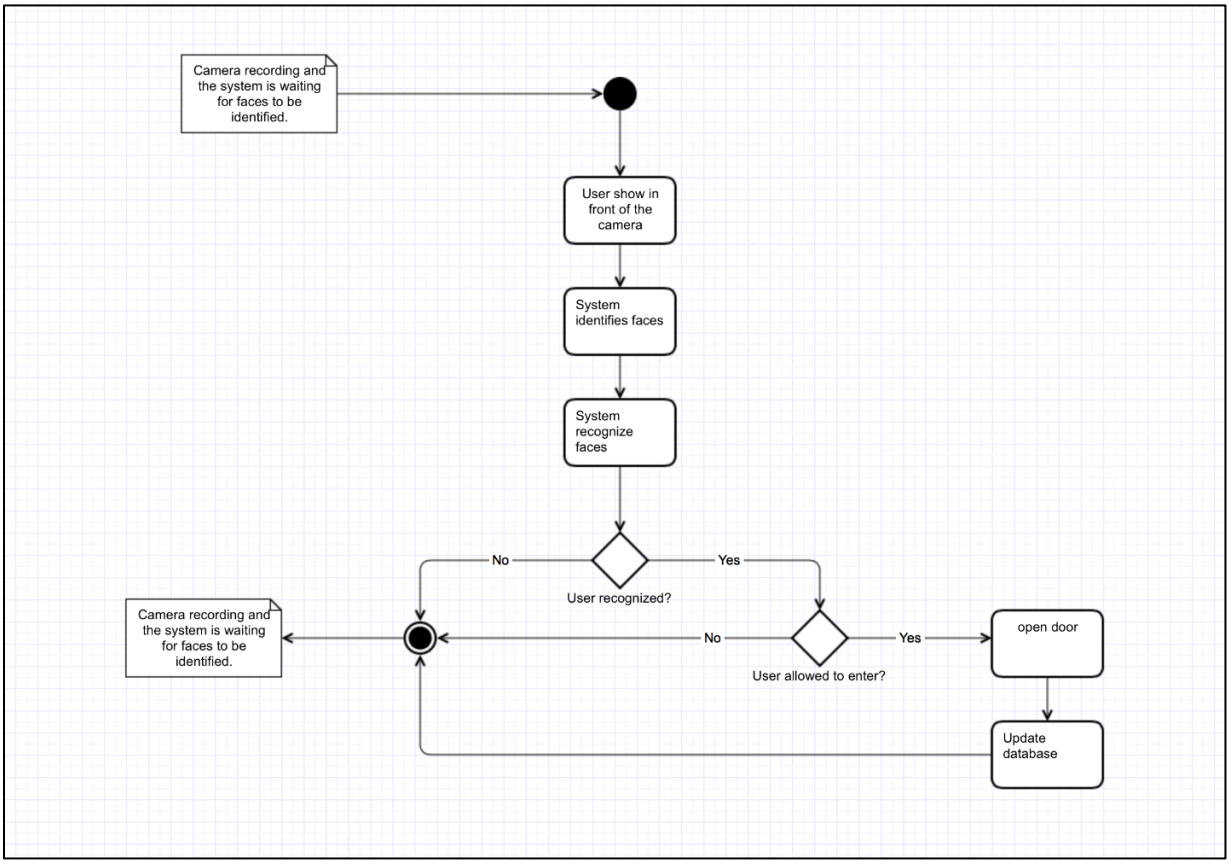

Figure 3 Authentication activity diagram.

C. Add User

Primary Actor(s): Admin

Description: Admin adds new user information and pictures to the database. Figure 4 Show an activity diagram for Add User.

Pre-conditions: A new user request addition.

Post-conditions: New user added.

Assumptions: None.

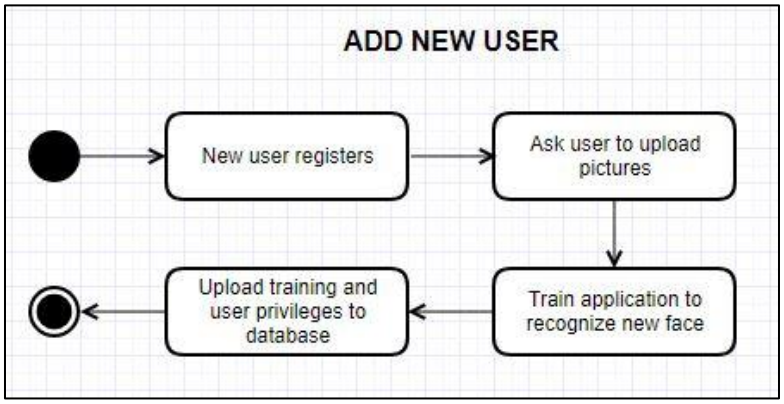

Figure 4 Add User activity diagram.

D. Remove users

Primary Actor(s): Admin

Description: Admin removes a user information from database upon the user leaving the system. Figure 5 Shows an activity diagram for Remove User.

Pre-conditions: A user is leaving the system.

Post-conditions: User removed.

Assumptions: None.

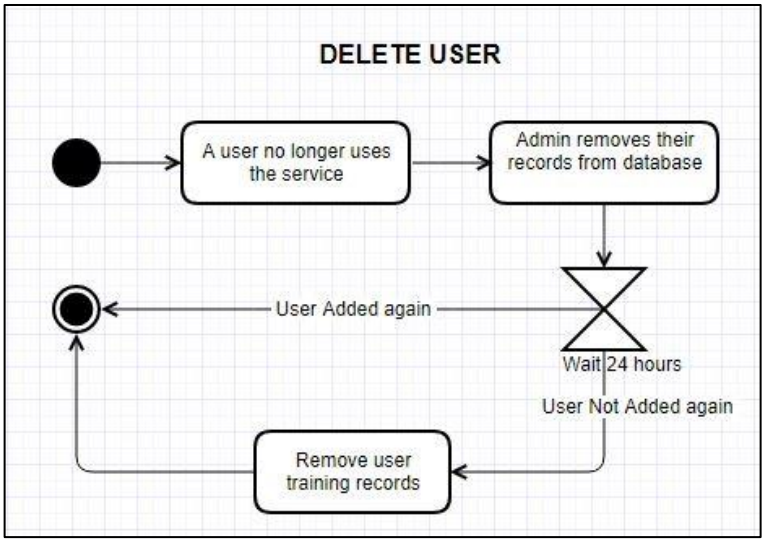

Figure 5 Delete User activity diagram.

E. Service Dispensing

Primary Actor(s): Service Dispenser.

Description: A service is dispensed by the service dispenser upon receiving a signal from application. Signal is sent to service dispenser in a specific node if and authentication and authorization happened from a camera in the same node. Figure 6 Shows an activity diagram for Service Dispensing.

Pre-conditions: A service requested upon authentication.

Post-conditions: Service dispensed.

Assumptions: None.

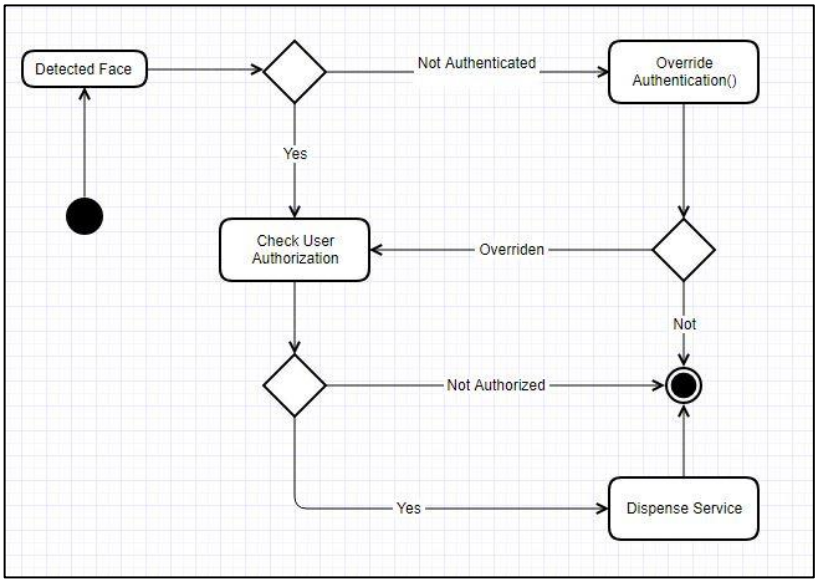

Figure 6 Service Dispensing activity diagram.

VI. SYSTEM ARCHITECTURE

Figure 7 shows the deployment of the system. The main node in the system is the **main server**. Administrators can interact directly with the server through a graphical user **interface**. **Authentication** through face recognition occurs on the server, this is achieved by fetching a video stream from a **camera** in a service node and applying detection and recognition on faces in the stream images. When a user is authenticated, they are then authorized according to their privileges which are specified in the **database**. A service signal complying with authorization is sent from server to the service node's **service dispenser**. The software on the server detects motion that is occurring in a video stream, when motion is detected, **surveillance** information are stored in the data center. Super users interact with the system through graphical user **interfaces** that can be installed on any Windows or Linux machine connected to the same LAN as the server.

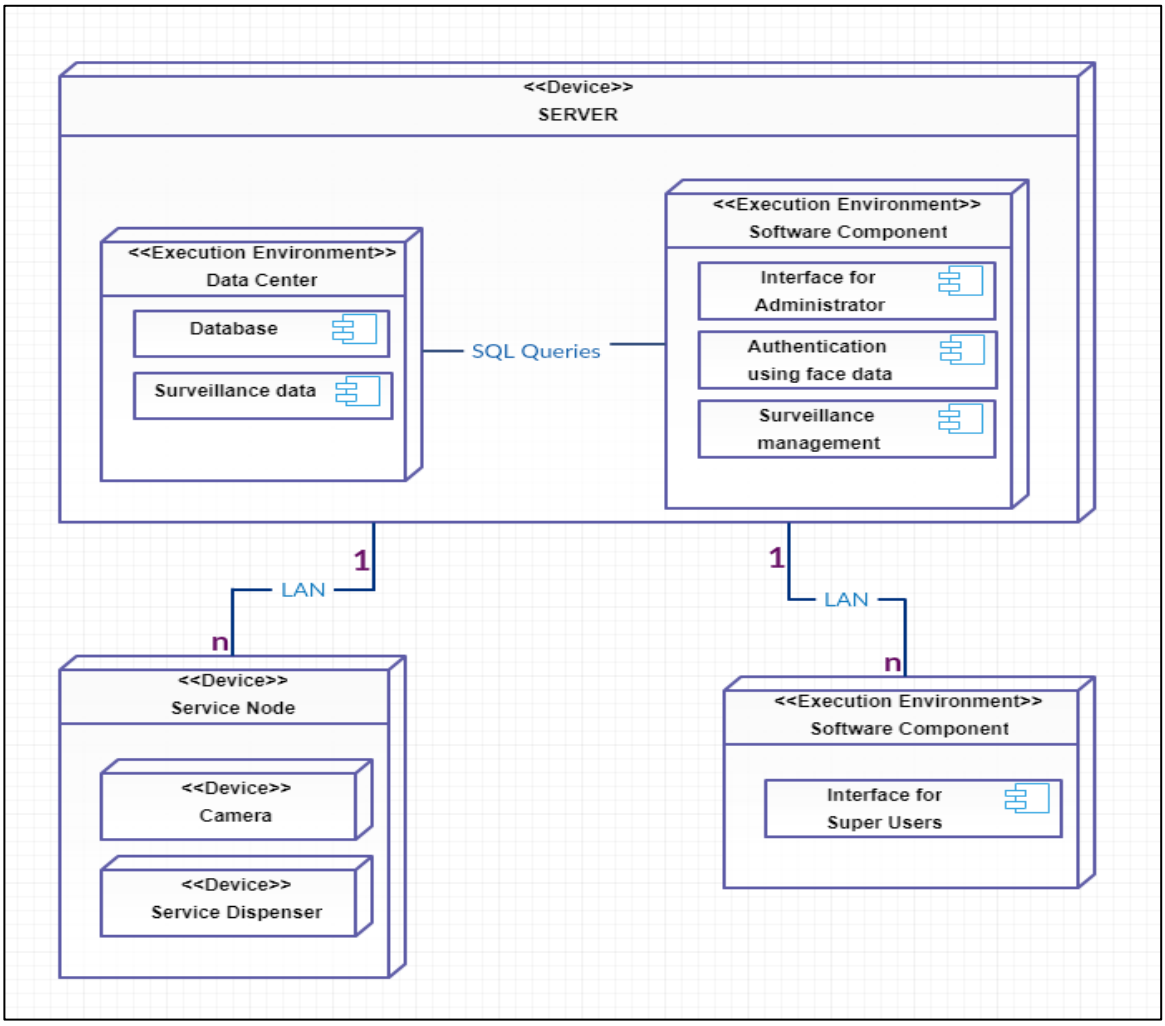

Figure 7 Deployment Diagram

VII. FACE DETECTION AND RECOGNITION

To use a person face as a biometric mean for authentication, there are two main steps for a machine to do so. First, identify where a face location is in an image, -usually this image is a frame from a video stream from a mounted security camera used for the purpose of face detection and recognition- this step is called face detection. After a face is detected in an image, it is cropped from the image and used in a face recognition *algorithm* to identify to whom this face image belongs to, from a known set of people.

A. Face Detection in OpenCV

OpenCV is a computer vision library for C++ and Python that provides many functionalities for dealing with single images as well as stream of images i.e. video. Object detection functionalities are provided in OpenCV using the Viola-Jones framework. Haar cascades are used to identify objects. A cascade can be trained to identify a specific object. For example: a hand watch, vehicle or a human face. Fortunately, cascades that are trained to identify faces were already developed and included in the library. The function `cv2.CascadeClassifier(cascadeFile)` takes an xml file that has been trained to identify objects as a Haar cascade, then returns a detector object. The `haarcascade_frontalface_default.xml` cascade file is used to detect faces. The function `detectMultiScale()` of the detector object takes an image and return the coordinates of the squares that contain the detected objects, in our case, squares that have a face fitted in.

B. Face Recognition in OpenCV

A face recognizer is supported in OpenCV library contribution, i.e. the face recognizer is OpenCV is not native to the library, but rather a contribution by other developers. The face recognizer has three methods each using a different algorithm to identify a new detected face from a set of known faces. The three methods are:

- **Local Binary Patterns Histograms (LBPH)**
  Compare a pixel in an image against surrounding pixels, the result shows the patterns the region has, eventually analyzing different patterns to infer information about image features that can be used in the comparison between other images features and identify whether it belong to the same person or not, according to some threshold.
- **Eigen Faces**
  Considers an image with h*w pixels, a matrix with h*w dimensions. According the principal component analysis (PCA) however, since not all dimension are equally important for

determining features of the image, only the few dimensions having most of the information are considered, and they are determined for having the highest variances.

- **Fisher Recognizer**

  An improvement on Eigen Faces, where not all dimension with low variance, that may account to features, are thrown. In addition, dimension with high variance that may be affected by external sources, such as light, and are not actually contributing to determining features, are examined.

The face recognizer has the following methods:

`Train()`: Takes a set of face images and corresponding IDs, train against them and output a yml file that contains training data.

`Update()`: Update the yml training file with new images. However, this method can only be used with LBPH algorithm.

`Predict()`: Takes a face image and compares it against the yml training data file, returns the predicted ID and a number stating distance. Lower distance indicates a higher confidence of the prediction.

The three algorithms were tested for a set of images for five known people, each having two set of images. The first set was used to train the recognizer, while the other set was used to test prediction results. While LBPH and Eigen Faces accomplished relatively lower success rates, i.e. less than 50% positive identification, Fisher algorithm obtained higher accuracy of 64% positive identification. Figure 8 shows success rate for fisher algorithm for each of the 5 people in the test.

Figure 8 Fisher algorithm positive identifications

C. Face Landmarks in Dlib

While the Fisher algorithm was the best of the three algorithms to identify and recognize face, results are still not optimal and do not comply with the system requirements. To improve the results of face recognition, there is a need, not only to detect faces, but also to detect

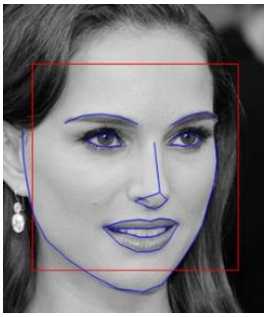

Figure 9 Face Landmarks posed on a face image of Natalie Portman

face landmarks within a face image. Those are coordinates of the image that specify locations of the chin, nose, lips, eyes and eyebrows. Figure 9 shows face landmarks posed on a face image. Figure 10 shows the 68 face landmarks that can be determined in Dlib functionalities. This information can be used to optimize the previous mentioned algorithms by providing a hint of what to compare between two face images. Dlib is a library that supports image processing functionalities using machine learning. `Shape_predictor(predictor_model)` is a method in Dlib that creates a landmark predictor object utilizing a trained predictor model. The `"shape_predictor_68_face_landmarks.dat"` predictor model was used to create an object for finding face landmarks. To utilize face landmarks in the face recognizer, the face picture with its landmarks has to be transformed so that the landmarks are in the same location for each image. This way the recognizer compares features from an image to the actual corresponding features in the other image. To transform an image with landmarks OpanFace `AlignDlib(predictor_model)` function is used to create an aligner object that will transform an image with Dlib landmarks. Figure 11 shows two face images of Adrien Brody and how his eyes location in one image is different from that in the other image. Figure 12 shows the two images after being aligned according to face landmarks. Average accuracy for fisher algorithm after using landmarks is 74%. Figure 13 show success rate for each of the 5 people in the test.

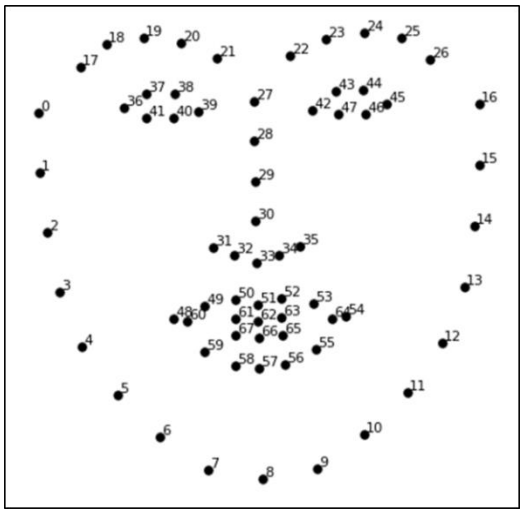

Figure 10: Face land marks found by Dlib's face landmarks predictor model

Figure 11 Two face images of Adrien Brody are not aligned when detected.

Figure 12 The two images after being aligned using Dlib face landmarks and OpenFace aligner

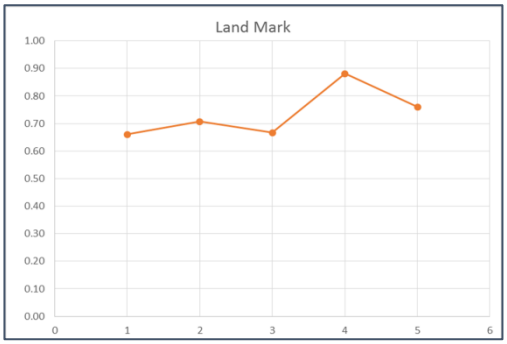

Figure 13 Fisher algorithm identification rate after images are processed with Dlib face landmarks

D. Drawbacks of OpenCV Face Recognizer

Even after adding face landmarks optimization, positive recognition averaged at 74%. That means, 26% of false positive for zero false negative identification. To even come close to zero false positive, identification threshold will have to be set so high that almost all identifications are false negative, i.e. to identify faces through OpenCV algorithms, perfect conditions have to be met, which is not practical. Clearly, face recognition algorithms do not resemble the accuracy that we humans have at recognizing faces. In fact, it would be surprising if they did, recognition algorithms are –more or less– comparing pixels of two pictures, not actual distinct face features, in that sense, any subtle difference like different lighting, would compromise the accuracy. On the other hand, humans do not compare differences between two faces, they just recognize them without much thought. There is a region in the human brain responsible only for that. In fact, people with damage in that brain region are described to have face blindness or Prosopagnosia. Those people would be able to describe a face they see, e.g. color of the eyes, whether the forehead is wide, estimate age and so on. However, it is almost impossible for them to differentiate between two different faces that have similar features. Face recognition algorithms can somehow resemble the process face blind people use in identifying faces. In order to reach any where close to zero false positive and less than 10% false negative, another school of thought than face recognition algorithms has to be followed, one that resembles normal human way of recognizing faces.

E. OpenFace Neural Network

In previous section, it was mentioned that face recognition algorithms will not be practical. But how else would a machine do the job of face recognition? To answer this question, we need to know what kind of task, if any, a machine can do that is not an algorithm, i.e. the process itself is too complicated to be described or explained in a flowchart. It turns out that artificial neural network comply with that definition. A deep neural network can take inputs and generate desired outputs, however, it is impractical to describe or justify what exactly is happening between the network layers. The same way that a face-blind person can describe the way they try to recognize faces, but normal people, though would try, but eventually reach a conclusion that they just do it not knowing how. Brandon Amos et al. developed an open source convolutional neural network (CNN) for identifying faces. The network has been trained against 500k labeled face pictures. The neural network is described as convolutional as some processing to input happens before it is passed to the

network. OpenFace CNN approaches human level accuracy at identifying faces, with **97.3%** positive identification under zero false negative (always making a prediction).

F. OpenFace Implementation

Table 1 OpenCV Face Recognizer Vs. OpenFace

	LBPH	**Eigen Faces**	**Fisher Faces**	**OpenFace**
Trainings Updateable	**Yes**	No	No	**Yes**
Accuracy	< 50%	< 30%	< 70%	**< 98%**
Training time	medium	high	high	**low**
Prediction time	medium	medium	**low**	medium

After a face has been detected and processed -this includes transforming face according to landmarks- from a frame, it is passed to the neural network as its input. The network outputs 128 floats called embeddings that range between -1.0 and 1.0. The network has been trained to output similar embeddings for pictures of the same person, and different ones for pictures of different people. One may wonder what each of the embeddings means, it turns out that we do not know. What is important is that embeddings are features that a machine can use to differentiate faces. This allows for an easy approach to handle data training. Each picture will have its own embeddings, therefore, adding to or removing from training is simple and does not require doing the whole training again. In OpenCV recognizer, this was possible only using LBPH algorithm. Table 1 compares OpenCV face recognizer and OpenFace.

G. OpenFace Drawbacks

OpenFace was originally designed to have high identification rates and identify faces in most situations even when an image is poor, has high noise or face features are not all apparent in the image like a sideway image or person in the picture wearing glasses. Thus, when high resolution images of faces with all features apparent are used in identification, the results are highly accurate with very high confidence for positive and very low confidence for negative. However, when images are not the highest resolution, we would expect, in our application, that results will tend to shift towards low confidence, but this is not exactly the case with OpenFace. The results keep on the same level of confidence as if the network is trying to make a best guess. To overcome this, information about noise and resolution of the input would be helpful, i.e. higher confidence would be required to register a positive identification if the input has higher noise or lower resolution. Hui

J. provided, using Python and OpenCV, a way to quantitatively determine noise in an image. This will be used after face detection to determine the required confidence for positive identification.

VIII. USER INTERFACE (UI)

A user interface is the means in which a person controls a software application, hardware device or a system in a natural and intuitive way. Our system has a database and some hardware component so we need a UI that can be integrated with many layers to allow users to interact with the system. This section shows decision, stages and implementation of a user-friendly interface for our system.

A. UI Choice

There are two main ways to implement a user interface for a system; programming an application or building a website. I started with choosing to write an application over building a web interface for many reasons. First, I'm very familiar with the GUI and because the time is the most important factor this was the main reason. Second, I thought that I don't have to write many interfaces since only admins and super users need an interface. This reason tricked me because writing a GUI is time consuming and in order to deliver a proper system, I need to write many interfaces. Then, I realized that it is better to build a website for many other reasons. First, a website is compatible with any type of computer around even smart phones can access it because they have browser. Second, a website is better because you can use it on all devices without installing extra applications. Third, websites have many frameworks so you don't need to build one from complete scratch unlike applications they don't have that much support nowadays. A website includes three main parts; backend, middleware and frontend.

B. Graphical User Interface (GUI)

When searching for GUI libraries you can find many such as Tkinter, GTK, PyQt and many more. We started using Tkinter then switched to PyQt because it is better for designing complex applications. Tkinter is Python's de-facto standard GUI package. It is simple and easy to learn but hard to align many components. It is best fit for simple application. PyQt is more than a GUI toolkit but it includes Qt Designer which is a graphical user interface designer. Qt Designer will generate a .ui file then PyQt can generate Python code from that. PyQt is more complicated to install and edit after designing but you can design a more elegant GUI. It is best fit for complex application.

C. GUI Implementation

I started with Tkinter library. It doesn't need to install anything and it comes with python only importing in the code is needed. I started to learn its functions and tested some of them. I did GUI application that has some buttons and more widgets to test them.

Alignment here wasn't easy. Because our interface will not be very simple, I decided to try other libraries.

Then, I installed PyQt and it needs some other libraries to be installed such as SIP. After that, I installed Qt Designer and started to learn how does work and how to design an application with GUI using it. Qt designer generate a .ui file. This file can be converted to a python code using the developer kit that you need to install first. Then, you will use the following command to generate a python code for the GUI you designed. `pyuic4 -x dbList1.ui -o dbList1.py`. I did an application that reads from a database and show list of users on a click of a button.

IX. BACKEND

The backend of a website is the database of this website and where the data is stored combining with a server to run the database.

A. Database

Database is a systematic collection of data that is organized and structured in a way to be easily accessible, managed and modified. It supports storage and manipulation of data and make data management much easier.

B. Database-management system (DBMS)

DBMS is a software application that act in the middle between a database and the other end either users or applications. They have queries to make applications uses and control the database. For example, you can create, modify, delete a table from a database and many more.

C. Database Choice

There are many popular DBMSs like MySQL, MongoDB, MariaDB, SQLite and many more. Different DBMSs are best fit for different solutions. Our project is a local system solution that requires speed, reliability and has normal data records (no multimedia stored in our database). Based on this, we used SQLite. SQLite is an embedded SQL database engine that reads and writes directly

to ordinary disk files. The application program, that wants to use the database, uses SQLite's functionality through simple function calls, which reduce latency in database access. We used SQLite because it is the best fit for local solutions, fast, simple, reliable and I'm familiar with SQL queries due to the database course I took earlier.

D. Data Modeling (DM)

For our system, we will be using Entity-Relationship (ER) model. An ER model is composed of entity types that are inter-related to each other. Every entity can be related to other entities or itself and every entity has attributes. For example, the relationship access has an attribute called Aceess and it is a many to many relationship. Given all of this, I know that there should be a separate table called access in my database.

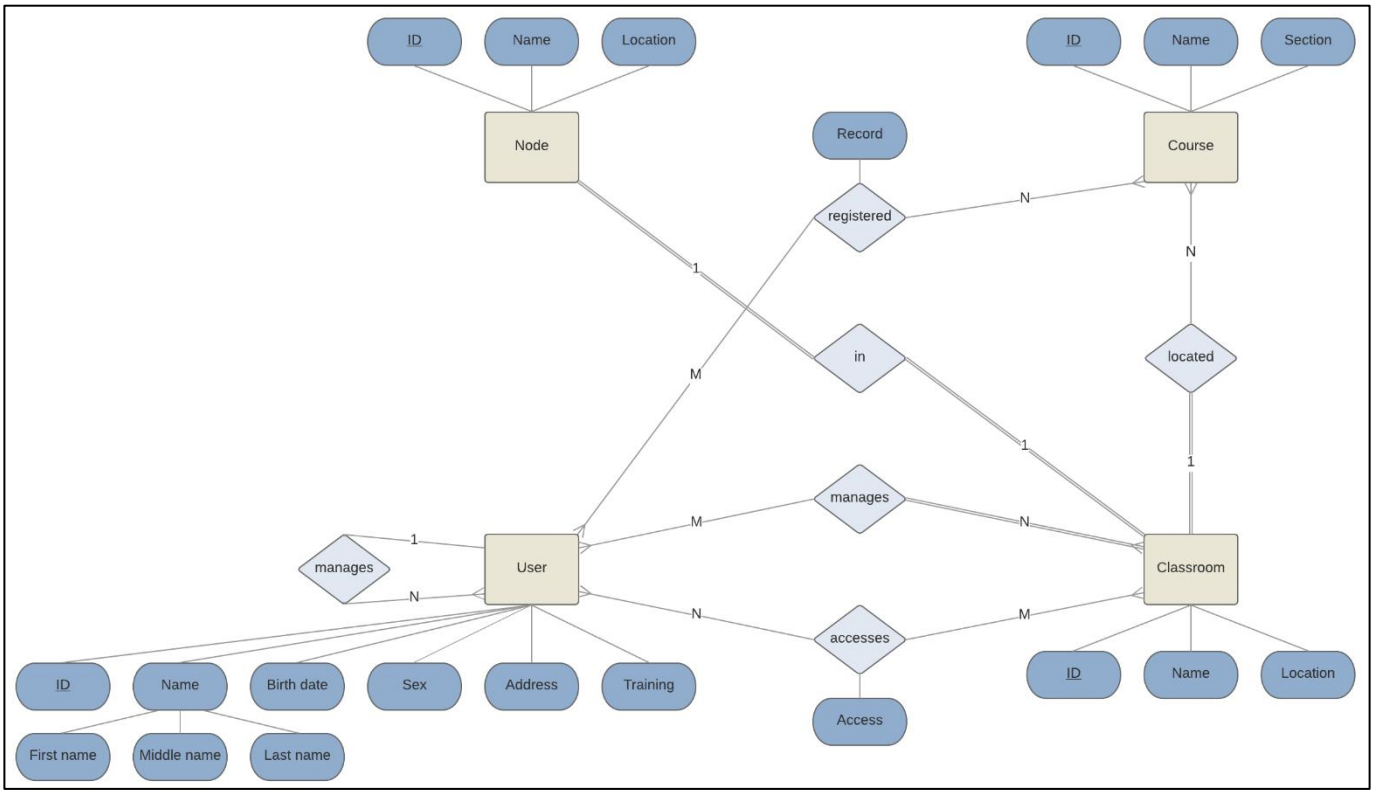

Figure 14 ER diagram

E. Relational Mapping

After drawing the ER diagram, we can get tables to be implemented in the final database using an algorism called relational mapping algorism.

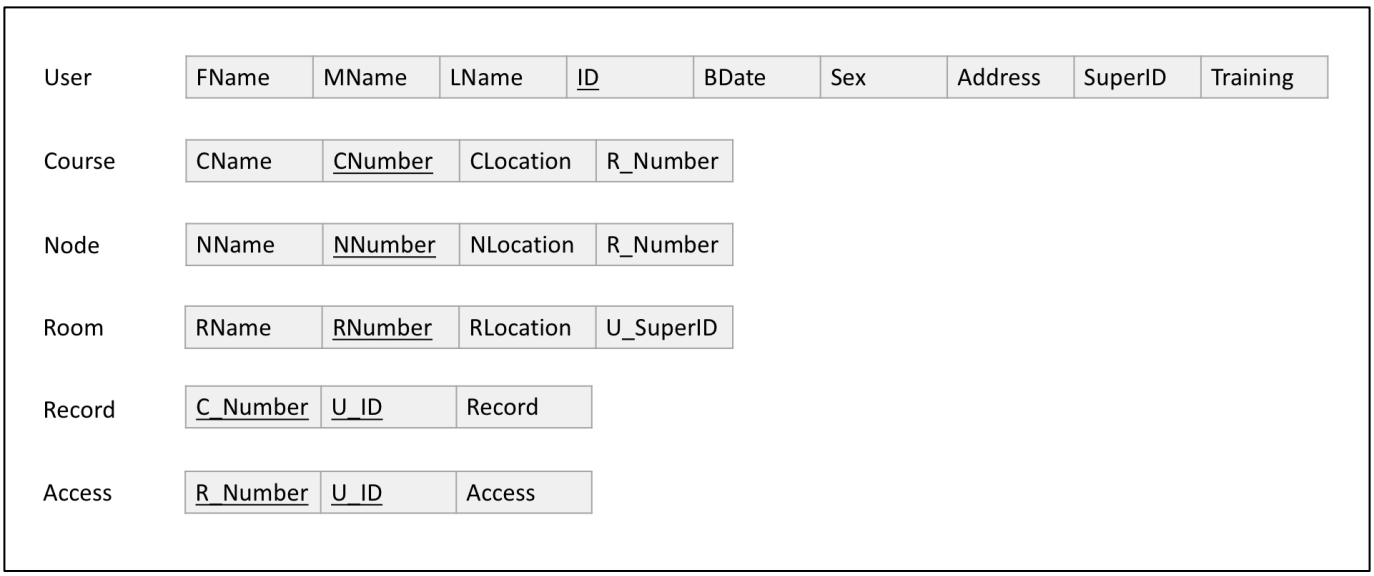

Figure 15 Data modeling

X. DJANGO AS MIDDLEWARE

The middleware is how frontend and backend are linked. Middleware can be a web server, application server, management system, and some other tools that support application delivery. There are many middleware frameworks to choose from such as; Ruby on Rails (RoR), Flask, Express.js, Django and many more.

A. Django

Django is a web development framework for Python. Using Django as middleware, we provide users with an interface that is easy to use, yet is secure. Django is being used by NASA, Pinterest, Instagram and many more. Moreover, it has rich documentation, a large community and a long term support. Django manages users accounts and privileges, authentication, web sessions and database through object-relational mapping (ORM). However, another web middleware than Django or any other user interface can be configured to work with our face recognition system, this is because the core functionalities of our system are implemented in a dependent application and are triggered through communication with other interfaces.

B. Models

In Django models, database tables are created according to Python classes. We will refer to classes as models. Each model will correspond to a table in the database. An instance of the model is a row in the table. A data attribute or a "variable" in the model is a column in the table. If no attribute was assigned as a primary key, Django will create one and will be referred to as `id`. Table 2 shows a model and its corresponding table in the database.

Table 2 A Django Model and its corresponding table

```
# MODEL
class Node(models.Model):
    camera = models.IntegerField()
    service =
models.CharField(max_length=10)


# INSTANCE
a = Node(camera=0, service='COM1')
a.save()
```

id	camera	Service
1	0	COM1

C. Users

Django provides a default user model which has a unique username attribute, password attribute in addition to other attributes. Passwords are not stored in plain in the password attribute, rather, iterations of hashing the password with a salt, and the salt itself. This provides protection for users password if the database is ever compromised. We created a Profile model that extends the user model so that we can add other fields to a user, most importantly, trainings and privileges.

D. Sessions

The `authenticate(username, password)` function returns a user object if there exists a user in the database with the provided credentials, otherwise it returns `None`. To create a session for the authenticated user the function `login(user)` starts the session, the session will keep on until `logout()` is used. `request.user.is_authenticated` is an attribute that is true when the current session is for user, otherwise it is false. When a user enters their credentials on the login page of the website, they submit a `POST` request, using the previous functions we can determine whether credentials are valid, and if so, create a session for the user. When a user submits a `POST` request to logout, the `logout()` function is called and the user directed to a page confirming logout.

E. Smart Login

If the user is using a machine with a camera, e.g. a webcam, a recognizer will try to identify them. If the person on the camera is identified they would be logged in automatically without the need to enter their credentials.

XI. FRONTEND

The frontend is the part of the website that the user interacts with. It includes core three components; HTML, CSS and Java script. For the frontend, we are using these three components along with bootstrap framework.

A. Integration

To integrate a simple HTML page with Django middleware, three things must be done; templates, urls and views. A template contains the static parts of the desired HTML output as well as some special syntax describing how dynamic content will be inserted. Then, we have use the view function which is a Django function that takes a Web request and returns a Web response. This response can be the HTML contents of a web page or anything. After that, we create URL path expressions and map them to our Django functions (views). This mapping can be as short or as long as needed. Also, it can be constructed dynamically. `url(r'^(?P<user_id>[0-9]+)/$', views.detail, name='detail'),` sets the URL to the user id as a number from 0 to 9 or more and map it to detail view.

XII. DESIGN

Now that basic components of the system are configured, they need to be connected together to perform the required tasks. Components are; the face recognizer, backend database and Django middleware. Tasks are; providing the system administrators with an interface to add/remove users and services as well as adding nodes and configuring them; integrate the face recognizer with the system backend as well as with the middleware; implementing a smart login.

A. Admin Interface

Django admin application is a simple way to implement an interface that allows admin to easily control database. Users can be assigned to have admin privileges, thus, they would be able to access the admin page and add/remove new users or services. To assign new trainings to newly added users, or update trainings for current users, admins can use a program in python to easily add trainings.

OpenCV takes care of capturing images from camera, pictures of newly added user are taken using a camera and run in a face detector, the detected face is shown on the screen and admin can decide to keep it as training or to capture a new frame. Once enough trainings have been obtained the user job is done, and trainings are stored in the user profile in the database.

B. Face Recognizer Integration

For the recognizer to retrieve trainings, nodes information and other meta data needed to perform recognition, it needs access to database. We set the recognizer to use Django ORM by setting the working environment to Django application that connects to the database. However, recognizer can be set to query database directly without the need for Django ORM, this would be optimal for a server-client database like MySQL. It is safer, though, to stick with Django ORM for an app-resident database like SQLite. `os.environ.setdefault("DJANGO_SETTINGS_MODULE", "website.settings")` sets the working environment to Django environment. The recognizer queries database for all trainings and corresponding users, it also queries nodes data and whether a node has a service dispenser. Then, it starts capturing frames from cameras of the nodes, perform face detection, when a face is detected it is run against the set of trainings and if a positive identification occurs, the services required at the node of the captured image are dispensed according to the identified user privileges.

C. Smart Login

When a user visits the main webpage of the system website, they have to enter their credentials to login and access their data. However, if the machine they use to access the webpage has a camera, it can be arranged that their homepage and data are directly accessed without the need to enter credentials. To achieve that, the client machine will have a program that accesses the camera, detects faces and generates their embeddings. When the users queries the home webpage from the web browser, the webpage with a login form appears. The webpage also contains a script that will try to connect locally with the program on the client machine. If connection is successful, the script generate a random token that we will call a facetoken. The program receives the facetoken and sends it accompanied with the most recent embeddings to the backend server where the recognizer is listening, at the same time, it will send a signal back to the webpage script. The recognizer receives embeddings and facetoken, recognizes who embeddings belong to, and saves that information with the facetoken. The webpage script having received a signal from the program, will send a POST request that contains the facetoken to Django server. Django server realizing that the coming POST

request has no credentials, will find the facetoken in the request. It will send it then to the recognizer asking whether it has a similar token and if so, who it belongs to. The recognizer responds back telling Django who the user with that token is. Django logs that user in and starts a session for them, as if they had entered their username and password, then returns a webpage telling them that they are logged in successfully, with their data available. As complicated as the process seems, it happens instantly without any inconvenience to the user. Figure 16 clarifies the smart login process in a sequence diagram.

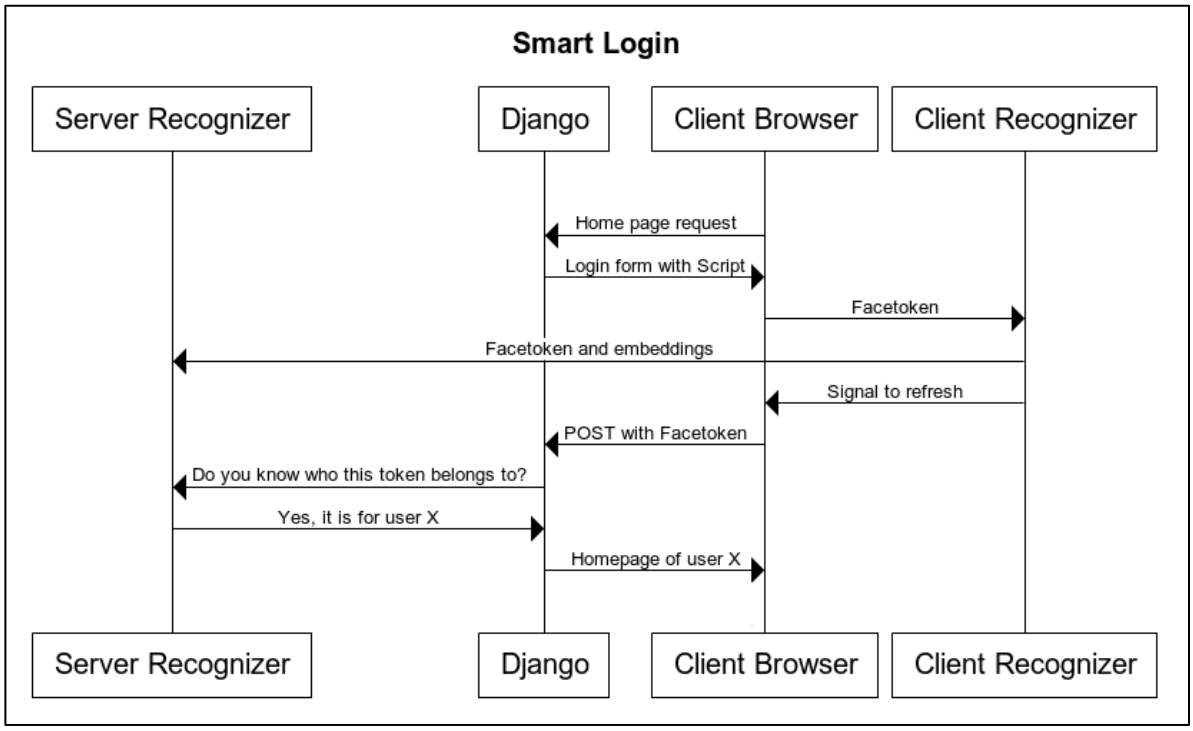

Figure 16 Smart Login Sequence Diagram

D. Class Attendance

Once the user is logged in or started a session, his ID is recorded until he logs out or ends the session. Then, the website request the listed courses according to his ID in another view or link. Once he chooses any course he can see his students' information including their attendance. Moreover, the super user or the teacher can override students' attendance in case of the student wasn't been recognized by our system.

E. Service Dispensing

A service dispenser can be used to communicate with an embedded device according to face authentication and authorization. The embedded device could be, for example, an automatic door

lock. To achieve communication between the recognizer and the embedded device, we use an Arduino as an embedded device and serial communication between recognizer and Arduino. However, Communication can be established through other means like Ethernet or Wireless communication. When a positive identification occurs at the node camera, a signal is sent to by the service dispenser to the embedded device at that node, according to the identified person privilege at that node. For example, if user x has access to room y, when they stand in front of a camera mounted at room y, the recognizer identifies that x is at y, then, it send a signal to an automatic door lock that will open room's y door. Figure 17 shows a the sequence of dispensing a service.

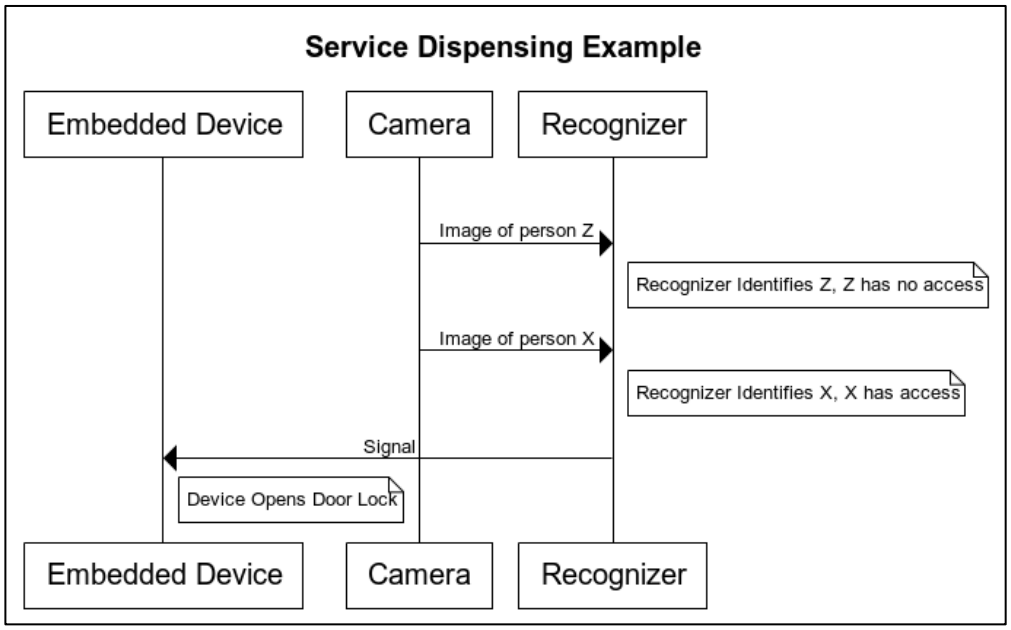

Figure 17 Sequence Diagram For Service Dispensing

CONCLUSION

Working in the senior project design project was a milestone in advancing my skills both in research and development. I have learnt a great deal about system construction and integration of many IT aspects. Open source software is a great way to both development of new product as well as learning. Biometrics, especially face recognition, is an evolving research area, and an essential one for a modern and smart technologies that strive to ease people's lives. The opportunities for further development are endless, those include; the advancement of machines recognition of faces through deep learning; utilizing cloud computing in the development of smart Internet of things applications that use face recognition. Furthermore, it is surprising how little research is dedicated for the development of solutions that utilize face recognition to help the visually impaired, or people with Prosopagnosia. Combining face recognition with embedded development would be a great tool to achieve that.

REFERENCES

B. Amos, B. Ludwiczuk, M. Satyanarayanan. "Openface: A general-purpose face recognition library with mobile applications," CMU-CS-16-118, *CMU School of Computer Science, Tech. Rep*., 2016.

Geitgey. A. Modern Face Recognition with Deep Learning. *Medium.* Jul, 2016.