Assignment 1: Image Gallery
ECE1779 A1

2020.02.16

Group member:

- Ruixin Huang 1001781565
- Lichen Liu 1001721498
- Baiwu Zhang 1001127540

1 Introduction

In Assignment 1 for ECE1779: Introduction to Cloud Computing, we build a website that can register users, upload photos from users, and detect objects in photos. The website is built on an AWS EC2 t2.small instance, with Flask as the backbone framework and MySQL as the database.

This document will provide a comprehensive review of our website. Section 2 will introduce the usage and navigation of the website. Section 3 will illustrate how the server code is architectured, as well as how modularization is achieved. Section 4 will describe each code module in detail.

To start the server, please start and log into our EC2 instance i-027c310cbfb8fb455 (created under baiwu.zhang@mail.utoronto.ca using AWS Educate Account), and run 'bash start.sh' from the Desktop. Once started, our website can be accessed either locally (http://0.0.0.0:5000) or from outside (http://3.231.61.127:5000/). If you use a ssh connection to the instance, please make sure the ssh terminal will not close during your testing. Otherwise, you can use tmux to create a detached terminal session and run 'bash start.sh' from it.

2 Usage

Once connected to the website, users will see a login page. The login page is the entry point for users to register and login to the website. For users that have logged in and with a session stored in the browser, the website will automatically take them to the photo management page.

2.1 Login page

- A new user is required to register a new account. This can be done so by clicking the "Register" button. A new form will pop up which requires the user to choose a username and password. Click "Confirm" to confirm new user registration and click "Cancel" to return to the login page. After clicking "Confirm" and if the username and password have been validated, a new account is created and the new user will be automatically logged in.
- A returning user can log in using their username and password at the login page.

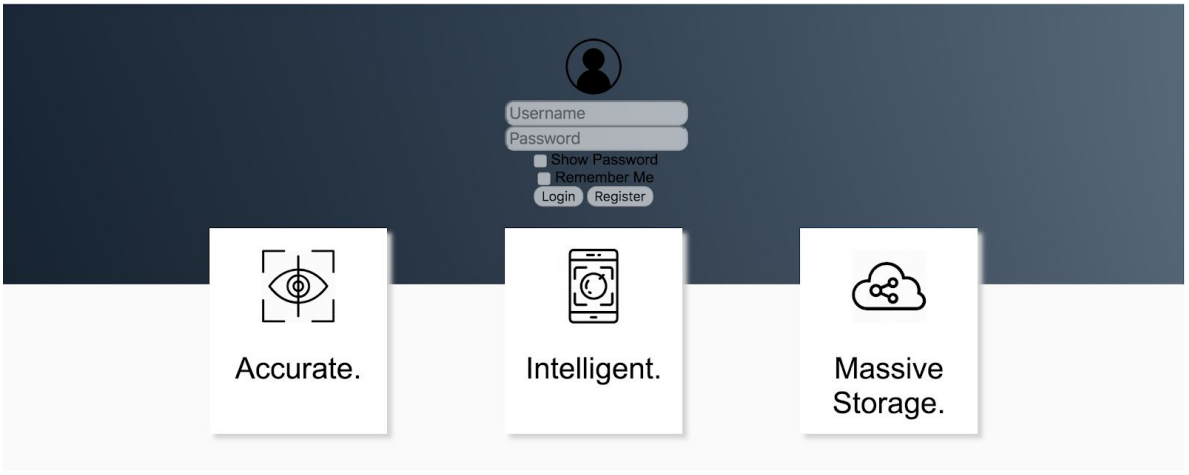

Figure 1: Shows the login page

2.2 Photo management page

- After a successful login, the user will be directed to the photo management page.
- A user can upload a new photo by clicking the “Choose photo” button, which opens the user's local directory and the user can navigate through his files and upload an image. After uploading, the user can click “Submit” to submit the image to the website for image detection.
  Note: The website supports files of .png, .jpg, .jpeg, and .gif formats. The maximum size for each file is 20MB.
- Thumbnails of processed photos of the current user will be displayed in the gallery on the lower part of the photo management page. By hovering the mouse over any thumbnail, the name of the corresponding uploaded image will show.
- All the thumbnails displayed are clickable, and by clicking on them, the user will be directed to a page that displays the original photo and the processed photo in full size.
- The user can also log out by clicking the “Logout” button.
- The “Hit to refresh” can be used as when an uploaded photo is not immediately displayed in the gallery section, because it is still being processed by the website. “Hit to refresh” will update the thumbnail gallery after a photo is processed completely on the website

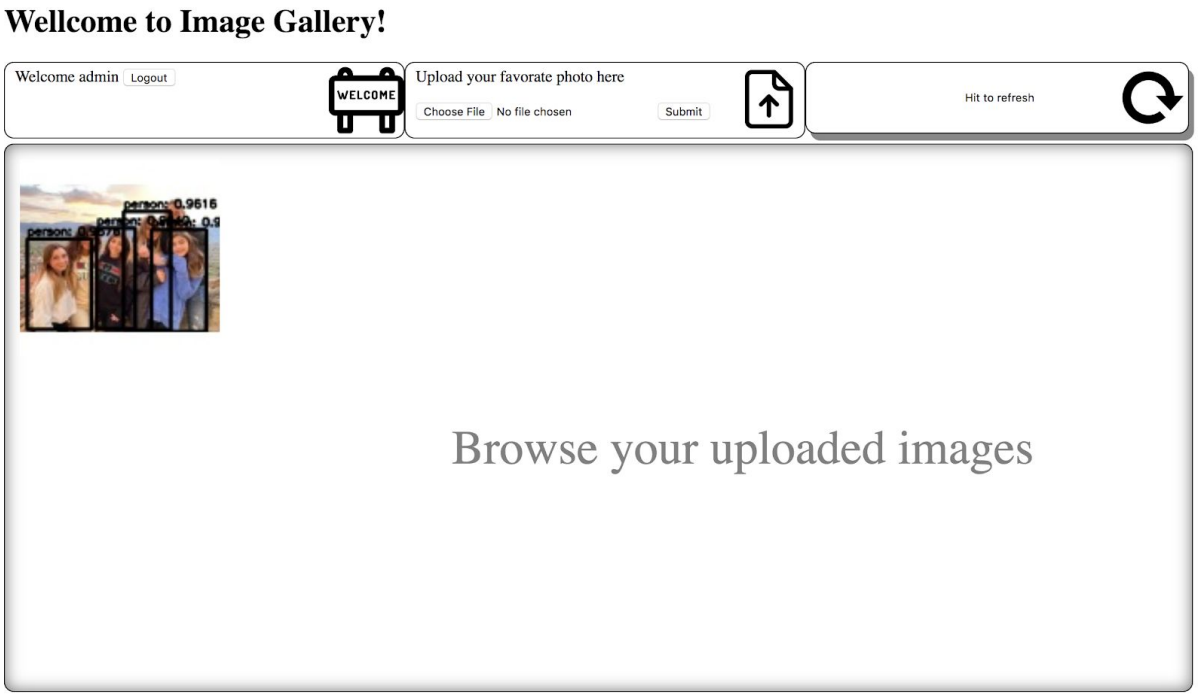

Figure 2: Shows photo management page

2.3 Full-size image display page

- Both the full-size original photo and the processed photo are displayed on this page side by side.
- The user can choose to go back to the “Photo management page” by clicking the “Home” button. Or delete the current photo (both original and processed version) by clicking the “Delete” button.

Figure 3: Shows full- sized photo display page

3 Server Overall Architecture

The server is responsible for handling all user requests and queries when they click buttons or view pages of our website. It is implemented using the Flask framework in Python 3.7. The server application in the framework is event-driven. The server is designed in such a way that it not only responds to requests correctly, but is also less error-prone to programming mistakes and easier for future developments. The server can be divided into three layers: request handler functions, business-logic functions, and persistent storage.

3.1 Request Handling

Request handler functions are directly tied to handling specific HTTP requests. These functions first grab arguments and forms from the request. The data is then passed into the business-logic functions. And finally, the server redirects users or renders corresponding templated HTML pages. The decouplings between request handling logics and business logics allows the server to be more flexible in terms of web Application Programming Interface (API). For example, the registration action performed on the HTML page, and performed directly from HTTP request might need to handle the requests differently but are using the same business logic function. In other words, the front-end designer now can focus on user interactions only.

3.2 Business Logic

Business-logic functions form the core layer that delivers our product to the user. Registering a new account into the database, confirming the user username and password for login, handling user photo updates and showing the photo that a user has uploaded, are all examples for business logic functionality. As mentioned in the previous paragraph, this layer is created to maximize code reuse and flexibility. That is, multiple request handler functions can call into a particular business-logic function easily. All top-level business-logic functions are assumption-free and will return failure messages if any. Assumption-free is essential for code reuse. For example, the account login function would take care of the situation that the user has already logged in. Caller functions in the request handling layer should not need to be aware of any business logics or even assumptions to them. The business logic functions also return failure messages to caller functions if necessary, so that the request handler side caller functions can decide the way to present the message by itself.

3.3 Persistent Storage

The website is using both MySQL database and filesystem for persistent storage. The file system is only used for storing images uploaded by the user, the generated object-detected images, and thumbnails. The three versions of the same user image are stored in three separate directories, with the same file name generated based on the photo ID. On the other hand, MySQL database stores account data, and photo data. The diagram below shows the schema of the database. The server application has a separate module for querying and updating the MySQL database, and it will be explained later.

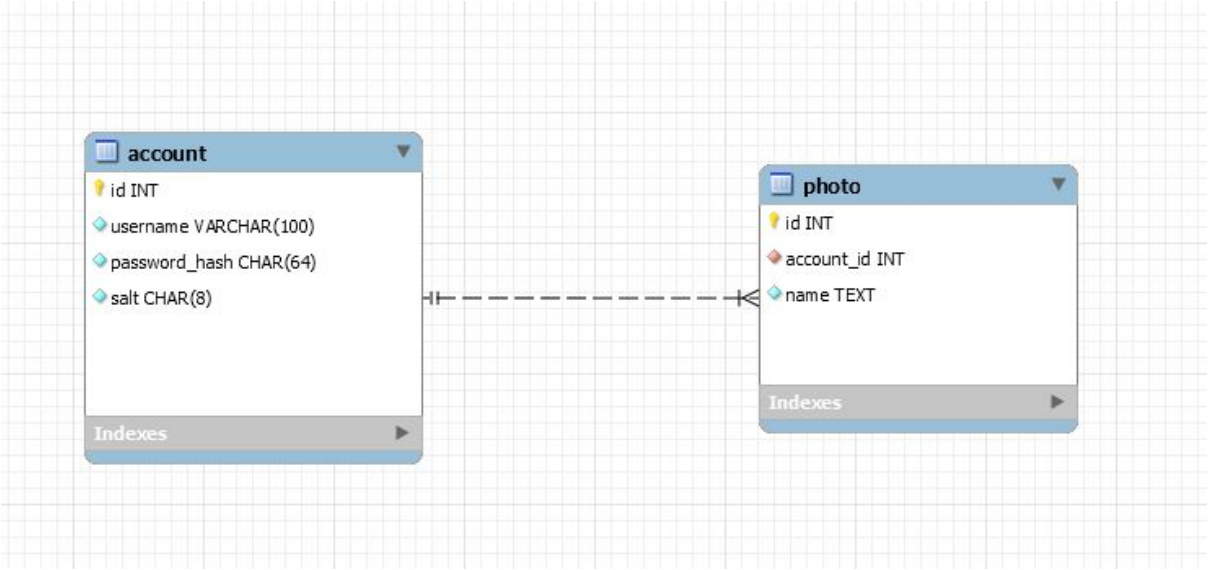

Figure 4: Database schema.

4 Server Modules

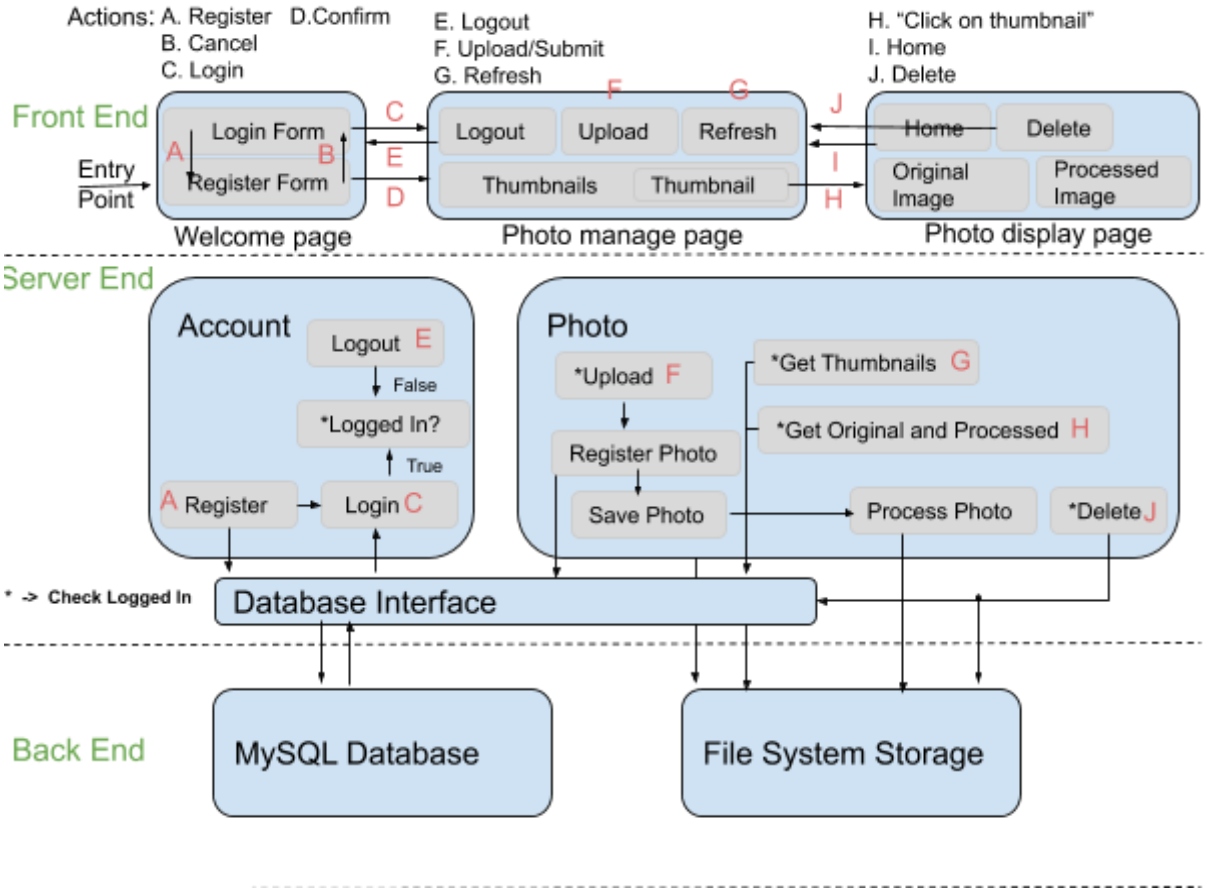

Figure 5: Webpage Interaction Logic and the Underlying Modules

4.1 Account

The account module is responsible for everything that is account-related, such as register, login and logout. It also has some utility functions including getting the current logged in username and user ID.

When a registration request is received, the validity of the input username and password is checked. Then a random 4 bytes salt value is generated, and it is appended to the end of the password after converted to bytes. Afterwards, this combined password and salt are then fed into a SHA256 hash function to produce a hash. Next, the salt value and the hashed password are converted from bytes into hexadecimal representation in string, with 8 and 64 bytes in size corresponding. Finally, the two values are passed into the database interface for insertion into the database. If the database already has the same username registered, it will return with the error.

The login process is similar but in reverse. If the database contains the user-input username, then the hashed password and salt value are returned. The salt value and the user-input password are then computed to produce a hash and is checked against with the hashed password retrieved from the database. If they match, then the user is successfully logged in. Finally, the username and user ID are stored into the session data structure as provided by the Flask framework.

4.2 Photo

The photo module is responsible for uploading user photos, handling the object detection, finding all photos for a particular user, and deleting photos.

The uploading process consists of two phases: checking and processing. The checking phase ensures all uploaded images are valid in format and size. Then the original photo filename and user ID are inserted into the database, and a unique photo ID is returned. Next, the photo is saved and fed into the OpenCV Yolo3 engine to locate objects, which will generate two more image files, one that has objected being marked, and another one is the thumbnail for it. All three versions of the photo are saved using the filename generated from photo ID and are stored separately in three directories. The photo ID approach allows the server to handle photos with unsafe file names, and also duplicated names.

To find all photos belonging to a particular user, the database is queried by user ID to collect all photo IDs. The list of photo IDs is filtered against the thumbnail directory to only contain the photo IDs that have already been processed and stored. The checking is necessary since some photos might still be in the processing queue, or some photos are already deleted by the user in another session. Finally, the static URLs are generated for the above photo IDs for display on the HTML page.

Photo deletion is slightly more tricky. The first thing is to verify the user session matches with the owner of the photo being deleted. Then the photo entry within the database is deleted, and all three files of the photo are deleted.

4.2.1 Object detection using OpenCV Yolo3

The object detection is achieved by using OpenCV2 library and is done by coco dataset trained yolo3 neural network, which is supported by OpenCV2 cv2.dnn.readNetFromDarknet API. The net object returned by readNetFromDarknet() is encapsulated in a class named YoloNet. By calling the pass_forward(images) method on a YoloNet, detected objects on images are

returned in the form of center location in x and y, width and height. The YoloNet object also supports batch processing.

4.2.2 Three photo processing modes

Detecting objects in an image is a relatively computation-intensive task, and usually takes a few seconds. This means the naive approach of processing the photo inside the request handler will increase the server response time, which is not a pleasant experience from the user's point of view. The team has been experimenting with two more approaches to decouple the image processing from the photo upload request handler. The server can easily switch between these three image processing modes by changing the configuration file.

Firstly, the naive approach is in a very streamlined version. The server registers the uploaded photo with the database, saves the original photo into the file system, computes for object detection, saves the result files into the file system, and then replies to the request with responses. The time duration between the request is received and the request being responded to can take up to 7 seconds.

The second approach is the batch processing mode. To optimize the memory use and massive require handling, we decide to implement a thread safe queue, which means each user's request is not handled immediately, but placed in a task queue. A worker thread that owns a persistent yolo net object sleeps on the queue when it is empty. The thread will be notified and start processing tasks in the queue in batches if any client places a task in queue. The benefit of this approach is that each client's request does not instantiate a new yolo net object. Also at the same time the server can handle more incoming requests, because the heavy lifting tasks are postponed and processed later in the backend by the worker thread.

The third approach is on top of the second approach, the pool processing mode. A pool of worker (processes) is created during server startup, and each worker is actively polling from a multi-producer and multi-consumer process-safe queue. The polling is blocking, so a worker can only continue to do the image object detection when it successfully pops a task from the queue. On top of it, the batch processing is added to fully utilize the Yolo3 net batch processing capability. After a task is successfully acquired from the queue's blocking get function, a few more additional non-blocking get functions are called to pop more tasks if possible. If it fails, then the worker proceeds with a single task, and otherwise, it can batch process multiple tasks at the same time. Of course, the server is capable of handling all kinds of errors raised during task enqueuing, such as the queue being full.

4.3 Database Interface

All database-related interaction routines are stored within a single module, for example, account query, account registration, and photo registration. A decorator function is created to hide the database connection and disconnection details from the actual query logics. Other functions can simply use “@database_operation” to establish a connection. All database related operations are carefully constructed so that all exceptions are properly captured. So when the server encounters some unexpected errors, it can fail with meaningful error messages.

4.5 Server Helper

A server helper script is created for infrastructure-related and testing tasks. The script can dump out the rows within the MySQL database, delete tables inside the database, and also clean up the file system storage. This is useful for resetting the server, and for debugging.

4.6 Configuration

All server-related configuration settings are stored within a single configuration file for centralized management.

4.7 File System Storage

Currently, other than the HTML icon and logo files, only the photos uploaded by users and their corresponding processed versions are stored in the file system. They are located in the “static” directory. The original copy of photos uploaded by the user are stored inside “static/data/photos”; the version that has detected objected being marked with rectangles are located inside “static/data/rectangles”; and the thumbnails are stored inside “static/data/thumbnails”. All image files are stored with file names that are based on their photo IDs, which are assigned by the MySQL database. All three versions of a photo have the same filename, which makes retrieving the processed versions of a photo very trivial.

To prevent arbitrary access to users’ photos, we overloaded the “send_static_file” function in Flask, in order to check whether the image requested belongs to the current session user. If the access is illegal, the user will be redirected to the main page.

4.8 MySQL Database

The MySQL database schema is straightforward. It contains two tables: account and photo.

The account table has 4 attributes: “id” of type INT, “username” of type VARCHAR(100), “password_hash” of type CHAR(64) and “salt” of type CHAR(8). The primary key is id, which is automatically incremented while creating a new row in the table. By creating a separate unique id as the primary key, username modification can be supported with minimum additional efforts. “Password_hash” and “salt” are all of fixed size CHAR type, which stores the hexadecimal string representation of their actual binary values.

The photo table contains attributes “id” of type INT, “account_id” of type INT, and name of type TEXT. Id is the primary key for the photo table, which is automatically incremented and is also used to generate a filename for storing the user uploaded photo in the file system. The “account_id” keeps track of the owner user “id” of the photo, it is a foreign key from the account table, which establishes a non-identifying 1:N relationship between the account table and the photo table. Next, the name attribute stores the original file name of the photo uploaded by the user.

5 Conclusion and Future Tasks

Our website is able to perform all functionality required by the A1 handout, including: register and login users, upload photos, display detection results, and support API. In the future, our team would like to implement features which will make the application more secure and useful. The following are items on our todo list:

1. Feature to allow the user to delete the account
2. Feature to support batch deletion of photos