

ECE1779 Assignment 1: Web Application

1. User Guide

1.1 Menu Bar

The menu bar is the most commonly used navigation tool for users. Depending on a user's login status, the menu bar would have different appearances as shown in Figure 1 & 2.



Figure 1. Menu bar when a user is not logged in



Figure 2. Menu bar when a user is logged in

There are six different buttons in total. Each of them would perform certain functionalities or lead users to a web page. Details of different web pages would be described in the next section.

1. “Home” button will lead users to the “Home” page where provides button that allows users to sign up or log in.
2. “Sign up” button takes users to the “Sign Up” page which asks a user for username and password to create a new account.
3. “Login” button takes users to the “Log In” page which asks a user to provide a username and password to log in.
4. “Logout” button is only visible to users that have already logged in. It allows those logged-in users to log out and redirects users to the “Home” page with a message displayed.
5. “Upload” button is only visible to users that have already logged in. Clicking it will take users to the “Upload” page where a user could upload an image.
6. “My Photos” also shows up only if user has already logged in previously. The button will lead user to the “My Photos” page.

1.2 Web Pages

The application contains a few different web pages allowing users to interact with the system. How each web page functions is detailed below.

1. “Home” page is the place that a user gets started. As shown in Figure 3 & 4, it has different appearances for logged in users and visitors.

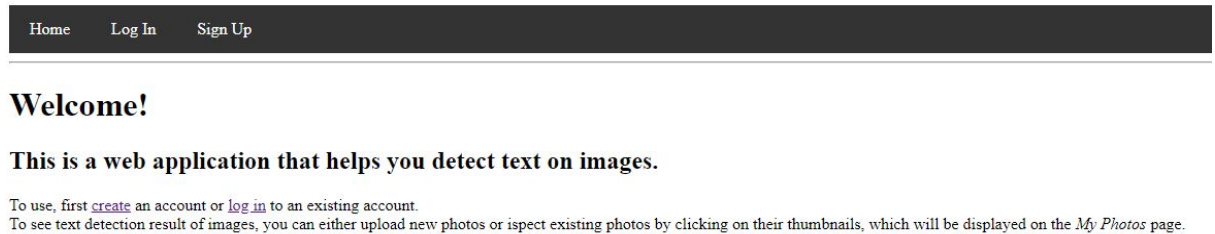


Figure 3. Home page for visitors



Figure 4. Home page for logged in users

It contains a simple welcome message for everyone. For visitors, it provides links to “Sign Up” or “Log In” page. For logged in users, it provides links to “Upload” page or “My Photo” page.

2. “Sign Up”, as shown in Figure 5 below, page is the place where a user could create an account.

The screenshot shows the top navigation bar with links for Home, Log In, and Sign Up. Below the bar, there is a large icon of a person with a plus sign. Below the icon, there are two input fields: "Username" and "Password". The "Username" field has a placeholder text "Enter Username" and the "Password" field has a placeholder text "Enter Password". At the bottom of the form, there is a green button labeled "Sign Up".

Figure 5. Sign Up page

In order to create an account, a user would need to provide a valid username and password. A valid username should be unique and shorter than 50 characters. Neither username nor password field can be empty. If successful, the user will be logged in automatically and be redirected to “My Photos” page with a success message displayed. Otherwise, the user will see an error message and a button that allows the user to go back to the previous page, as shown in Figure 6. Please be noted that only users who are not logged in can access this page, otherwise they will be redirected to “My Photos” page directly.

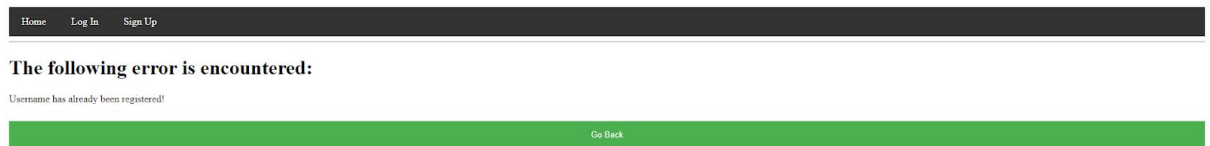


Figure 6. Error due to an invalid username

3. “Login” page, as shown in Figure 7, is a place where a user can log into an existing account.

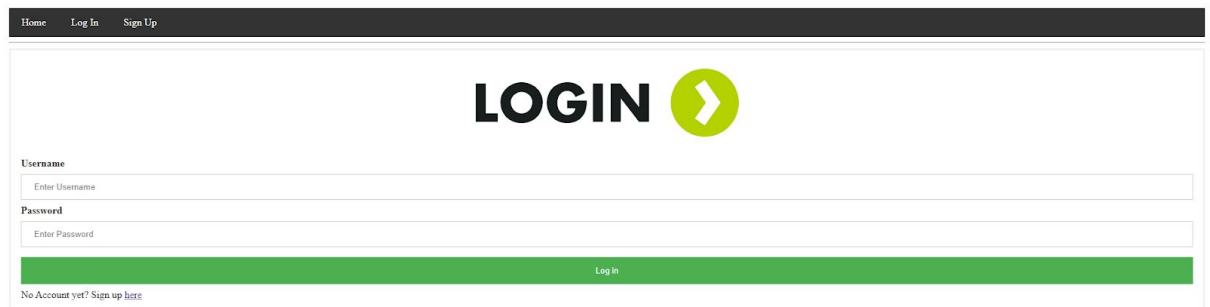


Figure 7. Log In page

If valid credentials are provided, the user will be redirected to the “My Photos” page with a success message displayed. Otherwise, an error message and a go back button will be shown as in Figure 8. Please be noted that only users who are not logged in can access this page, otherwise they will be redirected to “My Photos” page directly.

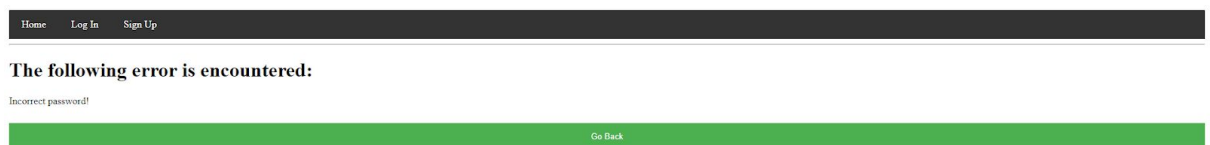


Figure 8. Log In error

4. “Upload” page, as shown in Figure 9, is a place where a logged in user could upload an image.

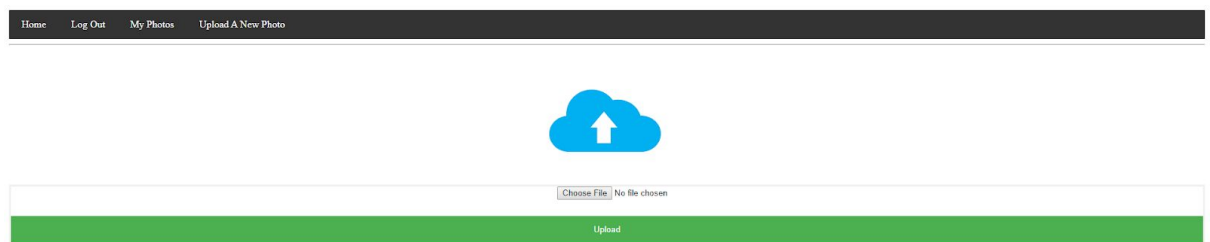


Figure 9. Upload page

If success, user can see a success message and be redirected to the “Display” page. If the image selected by the user does not meet requirements, user will see an error message and a button to go back, as shown in Figure 10. Please be noted that this page is only visible to logged in users.

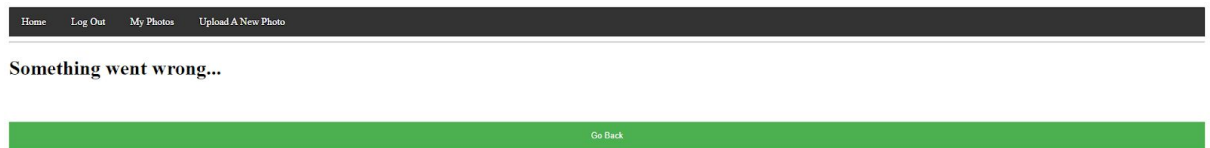


Figure 10. Upload fail due to invalid arguments

5. “My Photos” page is a place where a logged in user can preview thumbnails, as shown in Figure 11. Also, this page is only visible to logged in users.

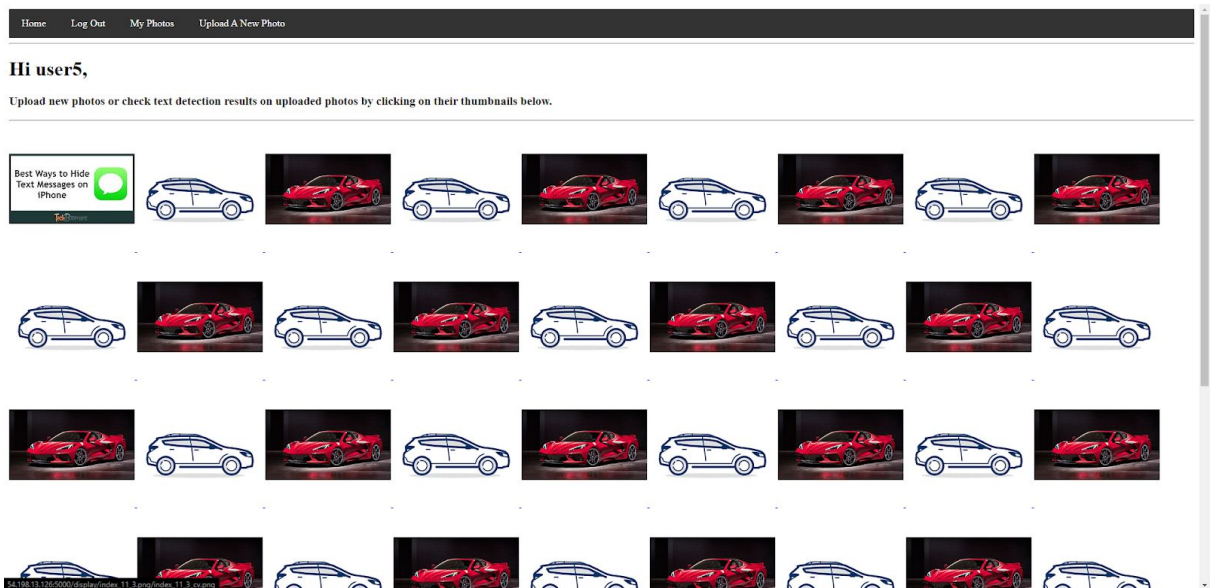


Figure 11. My Photos page

This page displays thumbnails of all the images uploaded previously by the user. Clicking any of the thumbnails will lead the user to the “Display” page of that image.

6. “Display” page can only be accessed from the “Upload” page and the “My Photos” page as described above. As shown in Figure 12, it display the original image and the text detection of the image.



Figure 12. Display page

However, if a malicious user tries to access other users’ images by changing file names in the url, an error message will be shown as in Figure 13.



Figure 13. Wrong access

2. Developer Guide

2.1 Architecture

There are eight web routes included in this web applications: Home, Signup, Login, Logout, My Photos, Upload, Display, and Error.

2.1.1 Navigation

Access to web pages depends on a user's login state:

- 1) Home and Error page can always be accessed.
- 2) Signup and Login pages can only be accessed when a user is not logged in.
 - If a logged in user tries to access them, the user will be redirected to the My Photos.
- 3) My Photos, Upload and Display pages can only be accessed when a user is logged in.
 - If a user who is not logged in tries to access them, the user will be redirected to the Login page.

There are three ways for users to navigate around:

- 1) Navigation bar containing various web links at the top of each web page;
 - As shown in Figure 14, the links included in the navigation bar depend on a user's state:

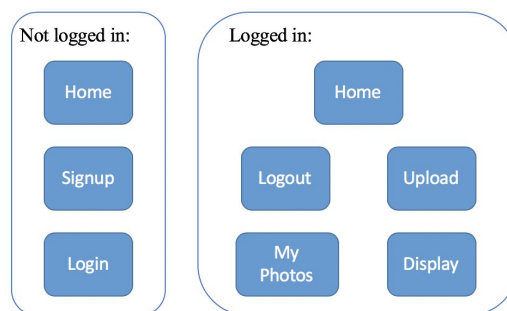


Figure 14. Links included in the navigation bar given a user's state

- 2) Automatic rendering of or redirect to another web page by Flask in response to user actions, including signup, login, logout, upload, and clicking on thumbnails displayed on My Photos;
 - The transition between web pages are summarized in Figure 15 below:

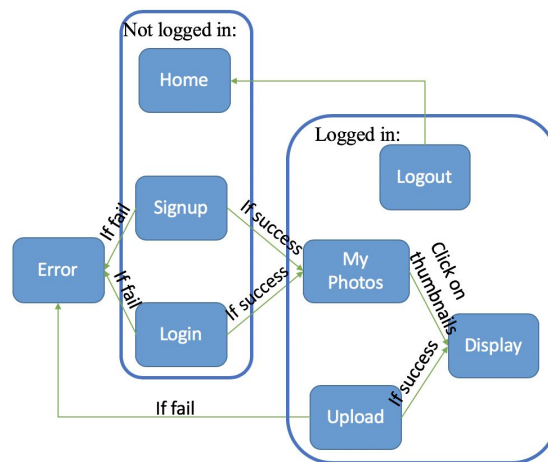


Figure 15. Transition between web pages in response to user actions

3) Type in URL directly.

2.1.2 Code Structure

The structure of the code of this web application is as shown below:

```

webapp (top level folder)
├── A1.py
├── README.md
├── app
│   ├── __init__.py
│   ├── helper.py
│   ├── models.py
│   ├── routes.py
│   ├── static
│   │   └── resource
│   │       ├── login.png
│   │       ├── signup.png
│   │       └── upload.png
│   └── templates
│       ├── base.html
│       ├── display.html
│       ├── error.html
│       ├── home.html
│       ├── login.html
│       ├── myphotos.html
│       ├── signup.html
│       └── upload.html
├── config.py
└── db_config.sql
  
```

```
|— frozen_east_text_detection.pb
|— load_generator.py
|— start.sh
```

Top level directory

A1.py and **start.sh** can be used to run the application. Command to use is `./start.sh` (non-debug gunicorn) or `"python3 A1.py"` (debug mode, since `debug=True` is set). Another option is command `"flask run"` (non-debug mode).

config.py defines environment variables used by Flask, including secret key used to handle sessions, limit maximum file size user can upload, path to the top level folder, and variables used to connect to the database server.

db_config.sql contains definition of the two tables, users and images, in the database that we use to store user account information and user-uploaded image information.

frozen_east_text_detection.pb contains the pre-trained neural network we use to perform text detection on user-uploaded images with OpenCV.

load_generator.py generates loads to our web application that we use to test its functionality.

README.md contains information important for developers to know when trying to work with the code in this directory.

“app” folder

__init__.py contains the initialization of the Flask instance and the database used for our web application, both of which are configured using environment variables defined in **config.py**. The LoginManager from flask-login package that helps with user session management is also initialized here. The route login-required pages will direct users that are not logged in to is set to be the login page.

helper.py contains the text detection function written with OpenCV. This function loads a pre-trained neural network and pass the original image through to identify regions as detected text after thresholding on probability and performing non-maxima suppression. The detected text will be marked with green rectangles, and this OpenCV-processed image will be saved into the local file system.

models.py defines the User class that we use to represent our user. The function `user_loader` that LoginManager from flask-login package uses to identify and load a user for sessions is also used here.

routes.py contains routing and action functions that tells Flask what URL will trigger what to happen. This is the core of the functionality of our web application, which will be detailed in the **Developer Guide - Function** section below.

“static” folder

This folder is where images used by Flask need to be stored. Since we need to display the original user-uploaded photos, their thumbnails and their OpenCV processed version with detected text enclosed in rectangles, this folder in the local file system is where we store them.

The **resource** folder here is where we store all images used in our UI.

“template” folder

This folder is where all html files used by Flask need to be stored.

base.html is what all other html files build on. It contains all css style and code for the navigation bar. The remaining html files, **display.html**, **error.html**, **home.html**, **login.html**, **myphotos.html**, **signup.html** and **upload.html** each corresponds to a different route, and is what is displayed as the webpage when the route its accessed. Note that there is no html file corresponding to the logout route.

2.2 Function and Implementation Considerations

2.2.1 Functions

Registration: With the username and password provided by a user read from the registration form, we first check if those are valid -- no empty field(s) will be allowed, and the username cannot already exist in our database. If the previous conditions are all met, after security checks also pass, we insert information about this new user into the users table in our database, automatically authenticate the user, and redirect the user to My Photos page. Otherwise, we redirect the user to Error page with a message indicating what went wrong.

Authentication: Registered users should be able to login using their username and password. They provide those information through the login form. Both fields need to be non-empty, and we then check if the username already exists in our database. If so, the password will be processed and compared to our records for match. After a user is logged in successfully, they will stay logged in for 24 hours unless they choose to log out, even when the user closes and reopens the website. This can be achieved by setting the session for current user to be permanent for 24 hours.

With the help of the flask-login package, our application can handle different users simultaneously logged in using different computers (through login_user and logout_user functions). A User class is defined to represent users with all required information, and functions are implemented to differentiate users from one another using their unique usernames. The package helps identify which session belongs to which user (through

load_user function and current_user variable), and this knowledge then helps control user access to regular and restricted (i.e. login-required) content and actions (by setting certain routes to have the login_required decorator).

If anything goes wrong during the authentication process, we raise an exception and handle it by redirecting the user to Error page with a message indicating cause.

Upload Images: Only logged-in users can upload new photos to get processed for text detection. Empty upload is not allowed. To allow for images with duplicate names, each uploaded image's name will be extended to include the user id of the user who uploaded it, and the count of photos this user has at the time of current upload. If security checks pass, we save the original image into the app/static folder under the top level directory, then process it to create a thumbnail and a OpenCV-processed version with detected text marked with green rectangles. If both thumbnail creation and OpenCV processing are successfully completed, we save them into the same folder as the original, with their names further extended with "tn" and "cv" respectively, and insert info about this uploaded image into the images table in our database. Only the extended image name, not the actual images, will be saved into our database.

If anything goes wrong during the uploading process, we raise an exception and handle it by redirecting the user to Error page with a message indicating cause.

View Images: Only logged-in users can browse thumbnails of all uploaded photos in My Photos page, and display an original photo and its OpenCV-processed version with detected text side by side in Display page.

The former is achieved by retrieving all extended image names from the images table in the database that have the same userid as the current user. Paths to those thumbnails can then be composed based on the retrieved names, and the thumbnails can then be accessed and displayed.

The user can be directed to Display page by two actions: after the uploading of a new photo is completed, and after clicking on a thumbnail displayed on My Photos page. For the later actions to work, link to Display page of each thumbnail needs to contain two variables indicating the names of the corresponding original image and OpenCV-processed image.

2.2.2 Security

To guarantee the security of our web application, we first control what to store into our database and file system. The username is required to have a length no greater than 50 characters. The password cannot be stored in plain text. We hash it with a per-user salt value and store both the salt and hashed password into our database. For files input by users, we first check if its extension is one of .jpg (.jpeg) or .png. The maximum size allowed for the image input is set to be 10M.

Secondly, we also control the images a logged-in user can access – an image can only be accessed by the user who uploaded it. Because our Display route is implemented in a way that takes in two variables from the URL to achieve being redirected to by clicking on the thumbnails, a logged-in user can see image names from the URL and might even be able to guess our naming convention. With that guessed naming convention, they might try to access images uploaded by other users through URLs with guessed image names. To prevent this illegal access from happening, we retrieve the entries that has those image names passed through URL and check if their corresponding user ids match that of the current user. If so, we proceed to display; otherwise we redirect user to Error page warning about the wrong access.

2.2.3 Database

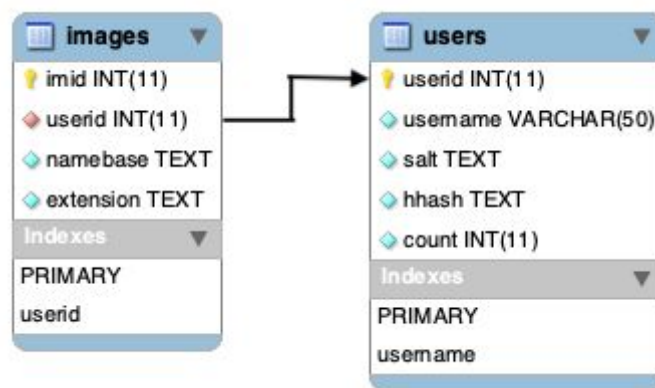


Figure 16. Database structure

The database contains two tables: users and images. The schema is shown in Figure 16 above. Details of the two tables are described below.

Table “users” stores user information. Column “userid” is the primary key. Each user also has a unique “username”. A per-user salt value would be stored in column “salt”. Column “hhash” stores hash value of (salt + password). Therefore, the system could easily verify a user’s password without storing it in clear text. Column “count” tracks how many photos a user have uploaded so far. By appending a user’s “userid” and “count” to each of his/her images, the system could ensure that no duplicate names are shared by two images.

Table “images” stores information of every image. Column “imid” is the primary key. Column “namebase” stores names, excluding extensions, of every image. As described above, a “namebase” is a combination of an image file name, its owner’s “userid” and “count”. Therefore, the system can ensure that no duplicate “namebase” across the whole system. Column “extension” stores file extensions. Column “userid” is a foreign key referencing “userid” in table “users”. It can be used to retrieve an image's owner’s

information, ensure an image can only be viewed by its owner and ensure that only existing users can upload images.

Since in both tables, every field depends only on the primary keys and there is no transitive dependencies, the database is 3NF normalized.

2.2.4 Success/Error Handling and Messages

To display appropriate error and success messages during user interaction with the web application, two mechanisms have been adopted.

For success messages, we flash them in the space right under the navigation bar on the web page the user is redirected to after a certain action. This includes the successful creation of a user account, a successful login and logout, and a successful image upload.

For error messages, we render the template of the Error page, with a specific error message passed with it. Failure cases include: empty user inputs, username input to signup form already exists in database, username is longer than 50 characters, username input to login form does not exist in database, input password to login form does not match the password from the database corresponding to the input username, input image exceeds 10M, thumbnail creation failed, and OpenCV processing failed.

Other than the failure causes specified above, we are also prepared to handle any other exceptions whenever database is accessed, although we don't normally expect things to go wrong. As a result, a general error message telling users that something went wrong and they should redo the actions will be displayed on the Error page.

If an error happens when we are trying to modify the content of the database, we will roll back to keep the database free of defects.

2.3 Testing

2.3.1 Unit Test

"Postman" is used to send different requests to each endpoint so we could test them one by one. Benefits of using "Postman" are that we could only test the backend logic without frontend and we could bypass frontend to test server side checks for corner cases. An example of sending request to register with empty username and password, which is not allowed by the frontend, is shown in Figure 17 below.

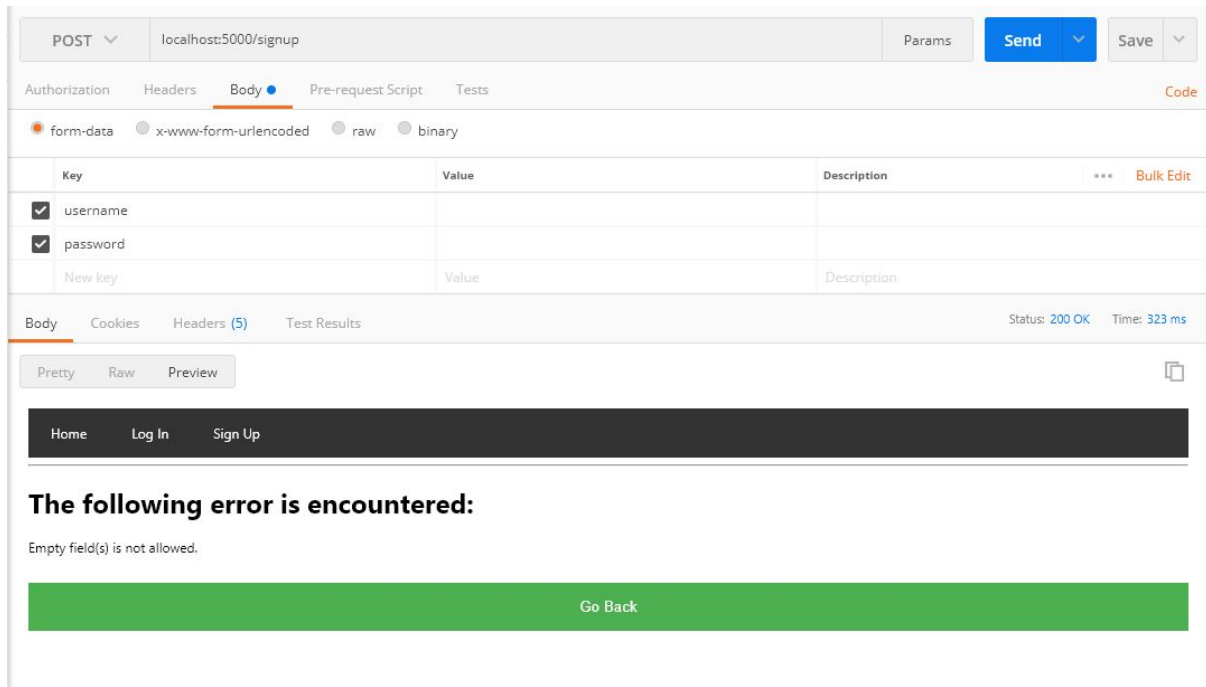


Figure 17. Registration with empty fields blocked by the server

2.3.2 Test Locally and on Cloud Environment

After the backend logic is properly tested, we could test the connection between backend and frontend as well as usability of the UI. The easiest way is to test the whole application locally. We let the application running on “localhost:5000” so that we could test each feature and make modification quickly. After local test is done, we migrate the whole application to our EC2 instance and allow inbound traffic on the instance. Therefore, we could test our website in everyday circumstances. Figure 18 below shows an example of uploading an image from one of ours personal computers.

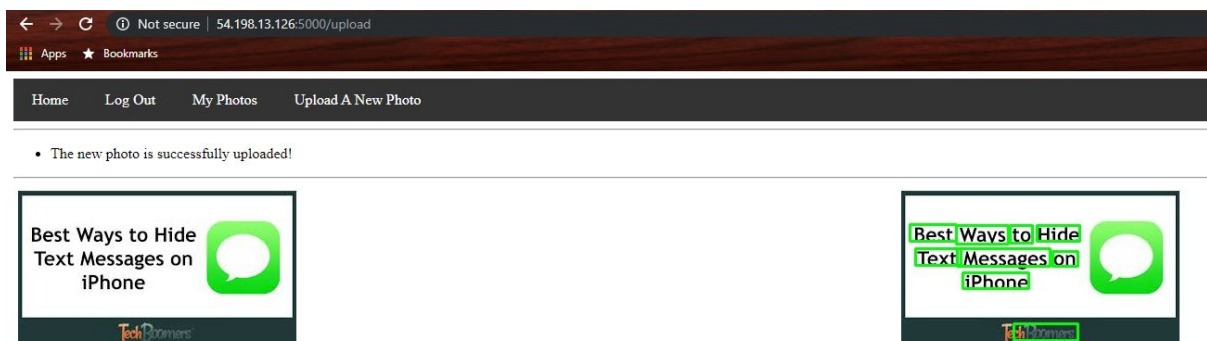
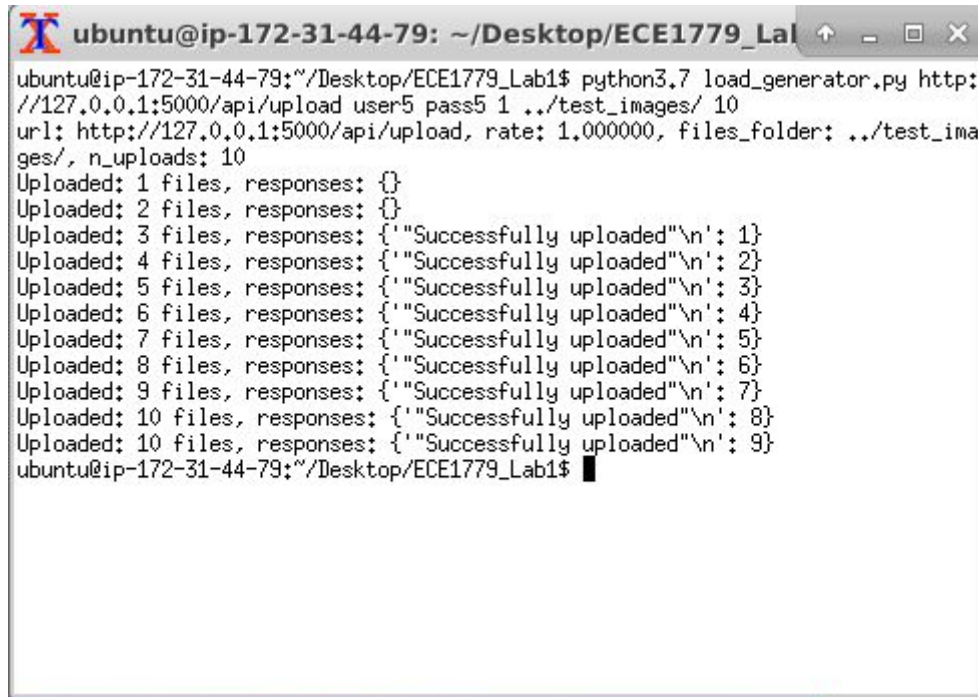


Figure 18. Successfully uploading an image to the server

2.3.3 Automated Test

Besides manual tests described above, automated tests are also done to test the system. Two endpoints “/api/register” and “/api/upload” are implemented. The first one can be tested by

using “Postman” as described in “unit test” section. The second one can be tested by the script “load_generator.py”. It would automatically call this endpoint to upload images. An example output of running the script on our weblocation is shown in Figure 19 below.

A terminal window titled 'ubuntu@ip-172-31-44-79: ~/Desktop/ECE1779_Lab1' showing the execution of a Python script. The command is 'python3.7 load_generator.py http://127.0.0.1:5000/api/upload user5 pass5 1 ../test_images/ 10'. The output shows the script uploading 10 files, with each upload step reporting 'Successfully uploaded' and a count. The final prompt is 'ubuntu@ip-172-31-44-79:~/Desktop/ECE1779_Lab1\$'.

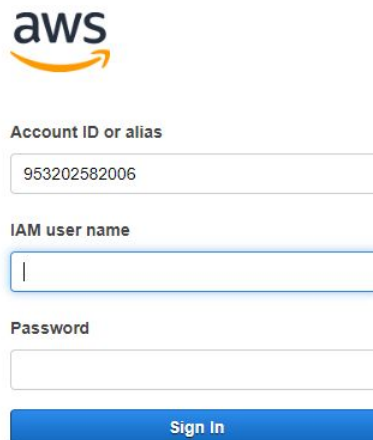
```
ubuntu@ip-172-31-44-79: ~/Desktop/ECE1779_Lab1$ python3.7 load_generator.py http://127.0.0.1:5000/api/upload user5 pass5 1 ../test_images/ 10
url: http://127.0.0.1:5000/api/upload, rate: 1.000000, files_folder: ../test_images/, n_uploads: 10
Uploaded: 1 files, responses: {}
Uploaded: 2 files, responses: {}
Uploaded: 3 files, responses: {'Successfully uploaded'\n': 1}
Uploaded: 4 files, responses: {'Successfully uploaded'\n': 2}
Uploaded: 5 files, responses: {'Successfully uploaded'\n': 3}
Uploaded: 6 files, responses: {'Successfully uploaded'\n': 4}
Uploaded: 7 files, responses: {'Successfully uploaded'\n': 5}
Uploaded: 8 files, responses: {'Successfully uploaded'\n': 6}
Uploaded: 9 files, responses: {'Successfully uploaded'\n': 7}
Uploaded: 10 files, responses: {'Successfully uploaded'\n': 8}
Uploaded: 10 files, responses: {'Successfully uploaded'\n': 9}
ubuntu@ip-172-31-44-79:~/Desktop/ECE1779_Lab1$
```

Figure 19. Load generator’s output

2.4 How-Tos

2.4.1 AWS Console Login

Go to <https://953202582006.signin.aws.amazon.com/console> (as shown in Figure 20 below) and use the IAM credentials provided in “credentials.txt” to sign in.

The AWS IAM login panel features the AWS logo at the top. Below it are three input fields: 'Account ID or alias' with the value '953202582006', 'IAM user name' which is empty, and 'Password' which is also empty. A blue 'Sign In' button is positioned at the bottom of the form.

aws

Account ID or alias

953202582006

IAM user name

Password

Sign In

Figure 20. AWS IAM login panel

2.4.2 Start/Stop the Instance

After logged in, navigate to “Services -> EC2 -> Instances”. Please make sure the location is “US East (N. Virginia)”. You should see the only instance for this project. Then, you could right click it to start/stop the instance, as shown in Figure 20 below.

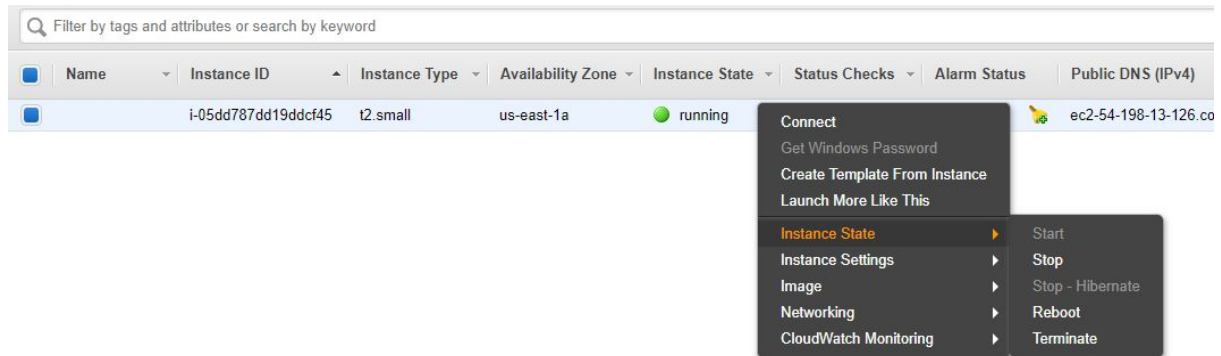


Figure 20. Start/Stop the instance

2.4.3 Connect to the Instance

You could SSH into the instance (`ssh -i keypair.pem ubuntu@Public IP`). The public IP of the instance is shown in the “Public DNS” field in Figure 20 above. A private key file “keypair.pem” is provided. You may need to give correct permissions to the file in order to use it if you are on Linux or Mac OS. Use command “`chmod 400 keypair.pem`” to do so.

2.4.4 Run the Application

Navigate to “/home/ubuntu/Desktop/ECE1779_Lab1”, which is the home directory for the application. Open terminal and simply type “`./start.sh`”. It starts the application on “0.0.0.0:5000” using Gunicorn. You could also type “`python A1.py`” to start the application in debug mode. Now, the application can be accessed anywhere in the world at “Public IP:5000”.

Team Member

Shi, Hu (1001416498, rock.hu@mail.utoronto.ca)

Sining, Qin (1001181106, sining.qin@mail.utoronto.ca)

Jian, Shi (1001841536, jianisme.shi@mail.utoronto.ca)