



Professional Practice in IT Project

Conor Shortt
Github

Blaine Burke
Github

Mark Reilly
Github

April 22, 2020

Contents

1	Introduction	2
2	System Requirements	3
3	Technology Used	4
4	Design Methodology	5
5	Limitations and known bugs	7
6	Features of the Implementation	8
7	Testing Plans	10
8	Recommendations for Future Development	11
8.1	Current State of the Software	11
8.2	Potential Future Uses	11
9	Conclusions	13

1 Introduction

For our 3rd year project for professional practice in IT, we decided to design a full stack web application which uses Facial recognition software[2] to log into a web page which in turn allows the user to search a database to view other users information.

For the Front-end portion we chose to use Flask which is a micro framework written in Python, we chose Flask because it is extremely flexible, relatively easy to learn and use, and allows for more control over what components to use such as the type of database.

For the Back-end portion after a lot of research and testing for different ways to achieve communication between Front and Back end, the decision was made to use MongoDB. During the first few weeks we used Flask-SQLAlchemy but later changed to MongoDB which allowed the database to be accessed remotely by each project member.

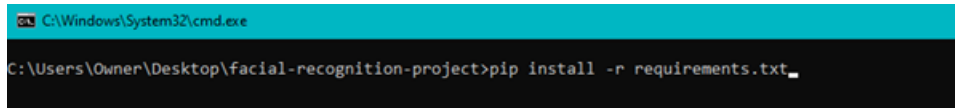
Our main objectives for the project were:

1. Have a working login system using facial recognition
2. Improve communication skills in a project based environment
3. Learning new technologies
4. Learning more about database communication

2 System Requirements

In order to run the project, there are several requirements needed.

- A webcam is required to use the App as when it comes to logging in you will be prompted to scan your face for two factor authentications. You are also required to upload a self-portrait Image of yourself in order to register an account.
- You are required to install the latest version of python 3.[7]
- The project dependencies are required to run our application. To install them follow the steps below!
 1. Clone the project to your desktop.
 2. Open the project and open command prompt
 3. Enter 'pip install -r requirements.txt' to download the dependencies



```
C:\Windows\System32\cmd.exe
C:\Users\Owner\Desktop\facial-recognition-project>pip install -r requirements.txt_
```

- You are required to have [3]Cmake and [4]Dlib installed
- A text/Script editor you have installed. For example Visual Studio Code



3 Technology Used

1. Python[7]
We used python as our programming language as it has wide variety of libraries we could use making it our best option over other languages. Python is also very simple to use , proof being that this is our first python project .
2. Flask [6]
We used Flask as our python framework for our web application. Flask is a small and powerful web framework for Python. It's easy to learn and simple to use, enabling you to build your web app in a short amount of time. Flask can be used for building complex, database-driven websites We chose flask due to its extensive documentation online so if we had issues we could easily find a solution to any problem.
3. MongoDB [1]
MongoDB is an open source database management system (DBMS) that uses a document-oriented database model which supports various forms of Data. This is one reason we chose it as we have several different types of data we needed to add such as numpy and binary arrays. It also has a very flexible data model and is very easy to learn!
4. Webcam
We chose to use a webcam as we required one for facial recognition. A webcam is essential for our application as without a webcam the user cannot login to our App due to our facial recognition two factor authentication. Any webcam should work that is connected to your Computer.
5. PIL [5]
PIL is the Python Imaging Library by Fredrik Lundh and Contributors. We used PIL to convert images to bytes then to base 64 so we can easily store it on our database.
6. Face recognition Module [2]
We used Adam Geitgeys [2] Face recognition Module in our project for our two factor authentication when logging in
7. Flask-Bcrypt
Flask-Bcrypt is a Flask extension that provides bcrypt hashing utilities for your application. We used bcrypt to hash the users password to implement a high level of security.
8. Login Manager
Flask-Login provides user session management for Flask. It handles the common tasks of logging in, logging out, and remembering your users' sessions over extended periods of time.Login manager made the login system run more efficiently.

4 Design Methodology

When starting our project we specified our language we would use, our end goals and what imports we could use.

Setting our End Goals

When we began our project, we, as a team decided on some end goals for our project. We chose that our project would be a web application made using Python that uses facial recognition when logging in, in order to access a database containing all the user's data. At first, we were caught between the Flask and Django framework. Ultimately, we chose flask because it's a small and powerful web framework, it's easy to learn, flexible and simple to use.

Choosing our storage method

After deciding which framework we were going to use, we had to then chose which type of database was best to use with Flask. At the beginning we picked Flask-SQLAlchemy as it work very effectively and easily with Flask. After much consideration we decided that it would be best to change to using MongoDB because it would allow all the data in the database to be accessed remotely by each of us.

Step One - Prepare the 3 main aspects individually

After this we all began to work individually to get the three main aspects of the project functioning. These are the Facial recognition API , the backend server and the frontend built with flask. We spent approximatly 2 to 3 weeks building these individually. Once completed we began to implement them together.

Step Two - Integration

We began by taking the server and integrating it into the flask app. This was due to the fact that implementing the Facial recognition required the login and register to be working before it could be used. Once the backend was implemented we added the facial recognition API as our two factor authentication.

Step Three - Improving efficiency

With the whole app finally together we set out on improving the apps overall efficiency by removing any unnecessary code that was left in. During this process we greatly improved the time it takes to launch the localhost server.

Step Four - Improve accesibility,Ergonomics and clean up code

Now that the bulk of the work was done we had to clean up our code and UI. We began to add more detailed,updated comments to our code to make it more accessible to others. We made our Flask app more ergonomic so anyone using the app for the first time could easily understand how it works! During this process we also improved the overall aesthetics of the Application.

Prof it Plan

1. All work individually to get the 3 main aspects functional

- Face Recognition
- Database + Server
- Frontend / Flask

2. Join them all together

3. Improve ~~prog~~ App to make it more efficient
e.g. remove unnecessary code and imports

4. Improve accessibility of the programme/Application
e.g. easy to navigate and understand

5. Improve aesthetics

Figure 1: Our initial plan

5 Limitations and known bugs

Throughout the course of making out project we encountered many bugs and limitations but we gradually over came them and worked around them.

1. MongoDB

When it came to connecting the MongoDB server, at first it would not connect and would throw errors whenever we tried to call from it. After several YouTube tutorials and after scrolling through several different stack overflow forums we figured it out and implemented the fixes.

2. Dlib

With Dlib we found it tough to get it installed correctly. In the beginning we could not figure out to get it installed properly as it would keep throwing errors every time, we called it. In the end with some research and determination we eventually got Dlib correctly working on our systems

3. Facial Recognition

In regards to the facial recognition side of the project, we originally had live facial recognition partially implemented. The idea was it would draw a box around the face of the user in the login screen, displaying their name at the bottom of the box if they were detected correctly, or "Unknown" if they weren't detected, but the limitation of this was that it was significantly more resource intensive to do live facial recognition. This is why we changed it to a single frame being captured and processed without drawing anything or altering the image taken.

4. Storing Images in the Database

Limitations of the database which we weren't aware of until later on was that we were not able to store images unless they were converted to bytes and then converted to a binary base64 array. We stored two arrays on the database, one for decoding to an image to be displayed on the search page, and one for decoding to numpy array to be used for facial recognition.

5. Capturing images with webcam

A bug that plagued us for quite a long time was that upon capturing and image with the webcam and saving it locally, trying to display that subsequent image straight away resulted in the webpage displaying the wrong image. To solve this we used ajax to update the webpage without reloading and this displayed the correct image.

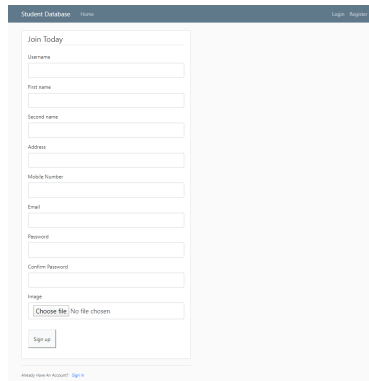
6. Flask

With Flask we found throughout the project when importing different functions from within the project, it could not find the function. This lead to a great deal of time being consumed with fixing the imports and solving the bugs. In the end after lots of testing the importing of various functions was solved.

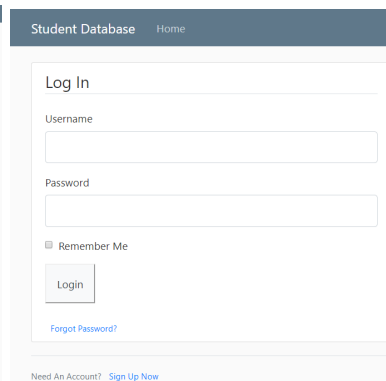
6 Features of the Implementation

There are several features available in our project.

1. A Register system that requires several details such as the username and stores the details to a database. Due to our databases primary key being the username, they must all be different. See image (a) below.

The screenshot shows the 'Register' page of a web application. The header includes 'Student Database' and 'Home' links, with 'Login Register' on the right. The main content area is titled 'Join Today' and contains a registration form with fields for Username, First name, Second name, Address, Mobile Number, Email, Password, Confirm Password, and Image. The Image field has a 'Choose file' button and a 'No file chosen' text. A 'Sign up' button is at the bottom of the form. A footer message says 'Already have an Account? Sign in'.

(a) The Register page

The screenshot shows the 'Login' page of a web application. The header includes 'Student Database' and 'Home' links, with 'Login Register' on the right. The main content area is titled 'Log In' and contains a login form with fields for Username and Password. There is a 'Remember Me' checkbox and a 'Login' button. A 'Forgot Password?' link is below the button. A footer message says 'Need An Account? Sign Up Now'.

(b) The Login page

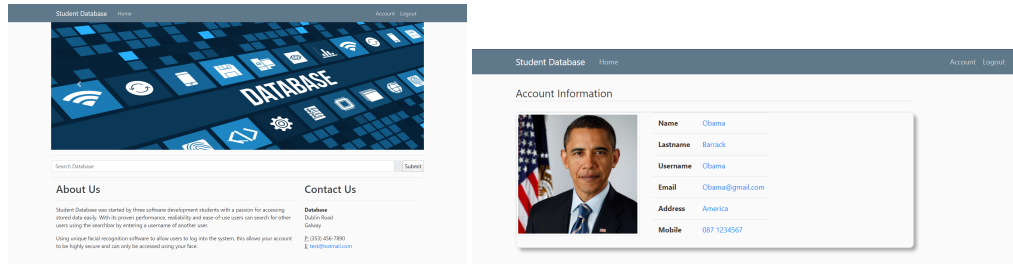
2. A Login system that validates the Users Username and password by checking if they match the records in the database. See image (b) above



(a) The Logout buttons

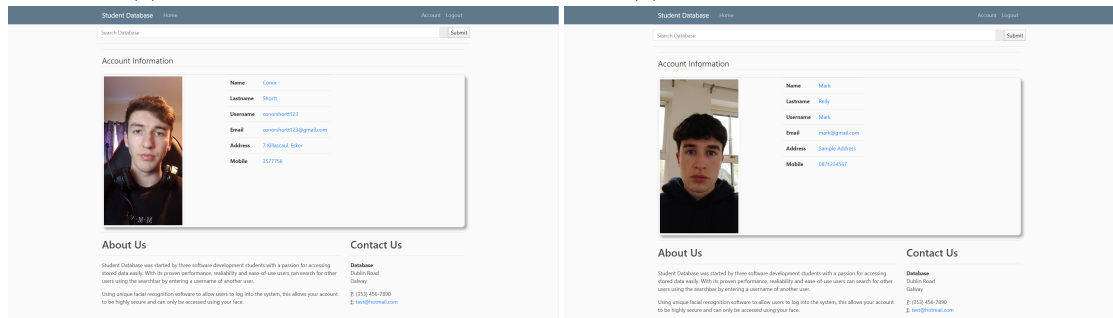
3. A Login / Logout system that allows the current user of a session to logout once logged in to allow another user to log in on the same device! See Figure (a) above.

- A search bar that allows the user to search through the database via username and retrieve the users profile. This contains their profile photo, username and email. See figure (a) below.



(a) The Search page

(b) The Account page



(c) Conors profile when searched

(d) Marks profile when searched

- An account button that allows the current user to view their own profile. See figure (b) above.
- Facial recognition Login which takes the users profile image from registration and compares it to the live image taken when the user enters their correct login details, this adds another layer of security to the application. See Figure (a) below



(a) The Facial recognition page

7 Testing Plans

1. Testing

When it came to testing our App we used mostly white box testing where the tester knows the internals of the code being tested and a small bit of black box testing where the tester knows nothing of the internals of the code.

2. White box Testing

White box testing allowed us to test our code faster and more efficiently but black box testing proved useful as it gave us the opinion of an outside source , helping us get more ideas and opinions and allowed us to see if our app is ergonomic to a wide range of users and easy to navigate through.

3. Black box Testing

When we were black box testing we deliberately attempted to break our app. This is so we could try be prepared for any errors that may occur making our application less likely to crash

4. Testing facial recognition

When it came to testing the facial recognition module we tried different variants of lighting and also tried taking images without any face in it. At first having no face in the image would cause an error to occur but we solved this issue.

5. Testing the Database

With the database we tried putting many different types of data into it to see what we were limited to. We discovered the images were too large for the database when not encoded so when we pushed them to the database we had to encode them and decode them when we pulled them.

6. Testing flask

When testing the flask application , we tried several aspects such as refreshing the page after logging in , entering incorrect details etc. We found a few bugs that would cause the login button to stay appearing when logged in but this issue was solved by setting the sessions state to logged in.

8 Recommendations for Future Development

There are many aspects of this project that could be expanded upon after the duration of the allotted project time. We were limited by time but certainly we could improve many aspects of our software. There were three main areas that we developed as part of our project. I developed the facial recognition scripts and camera, Blaine developed the MongoDB backend aspect[1], and Mark developed the frontend Flask web application.[6]

8.1 Current State of the Software

Currently the software is extremely malleable and not concrete by any means. It could certainly be taken in almost any direction. As of right now it features a database connected to the frontend flask application. With two factor authentication (Enter details and also facial recognition) as part of logging in. This software could potentially be used in several ways as I will highlight below.

8.2 Potential Future Uses

Student Database

We discussed many ideas and plans of what we were going to aim for when making the software, we settled on a database where we could search for a student username and it would get their details from the database and return them. The idea is that this is tailored towards teachers where they log in with two factored authentication using their details and their face.

Criminal Records Database

Another direction that this software could easily be brought in is a criminal records database. The user of this software would be a police officer or a prison warden perhaps. They log in with their details and face and then can subsequently scan a persons face who they have possibly stopped on the road, if the persons face is registered in the database it will pull down their details.

Database for Employees

The database could also be geared for industrial use. If an employer needed to keep track of where every employee was in the company site for example they could set up a camera system, connected to a computer which took frames from the cameras and compared faces in each camera to the database of employees, if someone not registered as an employee entered the building, an alert of some sort could be triggered.

Home Security

The database could be set up for a household where every person in the house is registered in the database. A camera is mounted at the entrance to the home and whenever someone is detected in the camera a computer connected to the camera performs facial recognition and lets them in depending if their face is registered in the database or not.

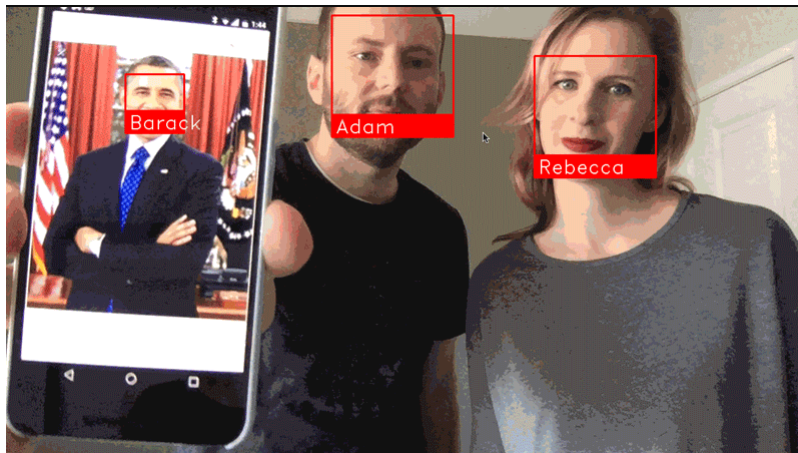


Figure 6: Live facial recognition - Adam Geitgeys
[2]

As explained above there are many options for expanding on this software and it could be taken in several directions.

9 Conclusions

In conclusion we feel we achieved all the main objectives set out at the start of the project. Have a working login system using facial recognition was the main component of the project and this was achieved through a lot of research and testing, we also feel our communication skills within a project environment have improved immensely, which will benefit us greatly in the future. Having been allowed to explore different technologies from the very start of the project we feel it has given each of us, a taste of different technologies we would have never come across, if there was a set technology to be used. **Some of the different learning's/findings are highlighted below:**

- Working as a team can be extremely difficult, but throughout the duration of this project we feel we all gained a lot of experience and skills necessary for collaboration.
- How to effectively use collaborative software such as Github, this was extremely useful for keeping everyone updated and working on the same code.
- How to distribute the work load of our project evenly among our group to ensure fairness and that the work load is manageable.
- We each learned significant amount about our respective areas, in developing the frontend, backend, and the facial recognition scripts that were written.
- Python was relatively new to all of us when we undertook this project, and we all took the risk in choosing it as the language we would utilize to create our software, but we all learned a lot about the language and how simplistic but extremely effective it is.
- Flask was also completely new to all of us and we had never heard of it before the initial project planning phase. It came down to Flask or Django for our frontend development and then everything else would've been developed from it. We chose Flask as it was more lightweight and intuitive than Django. We were glad we chose Flask in the end because development went smoothly for everyone.

References

- [1] Eliot Horowitz Dwight Merriman and Kevin Ryan. *MongoDB Database*. <https://www.mongodb.com/>.
- [2] Adam Geitgeys. *Facial Recognition Module*. <https://github.com/ageitgey>.
- [3] Bill Hoffman. *CMake*. <https://www.python.org/downloads/>.
- [4] Davis E. King. *Dlib*. <http://dlib.net/>.
- [5] Fredrik Lundh. *Python Imaging Library*. <https://pillow.readthedocs.io/en/stable/>.
- [6] Armin Ronacher. *Flask Framework*. <https://flask.palletsprojects.com/en/1.1.x/>.
- [7] Guido Van Rossum. *Python Language*. <https://www.python.org/downloads/>.