EECS 485 Project 1: Interactive Photo Album

Group membership due 9pm Thursday January 12, 2017
Project due 9pm Friday January 20, 2017

This assignment is an introduction to the software and tools for building server-side web applications. In this project, you will create and deploy an online photo service. It includes using a web server, a SQL database, and a server-side programming language. This project is in python3. Please double check which version you are using by typing `python --version` in your terminal.

Table of Contents

- **Setup**: Installing your local machine
- **Part 1**: Schema creation & Loading Data
- **Part 2**: Building a Photo Album
- **Part 3**: Deploy
- **Deliverables**

Setup: Installing your local machine – 0 points

We will be going over how install your virtual machines in the first discussion. If you run into issues with this process, please check google and stack overflow before reaching out for help. There is a lot of useful debugging information online! :)

Click here for instructions.

Part 1: Building the Database – 20 points

In this part, you will build the back-end database for your photo service using MySQL.

Basic MySQL Commands

MySQL is run in the terminal. It is already installed in the virtual environments you set up. You can use the following commands to access MySQL.

mysql -u <*username*> -p	This command is to start up MySQL in your terminals. Once you type it you'll be asked for a password. By default both your username and password are root.
CREATE <*database_name*> USE <*database_name*>	Type these commands into the MySQL prompt you should have entered now. These commands will allow to create and access a specific database. In your local environment you may call this database whatever you want. From here, you can type any SQL statements directly into the same command prompt.

Website Database Structure

Your website will have registered users, albums, and photos. We will store all of this information using the 4 tables as specified below. A user will be able to create, update, and destroy albums. You can assume there will be no duplicate photos in albums. If the same photo is in separate albums, please create two copies of the photo on the server, each with a unique picid.

Database Schema (primary key is in red):

1. User – username, firstname, lastname, password, email
2. Album – albumid, title, created, lastupdated, username
3. Contain – sequencenum, albumid, picid, caption
4. Photo – picid, format, date

You will need to generate a hash to serve as a unique picid for each photo. This hash will be computed based on the albumid of the album it is being added to and the filename of the image. Your filename used in computing the hash should include the extension portion of the name as well (ie: image.jpg).

Here is you to do that in Python:

```
>>> import hashlib
>>> m = hashlib.md5((str(albumid) + filename).encode('utf-8'))
>>> print (m.hexdigest())
```

This code prints a picid, for example: b94f256c23dec8a2c0da546849058d9e. When you actually name the pictures on your server, remember to add the extension to the end of the picid.

Part A: Create Tables

You have to map the above schema to relations in SQL. In mapping the schema to relations you must adhere to the following guidelines:

- **username/password/firstname/lastname** are all at most 20 characters
- **email** is at most 40 characters
- **titles** are at most 50 characters
- **picid** is at most 40 characters
- **captions** are at most 255 characters
- **format** is a fixed length 3 characters
- all **dates** is the SQL timestamp data type
- **albumid** is an integer who's value starts at 1 (look up auto-increment)
- **sequencenum** is an integer that starts at 0, and increments by 1 each time you add a picture
- add appropriate foreign key constraints

Put the SQL statements you used to create tables in the file sql/tbl_create.sql.

Part B: Load Data Into Tables

Use the following information when loading your tables. Download the images from Google Drive and load them into your app (in the `/static/images` folder, which is never committed to Github). There should be 30 jpg files prefixed by football, sports, space, or world (Tip: you may find writing a Python script to generate the SQL insert commands for each image helpful. This is not required.).

Here are a few notes for the SQL data:

- The caption values for all images should be empty strings (not null or undefined)
- The date, created, and lastupdated values should be the current timestamp
- The sequence numbers should begin at 0 and increment for each picture added on the site.
- Add the albums in the order listed below. The albumid should match up with the numbering in the table below.
- Make sure to insert the pictures in the order they are given in the folder we provide (alphabetical), don't rearrange them. This matters for sequencenum.

The website currently has **three users** listed in the table below (this information is case-sensitive):

Username	Name	Email	Password	Albums (prefix)
sportslover	Paul Walker	sportslover@hotmail.com	paulpass93	**1)** I love sports (sports) **2)** I love football (football)
traveler	Rebecca Travolta	rebt@explorer.org	rebeccapass15	**3)** Around The World (world)
spacejunkie	Bob Spacey	bspace@spacejunkies.net	bob1pass	**4)** Cool Space Shots (space)

Put the SQL statements you used to load data in the file `/sql/load_data.sql` in your group git repo for this project.

Part 2: Building a Photo Album – 80 points

In this project, you will learn how to use a back-end web programming language to interact with a MySQL database to generate dynamic web pages. By the end of this assignment you should feel comfortable using HTML, a back-end language (Python), and MySQL together.

We will not be teaching you any of theses languages in class. If you need help learning them here are some references:

- HTML: W3 Schools Beginner's Guide
- Python: Python 3 Docs
- MySQL: MySql Docs

If these are new to you, please make a point to go to discussion for info and tutorials as well as office hours for advice. Also in order to autograde your website, we require you to add specific id attributes within your HTML elements for each route. These will be specified in the requirements for that route.

GET and POST Requests

To send information to a webserver, HTML provides forms which can be submitted via HTTP GET or POST requests. You may want to refer to this page to understand basics of forms. HTTP GET requests do not make changes to the content that they are viewing, and SHOULD NOT affect the sate of the information stored in your site. HTTP POST requests are reserved for editing or adding content (e.g., saving a new user to MySQL).

Note that GET is the default when a user browses the web. When someone type in a url your browser automatically sends a GET request to the correct URL, but you will need to send POST requests manually form your website. Since we are auto grading make the body of your POST requests exactly as we specify. You will need the requests for you edit pages, to change content on your website. I cannot emphasize this enough. If your forms are incorrectly formatted, or do something like try to access feeds that don't exist (ie: you expect but we don't send), you will get a BAD REQUEST ERROR.

Last note, a request to GET /albums?username=<username> should show different results depending on the user specified by the username query parameter.

Routes:

Home Page

Route: http://{host}:{port}/{secret}/p1/

A good webpage should contain a **<title> tag,** other **<meta> tags,** a **header** and a **footer.** These tags are a good way to structure a webpage, and some are used by large crawlers to get quick information about the content on your webpage, and a good practice to always have. We will simply be checking to see that these tags exist. Feel free to put whatever content you want in them.

It is recommended that you actually put all this information in `base.html`, so it included on all the pages, and only the "block content" portion changes for a specific route. If this is unclear don't worry we will be going over how to do this in discussion.

Additionally the home page specifically should have some text describing the website and a list of users (by username) whose albums can be browsed. The usernames should have links to browse their albums at `/albums?username=<username>`. These links should have an `id=user_albums_<username>`.

Albums Page

Route: http://{host}:{port}/{secret}/p1/albums?username=<username>

On this page you will list all the albums a user has. For each album (1) display the name of the album *(these are diffed, so they are case sensitive, and thus make them **exactly** what we specified in the SQL section)*, and (2) include a link to view the album on the album page. These links should have an `id=album_<albumid>_link`.

There should also be a link to `/albums/edit?username=<username>`.This will allow a user to see an editable view of all of their albums. These links should have `id=user_albums_edit_<username>`.

Albums Edit Page

Route: http://{host}:{port}/{secret}/p1/albums/edit?username=<username>

This page presents the user with an editable list of his or her albums. Here is a very basic interface (this is for your reference only; you can have a fancier design):

```
<table>
  <tr>
    <td>Album</td>
    <td>Edit</td>
    <td>Delete</td>
  </tr>
  <tr>
    <td>Summer 2011 in Iceland</td>
    <td>[Edit]</td>
    <td>[Delete]</td>
  </tr>
  <tr>
    <td>Spring break 2010 in Brooklyn</td>
    <td>[Edit]</td>
    <td>[Delete]</td>
  </tr>
  <tr>
    <td>Thanksgiving 2010</td>
    <td>[Edit]</td>
    <td>[Delete]</td>
  </tr>
  <tr>
    <td>New: _____</td>
    <td>[Add]</td>
    <td></td>
  </tr>
</table>
```

When creating your interface, be sure to give each delete button an HTML `id=delete_album_<albumid>`. Similarly, the text box for adding a new album name should have `id=album_add_name` and the submit button should have `id=album_add_submit`.

As in /albums, different pages should be presented depending on the username query. We do this in the same way as above (e.g., /albums/edit?username=<username>).

On the other hand, we use POST method to the same URL for [Delete] and [Add] buttons. To help us use the autograder, please follow the interface defined below. When the HTML form is submitted with method="POST", the browser will attach a set of key-value pairs to the POST body. If the key op exists in the POST body, it means either Delete or Add was clicked in the previous window. The op can have two values, delete or add. If the value of op is delete, then the additional key albumid should be provided and have the value indicating the primary key value of album table in the database.

For example, an HTTP request to POST /albums/edit would have a body **exactly** like:

op: "delete"
albumid: <integer albumid>

If the value of op is add, the additional variables title and username should be provided together with the appropriate values in the POST request. Be sure to manage the created date for new albums.

In order to add an album, an HTTP request to POST /albums/edit would have a body **exactly** like:

op: "add"
username: "<username>"
title: “<album title>"

Clicking [Edit] link directs a user to /album/edit?albumid=<albumid> and should have the id=album_edit_<albumid>_link.

Album Page

Route: http://{host}:{port}/{secret}/p1/album?albumid=<albumid>

This page should display a thumbnail view of the pictures in the specified album ordered by increasing sequence number. Clicking on the image should take you to /pic?picid=<picid>. These links to the pictures should have id=pic_<picid>_link. Each album page should also include a link to the album edit view /album/edit?albumid=<albumid>, with an id=album_edit_<albumid>_link.

Album Edit Page

Route: http://{host}:{port}/{secret}/p1/album/edit?albumid=<albumid>

This page should be almost identical to /album **with the addition** of enabling the user to perform the following operations.

- Add pictures to the album.
  - When adding a picture you must generate a unique hash for *each* picture, the picid. Change the filename of the image to this hash before you save it to the server (but make sure to maintain the original extension), and also include the entire filename (including the extension), when generating the hash. Each picture will be saved as /static/images/{hash}.{type}.

  - Determine the format of the image during the upload using the file extension. Acceptable image formats include png, jpg, bmp, and gif (case insensitive). All other extensions should not be accepted by the server.
  - Automatically set the date to the moment the picture was uploaded.
  - Pictures uploads should be kept in a /static/images folder.
  - You should automatically assign a sequence number to a picture, which is one larger than the current largest sequence number.
- Delete pictures from the album.
  - Be sure to remove the file in the /static/images folder as well delete the correct entries from the database.
  - You must manage the lastupdated date in the album whenever an album is modified e.g. upload a new photo in an album, the lastupdated column in this album should change automatically. This can be done using either SQL statements within your app or SQL triggers (you may find the latter easier).

There are two POST requests you send on the album edit page. Be sure to read a tutorial on how to accept multipart/form-data via a HTML form POST request: Guide for Flask-Python.

The add request MUST have the format:

```
op: "add"
albumid: <integer albumid>
file: <file upload data>
<multipart/form-data also part of post>
```

The file input button for this form should have HTML `id=file_input` and the submit button should have `id=file_submit`.

To Delete a photo use the format:

```
op: "delete"
albumid: <integer albumid>
picid: "<picid>"
<multipart/form-data also part of post>
```

Each delete button should have HTML `id=delete_pic_<picid>`.

Pic Page

Route: http://{host}:{port}/{secret}/p1/pic?picid=<picid>

On this page show display each full-sized picture. It must also have navigational elements to go to the next and/or previous picture (if it exists) within the album with HTML id of `id=next_pic` and `id=prev_pic`. If a next or previous picture don't exist, don't display the links accordingly. The next and previous pictures are determined by increasing sequence number ordering, just like in the album page. There should also be a link back to the whole album page with `id=parent_album`.

Invalid Pages

Any invalid page request in the URLs listed below (including invalid or missing URL parameters) should be aborted with a 404 Error (Page Not Found). Flask will automatically handle routes you do not explicitly create. For routes you have created you will need to 404 on pages with incorrect information. Think of when this can happen (ie: no user exists, or no username is provided).

Part 3: Deploying – 0 points

Your website much be deployed for us to autograde Part 2.

This is a group assignment. You should have registered your GitHub username and joined a group via this link before continuing. You will receive an email with the following information after the group signup deadline:

- Your host & port number (e.g. 5000)
- Your group's MySQL username & password
- A secret string of random characters (e.g. abcdefghij), this will be used to make sure other groups can't alter your website. Please keep this secret! This will be the secret used in Part 2.

MySQL Password

Your MySQL account will be used throughout the rest of the semester for your website database backend. You are encouraged to change your MySQL password (the initial password is in the email). To change your MySQL password, follow these steps:

1. Log into your designated machine via SSH using your uniqnames
2. Connect to MySQL server using: `mysql -u <username> -p`. Enter the password we sent you in the email.
3. Set your new password, while inside the MySQL prompt by typing:
   ```
   SET PASSWORD = PASSWORD('YOUR NEW PASSWORD');
   FLUSH PRIVILEGES;
   ```

If you are unable to log in to your development machine or you cannot connect to MySQL server, please let us know right away.

URL

For project 1, we will be checking at the following URL. Please be careful to make sure this is your exact URL. Missing or extra slashes commonly cause issues with the autograder.

Format: http://{host}:{port}/{secret}/p1
Example: http://class4.eecs.umich.edu:12345/abcdefghij/p1

Deployment

When your project is finished, you should have a running website at the provided endpoint. Your application must be robust - we should not be able to crash your web server, or cause database timeouts, while testing your application. Evaluation will be done by examining your website through a web browser (by an autograder). Since we interact directly with your websites, we will modify

your database. Be sure to reset your website and database to its original state – all given images and users should exist – prior to submitting.

Also make sure you are not developing your code on your live site. Good practice is to develop locally and then move code to over to the server only to deploy.

Steps to Deploy

When you deploy your site you will follow the same steps you did for running your code locally (make sure to install a visual environment and all dependencies again). The only step that is different is when you get to running `python app.py`. This server is single threaded at not suitable to handle live production. Instead we will be using Gunicorn, a WSGI-compliant HTTP server.

1. `cd` into `/var/www/html/groupXX/`. This is your groups folder on the server.
2. Pull your code from github. Remember not your commit your images to github.
3. Set up your database. Run your create and load scripts, with the database we have provided you. Remember, your login credentials locally are not the same on the live server.
4. Copy the images over from your local computer to the server. We recommend looking up the `scp` command.
5. We have created a config file, which specifies where your website is deployed. Change this from localhost to the correct url, port, etc…
6. Type `gunicorn -b <HOST>:<PORT> -w 2 -D app:app` in your virtual environment to deploy your app. Remember you must have reinstalled vagrant and your dependencies, and be in the virtual environment.

This will start a background gunicorn server process that a Flask app instance called app. To verify, you can view a list of processes with `ps aux | grep <uniqname for user who started it>`. You should see a list of processes. If you ever need to kill/restart your processes, find the correct one, then and type `kill <process ids>`.

Deliverables

Please note: If something is not explicitly specified in the spec, we are probably not testing it, and you have the liberty of implementing it how ever you feel like.

Routes

Make sure that all these URLS are present in your web application:

- `/` Homepage, Browse List of Users
- `/album` Thumbnail View of an Album
- `/album/edit` Editing an Album -- Add/Delete Pictures
- `/albums` Browsing Albums for a Particular User
- `/albums/edit` Editing the List of Albums - Delete/Add Albums
- `/pic` View Picture with Prev/Next Links

HTML ID Attributes

Make sure that all these element IDs are present in your HTML templates. Additionally, when you view the live page, verify that they are present via (Right Click->View Source):

- /
  - Links to user albums should have an id “user_albums_<username>”
- /album
  - Each link to /pic?picid=<x> should have an id “pic_<picid>_link”
  - Each link to /album/edit?albumid=<x> should have an id “album_edit_<albumid>_link”
- /album/edit
  - Each link to /pic?picid=<x> should have an id “pic_<picid>_link”
  - Each delete button should have an id “delete_pic_<picid>”
  - The file input should have an id “file_input”
  - The submit button should have an id “file_submit”
- /albums
  - Each link to /album?albumid=<x> should have an id “album_<albumid>_link”
  - The link to to /albums/edit with id “user_albums_edit_<username>”
- /albums/edit
  - Each link to /album?albumid=<x> should have an id “album_<albumid>_link”
  - Each delete link/button should have an id “delete_album_<albumid>”
  - The text box for the new album name must have id “album_add_name”
  - The submit button to add a new album must have id “album_add_submit”
  - Each link to /album/edit?albumid=<x> should have an id “album_edit_<albumid>_link”
- /pic
  - The link to the next picture in the album should have an id “next_pic”
  - The link to the previous picture in the album should have an id “prev_pic”
  - The link to the parent album should have an id “parent_album”

Submit

For testing Part 1, please run the following SQL commands from inside your database, and submit a tarball of the output files to the autograder. Name the files sql_output_<NUMBER>.txt, where number is the command number below, and name the tarball sql.tar.gz.

Commands:

1. SELECT (username, firstname, lastname, password, email) FROM User ORDER BY username
2. SELECT (albumid, title, username) FROM Album ORDER BY albumid
3. SELECT picid, format FROM Photo ORDER BY picid
4. SELECT albumid, picid, caption FROM Contain ORDER BY picid

The autograder will hit your deployed site for testing the website, so make sure that is up and running as well.
In the README.md at the root of your repository please provide the following details:

- Group Name (if you have one)

- List the contribution for each team member:
  User Name (uniqname): "agreed upon" contributions
- Any need-to-know comments about your site design or implementation.

Please do not modify the files in your git repository or deployment after the project is due! The deployed version of your app also cannot have modifications after the due date.