INFO/CS 2300: Project 2

Create a small online *catalog*.

Overview

Project 2 is about learning how to use a database to render the contents of a web page. To develop these skills you'll build a small catalog website. You'll store the data for your catalog in a SQLite database. You'll then write a PHP web page to read the contents of the database and then present the data in a usable form for the web.

Learning Objectives

- Continuation of developing your algorithm based problem solving skills.
  - Develop good programming habits: planning first and then implementation.
  - Further development of your *programmer's toolbox* to solve problems with code.
- Exposure to the standard practice of backing website content with a database.
  - Using a database with PHP to populate the content of a web page.
  - Experience with using a development database: SQLite.
  - Basics of database schema design.
  - Exposure to basic SQL queries and security measures to prevent SQL injection.
  - Additional practice with *filter input, escape output* for HTTP parameters to help secure your web site.
- Using personas to inform the design and presentation of information.

Deadlines & Receiving Credit

Milestone	Points	Grade	Deadline
Project 2, Milestone 1 (p2m1)	10	Feedback (*completion*)	Tues. 3/6, 5:00pm
Project 2, Final (p2final)	100	Rubric	Tues. 3/13, 5:00pm

Slip days permitted for all deadlines. Maximum of 2 slip days per deadline.

Submit all materials, including plans, designs, and reports to your GitHub repository for this assignment. See each milestone's respective **submit.md** file for directions.

Assignment Repository

`git@github.coecis.cornell.edu:info2300-sp2018/YOUR_GITHUB_USERNAME-project-2.git`
Replace **YOUR_GITHUB_USERNAME** in the URL with **your GitHub username**.

Description

Create a small catalog website that displays any collection of objects in a database.

Examples of catalogs you might consider include:

- A music catalog like Billboard or Discogs.
- A movie catalog like IMDB (but more simplified)
- A video game catalog like VGCollect or Steam (without the download part).
- An online store like Amazon, Ebay, Baker Creek, or Rogue Fitness (without the payment part).
- A visual dictionary like a PokéDex or Bird Guide.
- A simple blog or forum like Tumblr, Twitter, or Reddit (but without user accounts).

Requirements

1. Web Site

You will develop a website with at least 1 page. The site must be backed by a pouplated database. You must include an HTML form for inserting new entries into the database. You must also include a search form for returning search results from the database. You should also thoroughly plan and design your website. Your designs should be informed by the provided personas.

2. Populated Database

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

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

Title	Actors	Genre	Rating
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

Before populating your database, make sure you thoroughly plan your database schema. Your database should make appropriate use of SQL constraints, like primary keys.

Note: You would not typically display the data directly from the database. Instead, use PHP, HTML and CSS to create a template to provide formatting and then read the data for all entries into the template from the database.

3. Data Entry Form

Use an HTML form that allows users to add an entry into your database. 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.**

4. A Search Form

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 search by multiple fields.

> Example: Search for movies where **Rating = PG** or search for movies where **Title = "Toy Story"**.

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

5. Filter Input, Escape Output

Your code should filter input and escape output.

Filter Input

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).

You do not need to to validate your forms (what you did in INFO 1300). Form validation is checking the form input and sending a populated form with error messages back to the user if their submission was incorrect. While this is always a good idea, there probably isn't time for this for on this project. Instead make sure you filter your inputs to secure your web site. If you choose not to validate the HTML forms, you should still provide some indication of an error if the user submits invalid data. Keep your persona in mind when providing error feedback.

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?
- How will you handle extraneous spaces before and after parameter values?

Note: If your first instinct is to use regular expressions 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. Lab 4 has links to reference documentation for these helper functions.

Escape Output

Make sure you escape any dirty inputs. Be careful to escape for the appropriate context. You should escape for HTML and SQL. We will try to break both when grading your assignment.

6. Design Guidelines

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 appropriate. The design should work well for your selected persona. You design does not need to be fancy, but it should look nice.

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 persona should never get trapped by navigation.

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.

Your site should display reasonably well (not necessarily identically) across Firefox and Chrome. We will check your website on any one of these browsers.

You may assume that your web page will be viewed in a desktop browser. You do not need to design a web page for mobile browsers. You do not need to use media queries for responsive design.

7. Coding Standards & Design Guidelines

Your assignment should follow the coding standards, conventions, and expectations of this class. See Project 1 if you need a reminder for some of them.

All code should be easy to read and understand by anyone. Use functions and includes to help organize your code. Make use of comments to explain things that aren't obvious. 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).

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

Your website should be under 10MB.

All code must be your own for this project. You may not use frameworks such as bootstrap.

Milestone 1

10 points.

In Milestone 1, you'll design and plan your website and database. You'll also inform your design based on a persona of your choice. Like Project 1, you'll author your plans in Markdown in the *design-plan* folder of your Git repository.

1. Pick a Persona

You have 4 personas to choose from located in the **design-plan/personas.pdf** PDF document. You may pick any persona you like. Make a note of your selected persona in your Design & Planning document: **design-plan/design-plan.md**.

2. Describe your Catalog

Take some time to think about the type of catalog you will create. What type of *thing* will you catalog in your collection (database)? What types of attributes (database columns) you will store for the *things*.

For example, for a movie collection, you might store the title, main actors, the genre and rating.

2. Sketch & Wireframe

Now that you know what type of collection you're going to create, make a quick sketch of your website. Use this sketch to help you figure out the location of your collection (probably an HTML table), the entry form, and the search form. As always, keep your persona in mind when authoring the sketch.

After making a quick sketch, create a wireframe for your page. When finalizing the design details in your wireframe, make sure that it's a design that will work for your persona. Take the time to layout the page elements carefully and to make sure your tables and forms have proper labels.

4. Plan your Database

Plan out your database schema. Think about the following questions.

- How many tables will you have? (you only need one)
- What columns/fields will you have in your table(s)? (you only 4 columns/fields)
- What type will each column/field be?
- What constraints will your fields have? (primary key, not null, auto increment, etc.)

In the **design-plan.md** file, describe your database plan. You may do this anyway you like. We just want to see that you've thoroughly thought through the structure of your database.

Tip: use a bulleted list where each bullet is a field. Describe what that field will hold, its type and constraints, etc.

5. Plan your Database Queries

You'll need at least three (3) database queries: 1) retrieve all results, 2) search the results by a user selected field, and 3) insert a record into the database. Describe your plan for these queries. You may use natural language, pseudocode, or the SQL queries themselves.

Example:

> **Natural Language/Pseudocode**: Select all records and all fields from the movies table.
> **SQL**: `select * from movies;`

6. *Filter Input, Escape Output* Plan

Describe your strategy for *filtering input and escaping output* for your HTTP parameters. You may use a combination of natural language and pseudocode for your plan. Use whatever method you feel best communicates your plan to the teaching staff.

This should include at least 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?
- What values will you escape and what methods will you use? Don't forget about contexts (HTML vs. SQL).
- Any other design decisions, additional functionality, or notes you want us to know about.

Tip: Feel free to utilize user-defined functions to make this process easier for you.

7. Additional Code Planing

You may feel like the above planning left something out. Feel free to include any additional planning you need here.

8. Populated Database

Create your database based on your plan and populate it with data. Your database should be titled **data.sqlite** and should be located in the *root* of your repository (i.e. it should be at the same level as *index.php*). Make sure you have at least 5 records/entries in your collection's main table.

Grading - Feedback

This is a feedback milestone. Your grade will be based on whether you submitted a complete assignment. If it looks like you tried, even though there are some mistakes, you'll get full credit. If your submission is obviously incomplete, you'll get a 0. There is no partial credit, either you submitted the assignment or you didn't.

We'll provide feedback on your milestone to help guide your work for the final submission. **Our feedback is designed to help you learn more**; our feedback is not a "pre-grade". This feedback is designed to catch large problems (which we sometimes miss). **This does not resolve you of the responsibility of meeting the project's requirements, even if we miss something.** Your feedback will show up as comments for the assignment in CMS. Please read it and use it to improve your learning and your project.

Note: Your Milestone 1 artifacts will be graded for points using the rubric below (not completion) in the final submission. Feel free to update and change these documents based on feedback. In fact, it's a good idea to do so. It will likely improve your grade.

Submission

See **submit-*m1*.md**.

Failure to follow the submission instructions **exactly** will result in a late submission (2 slip days or a 0). **No exceptions.** Late submissions will not receive feedback until a least one week after the deadline. Using slip days has consequences. Not receiving timely feedback is one of these consequences.

Final Milestone

100 points.

You will code the web page you designed and planned in Milestone 1. Your entire web page should be authored in HTML, CSS using PHP with the PDO extension to query your database. You may optionally use JavaScript, however it is not required. When coding up your final submission, make sure you've met the requirements above.

Tip: Test your final web site thoroughly, especially your forms. We will try to break your web page during grading. Make sure that we can't inject HTML or SQL.

Grading & Submission

The final milestone is graded via a rubric.

See **submit-*final*.md** for submission instructions. Failure to follow the submission instructions **exactly** will result in a late submission (2 slip days or a 0). **No exceptions.**

Tip: Before submission, re-read this document and rubric and check that you've met the requirements.

Rubric

I reserve the right to change this, but this should be close to the final version.

Planning (20%)

- Complete and thoughtful Design & Planning document.
- Sketches and wireframes are informed by a persona and follow visual design principles.
- Includes a reasonable, justified plan of *filter input, escape output*.
- Database plans are thoughtful and complete.

Design (20%)

- Design is appropriate for persona.
- The design pleasant and follows visual design principles. Design uses appropriate color, typography, layout, and positioning.
- The HTML form reasonably limits your persona from entering incorrect data (letters for a number, etc.).
- The user (your persona) is appropriately notified of improper data values or duplicate entries.
- 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. No dangling pages (user shouldn't need to press the back button).
- Website renders correctly in Firefox and Chrome (minor differences acceptable).

Implementation (50%)

- Implantation follows your plan.
- Website is under 10MB (does not include the *design-plan* or *.git* folders).

Database

- The web page is implemented in PHP and connects to a SQLite database using PHP's PDO extension.
- Database follows your plan.
- All collection based data in the web page is queried from the database.
- The website allows the user to view all entries that exist in the collection.
- The collection contains at least five (5) complete entries.
- Each entry in the collection contains at least four (4) data fields.

Entry Form

- The website allows users to submit an Add Entry form.
- The form allows users to specify at least four (4) different data fields per entry.
- The collection on the website is updated when the Add Entry form is submitted.
- The form works without any errors.

Search Form

- The website allows the user to submit a Search form.
- Users can search for a subset of the collection based on the selected data field.
- The form works without any errors.

Filter Input, Escape Output

- Web page is secure from injection based attacks.
    - All input is properly and appropriately filtered.
    - All output is escaped for the appropriate context.
- The implementation matches the description in your plan.
    - Duplicate entries are only allowed if justified in your plan.

Coding Standards (10%)

- The code follows the standards, conventions, and expectations of this class.
    - All HTML output validates (http://validator.w3.org).
    - Are the HTML, CSS, PHP, and JavaScript (if any) well-formatted, commented, and readable?
    - Comments are used where needed (good comments make it easier to give partial credit!)
    - Lines are indented correctly.
    - Variable and function names are human-readable.
    - External styling via CSS. **No inline or internal styling**
    - All files are well organized (styles folder, images folder, no redundant code).
    - Website is free of PHP and SQL errors.
    - External resources (images) have appropriate credits.
    - etc.
- No external code or libraries used. All code is your own work.