

ECE1779 Assignment 1

Mask Detection Web

Xiaoxi Yu
1005654748

Jiani Jia
1002226245

Shuya Wang
1006549503

Submitted on October 19, 2020

1 Project Summary

In this project, we create a Flask web app with mask recognition integrating into the browser. The website allows the administrator to create and delete normal user accounts, which requires username, email and password. Once logged in, all users can upload photos to do mask detection, where openCV is automatically applied for. Our website is able to record both original and processed photos in separate folders. After an image is uploaded, the website will show a new version of the image with red or green rectangles, which helps users know how many people are choosing to wear masks in the photo. When users browse their uploaded history, they will find that their photos are sorted into four categories, according to the number of people and the number of masks detected. The flowchart figure 1 in the next section summarizes the overall process for our website.

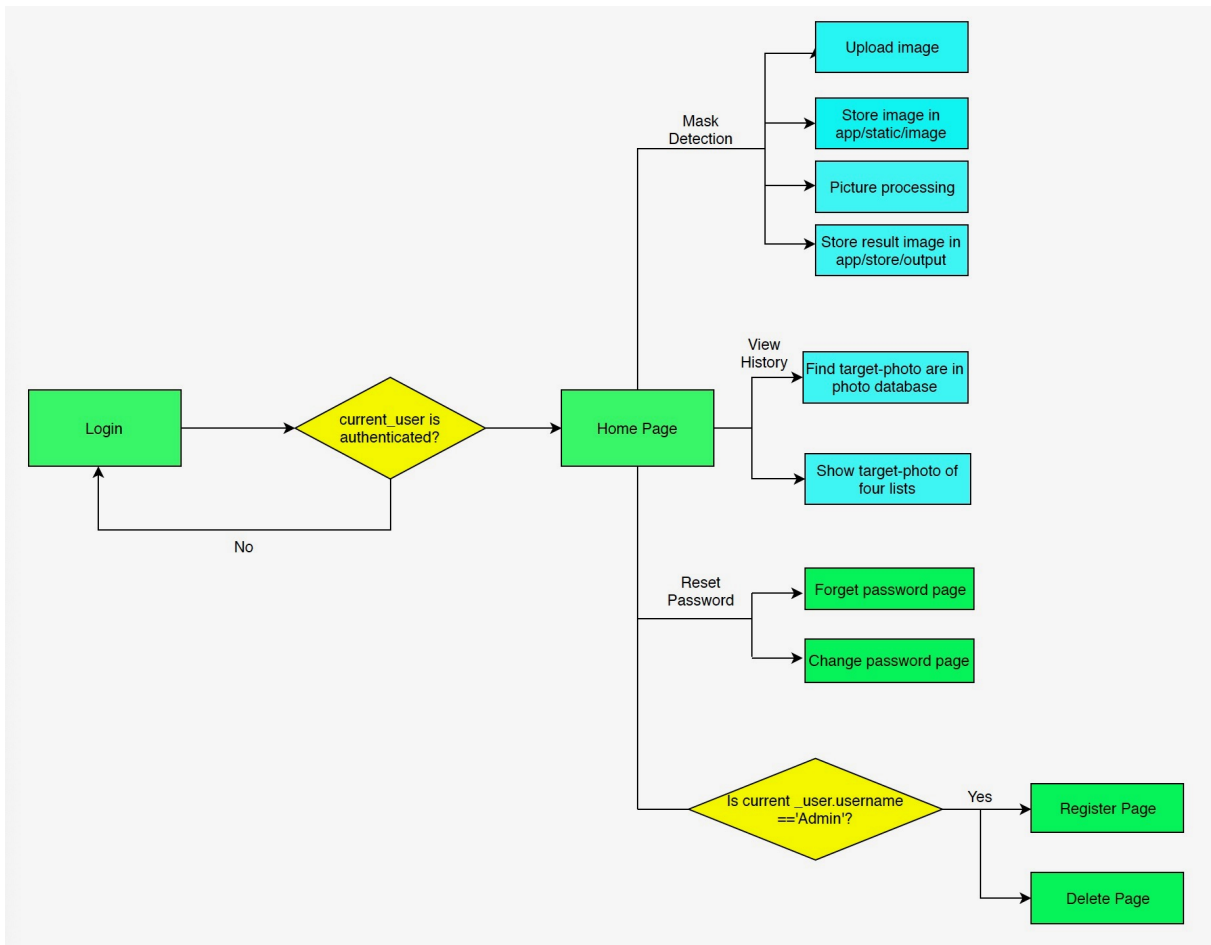


Figure 1: Block Diagram

2 Features

1. User Login/ Logout
2. Upload Photos from local or url
3. Mask Detection
4. Review History
5. Reset/ Change Password by Email
6. API Register/ Upload
7. Register New User, Delete User (Admin only)

3 Flask Extension Packages

Our project requires the following dependencies to be installed,

1. flask
2. flask-login: handle the tasks of logging in and logging out
3. flask-wtf: use Python classes to represent web forms
4. flask-sqlalchemy: connect to MySQL database
5. flask-mail: send email

4 User Instruction

1. Through url **http://35.168.113.124:5000/**, the flask web application can be accessed.
2. All users need to log in with their username and password at the login page. We specify the administrator named "Admin".
3. Only the administrator can register or delete accounts for users.
4. After logged in, users can upload photos from their local computer or type the photo's url address. They can run Mask Detection of the specified photo by clicking the "Submit" button. The result photos and the uploaded history will be shown at the bottom of the page.
5. Every user's specified detection histories are sorted into four lists: no face in the photo, all faces have masks, no one has a mask, some faces have mask. These information will also be represented at the top of page: how many face in the photo, how many people masked, how many people unmasked.
6. If the user forgets his password, he can click the "Forget Your Password? Click to Reset It" link in the login page and submit his recovery email. After that, they will receive an email sent by our admin email with a reset password link. Users can also change their password in the same way.
7. Users can log out their account by clicking the "Logout" button on the top navigation bar.

5 Program Description

5.1 File description

1. venv File: create virtual environment for Python 3
2. config.py: configure parameter and initial setting for our program, including database information, photo format, mail server
3. microblog.py: add the shell context processor function

4. app/forms.py: include login, register, delete account, reset password request, reset password forms to let user interact with our website
5. app/-init-.py: initialize file for our program
6. app/User.py: create an instance linked with User Table in database, consisted of id, username, email, hashed password
7. app/Photo.py: create an instance linked with Photo Table in database,, consisted of username, photourl, imagetype
8. app/email.py: send recovery email
9. templates/login.html: login page
10. templates/index.html: register account, delete account, upload photo from local computer, upload photo from url address, review history
11. templates/register.html: only visible for administrator, create new user account
12. templates/delete-account.html: only visible for administrator, delete existed user account
13. templates/reset-password-request.html: enter user's email for resetting password
14. templates/reset-password.html: the link in the recovery email can link to this page to reset password
15. templates/base.html: base of our web page
16. templates/email/reset-password.html: content shown in the reset password email
17. templates/email/reset-password.txt: content shown in the reset password email
18. static/image file: store the upload photo
19. static/output file: store the result photo
20. app/routes.py: website request routes
 - (a) /login: login page and store user information into database
 - (b) /index: upload photo store in app/static/image, save result photo into app/static/output and store into database
 - (c) /logout: logout current user account
 - (d) /register: Admin create account for new user
 - (e) /delete-account: Admin delete user existing account
 - (f) /reset-password-request: send email to user to reset password
 - (g) /reset-password/token: store new password into database
 - (h) /api/register: allow for automatic register new user testing
 - (i) /api/upload: allow for automatic upload photo testing

5.2 Front End

Templates: This folder contains all HTML templates referenced by each route. First we have base.html for an overall page layout structure design. Then we have login.html used for users to login. Once logged in, we will be directed to index.html which is the homepage for all the users, but for admin we have a register function linked with register.html and a delete account function linked with delete-account.html which regular users do not have. In the homepage, all users have the ability to upload files from local repository or URL links, then get four image classes classified by the number of faces and number of masks in the image which are shown below as figure 2 and figure 3.

Hi,Admin

[Register For New Account](#)

[Delete An Account](#)

Local File Upload

No file chosen

URL File Upload

[URL File Upload](#)

Figure 2: Upload files

5.3 Back End

The Back End part is mostly stored in the 'app' folder, which contains '-init-.py', 'routes.py', 'forms.py', 'Photo.py', 'User.py' and 'email.py'. First, '-init-.py' will initialize the flask module every time we run the web, it sets up the path of getting configures, the connection with SQL, also the reset password by email part. 'Routes.py' is the most important part of this project, since it links to all the routes requests, routes requests are individually shown below:

1. **login:** This page is for users to login to their homepage and is usually the first page that we see. When users require to login, it will check if current user is authenticated, then redirect them to their homepage, else, it will get their username, email and password using `query.filterby()` function, then check whether their password match with the password that we stored in database before. If so, let users login into their personal homepage, else, it will stay at the login page and flash 'Invalid username or password' at the top.
2. **logout:** log out users using `logout-user()` function in flask-login library.
3. **index:** Homepage for all users. Users need to login first so that users can get into their personal homepage. First, it gets all the files that this page needs. Then it gives each file in lists a unique name using `uuid` function. After that, it will extract file extension to see whether it's satisfied with given formats. If satisfied, then save the file into the local folder, else, show the error message 'Wrong Format!'. Read the image out from the local folder and apply mask detection on it, save the information of output photo into Photo database and save files into local folder; `output_`. Also, we save the number of faces, number of masked faces, number of unmasked faces into different arrays.
4. **register:** In Registration form, form checks the new username and new password in database to see if it's unique. If unique, then set password and upload into User table in database. If not, then show error message.
5. **delete-account:** Check if the username and email are in database. If so, delete this user's account.
6. **api/register:** API register routes are used for automatic testing. It take username and password to validate them. It will use the username and password to check whether they are both valid first.
7. **api/upload:** API upload routes are used for automatic testing. It will validate the username and password first. It will read and past all the valid files, then save and create the detected version.

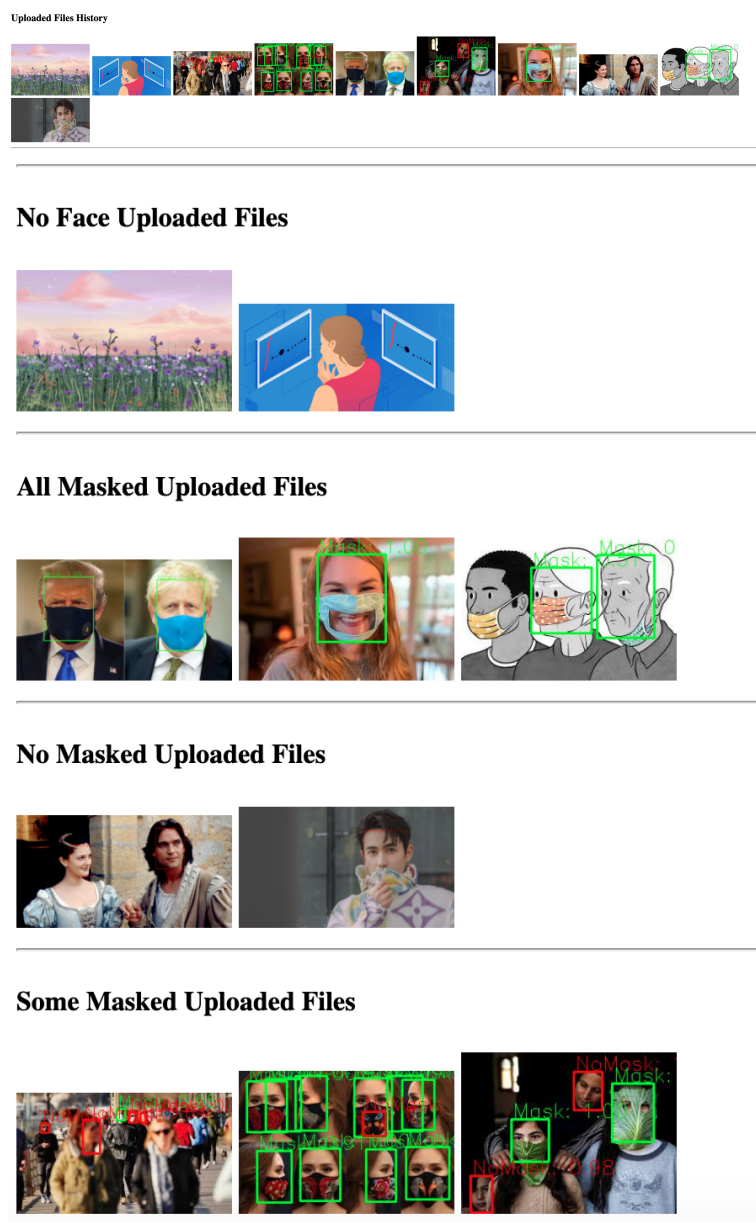


Figure 3: Uploaded photos + Four classes of detected photos

8. **urlupload:** URL address upload routes use the urllib library and save the photo in specified local file by function urllib.request.urlretrieve.

‘Forms.py’ stores all kinds of forms that have been used before in ‘routes.py’ file, each form is structured rigorously and is connected with its matching routes closely. ‘Photo.py’ and ‘User.py’ are two files that create tables that we need in the database.

6 Database

Database normalization, like figure 4, is a database design technique that divides a larger table into smaller tables and links them using relationships. The purpose of normalization is to eliminate redundant (repetitive) data and ensure data is stored logically.

In our case, the username is the link of two databases. When a user creates an account, the user information has a record in the User table, and the uploaded photos will have a record in the Photo table. This kind of relationship is called a one-to-many, since one username uploads many photos. Left table is to record User’s information including username, email, hashing password. When a user

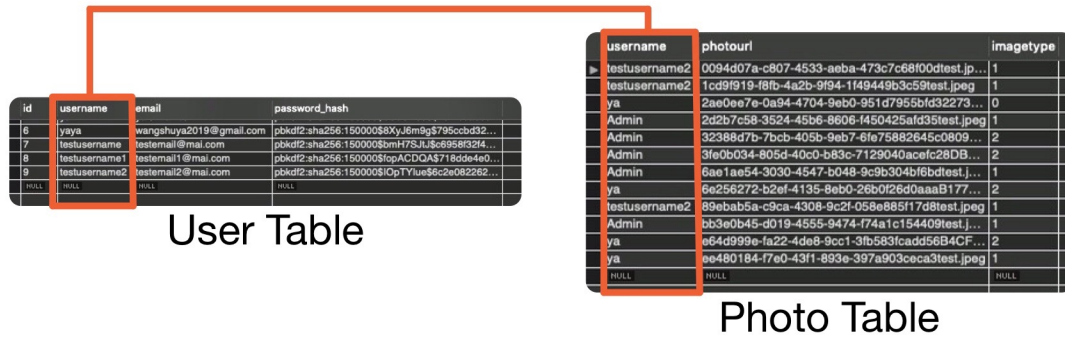


Figure 4: Normalized Database

login to our website, the program first checks whether the username is already inside the database or not. If the user does not exist, the program returns None, if exist, the program checks the correctness of password. When the password is not correct or the username is not in the database, our website will warn “invalid username or password” and not take the next step, otherwise, the user enters the mask detection page.

Right table is to record the photos history for each authenticated username. Mask detection can split the user uploaded photos into four lists, the column ‘imagetype’ in our database is the label for each photo that can indicate which list it belongs to. The column ‘photourl’ stores the unique id for each photo, therefore, the web page can find the path for each photo when needed.

7 Website Screenshot

Screen shots of all pages from the website application are presented as follow:

Mask Detection: [Home](#)

Sign In

Username

Password

Admin username: Admin
Admin password: Admin

Forget Your Password? [Click to Reset It](#)

Figure 5: Login page

Mask Detection: [Logout](#)

Hi,ya

Local File Upload

No file chosen

URL File Upload

[URL File Upload](#)

Figure 6: Regular user functional page

Mask Detection: [Logout](#)

Hi,Admin

[Register For New Account](#)

[Delete An Account](#)

Local File Upload

No file chosen

URL File Upload

[URL File Upload](#)

Figure 7: Administrator functional page (register and delete visible)

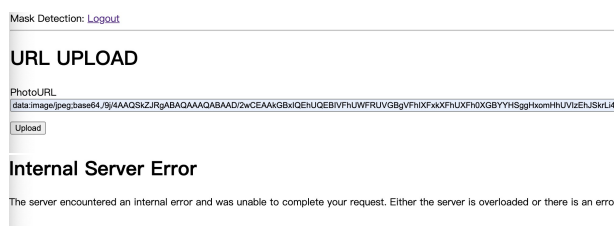


Figure 8: URL upload page (url shouldn't be too long since we set length limit)

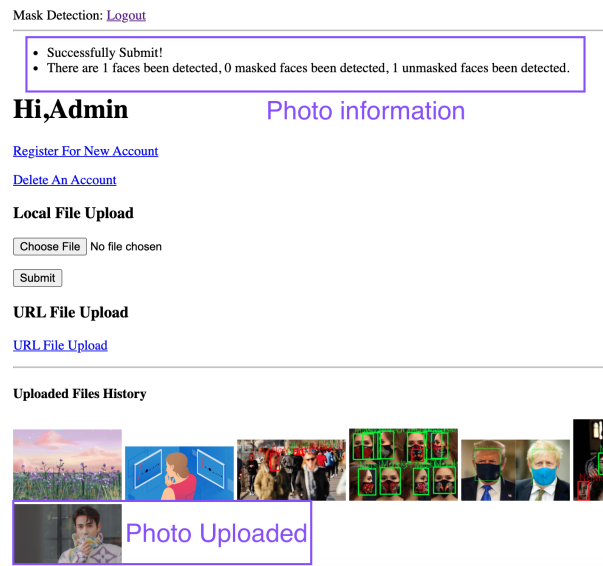


Figure 9: Upload succeeds and presents detected information

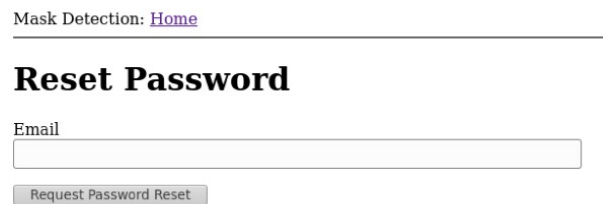


Figure 10: Forget password page

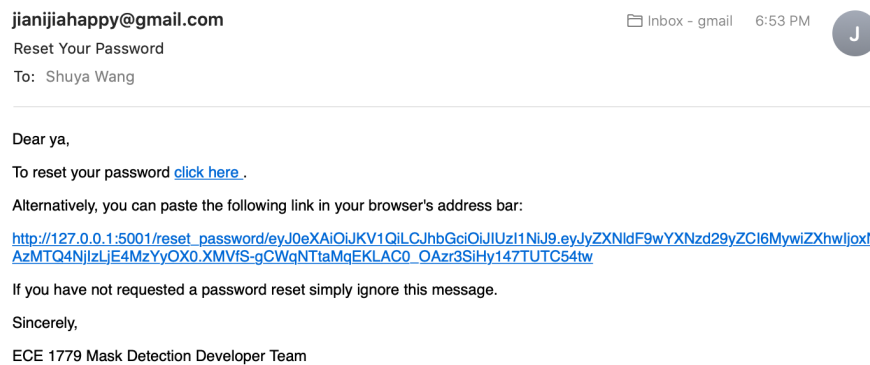


Figure 11: Reset password email

Mask Detection: [Home](#)

Reset Your Password

Password

Repeat Password

Figure 12: Reset password form

Mask Detection: [Logout](#)

Register

Username

Email

Password

Repeat Password

Figure 13: Register page

Mask Detection: [Logout](#)

Register

Username

[Please use a different username.]

Email

[Please use a different email address.]

Password

Repeat Password

Figure 14: Register when user has already existed in database

Mask Detection: [Logout](#)



Delete Account

Username

Email

Delete


Figure 15: Delete page

Untitled Request BUILD  

POST localhost:5000/api/register Send Save

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings Cookies Code

☐ none ☒ form-data ☐ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/> username	testusername3			
<input checked="" type="checkbox"/> password	testpassword3			
<input type="checkbox"/> file	 截屏2020-10-19 下午3.19.48.png X			
<input checked="" type="checkbox"/> email	testemail3@mail.com			
Key	Value	Description		

Body Cookies Headers (4) Test Results Status: 200 OK Time: 370 ms Size: 162 B Save Response

Pretty Raw Preview Visualize JSON ...

```
1 {  
2   "Success": true  
3 }
```







Bootcamp Build Browse    


Figure 16: API register succeeds

Untitled Request BUILD  

POST localhost:5000/api/register Send Save

Params Authorization Headers (8) **Body** Pre-request Script Tests Settings Cookies Code

☐ none ☒ form-data ☐ x-www-form-urlencoded ☐ raw ☐ binary ☐ GraphQL

KEY	VALUE	DESCRIPTION	...	Bulk Edit
<input checked="" type="checkbox"/> username	testusername2			
<input checked="" type="checkbox"/> password	testpassword2			
<input type="checkbox"/> file	 截屏2020-10-19 下午3.19.48.png X			
<input checked="" type="checkbox"/> email	testemail2@mail.com			
Key	Value	Description		

Body Cookies Headers (4) Test Results Status: 400 BAD REQUEST Time: 27 ms Size: 172 B Save Response

Pretty Raw Preview Visualize JSON ...

```
1 {  
2   "Success": false  
3 }
```





Bootcamp Build Browse    

Figure 17: API register Fails

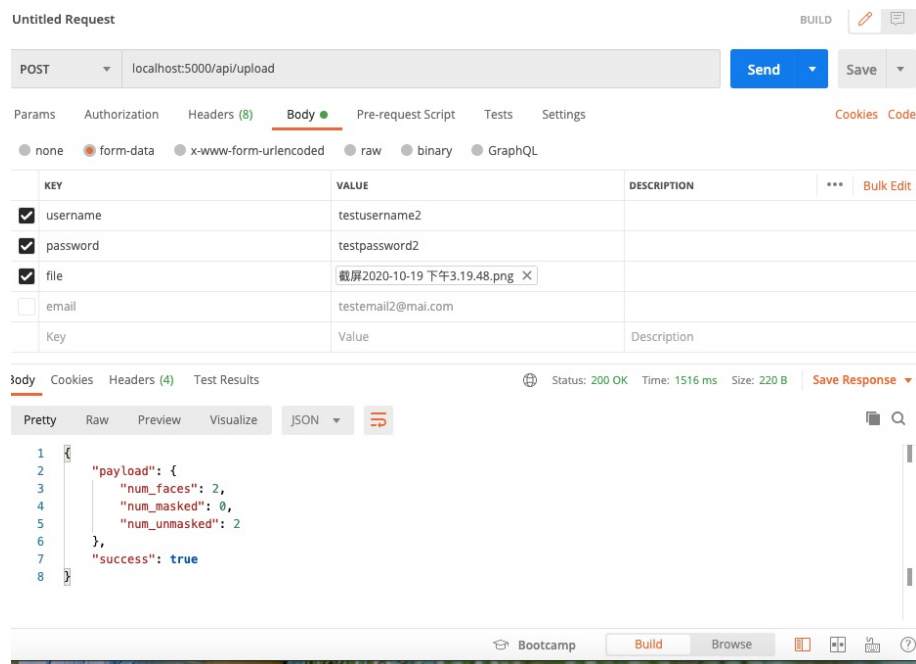


Figure 18: API upload succeeds

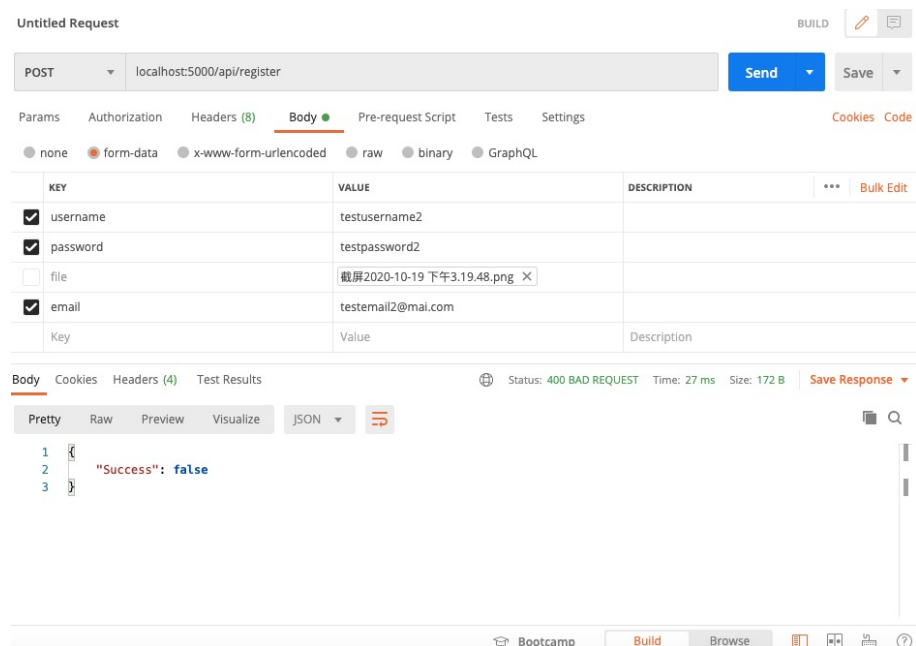


Figure 19: API upload fails