Project F.R.A.M.E
2019-2020

APRIL 9

Project F.R.A.M.E
Authored by: Samuel Tredgett, James Clark, Hugo A'Violet

Contents

Abstract

Project F.R.A.M.E (Facial Recognition Attendance Monitoring Engine) is a lightweight program intended for use by the lecturers at the University of Kent, with potential expansion for use in other universities.

F.R.A.M.E offers several core features including inbuilt scheduling, fully functional facial recognition software, back-end database storage and direct data relay to lecturers' emails.

1. Introduction

F.R.A.M.E is a general purpose attendance recording software with the intended goal of taking work out of the hands of lecturers whilst simultaneous streamlining and improving the accuracy of recording which students are in attendance of any given event. Our objective was to create a system that could be easily integrated with minimal interruptions to university workflow, allowing lecturers to maximise the short amount of time they have for face-to-face student interactions.

1.1 Background

Before we began working on project F.R.A.M.E, we had several skills to learn and several materials to learn how to use. This was a crucial stage in the process as all members of the group were required to learn a varied set of complex skills in a relatively short space of time, making the task difficult.

1.2 A Critique of Attendance Monitoring

The current attendance monitoring system in place at the University of Kent falls short in several areas. Namely, convenience, simplicity, reliability and accuracy. Having to hand pieces of paper out that require signatures every lecture, too often in excess of a hundred plus students, is not only an inconvenient for lecturers and students alike, it is also wasteful in times where being paperless is becoming more expected of institutions.

It also by nature facilitates the ability for students to sign their friends into lectures when some are still at home sleeping off the night before. This causes inaccuracies in the attendance data, and should a lecturer or student lose a piece of paper then this problem is only exacerbated.

1.3 Python

As a group we decided on Python as our programming language for several reasons. Python can be a lightweight tool to design simple prototypes quickly. We used this to our advantage in early stages of design to see what possibilities and limitations we would run into during development. An example of this was deciding what facial recognition system was best to reach our goal.

1.3.1 Facial Recognition Libraries

As a group we decided on Python as our programming language for several reasons. Python can be a lightweight tool to design simple prototypes quickly. We used this to our advantage in early stages of design to see what possibilities and limitations we would run into during development. An example of this was deciding what facial recognition system was best to reach our goal.

We originally planned to create a machine learning algorithm that would learn overtime the faces of students to gain a better accuracy score. However, if we would have taken this method, we would have to have a large dataset of existing students to begin testing. A learning algorithm would also take a long time to gain precision. Therefore, instead of this we concluded that the face-recognition python library was more suitable when iterating over faces who attended a specific module.

The advantage this library had was it handled the face embedding and face matching in one sequence, speeding up the time it would have to recognize a specific face. We also wouldn't have to update the entire face database if a new user was added. When programming the facial recognition function, we originally had the database set up where all the students were in one folder. This iteration method would be relatively slow and unnecessary when certain students weren't attending the module. Therefore, this was remedied so when a class started the facial recognition database would point to the specific faces in that module.

MatPlotLib was the data analytics python library we used to provide visual representation to our gathered data. This library's focus was to create visually interesting and useful graphs for lecturers and schools to use. Difficulties faced with using this library was understanding how data had to be inputted into the system. An example we faced with this was making sure the X and Y axis were showing the correct data. We had to extrapolate our list of lists data generated by our database into two separate lists, then input these into our graph separately.

1.3.2 MySQL Development with Python

MySQL was our database of choice for several reasons. Firstly, MySQL benefits from scalability, therefore with a system that's designed to create new data every time a class has finished, it's essential to use such a framework. Using MySQL with Python was difficult and had its fair share of bugs. Executing SQL statements within Python requires the "MySQL Connection" library. It also requires every time a SQL statement is called to create a connection to the server. Originally, all the SQL connections shared a common variable name, however once we had to run parallel connections, we needed to separate these out so our program wouldn't be slow when interacting with our server. The former issue we had was our class timer being too slow, due to the amount of linear SQL queries. This was fixed with parallelism.

1.3.3 Amazon Web Services

Amazon Web Services was the platform we chose to host our MySQL database on. The reasons for this was because it was important to always have uptime and for multiple people to be working on the system at once. Having many connections connected to the database on a local machine would be taxing and inefficient. AWS provides a free plan that is as competitive as other paid plans.

1.3.4 User Interface (TKINTER)

Tkinter was another python library that we utilised. It's important that the application is as user friendly and interactive as possible. Tkinter was chosen because of the vast amounts of existing documentation available, along with its lightweight usability and customization. During development, understanding Tkinter and how we can customize our GUI to provide a minimal and professional experience to our user's was important. Originally, certain key components such as the 'Select Room' menu option appeared as a text box that Staff would have to manually type in. This type of interactivity wouldn't work when the system has been designed to be a 'touch-screen' experience. Therefore, we had to develop the system to incorporate a touchscreen keyboard. This was done by creating a variety of buttons that when pressed would update or clear the room number. Understanding Tkinter and its relationship with Windows, also gave us the ability to create a Student Mode, which was essential to incorporate.

2. Requirements

2.1 User Stories

As a university student, I need to sign into my university lectures and seminars so that I receive attendance marks.

As a lecturer, many times I need to record the attendance of the students present at my lectures. Using paper this becomes a battle for accuracy and finding the papers as well as causing interruption to the lectures. I do this so that I can predict grades and see which students are lagging.

As a PHD student running seminars, I need to record the attendance of my seminars in order to award presence marks. I do this so that I may know who requires more tutoring on the topics at hand.

2.2 Non-functional Requirements

Creating a system that was lightweight and easy to use was one of our main goals for the deliverable software we created. We've set ourselves three main non-functional requirements, ease of use, speed of use, and reliability.

Ease of use – having a deliverable product that could be used by lecturers and students with ease was one of our most important non-functional requirements. We wanted the system to run smoothly so that it can be almost forgotten about except by the students who need to stand in front of it.

Speed of use – a fast run-time from recognising one student to being ready to recognise the next is also a high priority, as it prevents congestion at the entry of lectures allowing the lecturer to begin much more quickly, only needing the students to show their face to F.R.A.M.E and then take their seats while the lecturer sets up.

Reliability – lecturers are time strict, therefore having a reliable system was our most important non-functional requirements. Ensuring that all data recorded is stored properly and with accuracy is the entire purpose of the project, if it came to it we were prepared to make sacrifices in regards to the goals of the other non-functional requirements so long as reliability is not compromised.

2.2 Prototyping

Features: During the prototyping stage, it was important to understand how an intuitive user interface should be designed. Therefore, an important aspect of prototyping was designing a GUI that would be relevant to our hi-fi and lo-fi designs. While learning and improving upon the system through understanding new components, removing redundancies and updating the code to be more efficient. Towards the end of prototyping, we had a working GUI with minimal back-end support to allow for testing that everyone worked as required.

Challenges: There were several challenges faced by creating a working interface. Once the initial templates were running, we would encounter issues with the relative sizing of buttons and images. Code was added to resize these components when a User would change the window size, however as more components were being integrated into the system, we settled on fixed window size. This was mainly due to prioritizing development to meet strict weekly deadlines. We also discussed that once the system has initially been installed on a device, there is no need to resize the GUI. 'Student Mode' makes resizing redundant and as a "touch-screen" system, users shouldn't need to control window size. Another challenge we faced with developing a GUI is the overall aesthetic and design. This needed to be professional, yet minimalistic so users can easily navigate the interface.

We settled on a basic colour palette that matched our Universities and used a font that came with the Tkinter Python library. The design was a complication throughout, with several design changes focused on redesigning buttons, the background, and pop-up labels. We originally had trouble with designing the on-screen keyboard, and where to position it. It was originally in the same box as the room number, but we agreed that the room number would need to be shown larger than it was.

Therefore, the on-screen keyboard was designed to overlap the ‘Sign In’ button. This was because the button didn’t need to be pressed while selecting a room number.

Learning outcome: We learned a lot from this experience. We understood how important it is to spend time prototyping different designs and making sure it was user friendly and reliable. The result we achieve was a working GUI that clearly displays basic information, ready to be developed further.

Facial recognition iteration one

Features:

Prototyping different ways to detect faces was a necessary step to completing a working product. During this phase, we discussed at length different python libraries or ways this could be done. The original idea was to have all students listed in a single file directory, and then have the facial recognition model would check if a face matched in the directory. The original method used the OpenCV Python library to train and analyse the faces. This is a more accurate method of facial recognition, but we would need a relatively large training set to test the project. Therefore, we settled on a different Python Library called ‘Face-Recognition’ in the final project.

Challenges: As we briefly outlined above, the main concern with using this way of analysing the data is that the algorithm would have to be trained a lot to get positive results. We would also need to train the algorithm on unknown faces, this couldn’t have been faces of people within the database, so we would have needed to aggregate unknown faces to populate this section. Another challenge we faced when testing with the ‘Face-Recognition’ library was the time it took to iterate over the face database. The algorithm needed to check each face individually and if a face existed that wasn’t on that module it affected the performance to solve this problem, we split the face database directory into subcategories that were sorted by module code. This sped up the search drastically. Learning outcomes: We learned how to understand the importance of researching different ways in which a problem can be solved and understand what’s the most fitting technology to use that’s within our scope. The resulting prototype we achieved from this was a quick face matching system that worked as expected and combined with the GUI, created an adequate first prototype in which to expand.

2.3 Delivered Project

Retrieving/inputting database information

Features:

Once the prototype was complete and had a working GUI and facial recognition system, we needed to retrieve data from within the database. The database stores all the User and class information so developing a way to query this information and then present it to the screen was needed. Using the "MySQL Connection" library in Python we coded a variety of queries that could collect the data and store them within lists and variables.

An example of a query was collecting the class data when a class was found and outputting the information to the GUI. To do this we used threads. Threading allows for different processes to run parallel to each other, so the program doesn't halt when it's still needed. We created several threads, the first being a class check timer. This ran as soon as the program opened and would run a SQL query every second, checking to see if the current time matched a class time. Once a match was found, the database entries were converted to variables and printed to a label that appeared on the GUI. Another two threads would then begin, a class timer and a late timer. These would run in parallel; the late timer would finish in half the time and then mark users as late. Once the class had ended the whole process would reset, searching for the next class. The information gathered by the sign-ins would then be sent to the database, creating a new table with the user's information and whether they attended/were late.

There were several other types of queries created, these included: resetting the room class database once a class was over. This would clear the table, once the information had been copied over to a new table. It was important to do this, otherwise, we would have many different tables being created for each class, filling up the database. Another query would select the current User's information from the table and output it to the screen. This would happen if a user successfully signed in.

Challenges: There were many challenges associated with retrieving and updating the database. The biggest one was making sure there were parallel SQL queries or commits taking place. This is because the system relies on multiple actions happening at once. For example, updating the class database, creating a new class table, and exporting the attendance list. We discovered that having multiple variables for the connections fixed the problems we were having where connections weren't happening in parallel. We discovered that the MySQL connection library can only connect once to the database, so this was fixed with the different variable names.

Another challenge we faced was making sure the SQL queries were dynamic. For example, we needed the class check to check the correct room, if the room changes, we needed the system to check that table without having to restart the system. This was done using variables within the SQL queries. A placeholder would be inputted into the query that represented the current variable value. When this value changed the SQL query would still produce the correct results. This was needed for every SQL query. When creating a class table with the copied data from the room we needed variables to check what database (module code) it should be created in, what user information should be outputted to the screen depending on the current user-id and where to retrieve the data that needs to be exported.

Learning outcomes: We learned how to successfully retrieve and input data into a database using the MySQL connection Python library. We also learned a lot about parallelism and how to efficiently structure the program to produce an efficient workflow for updating and searching for classes.

Populating and testing the system

Features: During designing the final version of the system we needed to make sure the system could handle large datasets; this would be a valuable test to see if the system would struggle in a real-world scenario. Therefore, we downloaded a facial recognition database from MIT that contained over a dozen faces, as well as training images that had the faces at different angles. We populated the database with new students and correlated a face to each student. We also created several more modules and class dates and scattered the users across these modules. This allows the system to have a larger dataset we can work with and to test the system for an increased period. When testing the system to see if it recognized the faces, we selected images that were from different angles, and the system successfully outputted the correct user details. We also had classes scheduled consecutively to see how the system timers would cope with the continuous function calls.

Challenges: There were several challenges that were associated with testing. Several major bugs were found that needed to be fixed. One of these was the system wouldn't correctly clear the email list after each class, so the following attendance list for the next class, if a user attended both would have their previous details. The fix for this was to make sure all the Python lists would reset after exporting the email. Furthermore, we also had problems with the system timer not searching for the next class quick enough. This meant after the first class would be updating, it would miss the timer for the next class altogether. The reason for this was we had a lot of MySQL queries being sent at once, so we were waiting for the database server to retrieve the information. This, along with exporting the data to an email, cleaning the room table and creating a new table took time. We decided that the best approach would be to do these SQL queries before the class had finished. This worked well and meant Students couldn't sign in at the end of the class or just turn up at the end to sign in. This gave the system enough time to perform the operations and have the room ready for the next class.

Learning outcomes: Throughout this process we learnt as a team that populating and testing the project reveals bugs. So, doing this sometime before deadlines gives us time to improve upon the system and make sure its functionality is as it's supposed to be.

Data analytics

Features: Originally, we weren't designing a data analytics system to accommodate the system. However, once we had a variety of tables containing User information and had time to expand the project, we decided on implementing a form of data analytics. This feature allowed staff to access another application, where they can input in the module code and an email, the system would retrieve all students and classes and correlate how many classes each student had attended. It would then email the address provided with a bar chart containing this information.

This was done using the matplotlib Python library and data gathered from the MySQL database. The process this took was it would first get the students ids that attended that class and populate them into a Python list where a default attendance value of 0 was given to each id. The SQL query would iterate over the module code's database and check each table to see if a 'YES' was found in that row. If a 'YES' was found, it takes the user id for that row, enumerates over the attendance list to check if the user id matches. If it matches it appends a value of plus one to that specific list item. This process is repeated until all classes have been checked.

Once all classes have been checked the system splits the attendance values and users into two lists, to be made into x and y axis on a bar chart. The length of the y axis becomes the length of the attendance values, and users are inputted to the x axis. The system then creates a bar chart and emails the email, converting the bar chart into a PDF format.

Challenges: Understanding how matplotlib worked was the biggest challenge during this stage. The other processes were already in place, exporting to a pdf and emailing. There were many iterations of bar charts that weren't correct due to understanding the relationship between the x and y coordinates. Once this was resolved the bar chart worked correctly. Another challenge that we needed to approach was iterating over the rows and appending the correct list with matching user ids. This was done using n nested for loops and if statements. If the list at index zero matched the userID then the index at position one in that list gets added a 1 to its current value.

2.4 Potential Future options

Expanding the data analytics system

Features: Expanding the data analytics system is a potential future update that would benefit staff members in gathering a variety of different analytics quickly and reliably. Expanding upon our current system, we could add several different methods to search individual user attendance to see how they are performing over the course of the year. A plot graph could be useful to analyse a student's progress and reveal if they require support. The current system doesn't take individual users into account and reveals a bigger overall picture. This is useful but can also be implemented to include individual lecturer class information. It can be used for annual reviews to see the overall progress of employees. Once the system has collected a large amount of data, a variety of analytics methods can be used to create meaningful data. The graphs could also have lines that could notify the senior lecturer if a student falls below the expected attendance marks. The system would then email the appropriate staff

members notifying them, this can be useful to keep track of a large portion of students who need support.

Challenges: Challenges that could come from expanding the data analytics include the time it may take to analyse data if say, at the end of the academic year for a single student. Therefore, a database could be created that stores this academic value regularly so when using the tool, the process won't take a long time to extract the data, as it's already been stored before.
User Interface Optimization

Features: There are several features that could be added to the user interface to help optimize the system and improve the overall workflow for lecturers and students. Firstly, to access the menu bar staff could require a sign in option. This would remove the need to have a menu bar as another window could open that could be used to select the menu options. Creating a more efficient touch-screen system. Staff could type in their user id and password to access this window and either close the window manually or set it to student mode to close the window. Currently the menu bar requires the system to not always be in student mode. This could be changed so that it could always be in student mode with this optional window. Another optimization feature we could add would be changing the room select keyboard to accommodate letters as well as numbers. Most rooms are categorized by building name and then the room number. So, this could eliminate the need to remember a longer number. To expand upon this idea, the database could have databases for individual building names that could be selected like the room number. Another potential future feature we could include is optimising the design and style. Currently the design is somewhat professional but could use a facelift on the button styling to provide a better professional look.

Challenges: A challenge associated with a login system is making sure the passwords are stored securely. To do this we would need to hash the passwords and provide a salt to accommodate the password. As these would be stored in the database, we might consider storing these in a separate database so administrators adding to class schedules wouldn't have the option to look at these salts.
Expansion to other industries Features: F.R.A.M.E is a diverse system that could be expanded to other industries outside of University. It could be optimized to be used in workplaces and offices. For meeting attendance and clock in / clock out processes. The system could also be used as a secure way in combination with other biometrics to allow users into specific buildings and show when they entered and left.

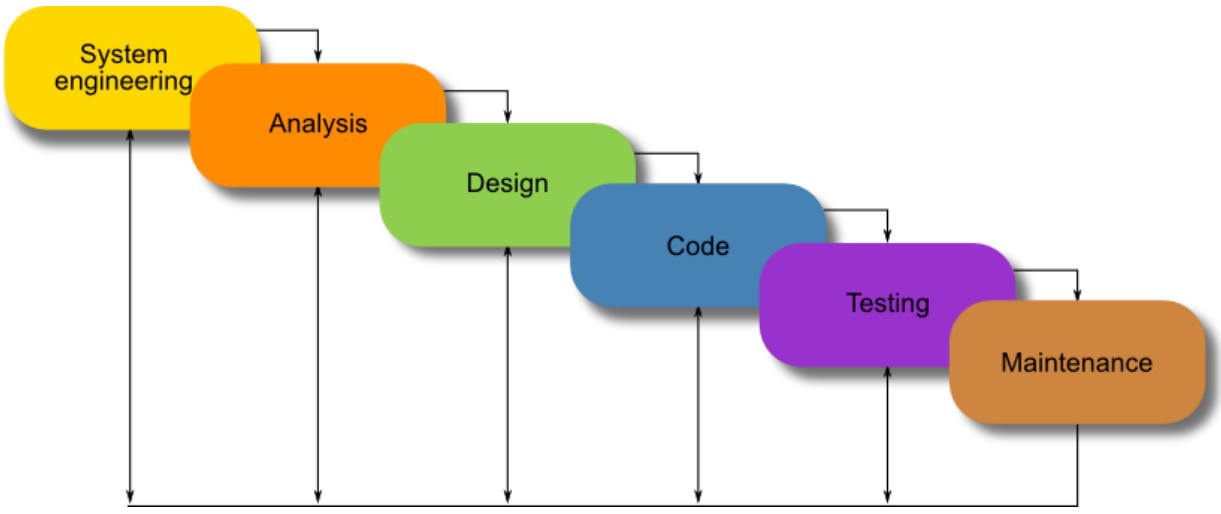
Challenges: The major challenge this feature possesses is that the system would need a complete rework to accommodate for these features. However, the processes in place allow for expansion and customizability for specific databases and GUI styling. If the menu bar was kept these options could be changed quickly without having to design the GUI to fit the specific organization's needs.

Optimising Code Linearity

Features: Throughout coding the project there are many ways the code could be optimized so the processes would be more efficient and linear. When a class has finished, the system generates a new table with user information. This process if wanted could be made redundant by removing the temp classrooms and directly creating a table in the specific module code database. This would speed up the process when checking for a next class and remove unwanted SQL queries. Another feature that could be optimised includes the way the attendance list is exported. To speed up the class checks the attendance list could be added to a stack. This stack would then slowly be updated, and the attendance list could be sent to the lecturer an hour after the class has taken place.

Challenges: The challenge with redesigning certain parts of the code relies on the current way the database has been implemented. The database would need a redesign to ensure these changes would be efficient.

3. Waterfall Development



[Figure (1.1) [1] Waterfall model]

When deciding what SDLC to use, we initially were settled on agile development in short scrums. However, we considered the length of the project at large and the other things we'd have to balance with it. This led us to decide to use the Waterfall development life cycle.

This is a sequential model that outlines the process as a chain of successive steps (see figure 1.1). this allowed us to keep a clearer overview of where we were at every point in time as well as focus solely on the context of the section that we're working on i.e. design.

4. Quality Assurance

During our development of the system, we have continued to ensure that the project is meeting the specified requirements outlined. To achieve this, we have been using various quality assurance and testing methods to examine different parts of the system and improve/fix these areas where required. Testing is a vital component to any software project, as any issues that aren't resolved can significantly affect the system performance and users experience.

During development we focussed on localization. This type of testing is useful when creating a large system. Being able to create independent code that works as intended, before adding them into the main system is an approach that allows us to easily find bugs relating to that specific component. Many of our components, including the GUI, facial recognition, splash screen and data analytics were all developed independently before being merged. However, this process can allow the coding to take a longer time, this is due to having to implement each component into the main program and to make sure they all work in conjunction with one another. This is a sacrifice we were willing to take, as the system has many different components working independently.

Another approach we took during development was to continuously test after a feature was added. This was usually done at the end of the working day; the system would be run through a variety of user and system testing to ensure the added components worked as intended. This type of testing revealed some major bugs, including many database issues, such as tables not being created properly, or user information not being retrieved when required.

User Acceptance Testing is the process of verifying that a solution works for the user. The document UAT Form goes into this in more detail with examples, however this was a core part of our quality assurance during this project. As the system needs to be efficient, as our idea was it would be used at educational institutions, we needed to make sure the application didn't crash, the functions all work as intended and the application consumes the minimum amount of resources (relating to threading). UAT was useful to us to see is a user could really use the software, we separated this process into two sections, the GUI navigation and the facial recognition system. We asked questions such as: "Does it behave as intended?" and "Do they have trouble navigating the system?". These questions helped shape our future iterations of the project.

Interoperability within the software ecosystem allows systems to be integrated together to use data in a coordinated manner. F.R.A.M.E incorporates the Amazon Web Services platform with RDS to run the MySQL database. It was an important quality assurance we needed to provide as AWS is a reliable way to store information. Their services are always online, therefore F.R.A.M.E does not have to worry about not being able to access the database. The type of interoperability used is Level 1 Foundational. This establishes the requirements for one system to connect with anyone, in this case F.R.A.M.E with AWS.

Furthermore, F.R.A.M.E also incorporated different quality control methods throughout. The primary method was a checklist system. This system was documented on the platform Trello and allowed our team members to easily understand what they needed to do to achieve a working system. The checklist laid out different components needed to have a basic working system, after the working system was complete the features would be less functional and more about optimization. This broke down our project into small target goals, allowing us to not be overwhelmed with the size of the task we had. It kept us focussed on on-track with small deadlines that needed to be met, rather than one overall deadline.

During the project we needed to test that the system could work on a larger dataset to replicate a real-world situation. Testing the system on a single user (the developer) was efficient enough until we started to include multiple classes and modules. Therefore, we settled on using an MIT face database called ‘The MIT-CBCL face recognition database”. This database contained face images of 10 subjects, these were high resolution photos, from varying angles. We used this database as training images to see how well our system would cope. We added these users to the database, using their front facing image as their user photo. We populated the database with these users and added them to a variety of modules. Once the system was setup, we could run the class schedule. During this process classes would begin, and the developer would select a training image from a given student who attended that class. We discovered that the accuracy of the system was 100%, with no wrong user details printed. The varying training images were comprised of different ages and ethnicities. This testing method proves to be effective and a quick way to see how the system would cope, without having to prototype the project in a real-world situation first. F.R.A.M.E handled the tests well, and we concluded that the system would be reliable enough to be used.

Finally, towards the end of the project we wrote a User survey, where we chose a large audience range and asked them to watch a video and fill in several answers, relating to questions. This testing is discussed fully in our Testing Analysis document. However, this method of testing our system related to quality assurance. This, along with our User Acceptance Testing provided meaningful user data that allowed us to understand our positive and negative aspects of the project. We can then use this information to improve upon the project in later versions.

Our testing methods overall proved effective, there was both functional and non-functional testing involved. We also made sure to include quality assurance methods that our team could follow so our project would meet the specified deadline with a working prototype. The testing also revealed that the project works as intended and that many users would be interested in such a product being made available at their educational institutions.

5. Bibliography

1 - Airbrake Blog. 2020. Waterfall Model: What Is It and When Should You Use It?. [ONLINE] Available at: https://airbrake.io/blog/sdlc/waterfall-model. [Accessed 08 April 2020].

2 - PyInstaller Quickstart — PyInstaller bundles Python applications. [ONLINE] Available at: https://www.pyinstaller.org/. [Accessed 08 April 2020].

3 - docs.aws.amazon.com. 2020. No page title. [ONLINE] Available at: https://docs.aws.amazon.com/. [Accessed 08 April 2020].

4 - 3.8.2 Documentation. 2020. 3.8.2 Documentation. [ONLINE] Available at: https://docs.python.org/3/. [Accessed 08 April 2020].

5 - Thread-based parallelism — Python 3.8.2 documentation. [ONLINE] Available at: https://docs.python.org/3/library/threading.html. [Accessed 08 April 2020].

6 - Graphical User Interfaces with Tk — Python 3.8.2 documentation. [ONLINE] Available at: https://docs.python.org/3/library/tk.html. [Accessed 08 April 2020].

7 - Overview — Matplotlib 3.2.1 documentation. [ONLINE] Available at: https://matplotlib.org/3.2.1/contents.html. [Accessed 08 April 2020].

8 - MySQL :: MySQL Connector/Python Developer Guide. [ONLINE] Available at: https://dev.mysql.com/doc/connector-python/en/. [Accessed 08 April 2020].

9 - GitHub - ageitgey/face_recognition: The world's simplest facial recognition api for Python and the command line. [ONLINE] Available at: https://github.com/ageitgey/face_recognition. [Accessed 08 April 2020].

10 – MIT Training Images – Face Recognition Datavase: Face Recognition Database [ONLINE] Available at http://cbcl.mit.edu/software-datasets/heisele/facerecognition-database.html. [Accessed 09 April 2020].