ECE1779 Project 1 Documentation

Hao Yin - 1007045001

Zhihao Yang - 1002270776

Leo Li - 1002598833

Description

In this project, we developed a web application using Python Flask framework. The web application contains a user management system, which requires a user to be registered by an administrator to use the application. Once the administrator registers the user, an email with the default password will be sent to the user, together with a link to change the password. The user can either choose to use the link to reset his/her password, or use the default password to login the application, and there will be a password change link there in the main page. Besides that, in the main page, the user can perform a mask detection function by clicking the corresponding button, which will navigate the user to a page where he/she can upload an image from local/url, and a result image showing faces with masks or not will be presented. In addition, the user can click on the upload history button in the main page to view the image he/she previously uploaded together with the result images. The administrator can use the user management system whose entry is also located in the main page to view/delete/create users. Lastly, if the user somehow forgets his/her password, there is a password reset page provided where the user input their registered email address, and an email with a link that will direct the user to a password reset page will be sent.

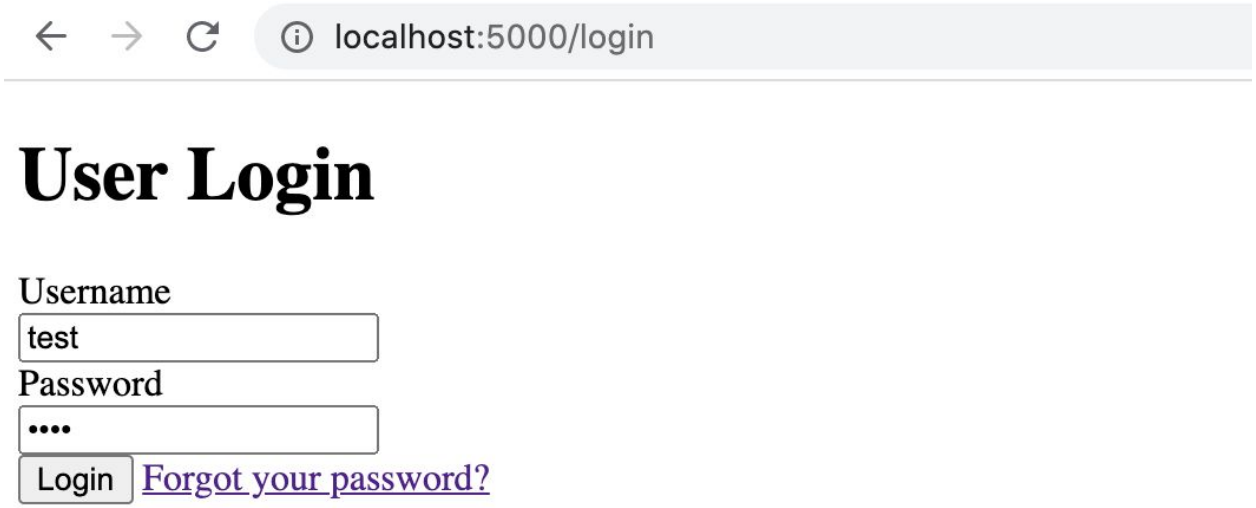
Prerequisites

Flask, flask_mail, passlib, requests, jwt

pip3 install Flask
pip3 install Flask-Mail
pip3 install passlib
pip3 install requests
pip3 install jwt
pip3 install opencv-python
pip3 install torch torchvision

Installation

We deployed our application on Zhihao Yang's AWS EC2 instance, and he uses a AWS educate account. Once you login to our AWS EC2 instance, the environment and the database has already been configured for you. You just need to use the command line to run the start.sh file located at the Desktop to start our web application server, e.g "bash ./start.sh". **The administrator's username and password have both been set to "test"**, which you can use to login at port 5000 route /login(web app entry) and register new users.

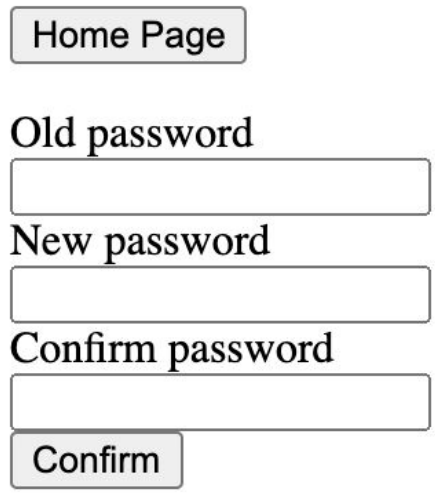


Once you login to the main page as administrator, you can click the "User Management" button.

You can view all the existing users here, and you are able to delete the existing user, or create a new one. The “Create a new user” button will navigate you to the user registration page.

By filling the text fields with username and user’s email address, an email containing a default password and a link will be sent to the email address(if the username and email address are not taken), with a link for changing the password.

The “Change your password” will lead the user to a page where he/she can change his/her password, where the old password is required.



In the upload image page, users can either select images from their local disks or download images by typing image URL from the web. After pressing the send button, the chosen image will be transferred to the server and be processed by the AI program to detect faces with masks. In addition, if you want to change the image file, press the reset button.

Upload Image

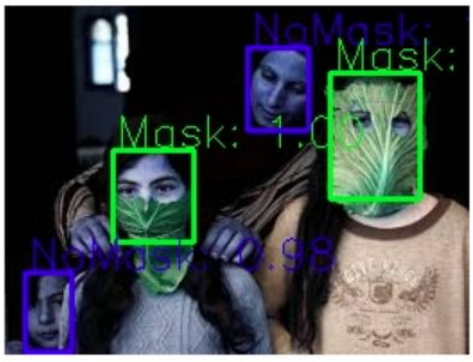
What is the image files to upload? Choose file test2.jpeg
Send Reset
What is the image file url? (Image URL here)
Send Reset

If the image file is valid, the browser will jump to the test result page as shown below. It shows the tested image and its result message, which shows if there are faces and whether they are wearing masks. If there are no images to be tested, an error message will pop up after pressing the send button. Also, if the Url is not valid, other corresponding error messages will pop up as well. By the way, the image size is acceptable, only smaller than 64MB.

Image Test

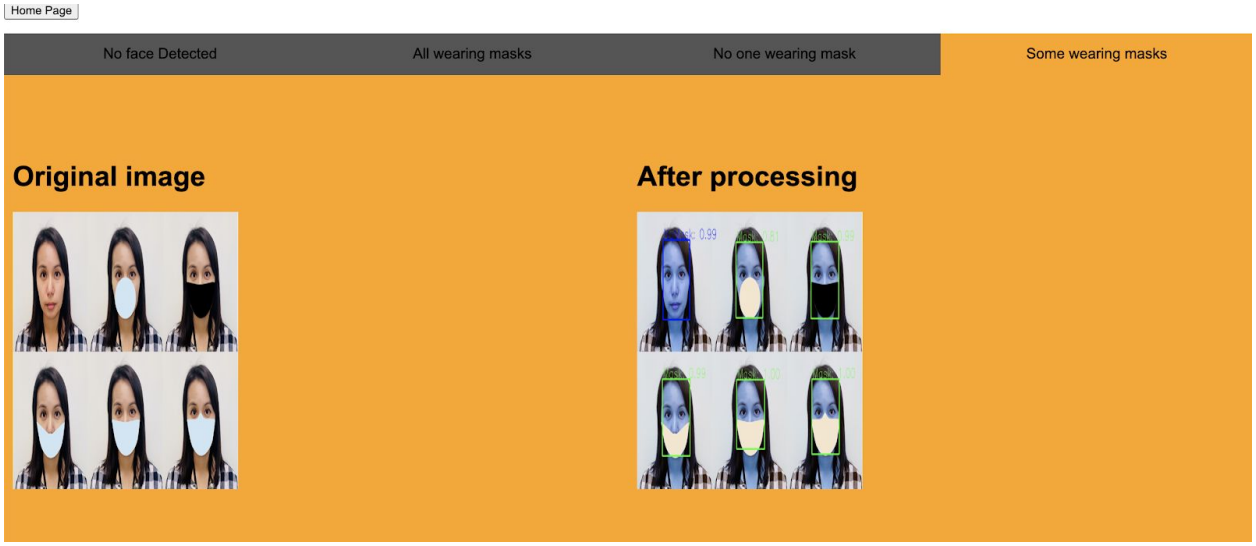
Back

some faces wearing masks



By pressing the ‘show upload history’ button, the user can browse all the images that were previously uploaded.

Four tabs on the top allow the user to choose which type of images he or she wants to see, which are: ‘No face detected’, ‘All wearing masks’, ‘No one wears mask’, ‘Some wearing masks.’



APIs and Important functions(technique) implemented

API for user registration
Route: hostname:5000/api/register
Method = POST

Description: register a user with username set to form.username, and password set to form.password.

Parameters:
Post form parameter: name: username, type: String
Post form parameter: name: password, type: String

What the API returns:

- The API returns a message of success and status code 201 on successful registration.
- It returns a status code 409 with a corresponding error message if the username is already taken.
- A status code of 400 will be returned with information of error if any of the required parameters is missing.

API for image upload
Route: hostname:5000/api/upload
enctype = multipart/form-data
Method = POST

Description: upload an image file and run it with the AI program for mask detection, then return the number of faces, number of faces wearing masks and number of faces not wearing masks.

Parameters:
Post form parameter: name: username, type: String
Post form parameter: name: password, type: String
Post form parameter: name: file, type: file

What the API returns:

- The API returns the success message including number of faces, number of faces wearing masks and number of faces not wearing masks with status code 201 on each successful test.
- It returns a status code 401 with a corresponding error message if the username does not exist or the password is wrong.
- It returns a status code of 413 with the error information if the uploaded image is too large(over 64MB).

- A status code of 400 will be returned with information of error if any of the required parameters is missing.

Session

Session was used to store information of the user who has logged in, such as user_id, username, etc, so that these useful information can be passed from one request to another without extracting it over and over again from the database, and it is secured with a secret key defined in the webapp instance. Most of the page will first check the session on the server side in order to prevent unauthorized visits such as trying to navigate to a page directly through the url. Once a user logs in, he/she will no longer need to go to the login page and sign in again before the session expires. The session was implemented with the session module in Flask. Once a user successfully passed the login verification, we put useful information of the user into session through:

```
session['username'] = request.form["username"]
session['role'] = row[1]
session['user_id'] = row[2]
```

If in some other request we want to use these information, we simply use the key to find the corresponding value stored in session:

```
user_id = session['user_id']
```

Once the user logs out, the session is cleared by:

```
session.pop('username', None)
session.pop("role", None)
session.pop("user_id", None)
```

Password Encryption

We do not want to store the user passwords in plain text in the database, in which case if the database's security is compromised, so will the user information. Instead, during the password reset and user registration step we take the user's password and add a random salt in front of it. After that use hash(sha-256 algorithm) to encrypt the "random salt + password", then store the result into the database. The cryptography process is done with:

```
encrypted_password = sha256_crypt.hash(password)
```

The sha256_crypt.hash() function takes a string as an input, adds a random generated salt before it and returns a corresponding hash value which is hard to hack through. Even a single password is compromised, since every password uses a random salt, it will be extremely hard for the hackers to break the entire user database.

To verified if the password user provides matches the hash value in our database, we simply use:

```
sha256_crypt.verify(password, verified_password)
```

The sha256_crypt.verify() function takes the user-provided password(unhashed) as first parameter and the password stored in the database(hashed) as second parameter, if the two matches it returns true, otherwise false.

Email Service

We set up a gmail SMTP server for sending emails to users during register and password recovery steps. The server config is indicated in the config.py file, and passed to the webapp instance by:

```
# smtp configuration
mail= Mail(webapp)

webapp.config['MAIL_SERVER']=smtp_config['MAIL_SERVER']
webapp.config['MAIL_PORT'] = smtp_config['MAIL_PORT']
webapp.config['MAIL_USERNAME'] = smtp_config['MAIL_USERNAME']
webapp.config['MAIL_PASSWORD'] = smtp_config['MAIL_PASSWORD']
webapp.config['MAIL_USE_TLS'] = smtp_config['MAIL_USE_TLS']
webapp.config['MAIL_USE_SSL'] = smtp_config['MAIL_USE_SSL']
```

We used the Mail and Message modules from Flask to send the email. In addition, we don't want the user to wait for too long while the email is sending. So we send the email asynchronously with a thread, below is how we defined the sending email function and how it is called:

```
mail = Mail(webapp)

def send_email(app, msg):
    with app.app_context():
        mail.send(msg)
```

```
Thread(target = send_email, args = (webapp, msg)).start()
```

Token

When an email with a password reset link is sent to the user, we add a token in the link which will automatically verify the user when he/she clicks on the link. The token created with JWT transmits information from the client to the server in a very secure way, and is also protected

with a JWT secure key defined in the webapp instance. The token will expire in 10 minutes, after that the user may have to request a password set email again. This is implemented by:

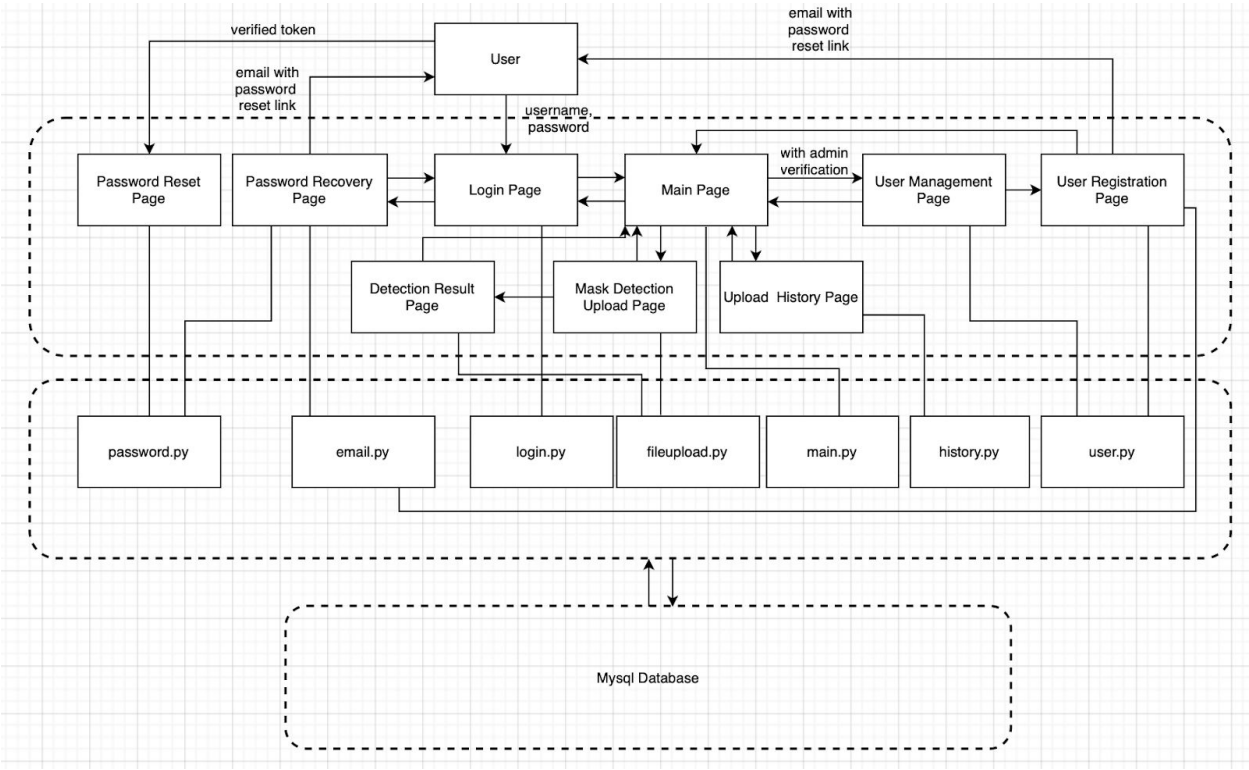
```
# get a token for the reset link, valid for 10 minute
token = jwt.encode({"reset_password": username, "exp": datetime.datetime.utcnow() + datetime.timedelta(seconds=600)},
                   webapp.config.get("JWT_SECRET_KEY"), algorithm='HS256')
# email display message
display_msg = "We have received your password recovery request. Please use the link below to reset your password:"
msg.html = render_template('/password/passwordreset.html', token = token, username = username, display_msg = display_msg)
Thread(target = send_email, args = (webapp, msg)).start()
```

Regular Expression

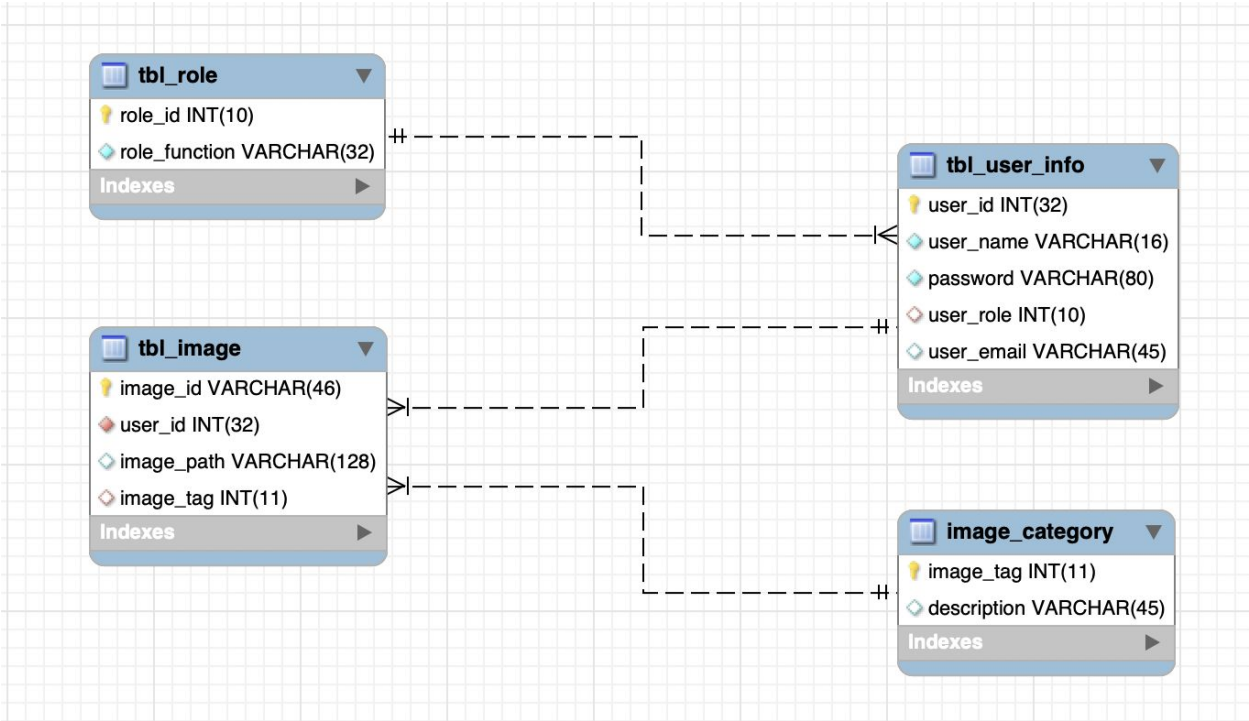
We used the regular expression module to check the validity of the user email during user registration, this is implemented by:

```
elif not re.match(r"^[A-Za-z0-9\.\+_-]+@[A-Za-z0-9\._-]+\.[a-zA-Z]*$", email_address):
    error_msg = "Please enter a valid email address!"
```

Application Architecture



Database Schema



History.py

Pass all the names of images to ‘history.html’ for the user who has logged-in. The images are divided into two groups, one group is the original image uploaded by the user, the other group is the images after processing. Different groups are passed separately to History.html

```
view = render_template("history.html",title="Upload history", beforeAI=beforeAI, afterAI=afterAI)
```

History.html:
Display history of images uploaded, all images passed from history.py is shown using”{{url_for(‘static’,filename=i)}}”

```
                {% for i in beforeAI[0] %}
                <image src="{{url_for('static',filename=i)}}" width="260" height="320"/>
            <br>
        <br>
          {% endfor %}
```

Contributing

Issue Tracker: https://github.com/Leoooooli/Web-dev-Flask/issues

Citation

Monaghan, S. (2020, January 25). Password Reset with Flask-Mail Protocol. Retrieved October 16, 2020, from https://medium.com/@stevenrmonaghan/password-reset-with-flask-mail-protocol-ddcdfc190968

W3schools.com HTML tutorial. Learn how to create full page tabs, that covers the entire browser window, with CSS and JavaScript.

https://www.w3schools.com/howto/howto_js_full_page_tabs.asp

Contact

leoo.li@mail.utoronto.ca
hy.yin@mail.utoronto.ca
kevinspace.yang@mail.utoronto.ca