Assignment 1: Hello, World Wide Web

Due: Thurs. September 19, 2013

This assignment introduces you to the software and tools which provide the backend for the website you will be building in this course.

- Web Server
- SQL Database
- Server-Side Programming Language

Table of Contents

Changes to this spec will be announced on CTools and visible on GitHub (https://github.com/EECS485/admin/blob/master/pa1.md).

Part 1 (0 Points): Log in to Your Dev. Machine

This is a group assignment. You should already have registered your GitHub username and joined a group via this link (http://eecs485.herokuapp.com). You will also receive an email with the following information:

- Two port numbers (e.g. `12345` and `54321`)
- Your MySQL username
- A secret string of random characters (e.g. `abcdefghij`)

The port numbers correspond to two virtual pages you will be hosting 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 (initial password is also same as your group name). To change your MySQL password, follow these steps:

1. Log into your designated machine via SSH using your uniquenames

2. Connect to MySQL server:

   ```
   mysql -u your-MySQL-username -p
   ```

   You will be prompted for your MySQL password and use the initial password this time.

3. Set your new password:

   ```
   SET PASSWORD = PASSWORD('YOUR NEW PASSWORD');
   ```

If you are unable to log in to your development machine or you cannot connect to MySQL server, please let chjun@umich.edu (mailto:chjun@umich.edu) know right away.

Part 2 (10 Points): Get a Server Running

For programming assignment 1, we will be checking your two websites at the following URLs:

```
http://{host}:{port1}/{secret}/pa1

http://{host}:{port2}/{secret}/pa1
```

For Example

```
http://eecs485-01.eecs.umich.edu:12345/abcdefghij/pa1

http://eecs485-01.eecs.umich.edu:54321/abcdefghij/pa1
```

*Note: We accept multiple server-side languages and therefor setting up a webserver may differ across these implementations.

We will provide specific tutorials for the following shortly, but feel free to get started right away:

- Apache and PHP
- Python
- We do not provide *official* support other languages/frameworks. *You may decide to complete this and other projects with another language/framework only if you feel comfortable without staff support*.

Part 3 (30 points): Schema creation and Loading Data

In this task you will start construction of your website. You will develop an online photo album service. 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, owner's username and a field that specifies the permissions to this album ("public" or "private"). Each album may have zero or more photos. Within the context of a particular album, a photo has a sequence number and a caption. Photos can be used within multiple albums (shared) and in all instances they will have the same URL, date taken, and format ("GIF", "JPG", etc.).

An initial 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 (**albumid**, **picid**, caption, sequencenum)
- Photo (**picid**, url, format, date)

Part 3a (20 points): Create Tables

You have to map the above schema to relations. In mapping the schema to relations you must stick 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
- captions are at most 255 characters
- web addresses (URLs) may be up to 255 characters
- all dates should be the SQL date type
- format should be a fixed length 3 characters
- the album id should be an integer
- picid is at most 40 characters
- the sequence number should be an integer. In different albums, one photo may have different sequence numbers.
- define suitable primary keys
- add foreign key constraints

Put the SQL statements you used to create tables in the file `/sql/tbl_create.sql` in your group git repo for this project. Refer to Deliverables section for more detail about how to submit.

Part 3b (10 points): Loading Data Into Tables

Use the following information when loading your tables. Please make up any information which is not given to you but *do not leave fields blank*. Download the images (http://www-personal.umich.edu/%7Echjun/eecs485/pa1_images.zip) to load here. There should be 30 jpg files prefixed by football, sports, space, or world.

The website currently has three users.

The first has username "sportslover". His real name is Paul Walker. His email address is sportslover@hotmail.com. He created two albums; the first one is titled "I love sports" which is public. The images in this album is prefixed by `sports`. Paul also has another album called "I love football", which is also public. The images in this album is prefixed by `football`.

The second has username "traveler". Her real name is Rebecca Travolta. Her email address is rebt@explorer.org. She created a public album called "Around The World". Her images are prefixed by `world`.

The third has username "spacejunkie". His real name is Bob Spacey. His email address is bspace@spacejunkies.net. He used the site to create one private album titled "Cool Space Shots". His images are prefixed by `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. 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. We need this to make this assignment autogradable.

MySQL Database

For this assignment each group will use its own database. Each group will have all privileges on its databases and no access to other databases on MySQL server. To run MySQL use the following command:

```
/usr/bin/mysql -u username db_name -p
```

Here `-u` identifies your username for MySQL. Password could immediately follow `-p` or it could be requested after hitting enter. Finally `db_name` identifies the database you want to use (your group's database). You should have received all the necessary information in an email to your group members, or you will soon.

From here, you may directly type SQL statements into the prompt, or use the source command of the monitor to execute SQL script files. If you run a script multiple times, you may need to drop tables or delete tuples from a table. When developing it is a good idea to test your SQL alone before using it in the app.

MySQL Deliverables

For part 3, put two sql files (`tbl_create.sql` and `load_data.sql`) in the directory `/sql` in 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 when you entered the following command:

```
mysql -u username db_name -p < script.sql
```

The above statement means running the mysql client with input from `script.sql` instead of standard input. For part 3a, 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 3b, the script will see if the data are populated by querying them.

We will run your .sql files with our own database. Therefore, no update will be made in your databases.

Please do not modify the files in your git repository or deployment after the project is due! The graders will look at the last commit to your repository before the due date. Also note that the deployed version of your app cannot have modifications after the due date. *We reserve the right to take away points from your group in either case.*

Part 4 (60 Points): Building a Photo Album

In this part we will start working on our database-backed website. You will learn how to use a back-end web programming language to interact with MySQL database to generate dynamic webpages. By the end of this assignment you should feel comfortable using HTML, a back-end language, 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 style but you are free to add CSS style to your website.

Part 4a (10 points): Getting Started

An "index page" at `http://{group_url}/` 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 the pages your server serves up to the users - this is possible in nearly every server-side framework.

Index: `/`

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. In PA1, no login is needed for your website.

View Album List: `/albums`

In order to send information to a webserver, HTML provides forms which are 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 (ex. saving a new user to MySQL). RESTful applications usually Create, Read, Update, Delete information using GET, POST, PUT, and DELETE requests respectively - more details on this on later projects when AJAX is introduced.

A request to `GET /albums?username=id` should show different web pages 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 example, if a user typed the URL with the query parameters (the query follows a url like: http://{url}?name=value&name2=value2), any server-side scripting language can retrieve this information when handling the request.

Part 4b (30 points): Editing Albums

Now you have to create at least two webpages as described below. In editalbumlist and editalbum, 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):

Album	Edit	Delete
Summer 2011 in Iceland	[Edit]	[Delete]
Spring break 2010 in Brooklyn	[Edit]	[Delete]
Thanksgiving 2010	[Edit]	[Delete]
New: ________________	[Add]	

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

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`, they additional key `albumid` should be provided together 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 variable `title` and `username` should be provided together with appropriate values entered by the user. Note that when you delete an album, you should also delete pictures in that album as long as they are not included elsewhere. 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"
```

Clicking [Edit] button directs a user to `/album/edit?id=albumid`.

Edit Album: `/album/edit`

This page is provided with an `id` (ex. `/album/edit?id=2`) this key is the primary key of `album` table. This page should enable 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`.
    - Determine the format of the image during the upload using the MIME type.
    - Automatically set the `date` to the moment the picture was uploaded.
    - Pictures uploads should be kept in a `/pictures` folder.
    - Each picture in the folder will be saved as `/pictures/{picid}.{type}`, the picture's `url`.
    - You should automatically assign a sequence number to a picture, which is one larger than the largest sequence number in *the picture's album* currently.
- Delete pictures from the album.
    - Be sure to remove the file in the `/pictures` folder as well as the database.

- You need to manage the `lastupdated` date in the `album` table.

Similar to above you may `add` or `delete` pictures via HTTP `POST /album/edit` from an HTML from. To Add:

```
op: "add"

albumid: 2

<multipart/form-data also part of post>
```

Be sure to read a tutorial on how to accept `multipart/form-data` via a HTML form POST request.

- Example in PHP (http://php.net/manual/en/features.file-upload.post-method.php)
- Example in Flask-Python (http://flask.pocoo.org/docs/patterns/fileuploads/)

To Delete a photo:

```
op: "delete"

albumid: 2

picid: "hashash"
```

Part 4c (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 `id`. For example: `GET /album?id=2` Clicking on the image should take you to `/pic?id=pictureid`.

View Picture: `/pic`

This page shows full sized picture. For example: `/pic?id=x83s7g5`. It must also have navigational elements to go to the next and/or previous picture in the album, as well as a link back to the whole album page.

Deliverables

Make sure that all these functions 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

Also be sure to turn in your `.sql` files in the `/sql` directory as mentioned above.

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 - don't worry if extra users and pictures are present when we test your website - just make sure the given content is present.

Code

Your server-side code will be graded by cloning at your GitHub repository. Be sure to use a `.gitingore` file to tell git to ignore any non-souce files (like `.pyc` files or `/venv`, etc) *and* the `/pictures` directory. Please do not put pictures anywhere in your git repo.

In the `README.md` at the root of your repository please provide the following details:

- Group Name (if you have one)
- List `User Name (uniqname): "agreed upon" contributions`.
- Details about how and if you deviated from this spec - avoid if possible.
- Extra details about how to clone and run your code - simple as possible.
- Anything else you want us to know, like how many late days you took.
- The formatting is not critical, we just need the information.

```
Group Name:

  Rabid Ocelots

Members:

  Otto Sipe (ottosipe): setup the database, setup the routes, did the project alone

  ...

Details:

  We called our /pic endpoint /foto

Deploy:

  (just an example)

  pip install < requirement.txt

  foreman start

Extra:

  I took 2 late days.
```

As mentioned above, **please do not modify your code after the due date - either on the repo or the server**, or else we will assume your submission is late. We then can assess late days or take off points.

Final Words

- Take the advice of the GSI and IAs in this course - including reading though the code samples (linked above) and attending discussion for additional tutorials and advice.
- Also note that there is no ***one way*** of doing things on the web. Don't be blatantly inefficient and try to follow standards, but at the end of the day - if it works well, ship it.
- Design your code together as a team and then split up the work.
- The same principles of any programming language apply to the web.
    - Avoid copy-and-pasting code that could easily be generalized or templated.
    - For example: The "view" and "edit" code and HTML for a list or single album should be similar or shared.