INFO / CS 2300 Project 2: Online Catalog

Due Tuesday, February 23rd, 2015 at 5:00pm

I. Introduction

Create a small catalog website that displays any collection of objects in your database. Instead of an actual database, however, you will be using your knowledge of file input/output in PHP to read and write information about your objects in a file (**data.txt**) on the server.

Examples of catalogs you might consider include:

- A music catalog like Billboard or Discogs
  - search by album, song, singer, genre, year, popularity
- A movie catalog like IMDB (but more simplified)
  - search by title, actor, genre, year, rating, awards
- A videogame catalog like VGCollect or Steam (without the download part)
  - search by platform, genre, series, developer, distributor, year, rating, tags
- An online store like Amazon or Ebay (without the payment part)
  - search by department, product name, gender/size/color, seller, price
- A visual dictionary (like a PokéDex or Bird Guide)
  - search by region, order, species, evolution, statistics, diet, time of activity
- A simple blog or forum like Tumblr, Twitter, or Reddit (but without user accounts)
  - search by tags, username, subject/title, description keyword, date, popularity

II. Requirements

1. The Full Collection (10 pts)

Your website must allow the user to view the entire collection of entries in your catalog at once, and your collection (in your **data.txt** file) should contain at least five (5) complete entries.

For example, your collection might look like the following on the index page:

Shrek	Mike Myers, Eddie Murphy	Adventure/Comedy	PG
Despicable Me	Steve Carell, Chris Renaud	Comedy/Animation	PG
Toy Story	Tom Hanks, Tim Allen	Fantasy/Adventure	G
Treasure Planet	Joseph Gordon-Levitt, Emma Thompson	Romance/Adventure	PG
The Iron Giant	Vin Diesel, Jennifer Aniston	Action/Adventure	PG

Note: you would not typically display the data.txt file directly. Use PHP, HTML and CSS to create a template to provide formatting and then read the data for all entries into the template from data.txt.

2. An Add Entry Form (15 pts)

Use an HTML form that allows users to add an entry into your data file. An entry must have at least four (4) data fields associated with it.

An example of this for a movie catalog would be:

1. **Title**: Shrek
2. **Actors**: Mike Myers, Eddie Murphy
3. **Genre**: Adventure/Comedy
4. **Rating**: PG

Note: If you want to include images for each entry, you do not need to support image uploads (that's a topic for Project 3). Instead, you can have a set of images on the server that the user can select from a drop-down list, or you can ask the user to type a valid image URL into a text field. **Remember, a source credit must display near each image unless you created the image yourself in which case the credit can be in an HTML comment.**

3. A Search Form (15 pts)

Use another HTML form that allows users to search for specific entries in your collection. Your search functionality does not need to have complicated natural language processing, but it must allow for each of two types of searches:

1. Filter Entries by a Single Field
   Example: Search for All Movies where **Rating** == **PG**

2. Filter Entries by Multiple Fields
   Example: Search for All Movies where **Rating** == **PG** and **Genre contains "Comedy"**
   If your search makes more sense as an or search, you may do that. Explain your choice of "and" vs "or" in your rationale.

Your filtered results will probably display in a similar manner as the full collection. Make sure it is clear to your viewers when they are viewing the full collection and when they are viewing the results of a search.

4. Form Checking and Data Validation (15 pts)

Your code should contain a reasonable amount of input checking. We will test your input checking rigorously, so please think hard about what you will allow and not allow as user input, and have a good reason for what you decide.

For example, if a form field is meant for searching the category "year," you should only accept 4-digit integers within a reasonable range (ex: movie year released might be restricted to no earlier than the release year of the first ever movie, and before a certain future year).

Other things you may want to consider:

- Should duplicate entries be allowed?
- Should special characters be allowed as input for a name?
- What is the maximum length of a field? Would you accept 1000 characters for a name?

Note: If your first instinct is to use preg_match for doing all your form checking and search functionality, stop and think about what it is you are actually trying to match using regular expressions. There are many PHP functions that already exist for comparing strings against other strings or replacing specific parts of a string. Take a look at them and see if one already exists that meets your needs.

5. Rationale (10 pts)

Describe your strategy for data validation and form checking in a PDF file (**rationale.pdf**) and **upload it to CMS**. Your **rationale.pdf** file should contain the following:

- A description of how you check user input and how you determine which data is valid
- A justification of why you allow some types of data and not others
    - What type of data did you check for in each field/category/filter and why?
        - Do you allow special characters ($, # , !) for entry names? Why?
        - Do you allow alphabetic characters in your date fields? Why?
        - Do you allow duplicate entries? Why?
- An example of a **search over multiple categories** that can be performed using the search form on your website
    - An example of an existing entry that would result from your example query
    - An example of an existing entry that would **not** result from your example query
- When searching multiple fields, is it an “or” search, an “and” search or do you offer a choice? Why did you do it this way?
- Any other design decisions, additional functionality, or notes you want us to know about

Note: You can view a sample rationale in the resources section on Piazza. The questions being answered in that rationale are a little different, but it will give you a sense of what we’re asking you to do.

6. A Persistent Data File (10 pts)

The data file (**data.txt**) on the server should always reflect the current state of your collection. If a user successfully submits a valid Add Entry form, then that new entry should appear in your data file even after the user closes and opens the browser again. The only exception to this is if the entry is a duplicate and you have explained why duplicates aren’t allowed in your collection. You should also fill this file with the first five (5) complete entries in your collection before submission.

A frequent problem programmers encounter working with a read/write data file such as data.txt is a PHP error like this “**Warning**: fopen(data.txt) [function.fopen]: failed to open stream: Permission denied in …”
If this happens, your permissions on data.txt are too restrictive. In your SFTP client, you can set permissions by right clicking the file and selecting "Properties" (WinSCP, FireZilla), or "Info" (Cyberduck). Make sure you set the permissions for the files to allow reads and writes (770 or rwxrwx---)

7. Appropriate Website Design (10 pts)

The design of the site should appropriately match the theme of your catalog site. The site should have a consistent look and feel, with clear, easy-to-follow navigation. If you display results on multiple different pages, you should be able to travel back to the previous page using navigation links or breadcrumbs, instead of relying on the browser’s back button. **Your menu should be consistent on every page and indicate in some way what page is currently being viewed.**

Note: For this project, the use of frameworks will not be allowed (see syllabus), but it is fine to use a preprocessor for your CSS as long as the output is legible to the grading TA.

8. Good Practices (10 pts)

All code should be easy to read and understand by anyone. Make use of comments to explain things that aren’t

obvious, and name your functions and variables appropriately. Your code should be indented properly so that it's easy to see which lines belong to each element (HTML/CSS) or function (PHP/JavaScript). It's also good practice to put all your CSS into external CSS files. Avoid using inline or embedded CSS unless you need to generate styles dynamically through PHP or JavaScript.

Remember, a source credit must display near each image unless you created the image yourself in which case the credit can be in an HTML comment.

When you upload your files to the server, keep them organized in appropriately named directories. CSS files should go into a **css** folder, image assets in an **assets** folder, and JavaScript files in a **scripts** folder. These subfolders and your HTML/PHP pages should go into the project folder **p2**.

Double check to make sure all your HTML output validates, which you can check at http://validator.w3.org/. Also test your add and search forms extensively, so you don't get PHP errors when you submit them. Check your site in Chrome, Firefox and Safari. We may use any one of those for viewing your site.

III. Submission

1. On the course server, upload your website to the **p2** folder. If you don't find a **p2** folder in your **www** folder, create one with that exact name.

2. Your **p2** folder must contain
    a. Either an **index.html** or **index.php** file
    b. A **data.txt** file containing your catalog collection and used for File I/O
    c. Any CSS, JavaScript, image assets, or additional HTML/PHP files needed for your site
    d. If you used a CSS preprocessor like SASS, make sure the output is readable, not compressed, so TAs can see what you did

    e. A backup copy of your rationale.pdf in case TAs aren't able to access CMS during grading.

3. Double-check that your website is uploaded to the correct location by accessing the following URL, replacing **username** with your server username (**netid**sp16):
    http://info2300.coecis.cornell.edu/users/**username**/www/p2/
    And make sure it works. Just because it works on your local computer doesn't mean it will work flawlessly on the server.

4. On CMS, upload your **rationale.pdf** file to the Project 2 assignment. This is your signal to course staff that you are ready to be graded. If it is not there, we will assume that your intention is to turn the assignment in late and accept the late penalty.

IV. Grading Rubric:

The Full Collection (__/ 10 pts)

- The website allows the user to view all entries that exist in the collection (__/ 2pts)
- The collection contains at least five (5) complete entries (__/ 5pts)

- Each entry in the collection contains at least four (4) data fields (__/ 3pts)

An Add Entry Form (__/ 15 pts)

- The website allows users to submit an Add Entry form (__/ 2pts)
- The form allows users to specify at least four (4) different data fields per entry (__/ 3pts)
- The collection on the website is updated when the Add Entry form is submitted (__/ 5pts)
- The form works as expected (__/ 5pts)

A Search Form (__/ 15 pts)

- The website allows the user to submit a Search form (__/ 2pts)
- Users can search for a subset of the collection matching a **single** data field (__/ 3pts)
- Users can search for a subset of the collection matching **multiple** data fields (__/ 5pts)
- The form works as expected (__/ 5pts)

Form Checking and Data Validation (__/ 15 pts)

- The user is appropriately notified of improper data values or duplicate entries (__/ 3pts)
- Duplicate entries are only allowed if justified in **rationale.pdf** (__/ 2pts)
- The implementation matches the description in **rationale.pdf** (__/ 5pts)
- Form data is handled appropriately under all common sense test cases (__/ 5pts)
    - The forms don't accept letters or symbols in a numerical field
    - The forms properly handle embedded HTML tags or JavaScript code from user input

Data Validation Description (__/ 10 pts)

- A PDF file **rationale.pdf** is uploaded to CMS (without this, we can't grade your project)
- Includes a reasonable, justified description of form checking and validation (__/ 5pts)
- The sample query returns at least one entry, and not all entries are returned (__/ 3pts)
- The sample query is not hard coded (other queries work just as well) (__/ 2pts)

A Persistent Data File (__/ 10 pts)

- The server contains a **data.txt** file containing all of the entries in the collection (__/ 5pts)
- This file is updated appropriately when users submit the Add Entry form (__/ 5pts)

Appropriate Website Design (__/ 10 pts)

- All content is legible and easy to read and understand (__/ 3pts)
- Design uses appropriate color, typography, layout, and positioning (__/ 4pts)
- Navigation is easy to follow and indicates current page: menu should be consistent on every page and indicate in some way what page is currently being viewed (__/ 2pts)
- No dangling pages (user shouldn't need to press the back button) (__/ 1pt)

Good Practices (__/ 10 pts)

- Code Clarity (__/ 3pts)
  - Comments are used where needed
  - Lines are indented correctly
  - Variable and function names are human-readable
  - Check HTML, CSS, PHP, and JavaScript files
- All HTML output validates (http://validator.w3.org/) (__/ 1pt)
- Files on the server are well organized in appropriately named directories (__/ 3pts)
- Website is free of PHP errors and images have appropriate credits (__/ 3pts)

WOW (__/ 5 pts)

Note: these points are NOT bonus points! They are PART OF the 100 points for project 2.

Website goes above and beyond the description in at least **one** of the following ways:

- Includes PHP functionality for sorting entries
- Includes PHP functionality for modifying or deleting existing entries
- Includes PHP functionality for resetting the data.txt file
- Includes PHP functionality for highlighting of search terms with CSS in search results
- Makes use of nontrivial Javascript or advanced PHP (like image uploads or sensible use of PHP objects)

(5 strong, 4 good, 3 okay, 2 minimal, 0 none)

V. Relevant Topics

This project is designed to give you more practice with the following topics:

- **File I/O**
  - PHP file(), file_get_contents(), file_put_contents()
  - PHP array(), implode(), explode()
  - PHP for(), foreach(), while() loops
  - PHP string functions str_split(), str_replace(), substr()
- **Forms and Data Entry**
  - HTML <form></form>, form elements
  - HTML <input />, input attributes
- **Form Checking and Validation**
  - PHP $_GET, $_POST, $_REQUEST
  - PHP if($condition), elseif($condition), else statements
  - PHP isset(), empty()

- PHP string functions strip_tags(), strip_slashes(), trim()
- PHP regular expressions, preg_match(), preg_replace()