GONZAGA UNIVERSITY
School of Engineering and Applied Science
Center for Engineering Design and Entrepreneurship

FINAL PROJECT REPORT
May 9 2018

Facial Recognition Log-in API

Prepared by:

[illegible]

Anton Sebastian Vargas

[illegible]

Elijah Michaelson

[illegible]

Brian Mackessy

1 Project Overview

Over the past decade, many major companies of have been hit with massive hacks that have resulted in hundreds of millions of people's private information to be stolen. The old method of username and password are far too susceptible to hacks, and just simply aren't good enough to keep people's private data secure. Vulnerabilities, such as the reuse of passwords or the use of weak passwords, are regularly exploited by malicious agents and it is hard to enforce policies on users that protect them. Another problem with passwords is that if you are smart and have many different passwords, it can be almost impossible to keep track of all of them. Because of these problems with usernames and passwords we think that the only way forward is biometric security. What we are proposing to use your face as your password. Advances in machine learning have made it possible for computers to recognize faces with a high degree of accuracy, opening the door for a secure and easy to use credential policy system.

We have built API for software developers to integrate into their login pages to allow their users to login using their faces together with/instead of their passwords. and automatically access these usernames and passwords when they pass a facial recognition test. Our software is easy to use and free, with the goal of attracting as many everyday users as possible. We also want to make sure that our software can work on any computer that has a camera on it, something that existing software doesn't do.

Our main deliverables consist of the software for the API which includes the facial recognition software as well as the Docker container we are using to deploy our software. This comes with API documentation for our users to easily use our product. Our software can be found at https://github.com/sebvargas/MLFacialRecognition.

Our project features checklist is as follows:

1) Provide an API that web developers can implement into their websites.

   We wanted to develop an API for web developers that they can easily use and implement into their projects. We wanted to ensure it was easy to read and implement.

2) Give access to user webcams.

   Our product had to be able to use the website user's webcam to take their picture for facial recognition.

3) Store webcam images for neural network usage.

   Our software had to be able to access the pictures that the user would take.

4) Provide additional help for users to capture their image

   Our software or API docs would help instruct website users as to how to take their picture

5) Have more than one user to be registered per website

   We wanted our software to not be limited on a local machine and to have multiple users be able to access their accounts through one machine.

6) Identify users with high accuracy

   We did not want our classifier to produce any false positives (which would mean someone would be able to log in when they were not supposed to)

7) Provide visual Indications of software in operation/success

   We wanted to log the user in if our software succeeded and to throw an error message if it did not.

8) Software must not be too slow or obstruct the user's website experience

   We wanted our software to be no slower than 5 seconds longer for someone to log in to their Facebook account and once they log in, they would be able to forget they even used our program.

9) Provide fool detection capabilities.

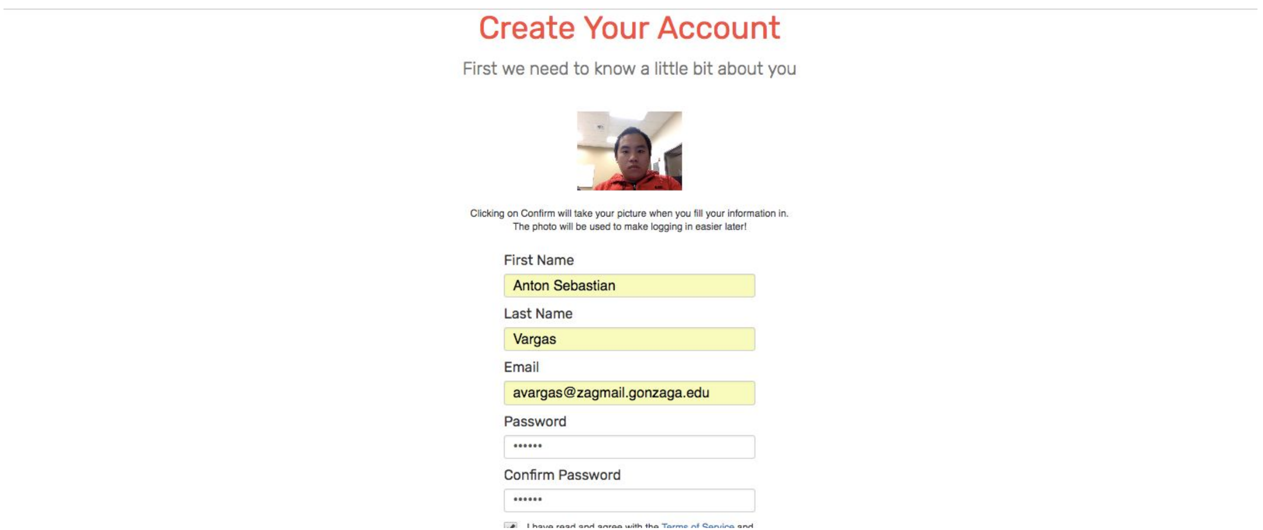
   We wanted our software to be able to distinguish if someone were to hold up a picture of the person they were pretending to be.

2 Work Accomplished

The system works as follows:
Note: When referring to *our users* I mean web developers, and when referring to *their users* I mean the web developers' users.

Our users would download our software in github and follow the install documentation.

Our software is packaged as a Docker container which allows our users to download and install our software without worrying about dependency issues. This also means that our software is compatible with all major operating systems (Mac, Windows, Linux).
A front end that either the user would implement or would use our default components to take a picture of their users and convert that picture into a URI to pass to our backend. If they were registering a user, it would use one image of the user facing forward and if it were logging a user back in, it would use three images: one of a forward facing user, one of their left facial profile and one of their right facial profile. This URI(s) are passed onto our backend.



Create Your Account

First we need to know a little bit about you

Clicking on Confirm will take your picture when you fill your information in.
The photo will be used to make logging in easier later!

First Name: Anton Sebastian

Last Name: Vargas

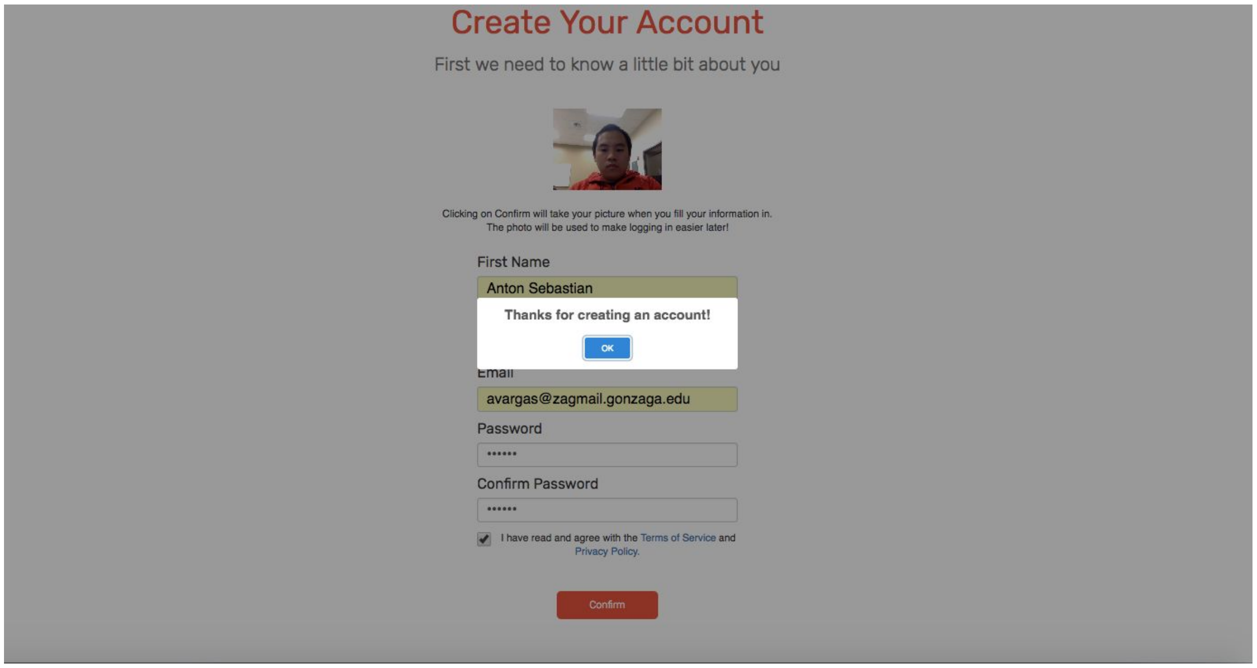
Email: avargas@zagmail.gonzaga.edu

Password: ••••••

Confirm Password: ••••••

✓ I have read and agree with the Terms of Service and

Above is our implementation of our API into a clone of the sparespace senior design team. We also developed a proof of concept website with a simple login/register functionality.



Create Your Account

First we need to know a little bit about you

Clicking on Confirm will take your picture when you fill your information in.
The photo will be used to make logging in easier later!

First Name: Anton Sebastian

Thanks for creating an account!

Email: avargas@zagmail.gonzaga.edu

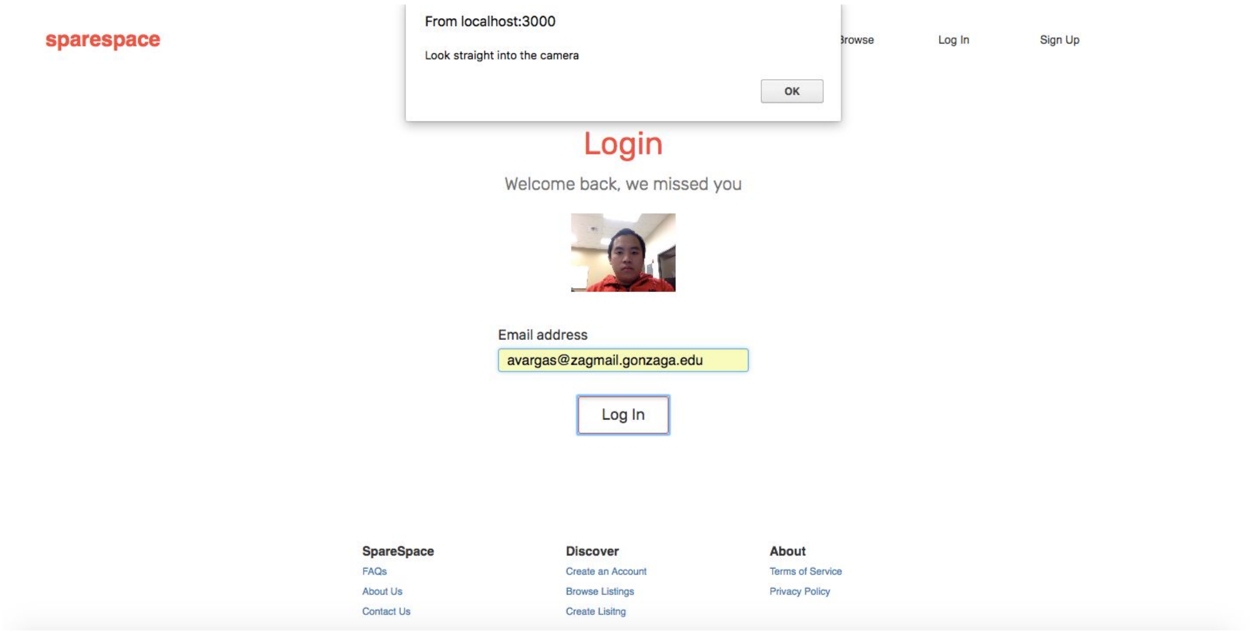
Password: ••••••

Confirm Password: ••••••

I have read and agree with the Terms of Service and Privacy Policy.

This confirms that the user has been registered in both their backend and our facial recognition software.

Then when logging back in, we currently have alerts set up to tell the user what to do



From localhost:3000

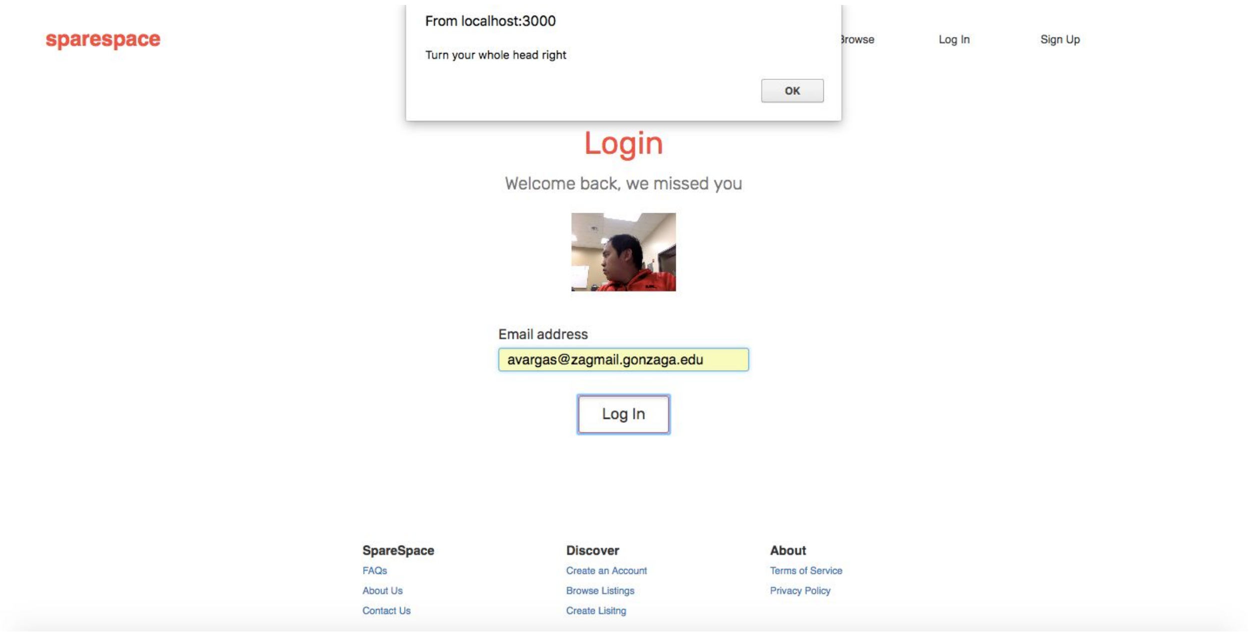
Look straight into the camera

Login

Welcome back, we missed you

Email address: avargas@zagmail.gonzaga.edu

Taking the right and left profiles of the face are used in detecting attempts to fool the system


sparespace

From localhost:3000

Turn your whole head right

Login

Welcome back, we missed you

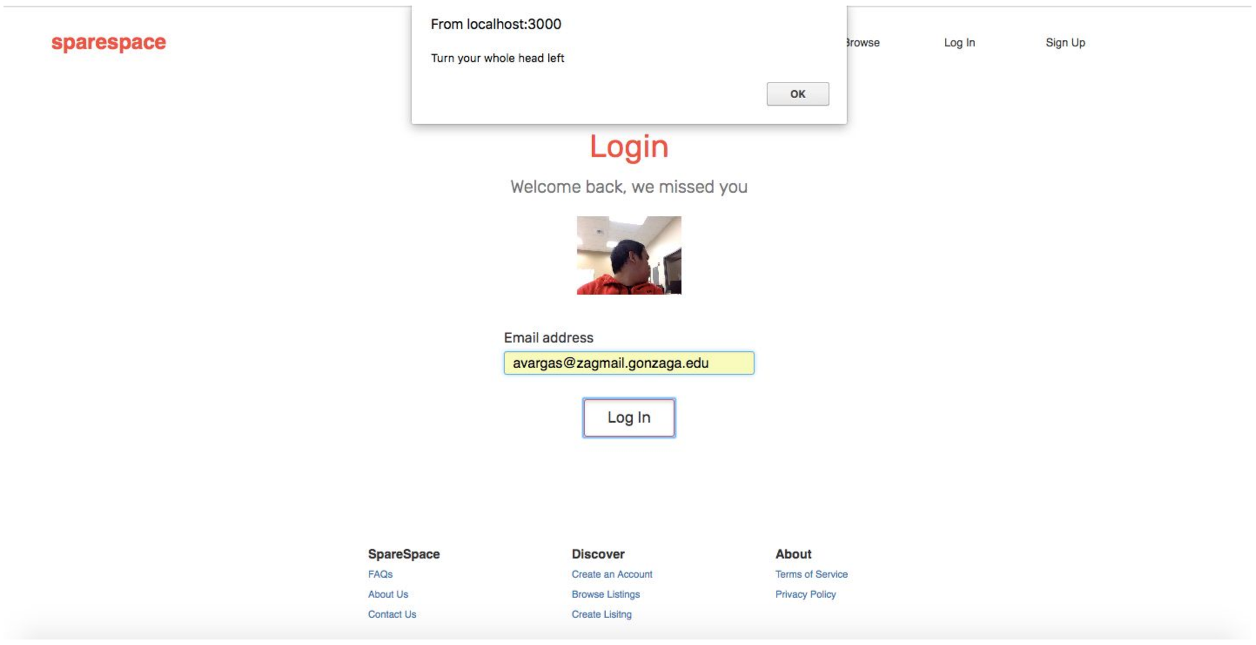
Email address

avargas@zagmail.gonzaga.edu

SpareSpace
FAQs
About Us
Contact Us

Discover
Create an Account
Browse Listings
Create Lisitng

About
Terms of Service
Privacy Policy


sparespace

From localhost:3000

Turn your whole head left

Login

Welcome back, we missed you

Email address

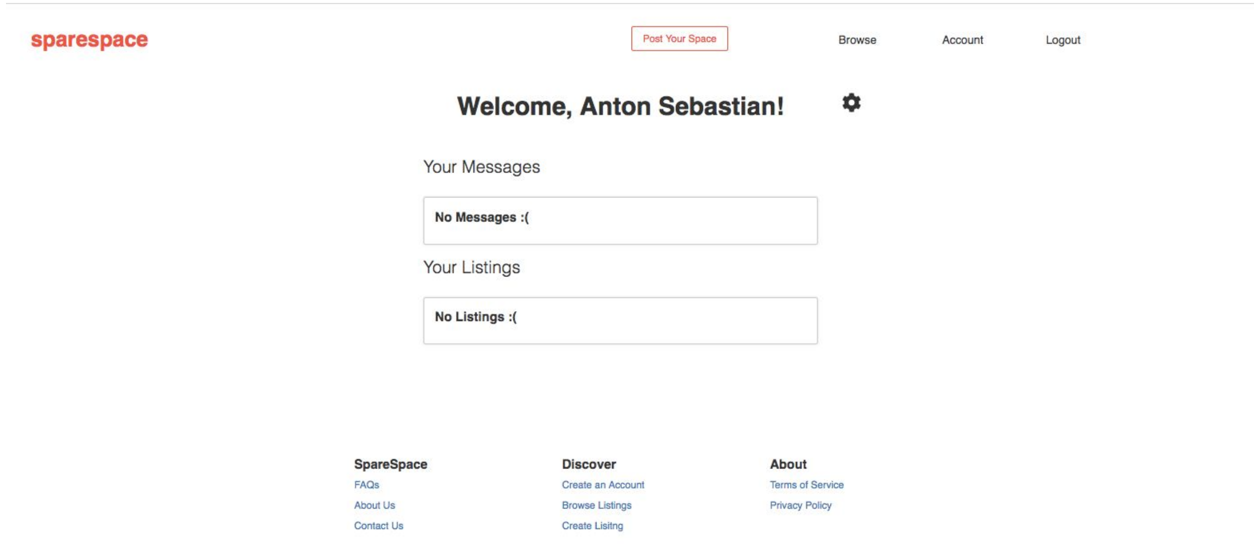
avargas@zagmail.gonzaga.edu

SpareSpace
FAQs
About Us
Contact Us

Discover
Create an Account
Browse Listings
Create Lisitng

About
Terms of Service
Privacy Policy

Then if the recognition is successful, it logs them into their account



Welcome, Anton Sebastian!

Your Messages

No Messages :(

Your Listings

No Listings :(

Our backend communicates with the user's backend to receive the URIs and use those for facial recognition and essentially tells the user's backend if the person registering was successful or if the person logging in was successful. Refer to the **4 System Architecture and Design** section for a more detailed explanation of our backend.

3 Work Remaining

Our software is considered complete as-is. However, as our project has evolved throughout the semester we have identified optimizations that would improve the operation of our application. In particular, we could improve the scalability of our system by only checking the supplied usernames file on sign-in. This means that we would only perform facial recognition comparison on one existing user rather than on all existing users. This has the potential to increase the log-in time of our system and improve the user's experience.

Another potential optimization we could make to our system is to provide more examples of how to implement our API calls into a variety of backends. Currently, we only provide examples for node.js and python backends, but this could be extended to encompass other popular methods in use today. We do not consider this to be additional work necessary for us to consider our project to be competed since our API is relatively simple and intuitive, but regardless it would likely increase the adoption rate of our application if we provided more diverse and detailed examples of our API in real-operating environments.

4 System Architecture and Design

Our system architecture has two major components: the programming interface and facial recognition. The programming interface includes a function for registration and another for login. The programming interface is written in Python. The programming interface handles all communication between the programmers backend and our backend. It will work for most backends, and code examples have been provided for using the programming interface with a React backend. A major library used by the programming interface is Pillow, which allows the system to pass photos between the users and our own backend. The Flask web framework is the library that allows us to call our mostly python code from a Javascript website.

The other major component is the facial recognition portion of the system. The facial recognition algorithm follows a multi-step process, which starts with preprocessing the data with an algorithm knows as a Histogram of Oriented Gradients. This algorithm replaces the individual pixels of the photo with oriented gradients that depict how light is bouncing off the face. This algorithm provides the topography of the face for later classification. The Histogram of Oriented Gradients is called using the OpenCV library, which provides algorithms useful in computer vision. After preprocessing, the image is then passed to a Convolutional Neural Network, which extracts features from the Histogram of Oriented Gradients. The CNN acts as a kernel for later classification by an Support Vector Machine. The CNN is powered the Tensorflow, and extracts 128 features from the image. These features are then passed to an SVM, for one versus one classification. The SVM we use is created by a call to the Sklearn library. By this point the facial recognition software has figured out who the owner of the face is, so the next step is figure out if is really them. The fool detection works by passing the right and left profiles of the user to another CNN. The CNN verifies whether or not the user is who they say are. The principle behind the right and left profiles, is simply to make fooling the software three times as challenging. The CNN used for fool detection also relies upon the Tensorflow library.

Both major portions of the system exists in a Docker container. A Docker container allows the system to achieve platform independence. It does this by running a mini-version of Ubuntu on the programmer computer, but also requires the programmer to have Docker installed.

Figure 4.1 is a diagram of our system architecture.

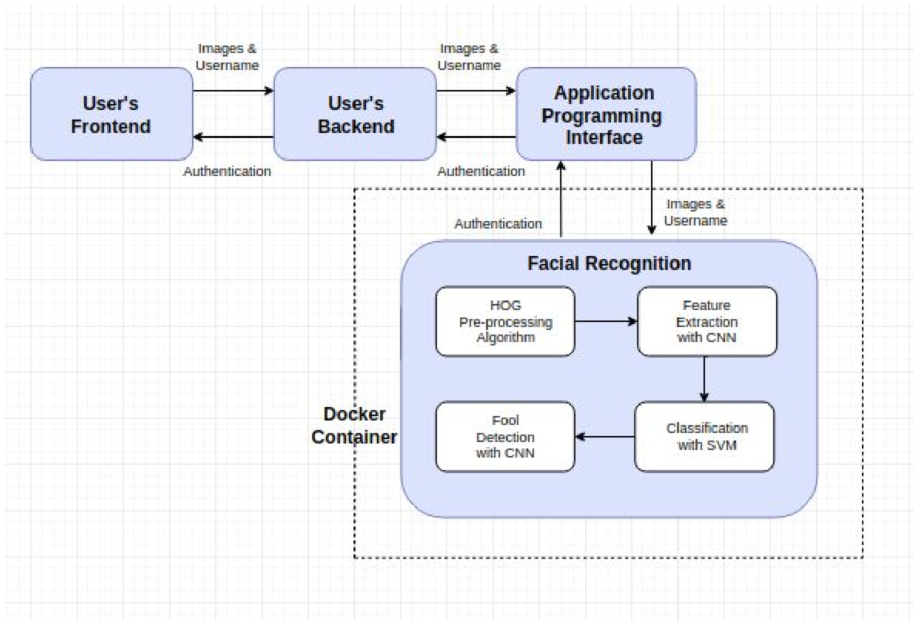


Figure 4.1

5 System Evaluation

Since we expect that our system is to be used by web developers, we have done usability tests on our API documentation and the functions we have written. We sat down with the sparespace senior design group and showed them our documentation and asked if they could follow our API and potentially implement it into their system. We received feedback and adjusted our system accordingly.

Our integration tests consisted of combining our front end portion and back end portions. For the front end, we had to ensure that the system was returning a username along with valid URIs to our backend and for the backend to return whether the person that tried to register or log in to get into our system.

Our unit tests involved ensuring that the functions of our API were taking the correct parameter types and returning consistent/correct values. Since Python is not a strongly typed language, this was especially important.

Our system tests involved getting people to try registering and logging in to our prototype website and ensuring that they were able to log in to the correct account and never into an account that they did not have access to. Throughout all our testing, we surprisingly have never had a false positive (which means a user logging in to an account that they do not have access to) but have had varying levels of false negatives depending on how high or low we set our confidence levels, with higher confidence levels producing more false negatives. We simply tweaked this to where false negatives would only happen if more than one person was in the picture.

6 Project Delivery, Deployment, and Maintenance

The system is being delivered as an open source software on github since we do not have a sponsor. It can be found at this link: https://github.com/sebvargas/MLFacialRecognition
For further maintenance, we have documentation for users to refer to and for us to come back to and improve upon.

7 Project Management

As a team, we worked well in the sense that we got the job done and that we were on the same page for the majority of the project. Our team's ability to estimate was a bit poor at the beginning of the year, but we slowly grew to improve on that and our estimates improved barring some unforeseen setbacks with stages in the project. Our project estimates were 136 hours of work while our actual hours were 210 hours. The project was generally split up as such-

Sebastian: Dockerization, Flask and React frontend, API documentation, Camera Access, Unit Testing, Sparespace Integration.
Accomplishments: Researched the optimal method for deployment (Docker) and how to stand up docker containers for Flask and React based websites to communicate to. Wrote up the API documentation and researched ways for camera access for the user as well as communicating front ends with a MongoDB database. Also worked on standing up the proof of concept site and the Sparespace clone.

Elijah: Dockerization, User Credential management, Implementation of Facial Recognition Component, Sparespace Integration
Accomplishments: Worked on allowing API functionality to be accessed from outside of the Docker environment. Added in additional functionality to the API (such as security levels). Worked on the credential management system and the secure identification of users using facial recognition. Helped implement API into Sparespace. Researched existing solutions to facial recognition problem.

Brian: Backend Development, Fool Detection Functionality, Implementation of Facial Recognition Component, Unit and Usability Testing
Accomplishments: Added a fool detection system to the facial recognition component of the application to increase security. Worked on implementing the facial recognition system and identifying the necessary dependencies. Researched existing solutions to facial recognition problem. Worked on development of the backend infrastructure.