

## **ECE 1779: Project 1**

Group Members:

Corey Kirschbaum, #: 997384165

Mohammed Baquir, #:1007489816

Sidharth Warriar, #:1006508940

Date: October 18, 2020

## Contents

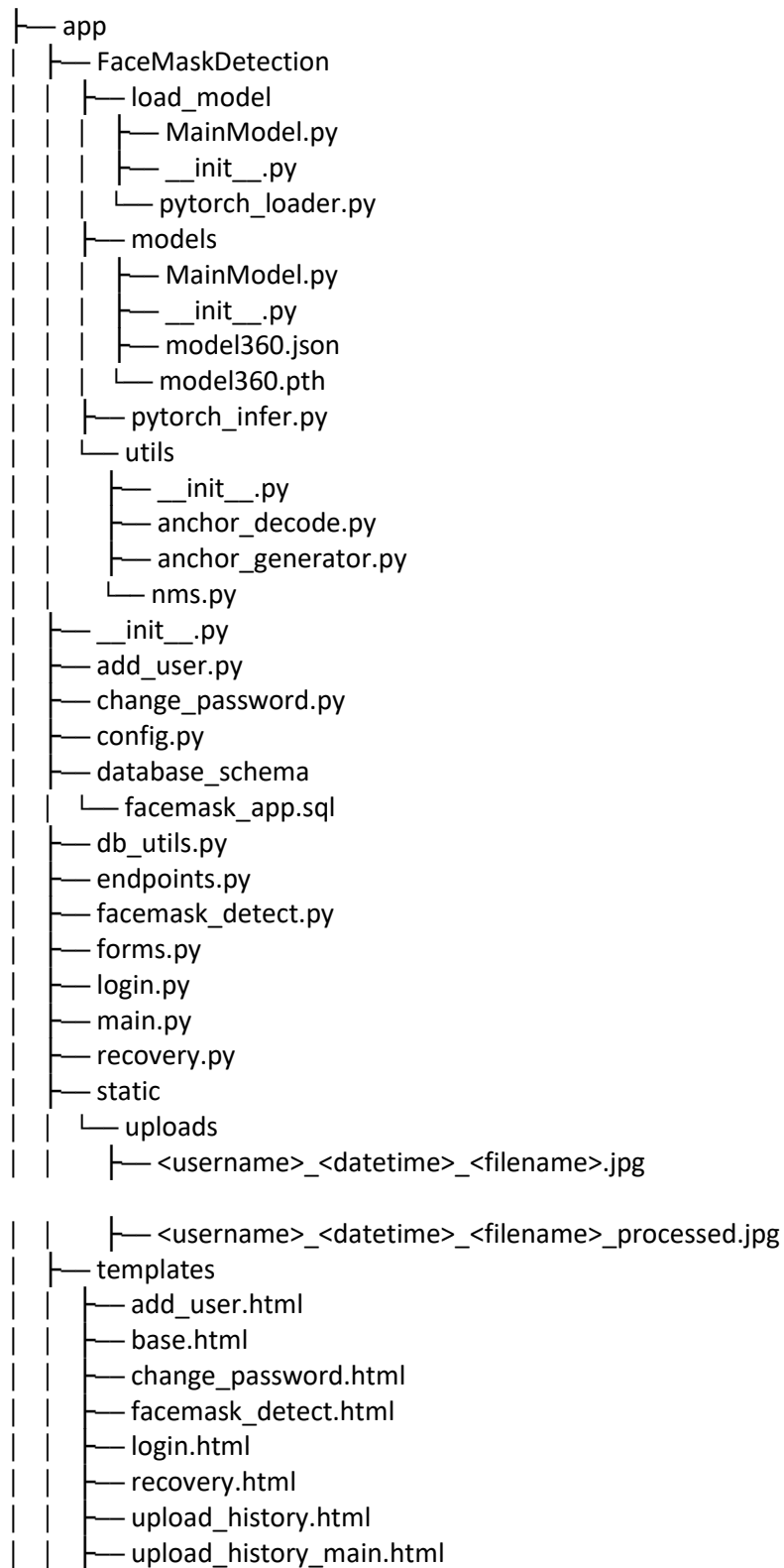
ECE 1779: Project 1	1
Table of Figures	3
1. Project Architecture	4
1.1. File Directory Layout	4
1.1.1. Flask Web Application	5
1.1.2. Face Mask Detect Integration	5
1.1.3. /app Web Application Files	5
1.1.4. Login - / or /login	6
1.1.5. Password Recovery - / recovery	7
1.1.1. Logout - /logout	9
1.1.2. Change Password - /change_password	10
1.1.3. Add/Delete User -> /add_user	11
1.1.4. Face Detect Upload - /facemask_detect	12
1.1.5. Upload History - /upload_history	15
2. Database Design	16
3. Testing Endpoints	18
3.1. Register - /api/register	18
3.2. Upload - api/upload	19
4. How to run the app	20
4.1 Amazon EC2 steps	20
4.2 Running the App	21
5. Future Improvements	21
References:	22

## Table of Figures

Figure 1: Login Page	6
Figure 2: Login Page Invalid Credentials	7
Figure 3: Password Recovery Page	7
Figure 4: Password Recovery - Bad Email Format	8
Figure 5: Password Recovery - Invalid Email	8
Figure 6: Logout Result	9
Figure 7: Change Password Page	9
Figure 8: Change Password - Incorrect old Password	10
Figure 9: Add/Delete User Page	10
Figure 10: Add User Page	11
Figure 11: Get for Face Mask Detection	11
Figure 12: Post for Face Mask Detection - After upload	13
Figure 13: Face Mask Detect - No image selected	14
Figure 14: Face Mask Detect - Invalid Image	14
Figure 15: Upload History Main Page	14
Figure 16: Upload History – List Example for All Faces Wearing Mask	15
Figure 17: Database ER Diagram	16
Figure 18: Register Json Success Message	17
Figure 19: Register Json Error Message	18
Figure 20: Upload Json Success Message	18
Figure 21: Upload Json Error Message	19

## 1. Project Architecture

### 1.1. File Directory Layout



```

| | └─ user_new.html
| └─ upload_history.py
|   └─ user_model.py
└─ requirements.txt
    └─ run.py
        └─ README.md

```

The above file structure output above was generated with the `>tree` command.

### 1.1.1. Flask Web Application

The applications flask web application is contained in the `run.py`. This will start the flask application in a single instance, we will be using gunicorn to execute our flask application as a threaded web server in order to provide higher performance at scale. As we noticed when uploading images using the provided `gen.py` script, which hits our file upload endpoint, that the webserver (local or EC2) was running every slow. However, configuring gunicorn was necessary as we found it to take high amounts of system memory and caused system performance issues.

Flask application name = `webapp`

### 1.1.2. Face Mask Detect Integration

As can be seen from the file structure, the entire face mask detection folder/code was integrated in to our project, placed in the `app/` folder. When our flask application wanted to use the face mask detection code it imported the inference function from the main python file within the face mask detection directory, `pytorch_infer.py`.

```
from app.FaceMaskDetection.pytorch_infer import inference
```

Integrating the face mask detection code did require a few code modifications within `pytorch_infer.py`:

- The main entry point and results code with in the `__main__` was commented out and the inference function arguments were extended to take in the processed file path as a parameter, such that allowing the inference function to now save the resulting generated image on the local file system.

### 1.1.3. /app Web Application Files

All code files for our web application is contained with the `/app` folder. At the high level this folder contains:

- `/Database` schema script
- `/FaceMaskDetection` code
- `/static/uploads` -> all images and their processed image uploaded to our application (local file system)
- `/templates` -> all our HTML files
- `__init__.py` -> contains our webapp configurations
  - o Upload folder
  - o Max image size
  - o Email info
  - o Secret key used by Flask for security in sessions (Flask Login) and to prevent CSRF attacks (Flask WTF for forms)

- Import all of our application python files
- \*.py -> These are all of implemented application files in python (flask application endpoints)
  - Login
  - User face detect upload image
  - User upload history
  - Password Recovery
  - Change Password
  - Add/Delete User

#### 1.1.4. Login - / or /login

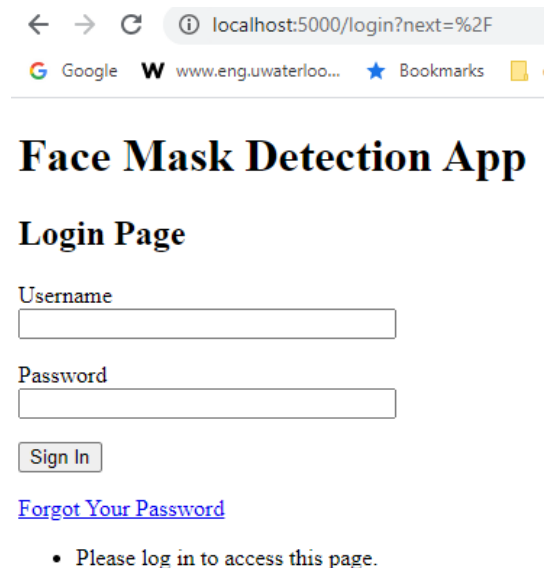
The main entry point to the web application is the login page. As we are using flask login package, we configured it to redirect all none-logged-in url requests to the login page. This is further enforced with flask login decorator `@login_required`, which enforces a flask route endpoint will only generate if user is logged-in, else redirects to login page. Once logged in the user will then be redirected to either the face detect upload page or the url they were originally attempting to access. Note: main.py only forwards between login and the face detect upload page, in case an actual main page is needed in the future.

The page contains a link to the password recovery page as well.

Flask flash messages are used to show additional information. The title and the flash message are part of the base.html page all HTML pages inherit from.

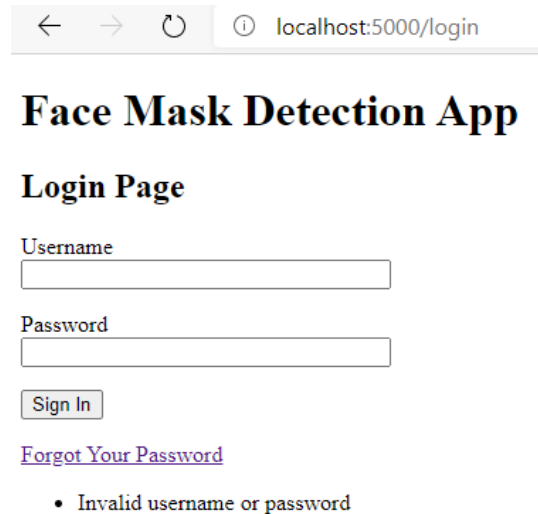
User login details are compared against our mysql database users table (error checking, if fail flash message below).

The werkzeug.security package is used to `generate_password_hash()` and `check_password_hash()`, see `user_model.py` for details.



The screenshot shows a web browser window with the address bar displaying `localhost:5000/login?next=%2F`. The page title is "Face Mask Detection App". Below the title is the heading "Login Page". There are two input fields: "Username" and "Password". Below the "Password" field is a "Sign In" button. Below the button is a link "Forgot Your Password". At the bottom, there is a bullet point: "• Please log in to access this page."

Figure 1: Login Page



A screenshot of a web browser showing the login page of the 'Face Mask Detection App'. The browser's address bar displays 'localhost:5000/login'. The page has a title 'Face Mask Detection App' and a subtitle 'Login Page'. It contains two input fields: 'Username' and 'Password'. Below these fields is a 'Sign In' button. A link labeled 'Forgot Your Password' is positioned below the button. At the bottom, a message states 'Invalid username or password'.

← → ↻ ⓘ localhost:5000/login

## Face Mask Detection App

### Login Page

Username

Password

[Forgot Your Password](#)

- Invalid username or password

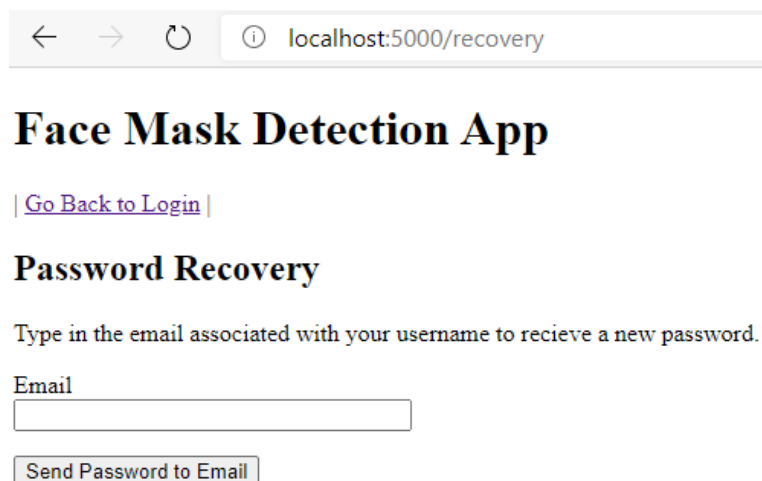
Figure 2: Login Page Invalid Credentials

Related files:

- login.py
- /template/login.html
- user\_model.py
- \_\_init\_\_.py -> sets /login endpoint as login\_view for flask login

### 1.1.5. Password Recovery - / recovery

If a user forgets their password, on the login page a link is provided that will take the user to a page that will allow them to enter their email address. If a user with the specified email address exists in the database, an email with a new password is sent to the user for them to login, where they can change their password. This is accomplished by generating a new random string as a password and updating the user's password in the database (users table) to be this new password. The email is sent using gmail.



A screenshot of a web browser showing the password recovery page of the 'Face Mask Detection App'. The browser's address bar displays 'localhost:5000/recovery'. The page has a title 'Face Mask Detection App' and a subtitle 'Password Recovery'. It includes a link 'Go Back to Login' and a text prompt: 'Type in the email associated with your username to recieve a new password.' Below this is an 'Email' input field and a 'Send Password to Email' button.

← → ↻ ⓘ localhost:5000/recovery

## Face Mask Detection App

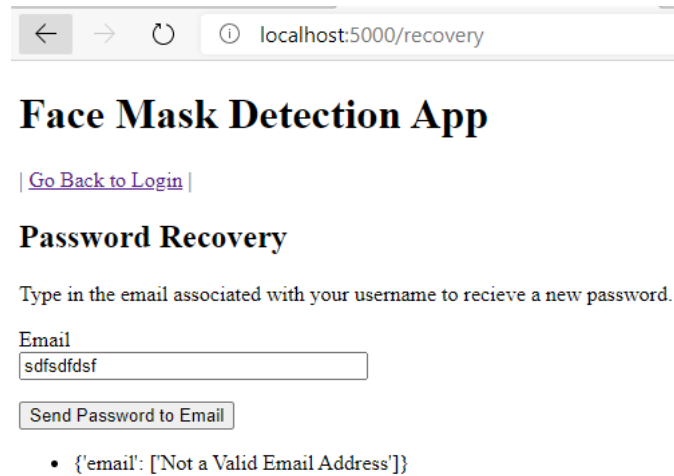
[Go Back to Login](#)

### Password Recovery

Type in the email associated with your username to recieve a new password.

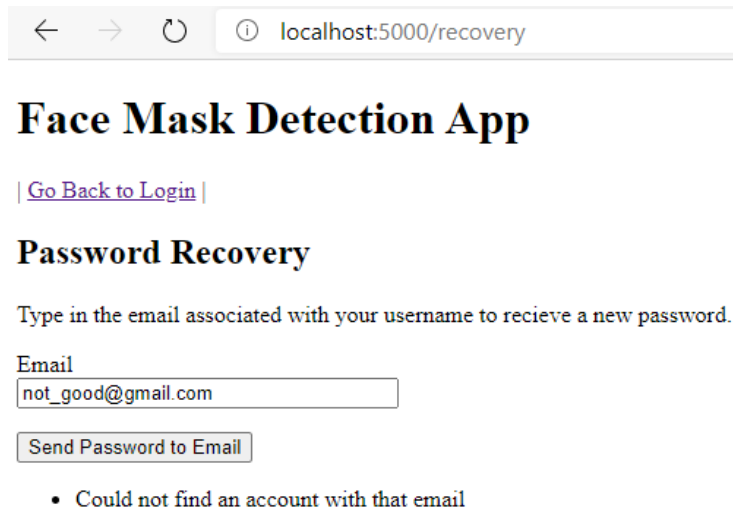
Email

Figure 3: Password Recovery Page



A screenshot of a web browser window. The address bar shows 'localhost:5000/recovery'. The page title is 'Face Mask Detection App'. Below the title is a link '| Go Back to Login |'. The section is titled 'Password Recovery'. A text prompt says 'Type in the email associated with your username to recieve a new password.' Below this is an input field labeled 'Email' containing the text 'sdfsdfdsf'. A button labeled 'Send Password to Email' is below the input field. Below the button is a red error message: '• {\'email\': [\'Not a Valid Email Address\']}

Figure 4: Password Recovery - Bad Email Format



A screenshot of a web browser window. The address bar shows 'localhost:5000/recovery'. The page title is 'Face Mask Detection App'. Below the title is a link '| Go Back to Login |'. The section is titled 'Password Recovery'. A text prompt says 'Type in the email associated with your username to recieve a new password.' Below this is an input field labeled 'Email' containing the text 'not\_good@gmail.com'. A button labeled 'Send Password to Email' is below the input field. Below the button is a red error message: '• Could not find an account with that email

Figure 5: Password Recovery - Invalid Email



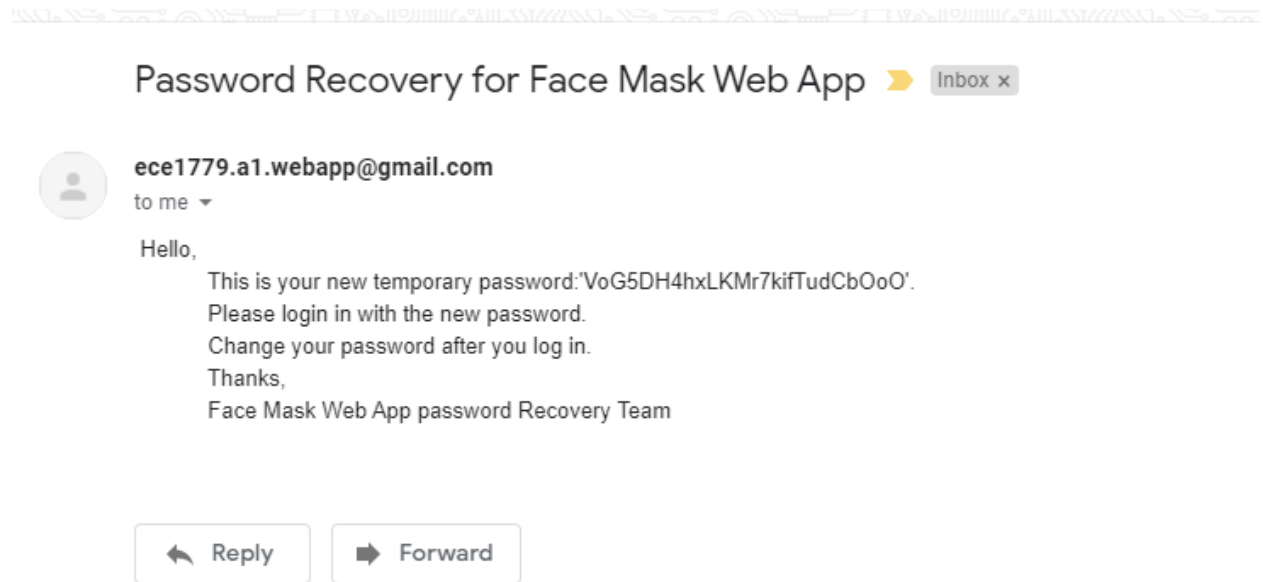


Figure 6: Password Recovery – Email message

Figure 6 shows the email the user would receive from the email address we created for this application.

Related files:

- recovery.py
- /template/recovery.html
- \_\_init\_\_.py -> sets up email in config

### 1.1.1. Logout - /logout

All HTML pages inherit the base.html template which contains a navigation bar. The navigation bar will only display the necessary URL links based on if the user is logged in or is an “admin”. The logout URL link will be available for all logged in user and will log the user out from flask login - `logout_user()`, and redirect the user back to the login page with a successful logout message.

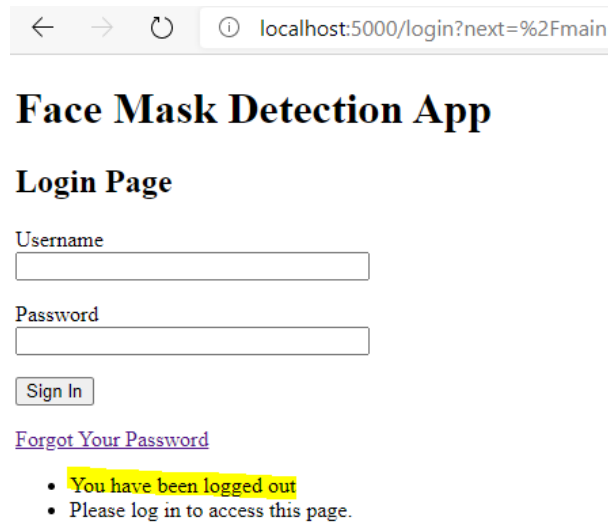


Figure 7: Logout Result

Related files:

- login.py
- /template/login.html
- /template/base.html

### 1.1.2. Change Password - /change\_password

All users have the ability to change their password. Once a user is logged-in the base.html template will generate a navigation bar which contains a link to the change password page. The base.html does this dynamic loading with the use of "{% if current\_user.is\_authenticated %}". Note: Just as the login page, flask forms are used to provide some validation and security on the client side. This will update the user's password in the database (users table).

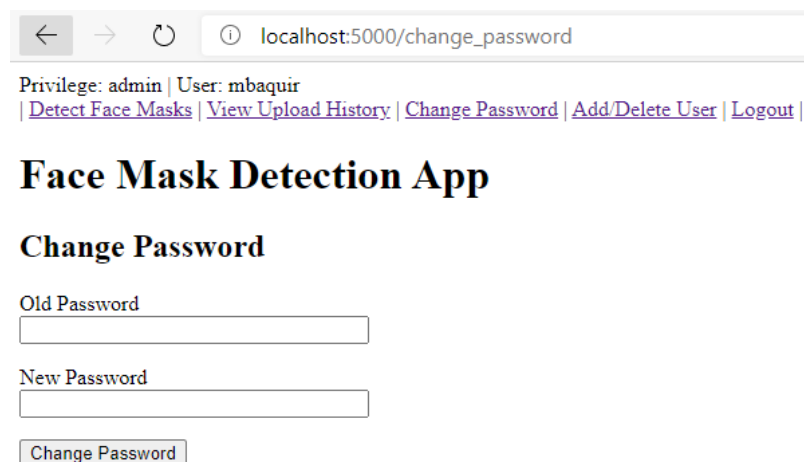


Figure 8: Change Password Page

← → ↻ ⓘ localhost:5000/change\_password

Privilege: admin | User: mbaquir  
| [Detect Face Masks](#) | [View Upload History](#) | [Change Password](#) | [Add/Delete User](#) | [Logout](#) |

## Face Mask Detection App

### Change Password

Old Password

New Password

- Incorrect Password for User

Figure 9: Change Password - Incorrect old Password

Related files:

- change\_password.py
- /template/change\_password.html

### 1.1.3. Add/Delete User -> /add\_user

This page is accessible from the navigation bar once logged in. However, it will only appear if the user account type is “admin”. Current user account type is visible at the top left corner of the navigation bar. If a regular user tries to access the page, it will be redirected back to the main page. The base.html handles the visibility of this link similar to that of change password.

The /add\_user page will collect all users from the users table and display it in table format. The ability to delete or create a user is provided, which on success will perform the correct operation in the database’s users table to reflect. Note: creating a user involves clicking the create button which redirects to “/add\_user/create”.

Privilege: admin | User: mbaquir  
| [Detect Face Masks](#) | [View Upload History](#) | [Change Password](#) | [Add/Delete User](#) | [Logout](#) |

## Face Mask Detection App

### Add, Delete or Edit Users

ID	Name	Email	User Type	
1	mbaquir	mohammed.baquir52@gmail.com	admin	<input type="button" value="Delete"/>
2	mbaquir2	mohammed.baquir@intel.com	regular	<input type="button" value="Delete"/>
3	corey	corey.kbaum@gmail.com	admin	<input type="button" value="Delete"/>

Figure 10: Add/Delete User Page

Privilege: admin | User: mbaquir  
[| Detect Face Masks](#) | [View Upload History](#) | [Change Password](#) | [Add/Delete User](#) | [Logout](#) |

# Face Mask Detection App

## Add Users

Username

Password

Email

User Type  

admin  
regular

Figure 11: Add User Page

Related files:

- add\_user.py
- /template/add\_user.html

### 1.1.4. Face Detect Upload - /facemask\_detect

Once a user is logged in, they are redirected to the upload page which allows users to upload images which then get applied to the face detection code. This page allows for either .jpg uploads from file system or a url. Max image size is set to 16MB, see configuration "MAX\_CONTENT\_LENGTH" in \_\_init\_\_.py.

← → ↻ ⓘ localhost:5000/facemask\_detect

Google W www.eng.uwaterloo... ★ Bookmarks ew interview Windows drivers ECE521 Inference A... Piazza

Privilege: admin | User: mbaquir  
[| Detect Face Masks](#) | [View Upload History](#) | [Change Password](#) | [Add/Delete User](#) | [Logout](#) |

## Face Mask Detection App

### Select a file to upload

No file chosen

Image url:

Figure 12: Get for Face Mask Detection

When file upload is successful the original, processed, and results (# of faces, # of masks, # no mask, and identified category for user to see in upload history page) are display on post and inserted into the

database. The image path on the local file system is what is saved in the database (images table), not the image file itself.

Images are saved to the local file system under /app/static/uploads. The file name is modified from the original uploaded image to:

- <username>\_<datetime>\_<filename>.jpg
- <username>\_<datetime>\_<filename>\_processed.jpg

localhost:5000/facemask\_detect

Privilege: admin | User: mbaquir  
[Detect Face Masks](#) | [View Upload History](#) | [Change Password](#) | [Add/Delete User](#) | [Logout](#)

Face Mask Detection App

Select a file to upload


Choose File

No file chosen

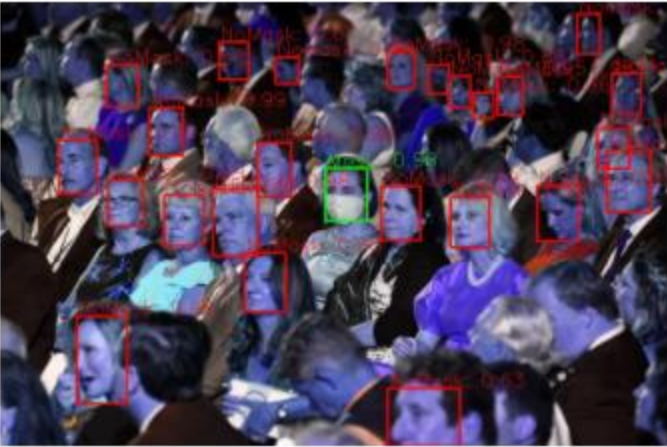
Image url:

Submit

Original Image:



Processed Image:



Number of Faces Detected: 25

Number of Masks Detected: 1

Number of No Masks Detected: 24

Category: some\_face\_mask

Figure 13: Post for Face Mask Detection - After upload

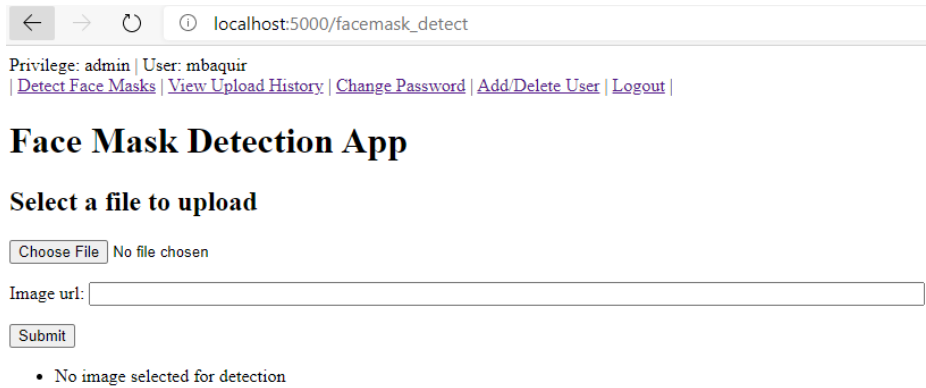


Figure 14: Face Mask Detect - No image selected

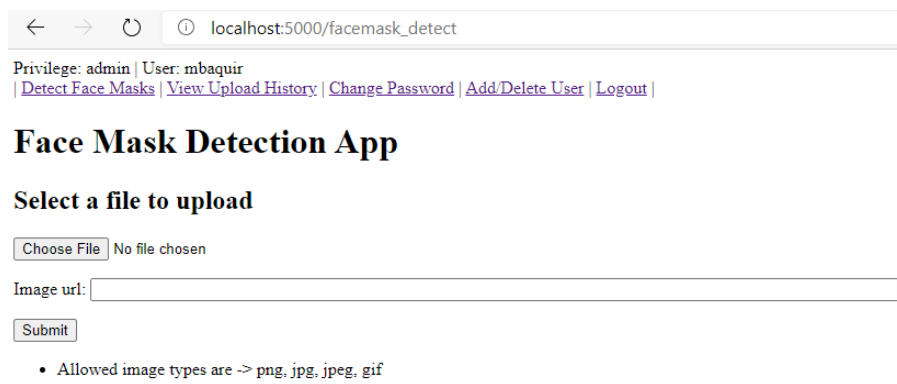


Figure 15: Face Mask Detect - Invalid Image

Related files:

- facemask\_detect.py
- /template/facemask\_detect.html

### 1.1.5. Upload History - /upload\_history

This page just contains the links to the 4 categories for an image. The links will take the user to /upload\_history/<category> url where they can see their history. The history is displayed in a table with original image, processed image, and details (# faces, # mask, # no mask). The history of images is pulled from the mysql database.

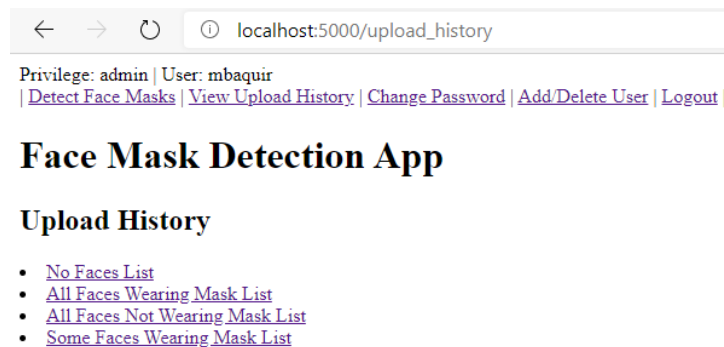


Figure 16: Upload History Main Page

localhost:5000/upload\_history/all\_face\_mask

Privilege: admin | User: mbaquir  
[Detect Face Masks](#) | [View Upload History](#) | [Change Password](#) | [Add/Delete User](#) | [Logout](#)

## Face Mask Detection App

### Upload History

- [No Faces List](#)
- [All Faces Wearing Mask List](#)
- [All Faces Not Wearing Mask List](#)
- [Some Faces Wearing Mask List](#)

### Upload History - all\_face\_mask

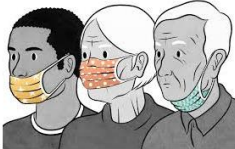
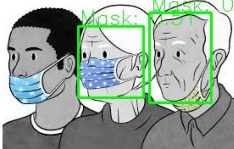


Original Image	Processed Image	Image Data
		Number of Faces Detected: 2 Number of Masks Detected: 2 Number of No Masks Detected: 0
		Number of Faces Detected: 1 Number of Masks Detected: 1 Number of No Masks Detected: 0

Figure 17: Upload History – List Example for All Faces Wearing Mask

Related files:

- upload\_history.py
- /template/upload\_history\_main.html -> list of links to the 4 category upload history
- /template/upload\_history.html -> table of upload history images for category

## 2. Database Design

Database utilities => db\_utils.py (automatic teardown)

Database credentials => config.py

```
db_config = {'user': 'ece1779',
             'password': 'secret',
             'host': '127.0.0.1',
             'database': 'facemask_app',
             'auth_plugin': 'mysql_native_password'}
```

To login to “facemask\_app” database, (our web application uses) database user:

Username: ece1779

Password: secret

To start mysql database, if it is not running, execute in terminal:

> sudo /etc/init.d/mysql start

Note: to generate database and tables run in mysql:

mysql> source app/database\_schema/facemask\_app.sql



The database schemas are in /app/database\_schema/facemask\_app.sql

The project uses one mysql database: "facemask\_app"

This database contains only two tables: "users", "images"

**users: (id, name, password, email, user\_type)**

- id = auto-incrementing integer (primary key)
- name = username
- password = hashed password
- email = user email
- user\_type = ('admin', 'regular')

**images: (id,**

**original\_img\_path, processed\_img\_path,  
num\_faces, num\_masks, num\_no\_masks,  
category\_name,  
userid)**

- id = auto-incrementing integer (primary key)
- original\_img\_path = file path for uploaded image on local file system
- processed\_img\_path = file path for process/modified upload image (through face detect) on local file system
- num\_faces, num\_masks, num\_no\_masks = integer number value from face detect
- category\_name = ('no\_face', 'all\_face\_mask', 'all\_no\_face\_mask', 'some\_face\_mask')

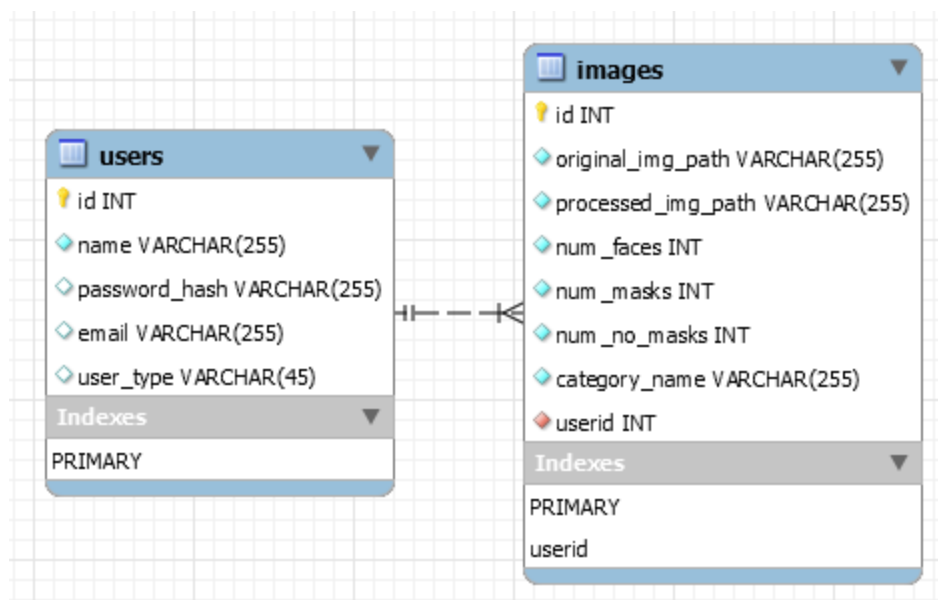


Figure 18: Database ER Diagram

Most database queries in our application involve select searches on:

- users.id (to get/check logged in user info, to get images)
- username (for login)
- email (Password recovery)

Hence, as most searches are on user id, having user id being the primary key will provide good performance. However, as on login user id is not known, it might be considerable to think about having username as primary key and remove id. This is thought by the group to not be a good idea as string comparisons are more expensive than integer comparison.

#### **Normalization Proof:** [7, 8]

1NF: Each column unique/single/atomic value

- ✓ Every entry in table is unique, and each column only contains a single value

2NF: Must be 1NF and no partial dependencies

- ✓ As tables contain a single primary key, no partial dependencies exist

3NF: Must be 2NF and no column is dependent on a none key column

- ✓ Again as tables contain a single primary key, all columns depend on the key only

BCNF: Must be 3NF and all tables have only a single primary key

- ✓ All tables contain a single primary key

Related files:

- config.py
- database\_schema/facemask\_app.sql
- db\_utils.py

### **3. Testing Endpoints**

To allow for automatic testing, as specified in the project description, two URL endpoints have been added.

#### **3.1. Register - /api/register**

This conforms to the post form specified in the project description.

If username or password is not provided => error code 400 with a “Bad Request - please ensure username and password are set” is sent back in json.

If the user already exists => error code 400 with a “Bad Request - please ensure username and password are set” message is sent back in json.

On success user is added to database (users table) as a regular user account.

Example:

curl -d "username=Corey&password=Kirschbaum" -X POST <http://localhost:5000/api/register>

```
{
    "success": true
}
```

**Figure 19: Register Json Success Message**

```

{
    "success": false,
    "error": {
        "code": servererrorcode,

        "message": "Error message!"
    }
}

```

Figure 20: Register Json Error Message

### 3.2. Upload - api/upload

This conforms to the post form specified in the project description.

If file is not provided as an argument => error code 400 with a “Bad Request - No file part” is sent back in json.

If username or password is not provided => error code 400 with a “Bad Request - please ensure username, password and file are set” is sent back in json.

If the file argument is empty => error code 400 with a “Bad Request - No image selected for detection” is sent back in json.

If the file type is not valid => error code 400 with a "Bad Request -filename:<filename> - Allowed image types are -> png, jpg, jpeg, gif" is sent back in json.

If the user does not exist => error code 400 with a “Bad Request - Could not find an account with that username. Please register the user first” is sent back in json.

If the password is not correct => error code 400 with a “Bad Request - Incorrect password” is sent back in json.

If could not save file in local file system => error code 500 with a “Bad image\_path - Could not save the image” is sent back in json.

If could not parse image (bad image file) => error code 500 with a “Bad Image - Could not parse the image” is sent back in json.

```

{
    "success": true,
    "payload": {
        "num_faces": number,
        "num_masked": number,
        "num_unmasked": number
    }
}

```

Figure 21: Upload Json Success Message

```

{
    "success": false,
    "error": {
        "code": servererrorcode,

        "message": "Error message!"
    }
}

```

Figure 22: Upload Json Error Message

## 4. How to run the app

### 4.1 Amazon EC2 steps

We are using the AWS educate account for running our EC2 instance.

The educate account we are using is: 'mohammed.baquir@mail.utoronto.ca'

The ec2 instance that we are using is called 'my\_ece1779\_instance' with the instance ID: 'i-02883f308e5931a9f'

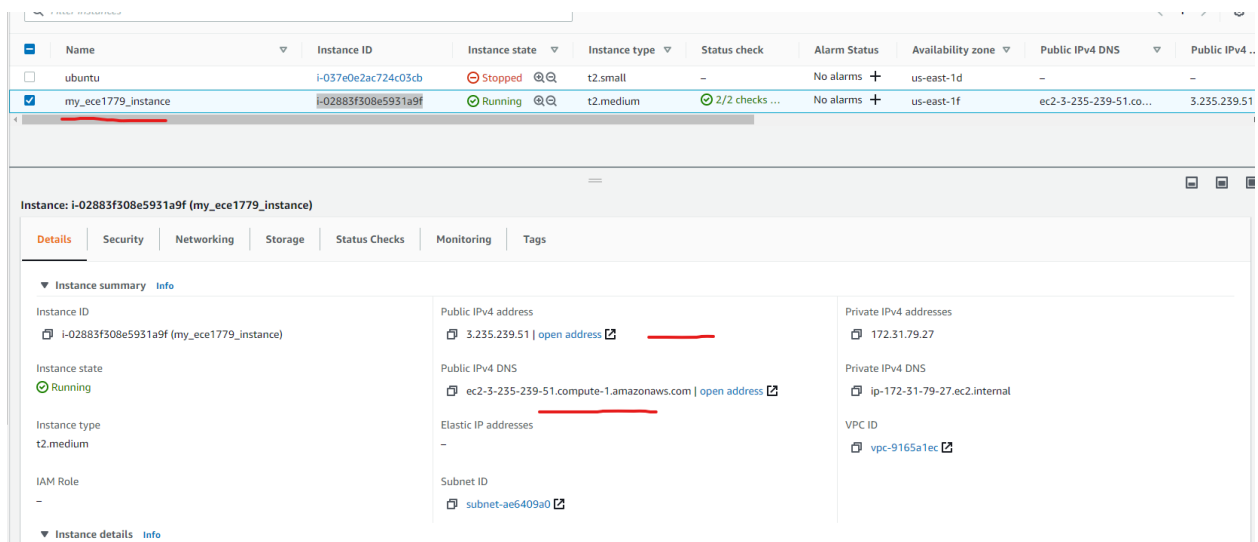


Figure 23: Amazon Ec2 instance

Note the Public IPv4 DNS and the Public IPv4 address which will be used to ssh into the ec2 linux server and the IP address to access the flask web application.

We have included the ssh keys in the tar file project called : 'baquirmo\_ec2instance.pem'. Also created a copy of the pem file called 'keypair.pem'

Once the instance is started, copy the public IPV4 DNS address to ssh into the server. For example:

```
`ssh -i baquirmo_ec2instance.pem ubuntu@ec2-3-235-239-51.compute-1.amazonaws.com`
```

Once sshed in, cd into the Desktop directory.

Under the Desktop directory we have ece1779\_a1 which is our git repo with the project.

Under the Desktop directory we also have the start.sh script that runs the project using gunicorn.

## 4.2 Running the App

We have already created an admin account in the webapp. To view pages that only admin is allowed to see for example “Creating and deleting users” use the following admin credentials:

username: mbaquir

Password: password

This will allow you to see the admin privileged pages.

The start.sh script contains the gunicorn command used to run the application:

```
` gunicorn --bind 0.0.0.0:5000 --preload --workers=5 --worker-class=gevent --worker-connections=1000 -  
-access-logfile access.log --error-logfile error.log run:webapp `
```

We decided to use my workers based on the following formula:

$(\text{Number of CPU} * 2) + 1$

Our linux server has 2 CPUs and thus we are using 5 workers. We played around with different gunicorn settings to see what gave us the optimal performance taking into account the memory requirements of the ec2 server.

We were having socket errors when running gunicorn without the preload flag.

## 5. Future Improvements

Due to the time constraint of the project and of the group’s members schedule, many improvements can be made in the future.

- Clean up code and modularize
- Improve performance for multi-users for scale
- Create a background task to delete image that are X days old to save space on system
- Push database to an AmazonS3
- Improve UI and visuals
- Provide more metrics on history page
- Perform more rigorous testing to handle edge cases
- Attempt to better optimize database performance
- Ensure all function return values are properly error checked
- additional checks for .exe

Issues:

- Tried upload large exe caused something to block my localhost

## References:

- [1] <https://docs.microsoft.com/en-us/office/troubleshoot/access/database-normalization-description>
- [2] <https://blog.miguelgrinberg.com/post/the-flask-mega-tutorial-part-i-hello-world>
- [3] <https://flask-login.readthedocs.io/en/latest/>
- [4] <https://pythonhosted.org/Flask-Mail/>
- [5] <https://flask.palletsprojects.com/en/1.1.x/>
- [6] <https://dev.mysql.com/doc/connector-python/en/connector-python-example-connecting.html>
- [7] <https://www.studytonight.com/dbms/database-normalization.php>
- [8] <https://www.w3schools.in/dbms/database-normalization/>