ORTHODOX

Testing Strategy

Killian Connolly - 17303116
William John O'Hanlon - 17477494
Supervisor - Prof Tomas Ward

1. Introduction

Testing is an important part of our project and we utilised some testing software throughout. We emphasised testing on key components that may be vulnerable. We would like to begin by explaining that both of us have been living with each other throughout final year. Thus, peer reviewing pull/push requests was simply done by asking each other to examine one another's code. Furthermore, a substantial amount of pair programming was also completed further emphasizing a peer reviewing environment.

2. Testing

The type that will be covered in this document:

- Flask Unit Testing
- Classifier Testing and Validation, roc curve confusion matrix
- Ad Hoc Testing
- API Testing
- User Performance Evaluation Testing
- Browser Testing

2.1 Git and CI/CD pipeline

Gitlabs built in CI/CD pipeline feature was utilised for testing purposes. Before implementing any component to our project we set up the pipeline through a .gitlab-ci.yml file and a Makefile. The pipeline uses a python 3.8 docker image which is run by the School of Computing.

Originally we had both a build and a test stage in our pipeline, however towards the end of our project we combined both stages due to the docker instance having memory problems as pointed out by Stephan Blott. Since deployment was not complete until towards the end of our project we did not have a continuous deployment element to our pipeline. However in the future we would like to implement this by including a web hook to facilitate continuous deployment. This would also involve implementing versioning control over the application.

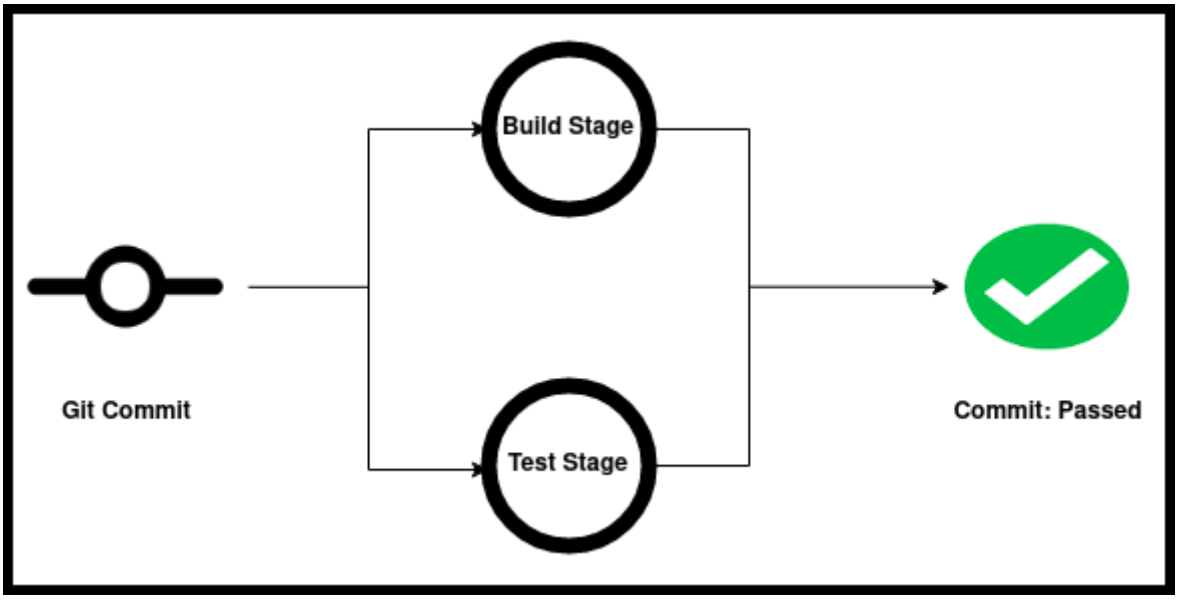


Fig. 2.1 Architecture of CI/CD pipeline on our gitlab repository

Our git branches are related to the components of our project: Frontend, Backend, Pose Estimation, API and testing. Frontend is responsible for frontend development such as the Flask application, HTML and CSS. The Pose Estimation branch manages dataset creation, pose estimation implementations and training our classifiers. The API branch was responsible for the API and database related code. Finally, the testing branch was responsible for writing tests for the Flask App and our python programs.

2.2 Flask Pytest unit testing

Test cases were written using the pytest library to test the web application's endpoints. In order to begin testing, a tests folder was created in the flask app directory. The test files for each user facing endpoint are placed in this directory. A testing configuration program was created called `conftest.py`. This file is called once pytest is run in the flask app directory, and sets up the testing environment. This program creates pytest fixtures, which run before each test function. These fixtures create an app instance which can be used during testing.

Each web page endpoint is tested with HTTP GET, HEAD, POST, PUT and DELETE requests. These unit tests are contained within methods prefixes with `test_` within python programs with the same prefix. In each test case the session 'loggedin' condition must be set to `True` otherwise the requests will be denied and a 302 error will occur. A check is done before this is set to ensure that no user can successfully access any of our endpoints without authorisation.

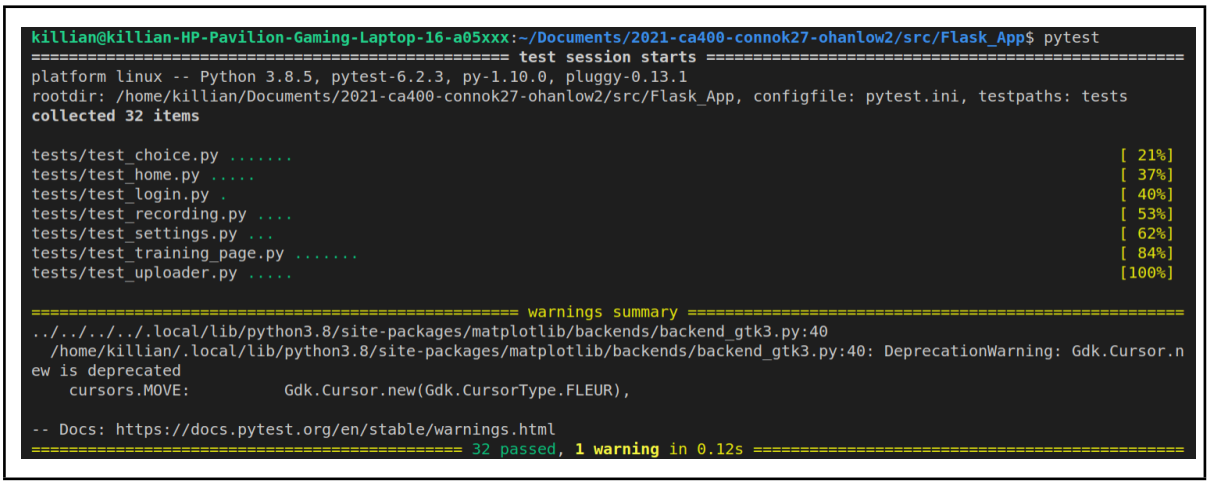


Fig.2.2 Pytest suite results

A vital part of the web application that required testing was the uploading of video files for analysis. We wanted to ensure that no user can upload anything other than an MP4 file.

2.3 Classifier Testing and Validation, roc curve confusion matrix

As mentioned in our technical guide, a benchmark program was used to determine which classifier would give the highest accuracy after training on a particular dataset. The benchmark results can be found within the `src\Model_Results` directory in gitlab repository.

The benchmarking program contains approximately 10 different classification algorithms imported from the scikit-learn python library. The classifiers were trained on each of our datasets separately. K-Nearest Neighbour and Gradient Boost Classifier achieved the highest validation accuracy across our datasets. The following table shows the validation accuracy, F1 scores and the testing accuracy of each of our classifiers. The validation dataset was obtained by splitting the original dataset at a ratio of 80:20 respectively, 20 percent being validation.

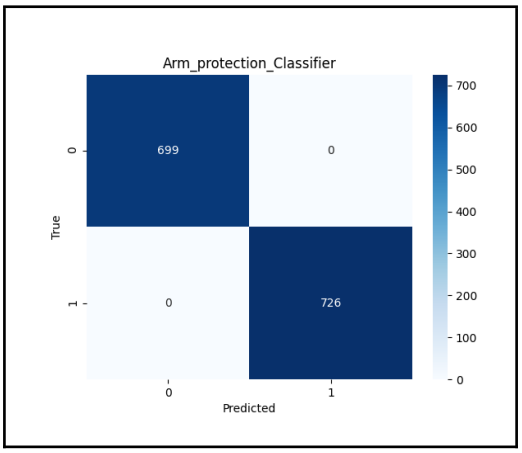
	Classifier	**Validation Accuracy**	**F1 Scores**	**Testing Accuracy**
Jab Elbow	KNN	0.98	0.98	0.6111
Over Commitment	GradientBoosting	1.0	1.0	1.0
Arm Protection	KNN	1.0	1.0	1.0
Chin Protected	KNN	1.0	1.0	1.0
Right Elbow In	KNN	0.99	1.0	1.0

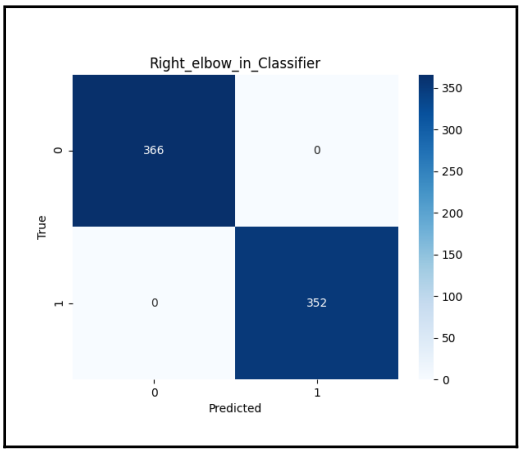
Fig.2.3.1 Table of all classifier and their testing results

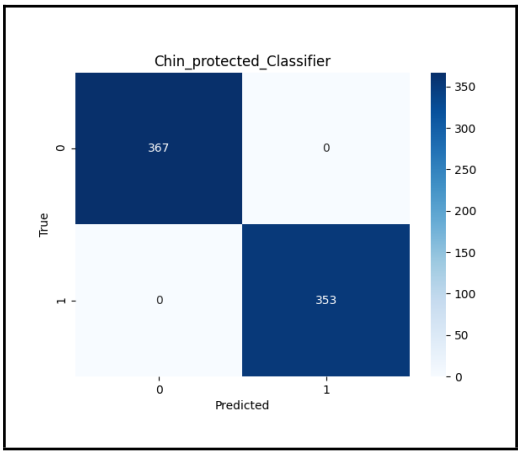
Testing accuracy was deduced from an external dataset, taken on different days in different environments for intersession variability. The testing datasets contain raw videos, which are individually fed to BlazePose for pose estimation, with an array of body landmarks corresponding to each frame being returned. The trained model then produces a prediction for each of the extracted frames from a video sequence. These predictions are added to a dictionary and a simple moving average filter is applied to the dictionary to get the overall prediction i.e. median prediction in the dictionary taken as prediction. An example of the output from this process is shown below.

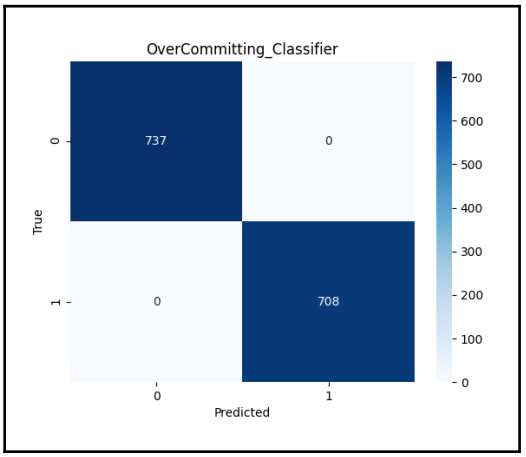
```
==========
Training :  QDA_right_elbow_in_02_05_21.pkl
==========
Model achieved the following validation score in training :  0.9972144846796658
{'0': 108, '1': 7}
{'0': 84, '1': 32}
{'0': 88, '1': 28}
{'0': 0, '1': 115}
{'0': 0, '1': 118}
{'0': 15, '1': 101}
Number correct in validation : 6, out of 6
Testing accuracy : 100.00
              precision    recall  f1-score   support

     Correct       0.99      1.00      1.00       357
   Incorrect       1.00      0.99      1.00       361

    accuracy                           1.00       718
   macro avg       1.00      1.00      1.00       718
weighted avg       1.00      1.00      1.00       718
```

Fig.2.3.2 File showing testing of Right Elbow Protection classifier model.









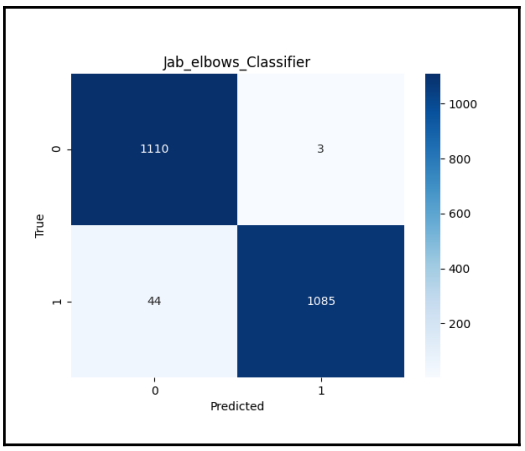
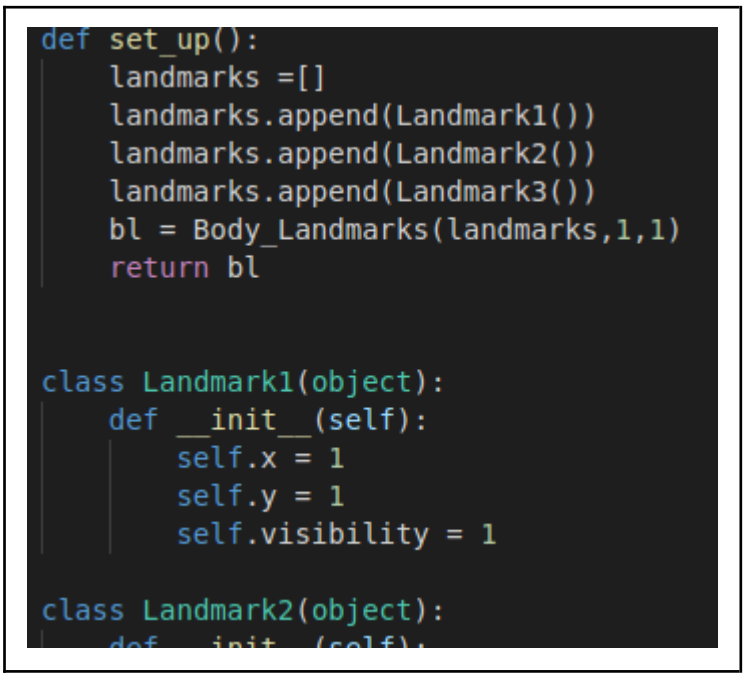


Fig. 2.3.3 Confusion matrix for all five binary classifiers showing True Positives, True Negatives, False Positives and False Negatives

2.4 Ad hoc Testing

BodyLandmark functions that contain various mathematical functions such as calculating the angle between landmarks, distances and slopes according to their respective X and Y coordinates. These tests are contained in the test directory within the Pose_Estimation directory. These functions require sample Landmark objects which are setup towards the bottom of the program and are initialised within the setup() function. Within our main implementation the BodyLandmark object comprises 33 Landmark objects however for testing only three landmark objects are created. Each function is tested two or more times evaluating on different scenarios, for example, testing an acute angle and right angle.



```
def set_up():
    landmarks =[]
    landmarks.append(Landmark1())
    landmarks.append(Landmark2())
    landmarks.append(Landmark3())
    bl = Body_Landmarks(landmarks,1,1)
    return bl


class Landmark1(object):
    def __init__(self):
        self.x = 1
        self.y = 1
        self.visibility = 1

class Landmark2(object):
```

Fig 2.4.1 Initial setup for each function

`generate_feedback_report` function takes the predictions given by our models once a frame collection is provided and generates a feedback report. This function is vital as it is the middleware between the model output and user by translating the results into a human readable report. Each classifier result consists of a binary dictionary. For testing these dictionaries are created with random values and compared with the correct output.

Both of the Ad Hoc test files are contained in the same directory and pytest is run against them.

```
killian@killian-HP-Pavilion-Gaming-Laptop-16-a05xxx:~/Documents/2021-ca400-connok27-ohanlow2/src/Pose_Estimation/test$ pytest
============================= test session starts ==============================
platform linux -- Python 3.8.5, pytest-6.2.3, py-1.10.0, pluggy-0.13.1
rootdir: /home/killian/Documents/2021-ca400-connok27-ohanlow2/src/Pose_Estimation/test
collected 17 items

test_body_land.py .............                                          [ 82%]
test_generate_feedback.py ...                                            [100%]

============================= warnings summary =================================
../../../../../.local/lib/python3.8/site-packages/matplotlib/backends/backend_gtk3.py:40
  /home/killian/.local/lib/python3.8/site-packages/matplotlib/backends/backend_gtk3.py:40: DeprecationWarning: Gdk.Cursor.new is deprecated
    cursors.MOVE:          Gdk.Cursor.new(Gdk.CursorType.FLEUR),

-- Docs: https://docs.pytest.org/en/stable/warnings.html
======================== 17 passed, 1 warning in 0.50s =========================
```

Fig 2.4.2 Pytest of Ad Hoc Functions

The deprecation warning can be ignored.

2.5 Testing API

Testing will be done via Postman, which is a simple API client that allows developers to test and develop APIs. For testing purposes, we created empty tables and commenced with testing seeding data such as creating users and feedback reports.

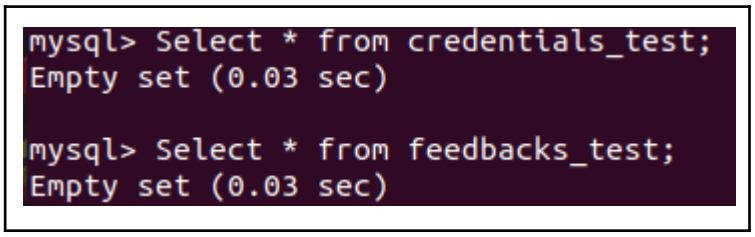


Fig 2.5.1 Empty Tables

The following will show images of the request being sent using Postman and the result of the database after the request was sent via command line interface.

How to read Postman images:

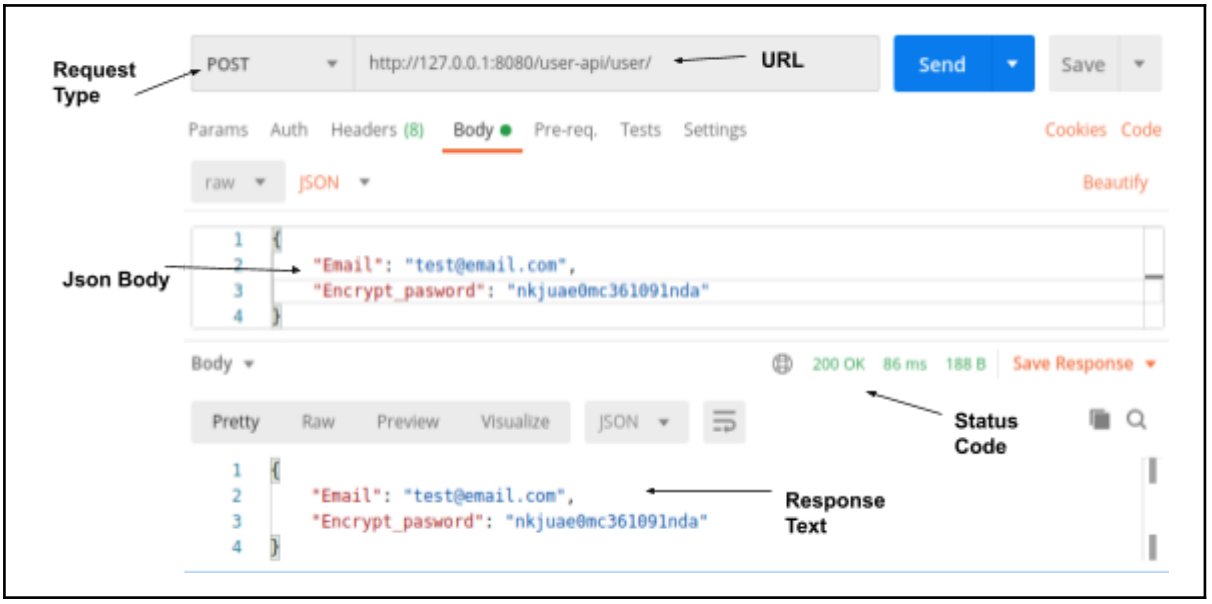


Fig 2.5.2 Components of Postman

Testing insertion of user credentials into database

Fig 2.5.3 POST Requests to Create a User

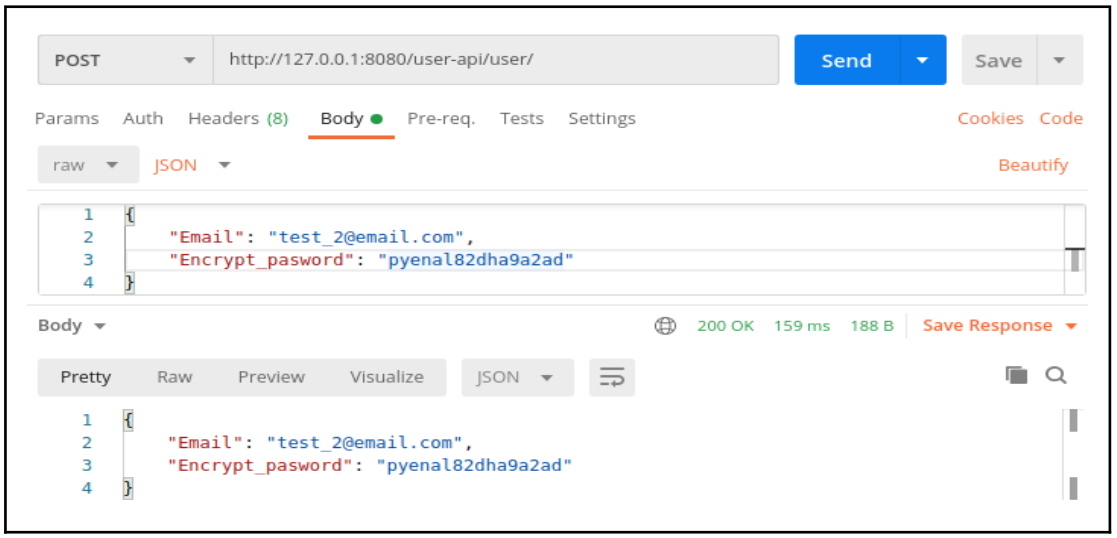


Fig 2.5.4 Another POST Requests to Create a User

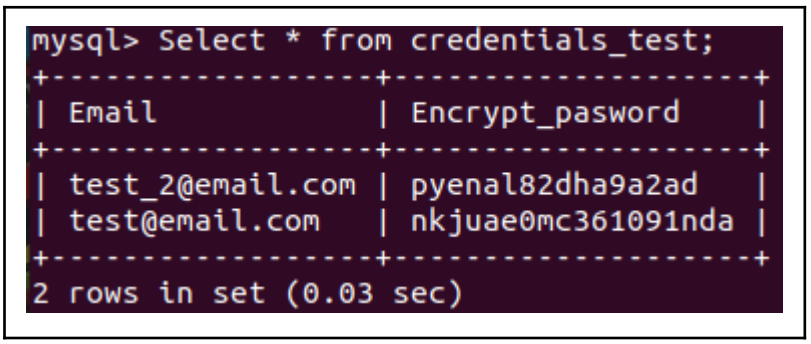


Fig 2.5.5 Results after Create User Requests

Testing GET request for obtaining all Users

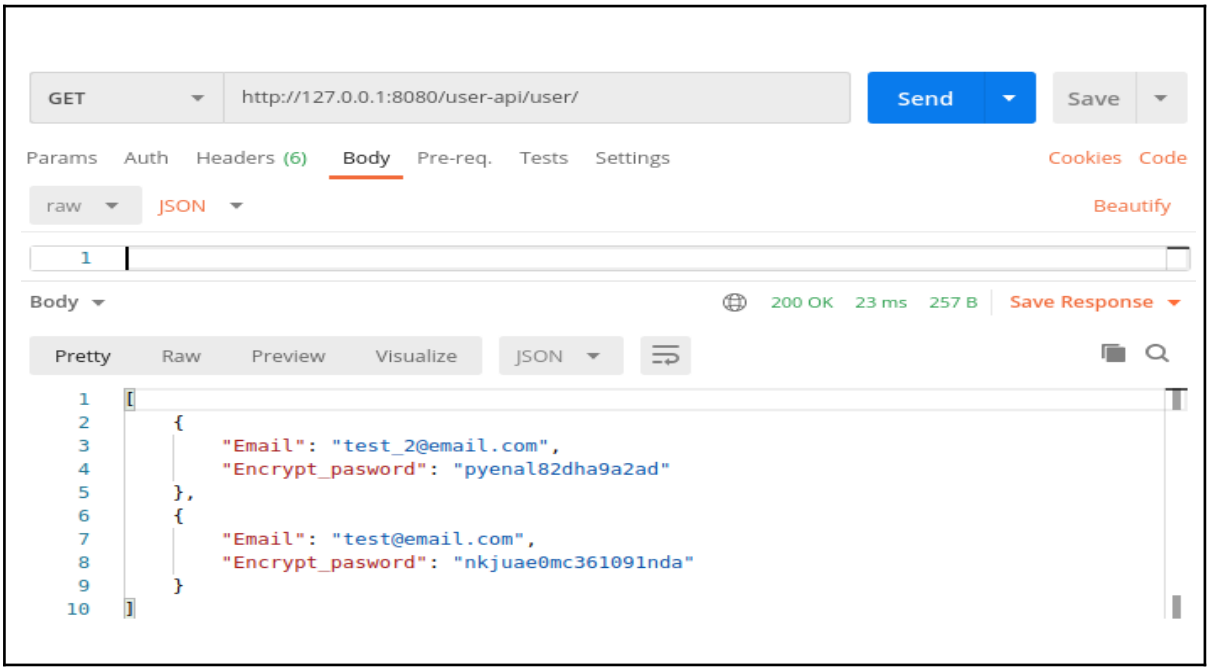


Fig 2.5.6 Results after using GET request showing all Users

Testing GET user by email address via URL parameter

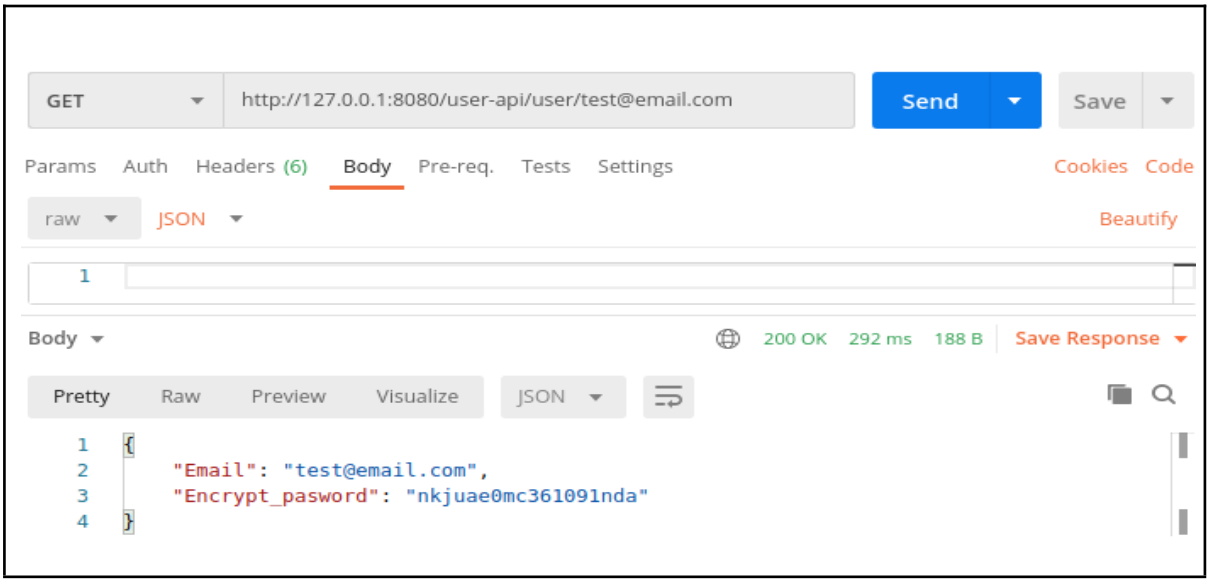


Fig 2.5.7 Results showing 200, meaning that the User in URL parameter exists

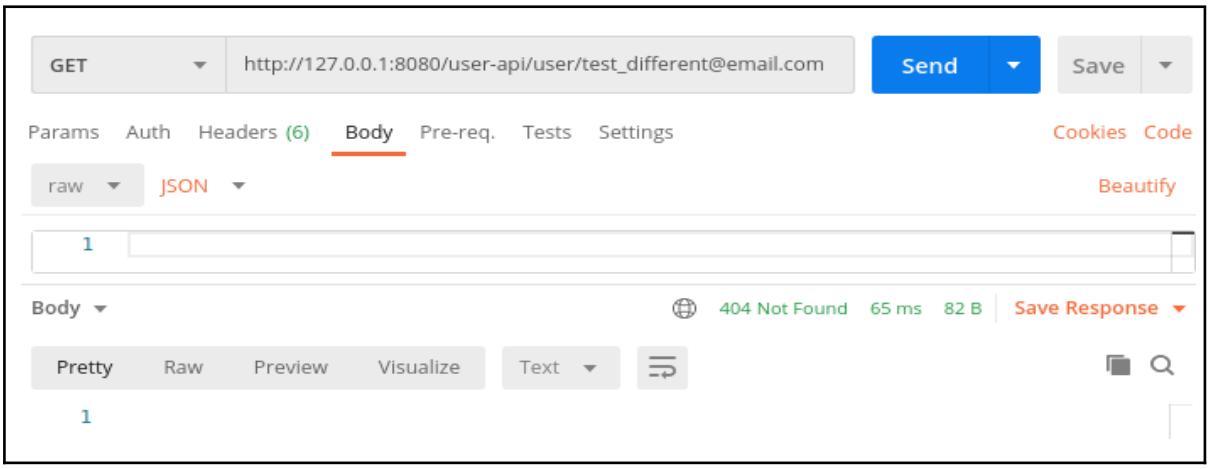


Fig 2.5.8 Results showing 404, meaning that the User in URL parameter does not exist

Testing PUT command for changing User Credentials

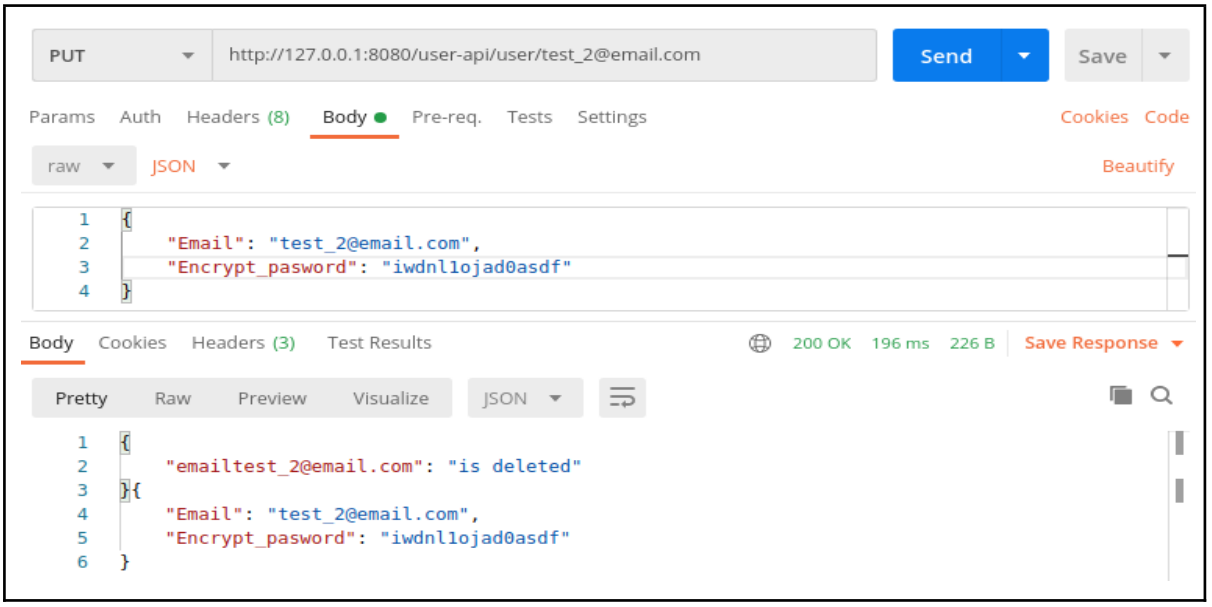


Fig 2.5.9 200 Showing that the Users credentials have successfully been updated

Fig 2.5.10 Database successfully holds the changes

Testing DELETE request for deleting Users Credentials

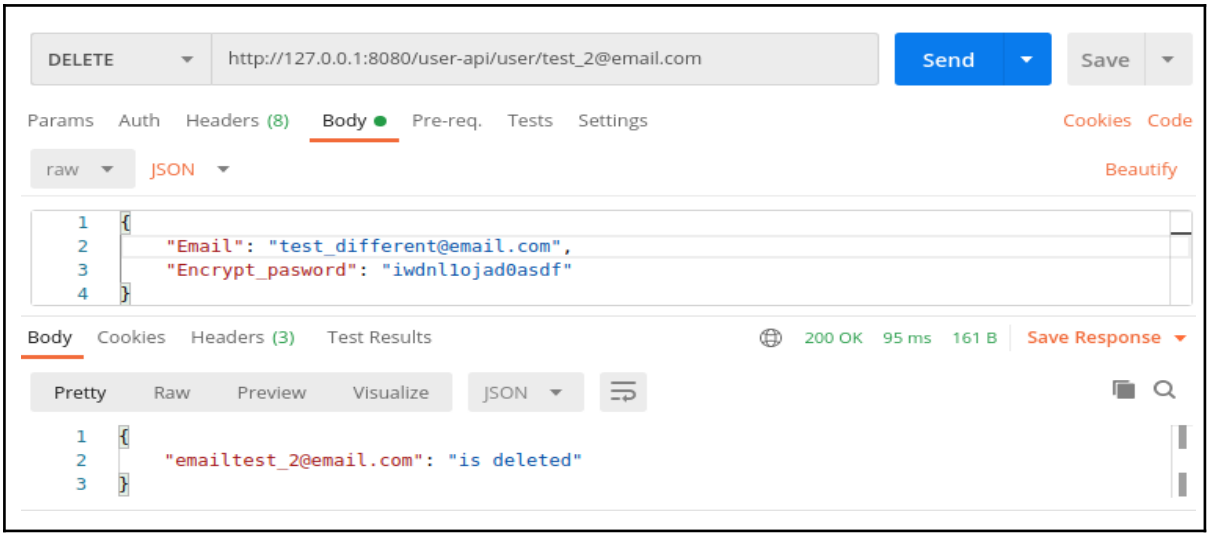


Fig 2.5.11 200 Showing that the Users credentials have successfully been Deleted

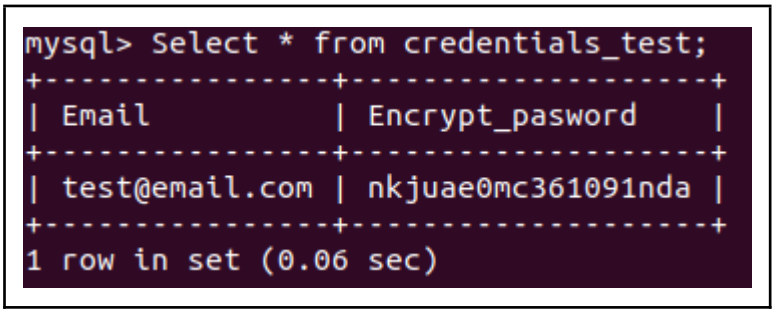


Fig 2.5.12 Database after DELETE request

Testing POST feedback report

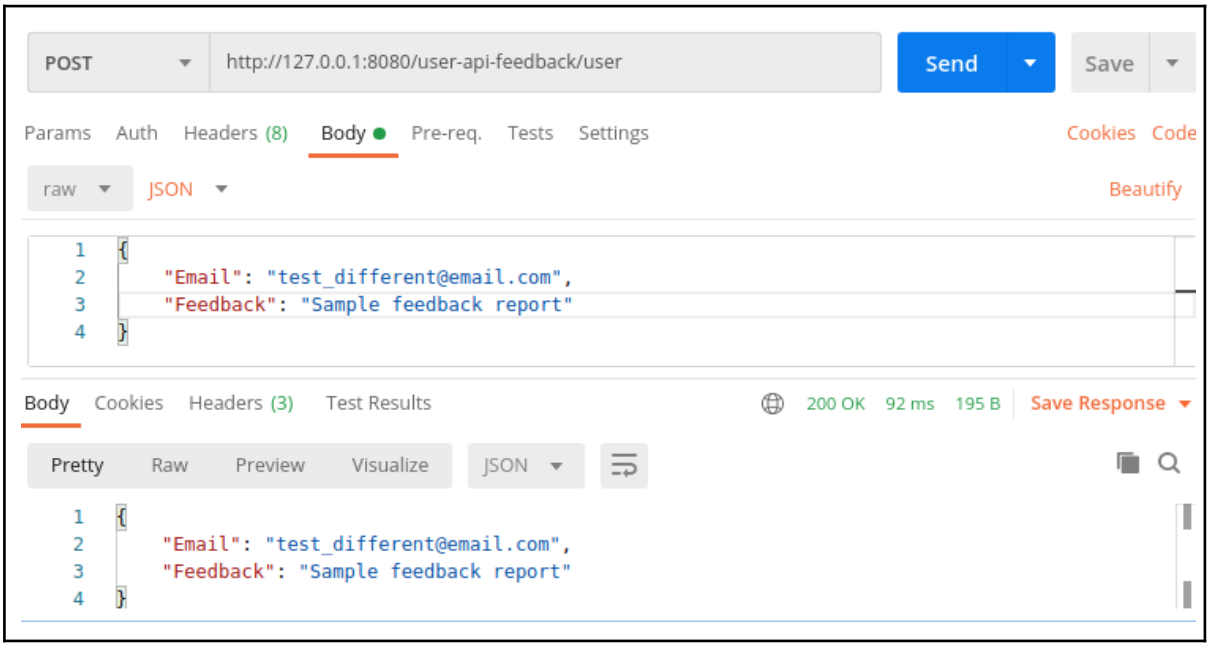


Fig 2.5.13 200 Showing that the Feedback Report has successfully been made

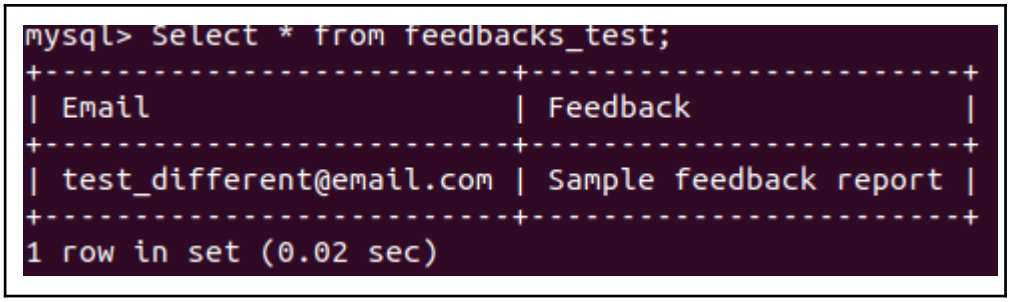


Fig 2.5.14 Feedback Database after POST request

Testing GET request returning all Feedback Reports given an Email address

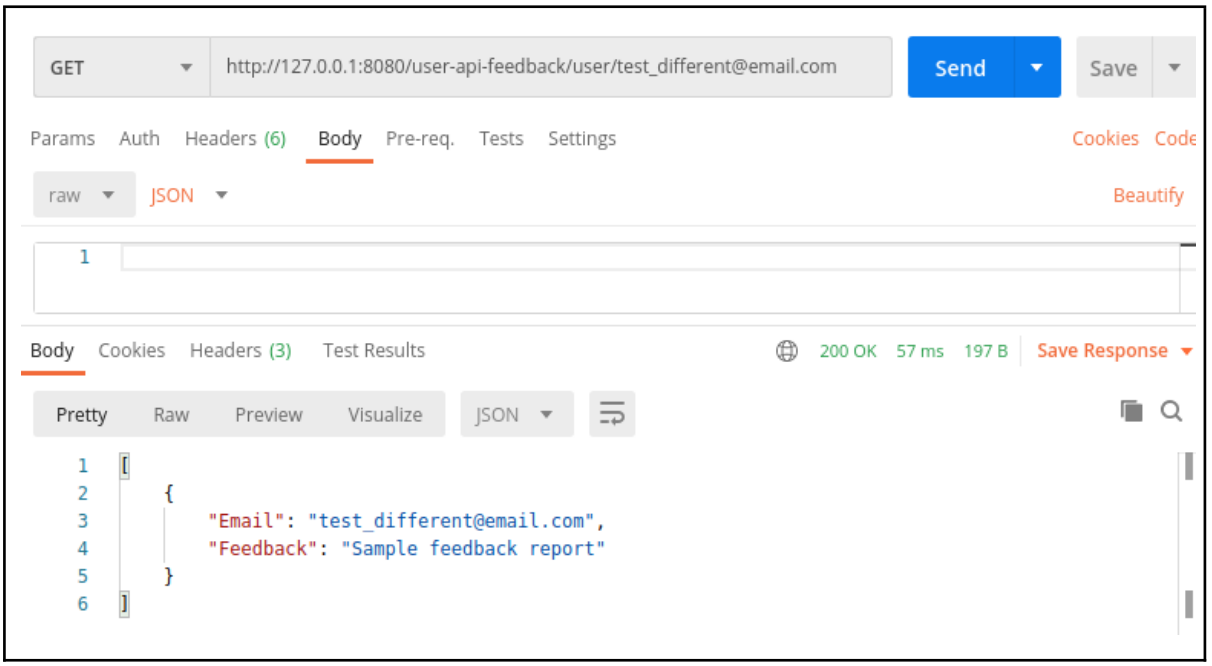


Fig 2.5.15 Returned text showing all Reports given email in URL parameters

2.6 User Performance Evaluation Testing

I think we have system testing to an extent, cause we have flask http testing, we have api testing and we have model testing. System testing i think is to have them one after the other.

Since there are 5 binary classifiers that the video is run on, this means that there are 32 possibile results that the user can get. The following are a few examples of a user testing our model via our web application. We will show images from within the video where the user is performing the technique incorrectly(if they are doing anything incorrectly) along with the produced report.
As with all machine learning models there are some limitations to the accuracy of the models whereby the results produce an output that may be inaccurate.

Fig 2.6.1 Screenshot from the middle of the given video(Not a feature)

Fig 2.6.2 Result given from our Web Application

As you can see from the analysis above three out of five classification came up incorrect. We have highlighted these errors in Fig 2.6.1 , the other two techniques were done correctly, their hand is correctly protecting the face and their chin is protected by their shoulder.

2.7 Browser Testing

There is one major component that we wanted to browser test and that is obtaining live feed from the users device. The overall application should work in the majority of modern browsers, this testing was done whilst testing live feed upload. Since all browsers are unique, the dimensions of the camera had to vary from browser to browser. On top of this mobile browsers were dissimilar to their desktop browser. This leads to us being unable to set universal dimensions for all browsers(Fig 2.7.1). Hence certain browsers will show skewed camera dimensions. This should not affect the accuracy of the model since they are normalised using the centre of the body landmarks.

	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	WebView Android	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet
aspectRatio	59	79	No	No	Yes	?	59	59	29	?	?	7.0

Fig 2.7.1

The following images are browser testing of the live feed video

Microsoft Edge:

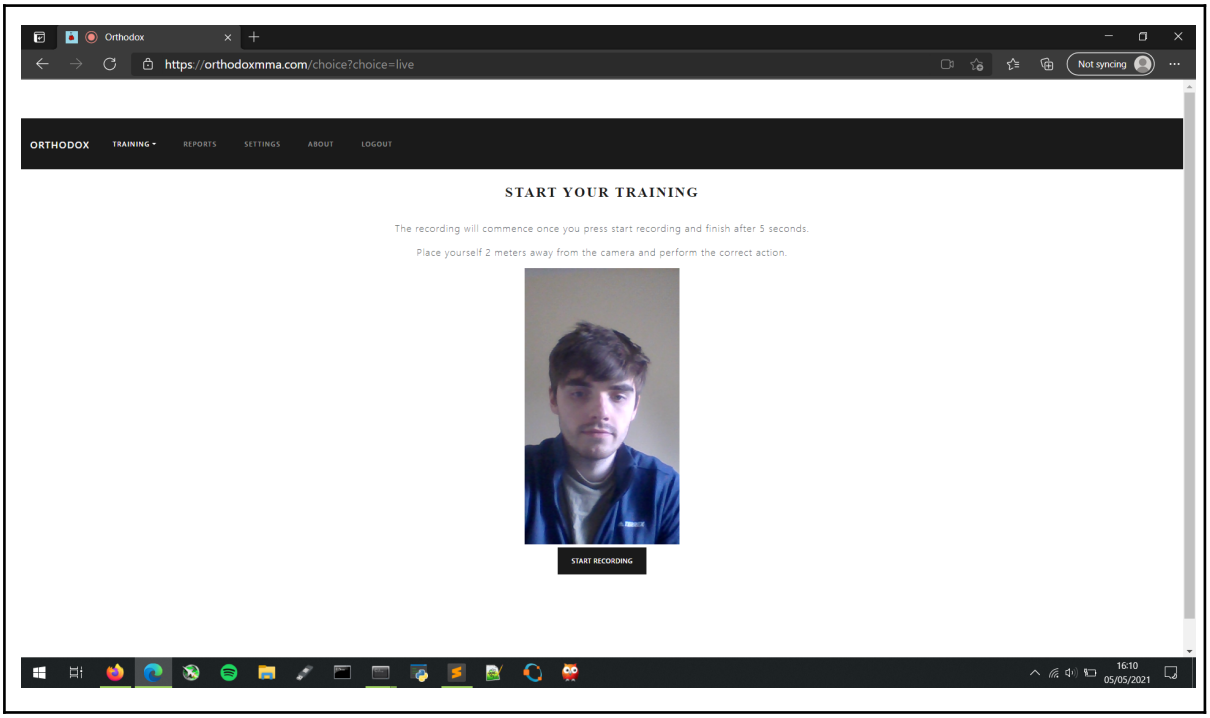


Fig 2.7.2

Google Chrome Desktop:

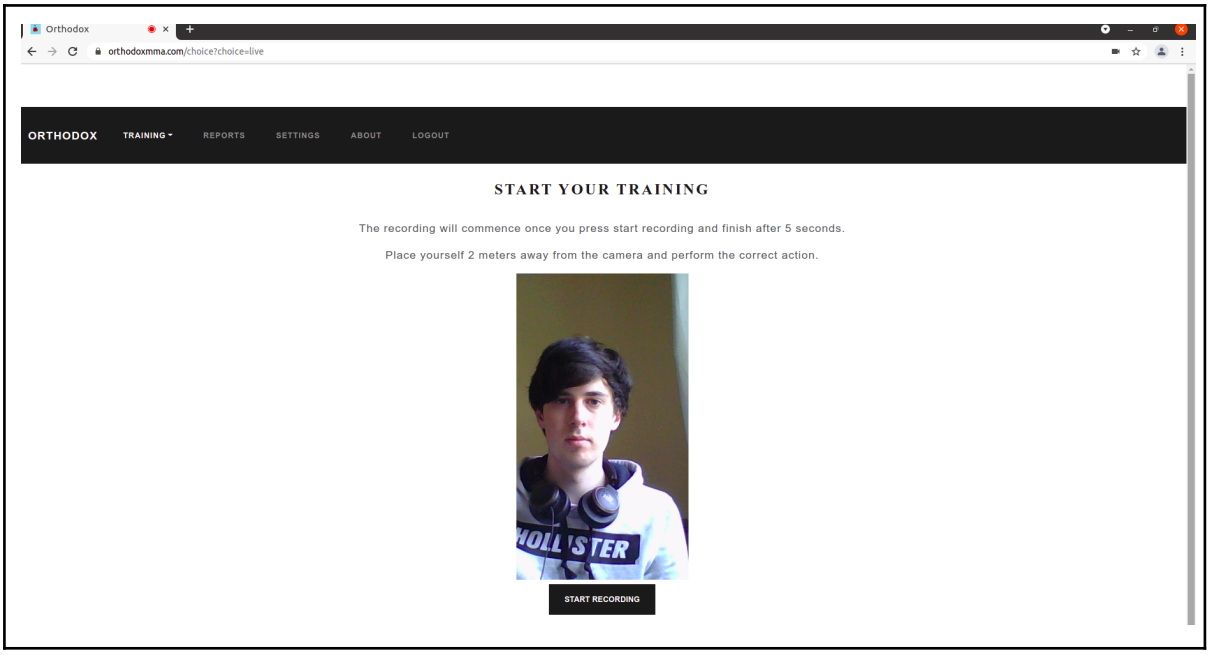


Fig 2.7.3

Chrome Mobile:

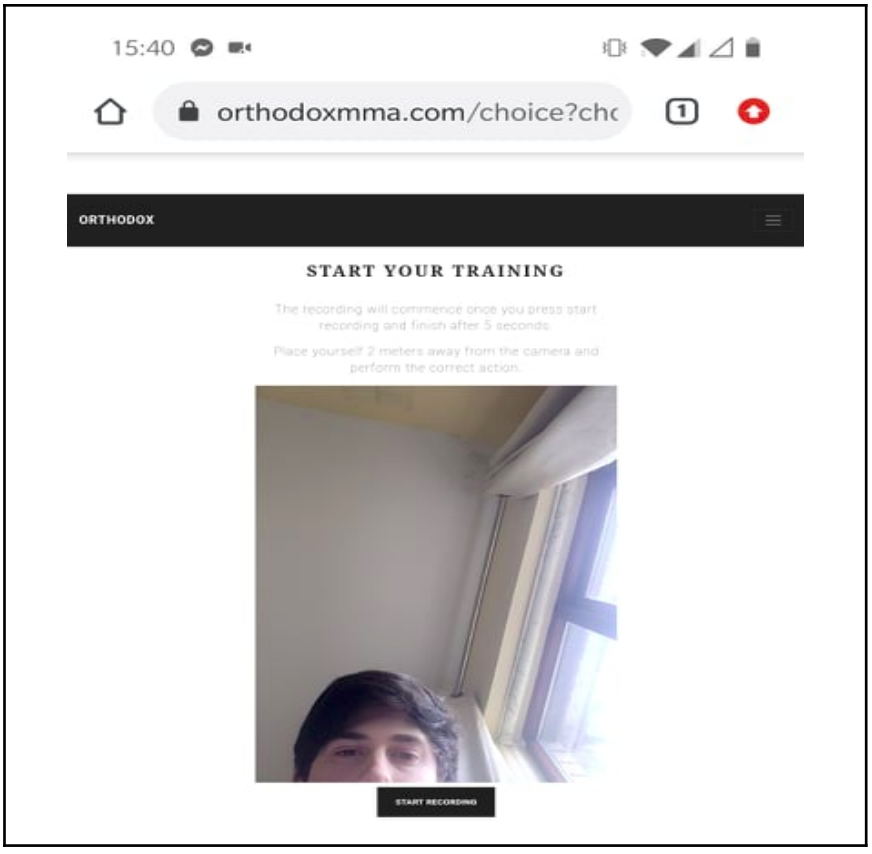


Fig 2.7.4

FireFox:

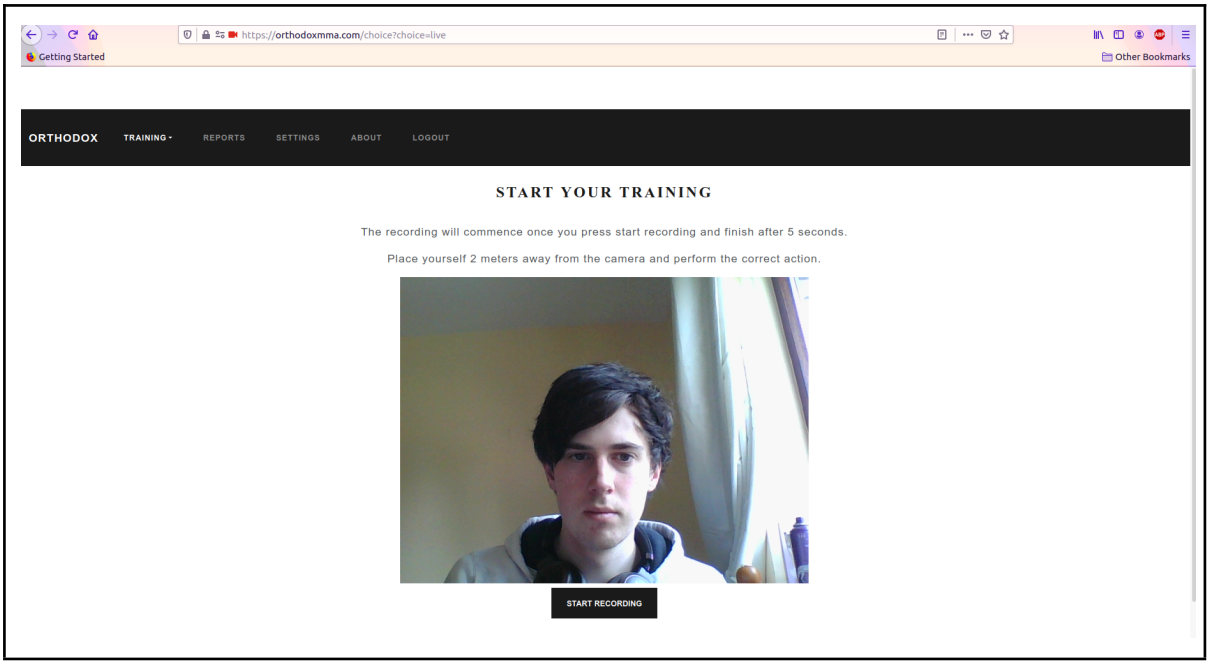


Fig 2.7.5

Safari:

We noticed that the libraries used for obtaining live feed held a question mark over safari on whether it supports it or not(Fig 2.7.1 & Fig 2.7.8). We tested it and found that safari on mobile work however the dimensions are nearly unusable. Safari on desktop showed that MediaRecorder was not supported. We researched this and found that depending on the version that safari is using will determine its usage.

Safari mobile:

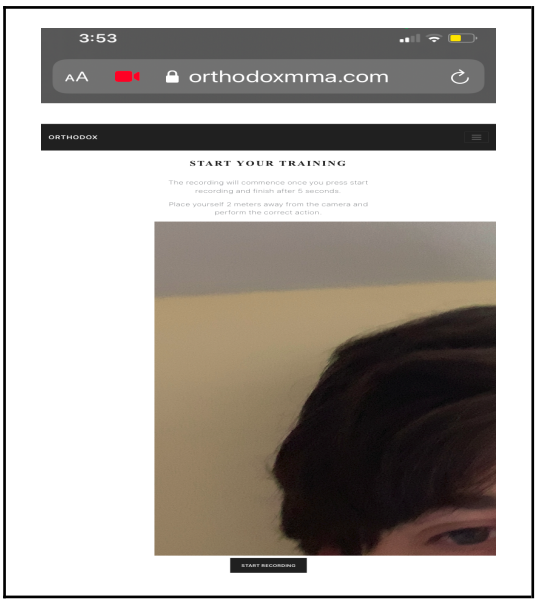


Fig 2.7.6

Safari Desktop:

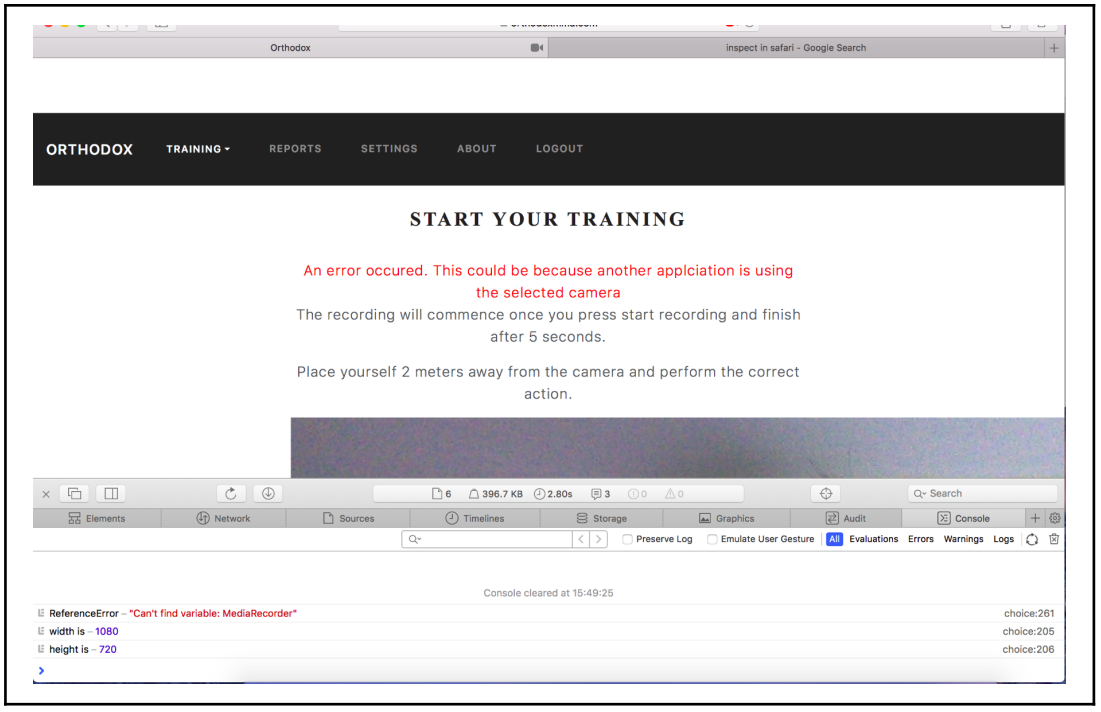
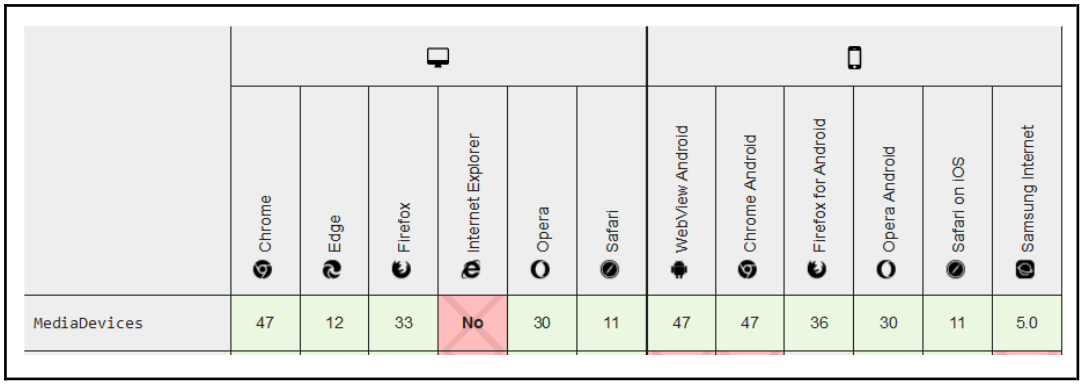


Fig 2.7.7

Internet Explorer:

The MediaDevices API used to facilitate the live feed functionality is not available in Internet Explorer. Therefore, this component cannot function properly in this browser.



	Chrome	Edge	Firefox	Internet Explorer	Opera	Safari	WebView Android	Chrome Android	Firefox for Android	Opera Android	Safari on iOS	Samsung Internet
MediaDevices	47	12	33	No	30	11	47	47	36	30	11	5.0

Fig 2.7.8

3 Conclusion

In conclusion, testing was a consistent part of our project. We tested on a number of different levels across the components of our system. We received positive results when testing our classification algorithms on external testing datasets. Despite our application being unable to run on certain browsers because of Javascript library being unsupported, namely Internet explorer, we believe our level of frontend testing was more than sufficient.