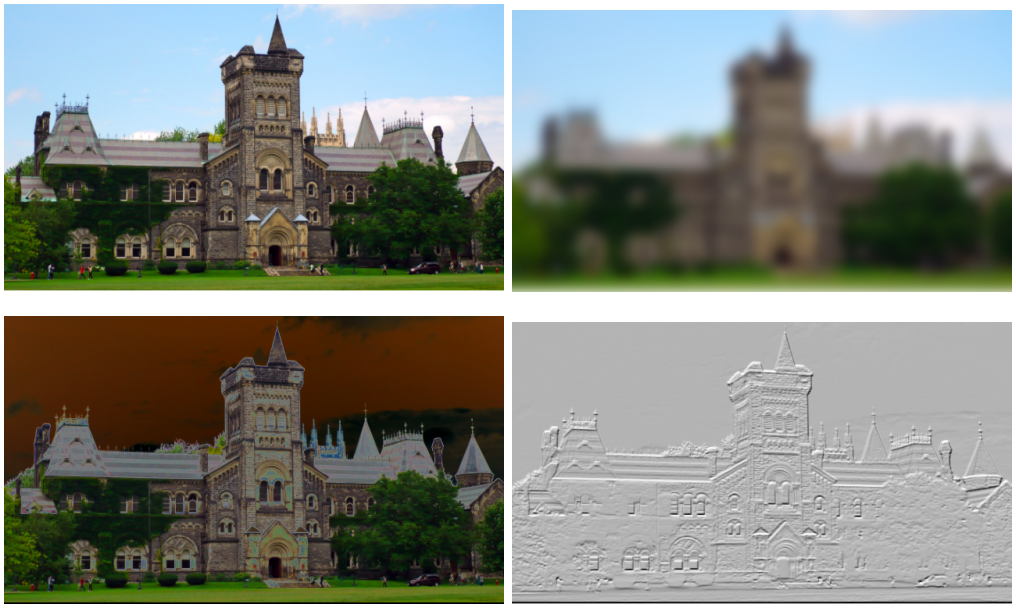
ECE1779: Introduction to Cloud Computing

Fall 2021

Assignment 1

Web Development



Due Date

October 19th, 2021

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 will develop a web application for storing and browsing photographs. Your application should provide the following functionality through a web interface:

1. Login panel. Access to the application should require authentication. Include support for password recovery.

2. User management. The administrator should be able to create additional user accounts. This is not a public application. The only way to create new accounts should be for the administrator to login and create the account. User accounts should have access to all features in the web site, with the exception that they should not be able to create or delete user accounts. All users should have the ability to change their password.
3. Image upload. An authenticated user should be allowed to upload a new image by selecting it from their local file system, or by typing a URL for a location on the web from which the image can be downloaded.
4. Image transformation. After a new image is uploaded, your application should automatically create an image thumbnail, as well as the following 3 specific transformations using the popular ImageMagick library:
    a. blur(radius=0, sigma=8)
    b. shade(gray=True, azimuth=286.0, elevation=45.0)
    c. spread(radius=8.0)
5. Browsing. Authenticated users should be able to browse through thumbnails of their previously uploaded images. Clicking on a photo's thumbnail should produce the full resolution version of the photo as well as its 3 transformations.

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.
2. Store information about user accounts and the location of photos 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. To allow for automatic testing, your application should **(in addition to your web interface)** include two URL endpoints to automatically register users and upload photos. Our automatic testing script will send an **HTTP POST** to your URL endpoints with the parameters in the body of the HTTP request. Your implementation should not accept HTTP requests where parameters have been appended to the URL. See this link for more information about how HTTP POST requests work: https://www.w3schools.com/tags/ref_httpmethods.asp

4. The two automatic testing 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
   ```

   - Sample forms that conform to these specifications are available here and here.
   - These endpoints should generate responses in the following JSON format:
     In case of failure:

     ```
     {
         "success": false,
         "error": {
             "code": servererrorcode,
             "message": "Error message!"
         }
     }
     ```

     In case of success for the ***register*** interface:

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

     In case of success for the ***upload*** interface:

     ```
     {
         "success": true,
         "payload": {
                 "original_size": number,
                 "blur_size": number,
                 "shade_size": number,
                 "spread_size": number
         }
     }
     ```

For example, a successful call to the upload interface with the file UofT.png should produce the response:

```
{
        "success": true,
        "payload": {
                    "original_size": 486695,
                    "blur_size": 137613,
                    "shade_size": 110209,
                    "spread_size": 583960
        }
}
```

5. Follow basic web application security principles.
    - All inputs should be validated with reasonable assumptions (for example, do not let the user upload files larger than 10 MB). Also, do not rely only 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.
6. Appropriate error and success messages should be displayed to the user during user interaction with the web application
7. Your application should support different users simultaneously logged in into your applications using different computers.
8. Deploy your application including the database on a single AWS EC2 instance of type t2.micro. This instance type is eligible for the AWS free tier.
9. 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 the home directory of the user associated with the EC2 keypair (e.g., /home/ubuntu). 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 the first name, last name, and student ID of each student in a separate line in a text file named group.txt. Please make sure that the first and last names in this group.txt file match exactly how your name is displayed in Quercus.
3. Include 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 (20 points)

1. UI/UX: Being able to navigate through all pages easily (using sensible menus and buttons), properly showing the photos. While this is not a design course, it is expected that you application will include reasonable styling to provide a pleasant user experience (**3 points)**
2. Functionality: implementation of features listed in the Web Application Description section. (**6 points**)
3. Correctness: Handling exceptions and corner cases, no crashes or bugs, correct database design, proper photo storage **(2 points)**
4. Testing interface: Our automatic testing requires the upload API(explained above) to function **(2 points)**
5. Security: Input validation, using salts and hashes for passwords **(1 points)**
6. 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 **(6 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 limited free access to its infrastructure to new users through its free tier. Information about the AWS free tier, including how to apply for an account is available at https://aws.amazon.com/free. You should complete this assignment using exclusively resources from the free tier.

Important note: You AWS account is fully functional which means that it will incur charges if you use resources that do not qualify for the free tier, or if you use free-tier qualifying resources beyond the quota that Amazon provides (e.g., a 750 hour per month of EC2 t2.micro, 30 GB of EBS). You will be responsible for any charges. To avoid incurring charges make sure to use only resources from the free tier and remember to suspend your EC2 instances when not in active use.

The following video tutorials show how to :

- Create an EC2 instance
- Connect to the new instance using ssh
- Suspend an instance

To help you get started, an AMI (id: ami-080ff70d8f5b80ba5) is provided with the following software:
Note: the AMI is only available in the N. Virginia AWS Region

- Python
- MySQL Server (root password ece1779pass)
- Database and AWS Flask examples covered in lectures
- ImageMagick
- Python Wand ImageMagick bindings
- gunicorn
  This is a high performance server for Python-based applications. For an example of how to run it, look at the run.sh file inside /home/ubuntu/extras/run.sh
- 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