ECE1779: Introduction to Cloud Computing

Fall 2019

Assignment 1

Web Development

Due Date

Oct 18

Objective

This assignment will provide you with experience developing a web application using Python and Flask. You will also get experience deploying and running your application on Amazon EC2.

Web Application Description

In this assignment, you should develop a simple web application for text detection on photos. Your application should support the following functionality through web:

1. Login and registration for users: New users should be able to register by providing a username and password in a registration form. Already registered users should be able to login using their username and password. Users should also be able to logout if they want.
2. Uploading new photo: A logged in user should be able to upload a new photo. After a new photo is uploaded, your application should automatically draw rectangles around texts detected in the photo and save the result, a thumbnail of the result, as well as the original photo.
3. Browsing uploaded photos: A page that lets logged in users browse thumbnails of their uploaded photos. Clicking on a photo's thumbnail should produce the original version of the photo as well the photos with rectangles around texts.

Notes for implementation:

1. Use Flask framework.
2. You can use the popular ImageMagick tool to create thumbnails.
3. Use the popular OpenCV library for the text detection and drawing rectangles around texts. You can find tutorials on how to do so here.

Requirements

1. All photos should be stored in the local file system (i.e., on the virtual hard drive of the EC2 instance). A user might upload different photos having similar names. In this case, your application should treat them as different photos, obviously.
2. Store information about user accounts and the location of photos owned by a user in a relational database. Do **not** store the photos themselves in the database. It is up to you to design the database schema, but make sure that you follow design practices and that your database schema is properly normalized.
3. Follow basic web application security principles.
    - All inputs should be validated with reasonable assumptions (for example, do not let the user upload unreasonably big files, do not let the user use a username with more than 100 characters). Also, do not rely on browser-side validation and do the validation on the server side too.
    - User passwords should not be stored in clear text in the database. Instead, store the hash of the password concatenated with a per-user salt value. Details for how to do this are available here.
    - Access to stored photos should be restricted to the user that has uploaded them.

4. Appropriate error and success messages should be displayed to the user during user interaction with the web application
5. Logged in users should stay logged in for at least 24 hours. Note that there might be different users simultaneously logged in into your applications using different computers.
6. To simplify testing, your application should **also (in addition to your web interface)** include two URL endpoints as APIs that makes it possible for the TA to automatically register users and upload photos. These endpoints should conform to the following interfaces:

   Register:

   ```
         relative URL = /api/register
         method = POST
         POST parameter: name = username, type = string
         POST parameter: name = password, type = string
   ```

   Upload:

   ```
         relative URL = /api/upload
         enctype = multipart/form-data
         method = POST
         POST parameter: name = username, type = string
         POST parameter: name = password, type = string
         POST parameter: name = file, type = file
   ```

   - These endpoints should generate appropriate HTTP response codes in success and error conditions as well as responses in JSON format
   - Sample forms that conform to the specification above is available here and here.
   - A load generator that conforms to the upload interface is available here (Note that the load generator should be run using python 3.7+). You can use the load generator to test your application. **Warning:** the load generator creates a lot of network traffic. To minimize bandwidth charges, you can use the load generator inside EC2 only.

   To run the load generator, run the python script using:

```
      python3.7 gen.py upload_url username password
upload_rate(upload_per_second)     image_folder number_of_uploads
```

   for example:

```
      python3.7 gen.py http://9.9.9.9:5000/api/upload user pass 1
./my-photos/ 100
```

7. Your application should be deployed on a single AWS EC2 instance of type t2.small
8. All code should be properly formatted and documented

Deliverables

We will test your application using your AWS account. For this purpose, your application should be pre-loaded on an EC2 instance. Please **suspend** the instance when you submit your project to prevent charges from occurring while the TA gets around to grading your submission. Make sure to not restart the instance from the moment you submit your project.

You should write a shell or bash script called **start.sh** that initializes your web application. This script should be put in Desktop folder inside the EC2 instance. Your web application should run on port **5000** and be accessible from outside the instance. So, make sure that you open this port on your EC2 instance. Documentation about how you can open the port can be found here.

It is preferred to use gunicorn or uwsgi to start your Flask web application.

Submit the assignment only once per group. Clearly identify the names and student IDs of group members in the documentation.
To submit your project, upload to Quercus a single tar file with the following information:

1. User and developer documentation in a **PDF** file (documentation.pdf). Include a description of how to use your web application as well as the general architecture of your application (using figures and diagrams if needed). Also include a figure describing your database schema. Click here for tips on how to write documentation. Your documentation will be marked based on how cohesive it is and how well you are able to explain the technical details of your implementation.
2. In addition to the documentation, put name and student ID of each student in a separate line in a text file named group.txt .
3. If you are using the AWS educate account (link provided in the class) with the $50 credit, then we don't need your AWS credentials as we already have access to your AWS console.

   Otherwise, please send us your AWS credentials in a text file (named credentials.txt). We will need these to log into your account. You can create credentials with limited privileges using AWS IAM if you don't want to share your password with the TA; however, make sure that you include permissions to start and stop EC2 instances. Test the credential to make sure they work.

4. Key-pair used to ssh into the EC2 instance (named keypair.pem).

   Do not forget to send the .pem files required for ssh-ing into your VM's.

5. Anything else you think is needed to understand your code and how it works.

Marking Scheme

1. UI/UX: Being able to navigate through all pages easily (using sensible menus and buttons), properly showing the photos **(4 points)**
2. Functionality: Text detection functionality, handling exceptions and corner cases, no crashes or bugs, your web application should take a reasonable time to process requests, correct database design, proper photo storage **(7 points)**
3. Test on load generator: We shall test your web application using the load generator provided. The load generator requires the upload API(explained above) to function. Make sure that your web application works using the load generator **(3 points)**
4. Security: Input validation, implementing sessions, using salts and hashes for passwords, correct mechanism for logging in and registering users **(4 points)**
5. Documentation: All code should be properly formatted and documented, explanations about how to log in to your amazon account, how to start the instance and how to run the code should be included. You should also write about the general architecture of your code and how different elements of your code are connected to each other. Don't forget about the database schema or group members **(7 points)**

Note that you might lose points on items 2-4 if the TA is unable to test your application (for example, if API endpoints don't work properly, your start.sh does not work, you sent wrong keypair.pem or you forgot to open required ports on EC2 instance). Make sure that everything is in place.

Resources

Amazon provides free credits for students to experiment with its cloud infrastructure, including EC2. To apply for an educational account go to Amazon Educate.
The following video tutorials show how to :

- Create an EC2 instance
- Connect to an instance using a VNC Client.
  - using the command: ssh -i key.pem ubuntu@IP -L 5901:localhost:5901 you should first do port forwarding using an ssh tunnel to your instance.
  - If you are using Putty in windows, you can add a tunnel after you are successfully connected to your instance. Right click on the top of PuTTY console window and select Change Settings.... On the left menu, select Connection->SSH->Tunnels. Enter "5901" on the Source port and "localhost:5901" on the Destination fields and then click Add. The tunnel should be added to the list. Now click Apply.

    - After creating a tunnel, you can use any VNC viewer to connect to "localhost:5901" using the password "ece1779pass"
- Suspend an instance

To help you get started, an AMI (id: `ami-080bb209625b6f74f`) is provided with the following software:
Note: the AMI is only available in the N. Virginia AWS Region

- Python (default 2.7), Python 3.5, Python 3.7
- PyCharm IDE
- Firefox
- MySQL Server (root password ece1779pass)
- mysql-workbench
- vncserver (password ece1779pass)
- Database and AWS Flask examples covered in lectures and tutorials.
- ImageMagick
- OpenCV libraries
- Python Wand ImageMagick bindings
- Python OpenCV binding (cv2)
- gunicorn
  This is high performance server for Python-based applications. For an example of how to run it, look at the run.sh file inside Desktop/ece1779/databases
- In addition the directory in Desktop/ece1779 contains the following:
    - **databases:** A PyCharm project with all examples from the databases lecture and tutorial
    - **extras:** A PyCharm project with code for transcoding photos and a sample form that conforms to the load testing interface