EECS 485 Project 1:  
Hello, World Wide Web

[Group membership](http://eecs485fall2016.herokuapp.com/) due 9pm Thursday September 15, 2016

Project due 9pm Thursday September 29, 2016

This assignment is an introduction to the software and tools for building server-side web applications. It includes using a web server, a SQL database, and a server-side programming language. In this project, you will create and deploy an online photo service.

# Table of Contents

[Part 0 (0 Points): Set up your local machine](#_py2288ogkxzx)

[Part 1 (30 points): Schema creation & Loading Data](#_l8kdf2ndyq9a)

[Part 2 (60 Points): Building a Photo Album](#_7cs8v3fkb15d)

[Part 3 (10 Points): Deploy](#_tamyyc4plqqa)

[Deliverables](#_aqi1xfzh6dwp)

# Part 0 (0 Points): Set up your local machine

To complete Part 0, follow the instructions [here](https://docs.google.com/a/umich.edu/document/d/1myLOStrVv03s5KNofJxBeCqmV2FZ9hrwAfqHHy172O0/edit?usp=sharing).

**Make sure to use the new project 1 starter files, and not ones provided in discussion.**

# Part 1 (30 points): Schema creation & Loading Data

In this part, you will start working on the back-end database for your photo service. Your website needs a database to store data and MySQL will be used to fill this requirement.

### Using MySQL

To run MySQL in the terminal, use the following command:

$ mysql -u <username> -p.

Here, -u identifies your username for MySQL (most likely root). There will be a prompt for the password when you hit enter (most likely root). If you’re on your local server, first create a database (one is already created for you on the live server called groupXXp1). Once you do, you can run the following to access the SQL server with the given db\_name:

$ mysql -u <username> <db\_name> -p.

From here, you may directly type SQL statements into the prompt, or use the command line to to execute SQL script files without entering the MySQL server. *If you run a script multiple times, you may need to reset the database to avoid errors.*

### Website Database Structure

Your website should have registered users (people), each identified by a unique username. For each person you should also maintain a password, a first name, a last name, and an email address. Each person may create/update/destroy as many albums that he or she owns as he or she would like. Each album has a unique id, a title, a date created, a date last updated, and the owner's username. Each album may have zero or more photos and there will not be duplicates. Within the context of a particular album, a photo has a caption (caption only lies in database for now, we will display it on website in pa2), as well as a string to serve as a unique picid. The same photo can be in two separate albums *but* there will be two copies of this photo on the server, each with a unique picid.

A relational schema for the data you will need is as follows (**primary key** is bolded):

* User ( **username**, firstname, lastname, password, email )
* Album ( **albumid**, title, created, lastupdated, username )
* Contain ( **sequencenum,** albumid, picid, caption )
* 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.

Here is an example in Python:

>>> import hashlib

>>> albumid = 1

>>> filename = 'sports\_s1.jpg'

>>> m = hashlib.md5()

>>> m.update(str(albumid))

>>> m.update(filename)

>>> print m.hexdigest()

b94f256c23dec8a2c0da546849058d9e

Alternatively:

>>> m = hashlib.md5(str(albumid) + filename)

>>> print m.hexdigest()

b94f256c23dec8a2c0da546849058d9e

## Part 1a (20 points): 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 should be a fixed length 3 characters
* all dates should be the SQL timestamp data type
* the album id should be an auto incrementing integer (value will start at 1)
* the sequence number should be an integer that is not auto incrementing (value will start at 0)
* add foreign key constraints

Put the SQL statements you used to create tables in the file sql/tbl\_create.sql.

## Part 1b (10 points): Loading 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
* **Make sure to insert the pictures in the order they are given (alphabetical), don’t rearrange them**

The website currently has **three users** (passwords are listed below as well):

The **first** has username "sportslover". His real name is Paul Walker. His email address is sportslover@hotmail.com and password is “paulpass93”. He created two albums; the first one is titled "I love sports". The images in this album are prefixed by sports. Paul also has another album called "I love football". The images in this album are prefixed by football.

The **second** has username "traveler". Her real name is Rebecca Travolta. Her email address is rebt@explorer.org and address is “rebeccapass15”. She created an album called "Around The World". The images in this album are prefixed by world.

The **third** has username "spacejunkie". His real name is Bob Spacey. His email address is bspace@spacejunkies.net and password is “bob1pass”. He used the site to create one album titled "Cool Space Shots". The images in this album are prefixed by space. **Add the albums in the order listed below**.

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

Put the SQL statements you used to load data in the file /sql/load\_data.sql in your group git repo for this project. Refer to Deliverables section for more detail about how to submit. Although you can load data directly from a text file, use insert statements instead.

## Part 1 Deliverables

For part 1, put the two sql files (tbl\_create.sql and load\_data.sql) in the /sql directory of your git repo. The .sql files should be normal text files which include proper MySQL statements separated by newline characters. The sql files should run OK on execution of the following command:

$ mysql -u <username> <db\_name> -p < script.sql

The above command runs the mysql client with input from script.sql instead of standard input. The username/password is what you entered during setup (or root for both in Vagrant). For part 1a, the grading script will evaluate by seeing if all necessary tables exist, test data can be inserted without errors, and test data are rejected if constraints are not satisfied. For part 1b, the Autograder will query the database to determine if the correct data exists.

# Part 2 (60 Points): Building a Photo Album

In this task, 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 in concert. You may use the following [W3 Schools Beginner's Guide](http://www.w3schools.com/html/html_intro.asp) to get familiar with HTML. **Please make a point to go to discussion for info and tutorials as well as office hours for advice.**

Your photo album website is meant to be used by a small number of users with a simple interface. Start thinking of the overall design of your website. In particular you will need to worry about a clean navigation interface and ease of editing albums. You will not be graded on aesthetics, but you are free to add CSS style to your website (check out [Bootstrap](http://getbootstrap.com/)). 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.

## Part 2a (10 points): Getting Started

An "index page" will serve as the homepage for the website. A good way to have consistent interface design is to create a template file (or set of files) that you include in all pages your server serves to users - this is possible in nearly every server-side framework (our projects use [Jinja](http://jinja.pocoo.org/). Locally, you will visit http://localhost:3000/{secret}/p1/ but the live URL will differ and look like the host:port template seen below. Since all route urls will begin with {secret}/p1, we recommend using [url\_prefix](http://flask.pocoo.org/docs/0.11/blueprints/#registering-blueprints). The secret will be given to you once groups are finalized (and is discussed in Part 3).

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).

### Index: / = http://{host}:{port}/{secret}/p1/

The index should contain a proper <title> tag, other <meta> tags, a header and footer for the page, some text describing the website and a list of users 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>.

In P1, no login is needed for your website.

### View Album List: /albums

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](http://www.w3schools.com/html/html_forms.asp) to understand basics of forms. HTTP GET requests do not make changes to the content that they are viewing. HTTP POST requests are reserved for editing or adding content (e.g., saving a new user to MySQL).

A request to GET /albums?username=<username> should show different results depending on the user specified by the username query parameter. Note that GET is the default when a user browses the web. Remember: since most browser requests are GET requests they SHOULD NOT affect the state of the information stored in your website. For each album listed, it should be a link to view the album GET /album?albumid=<albumid> so that way users can quickly navigate to the albums. These hyperlinks should have 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, which we will create in part (2b). These links should have id=user\_albums\_edit\_<username>.

## Part 2b (30 points): Editing Albums

Now you have to create at least two web pages as described below. An edit album list and edit album. You don’t need to consider the accessibility of the albums, and once getting the username you can edit all albums of this user.

### Edit Album List: /albums/edit

Presents the user with a 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 like:

op: "delete"

albumid: 2

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 like:

op: "add"

username: "spacejunkie"

title: "nicealbum"

Clicking [Edit] link directs a user to /album/edit?albumid=<albumid>.

### Edit Album: /album/edit

This page is provided with an id (ex. /album/edit?albumid=2) this key is the primary key of album table. 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). 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, 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).

Similar to above you may add or delete pictures via HTTP POST /album/edit from an HTML form. To Add, you must have the following POST data:

op: "add"

albumid: 2

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. Be sure to read a tutorial on how to accept multipart/form-data via a HTML form POST request: [Guide for Flask-Python](http://flask.pocoo.org/docs/patterns/fileuploads/).

To Delete a photo:

op: "delete"

albumid: 2

picid: "hashash"

Each delete button should have HTML id=delete\_pic\_<picid>.

## Part 2c (20 points): Viewing Albums and Pictures

### View Album: /album

This page should display a thumbnail view of the pictures in the album ordered by the sequence

number. The albumid is given via query parameter named albumid. For example: GET /album?albumid=<albumid>. Clicking on the image should take you to /pic?picid=<picid>. The links to the pictures should have HTML id=pic\_<picid>\_link. Each album page should also include a link to the edit view /album/edit?albumid=<albumid>.

### View Picture: /pic

At /pic?picid=<picid>, this page will show full sized pictures. It must also have navigational elements to go to the next and/or previous picture (if it exists) in the album with HTML id of id=next\_pic and id=prev\_pic. There should also be a link back to the whole album page with id=parent\_album.

# 

# 

# Part 3 (10 Points): Deploy

This is a group assignment. You should have registered your GitHub username and joined a group via this [link](http://eecs485fall2016.herokuapp.com/) before continuing. You should also have received an email with the following information after the group signup deadline:

* One port number (e.g. 12345)
* Your group’s MySQL username
* 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.

The port numbers correspond to two virtual pages you will host for this assignment. 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:  
   $ mysql -u <username> -p

You will be prompted for your group’s MySQL initial password, that we provided..

1. Set your new password, while inside the MySQL prompt:  
   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.

For project 1, we will be checking your website at the following URL:

http://{host}:{port1}/{secret}/p1

For Example

http://class4.eecs.umich.edu:12345/abcdefghij/p1

### Deployment

When your project is finished, you're expected to have a running website at the provided endpoint. Evaluation will be done by examining your website through a web browser (both by human graders and an autograder). Since we interact directly with your websites, some modification of your database can occur. **Be sure to reset your website and database to its original state ‒ all given images and users should exist ‒ prior to submitting.**

Your deployed code should not be deployed using the python app.py command, which is not suitable for live production environments. Your code and Python packages should be contained within a virtualenv. After sourcing/activating your virtualenv, you will use [Gunicorn](http://gunicorn.org/), a WSGI-compliant HTTP server.

You should not be developing remotely on the servers provided. Instead, develop your code locally, test it locally with the Flask development server, your own MySQL installation, etc. After verifying correctness locally, go into your server, create a folder for the current project in /var/www/html/groupXX/ and do a git pull from your Github repo on the server. Once your code is on the server and you have run the necessary SQL files similarly to how you do in Part 1. We have created a config.py file. Put all your server and login info in this file. **Remember, your login credentials locally are not the same on the live server.**

In addition to running your SQL files, you will have to setup a new virtual environment like you did in Part 0 (**remember to never commit the the /venv folder to Github**).

After installing the pip dependencies in the venv, you can deploy your app with Gunicorn as follows:

gunicorn -b class3.eecs.umich.edu:3000 -w 2 -D app:app

This will start a background gunicorn server process that a Flask app instance called app (in a file called app.py) that is running on two ports: 3000 and 3001 (change them to yours). To verify, you can view a list of processes with the following command:

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 run the kill command (there will be more than one but try killing just the first one):

kill 12345

where 12345 is the process id of your gunicorn server.

**Your application must be robust - we should not be able to crash your webserver, or cause database timeouts, while testing your application.**

# Deliverables

All deliverables for P1:

### 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?id=<x> should have an id “pic\_<picid>\_link”
    - These should also be present on /album/edit
* /album/edit
  + 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”
    - These should also be present on /albums/edit
  + The link to to /albums/edit with id “user\_albums\_edit\_<username>”
* /albums/edit
  + 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”
* /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

Submit the following files to the autograder:

* source.tar.gz A tar archive containing your application source code.
* tbl\_create.sql
* load\_data.sql

How to create a tarball of your source code, from your git repository:

git archive --format tar.gz HEAD > source.tar.gz

The autograder should be released on 9/16. Look out for a Piazza announcement.

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.**