Project 3: Well-Designed Photo Gallery

You will design and implement an online photo gallery in PHP. You will demonstrate your ability to design an aesthetically pleasing and usable interactive site. Your photo gallery will be backed by a database which you will use to store information about the images uploaded to your gallery. To help organize the photos in your gallery, you will implement the ability for users of your site to tag the photos.

1. Learning Objectives

- Develop a professional interactive website suitable for a portfolio.
- Develop the skills to translate client/customer requirements into a working implementation.
- Leverage design patterns to improve the usability of your site.
- Practice using your *web programmer's toolkit* to solve complex problems.
- Practice structuring a database with multiple tables and foreign keys.
- Practice building and querying relationships between tables using common fields (joins).
- Employ best practices for user uploaded content for dynamic websites.

2. Deadlines & Receiving Credit

Milestone	Points	Grading Method	Slip Days	Deadline
Milestone 1 (*p3m1*)	25	Feedback (*completion*)	Maximum: 2 days **slip days = no feedback!**	4/15, 11:59pm ET
Milestone 2 (*p3m2*)	25	Feedback (*completion*)	Maximum: 2 days **slip days = no feedback!**	4/22, 11:59pm ET
Final (*p3fin*)	**150**	Rubric (*p3m1* + *p3m2* + *p3fin*)	Maximum: 2 days	4/29, 11:59pm ET

IMPORTANT! No feedback is provided for milestones submitted with slip days! If you want feedback, please submit by deadline.

Milestones are graded *twice*. First for feedback (completion grade only). Lastly, for points at the final submission (via rubric). ***All* work is graded via rubric for points at the final submission.** Use your milestone feedback to improve your final grade; **revise milestone work prior to final submission**.

Completion credit will be awarded so long as you made a good faith effort to complete the milestone's requirements. Very obviously **incomplete milestone submissions will receive a 0** for the completion grade.

Milestone feedback is not a *pre-grade*. This feedback is designed to catch large problems (which we sometimes miss). Regardless of the feedback (or lack of feedback) that you get, ***you* are responsible for meeting all of the project's requirements for the final submission**.

Failure to *push* your submission to GitHub is equivalent to not submitting the assignment. You will receive a 0. It is your responsibility to verify that you submitted your assignment.

3. Project Tips

This project is *challenging*. Start working on this project now. Take the milestones seriously. **Seek help early.** Do not wait till the last minute to start the project or seek help.

Take the design of your website very seriously. Since Project 4 was eliminated, your Project 3 is considered your portfolio showcase for this semester. Your design should look professional, be aesthetically pleasing, follow visual design principles, and be usable by your target audience(s).

4. Git Repository & Submission

Clone `git@github.coecis.cornell.edu:info2300-2020sp/YOUR_GITHUB_USERNAME-project-3.git`. Replace **YOUR_GITHUB_USERNAME** in the URL with **your GitHub username**. This is usually your NetID.

Submit **all** materials to your GitHub repository for this assignment. **See *README.md* in your Git repository for submission instructions for each milestone.** Never email your submission to the instructor.

Tip: Commit and push your changes every time you work on your project. Every time you commit and push you store your changes on the GitHub server. This acts as a back-up for your work. It also means that if you forget to submit before the deadline, there's something already on the server that the TAs can grade for partial credit.

5. Documenting Design (Design Journey)

We will grade your **design-journey.md** in VS Code's Markdown Preview. **Everything, including images, must be visible in VS Code's Markdown Preview.** If it's not visible in VS Code's Markdown Preview, then we won't grade it. We won't give you partial credit either.

If you included images in your design journey, they must be visible in VS Code's Markdown Preview. No credit will be provided for images in your repository that are not properly linked in Markdown. No credit will be given for design work not included in the **documents/design-journey.md** file; **do not rename this file or put your answers in another file!** Remember to check and test all assignment submissions!

Requirements

Create an online image gallery.

- Your gallery must be designed for at least one specific target audience.
- Your design should be aesthetically pleasing and follow visual design principles.
- The site's navigation and all content should be well organized for your target audiences.
- You will implement your website in PHP with all data stored in an SQLite database.

  When generating the HTML for your web page, you should use SQL queries to retrieve specific data from the database and *directly* use that data for your HTML generation.

  Do not use PHP to *find* specific data; directly retrieve specific data from the database through well-written SQL queries. For example, do **not** retrieve *all* the records from the database using SQL and then use PHP to loop through and find the data you want. SQL is much more efficient at searching and we expect you use it where appropriate.
- You may use HTML, CSS, JavaScript, PHP, and SQL.
  - You may not use external code except for the Lab 5 Component Library.
  - You may use jQuery.
  - You may not use Bootstrap.
- You *may* optionally build on your Project 1 or Project 2 if you like. Document your existing design the design journey.

  Note: Use (*includes/init.php*) from Project 3, not Project 1 or 2.

Note: Your content can be something other than an image gallery but it must follow a similar structure. If you choose entities other than images and tags, you should explain in your design journey how your entities correspond to the requirements of the assignment as written in the language of images and tags.

1. Photo Gallery

- Users should be able to view *all* images in your photo gallery at once. (e.g. a gallery/thumbnail page)
- Users should be able to view *all* images for a *tag* at once. (e.g. tag page or filter by tag on gallery page)
- Users should be able to view a *single image* and all the tags for that image at once. (e.g. image details page)
- Users should be able to upload a new image.
- Users should be able to remove (delete) an image.
  - Make sure you clean up any relationships to the image in other tables. (Where the image is a foreign key.)
  - Make sure you delete the corresponding file upload from disk.
- Users should be able to view *all* tags at once. (e.g. a list of tags)

- Users should be able to add an existing tag to an image, add a new tag to an image, and remove a tag from an image.
- Tags must be unique. You cannot have duplicates of the same tag.
- You may store and display any other type of data you like for the images. Although none of this is required.
  - example: A user defined description for the image or the date it was taken.
  - example: The file format or size of the image.

2. Design

- Your design should be aesthetically pleasing and employ visual design principles.
- Your photo gallery should employ the common design patterns you identified in Lab 4.

  You may use the *Design Pattern Snippet Library* from Lab 5 **as a reference** for supporting these design patterns. If you use the component library, make sure you follow the citation requirements as outlined by the course's **External Code Policy**.

  DO NOT COPY AND PASTE CODE!
- **Do not design a website with "submit" style buttons everywhere.** None of the image galleries we analyzed in lab 4 did this. You shouldn't either.

  For example, delete is often implemented as a menu item.

3. Requests

- All requests made to the server must use an appropriate request type: GET or POST.

  See guidelines in your textbook or refer to INFO 1300 notes for guidance.

  Failure to do so will likely result in multiple deductions due to the cascading issues with using incorrect request types (i.e. security, usability, etc.)
- If you do **not** require user input (no input components), but need a GET request, use a URL with **query string parameters**.

  GET requests that do not require user input should use `<a>` elements for URLs with query string parameters. Style these elements to leverage existing design patterns.

  For example, to view the *balloons* tag on the image gallery page, you may have a link for the *balloons* tag that uses query string parameters: `gallery.php?tag=balloons` or `gallery.php?tag_id=3`.

  Another example, to view the details for a specific image, you may have an image page that accepts an id: `image.php?id=8` or `image.php?image_id=8`

  Hint: You will need to make extensive use of query string parameters for most of your links.

  Hint Hint Hint: https://www.php.net/manual/en/function.http-build-query.php

- All POST requests require a `<form>` element. (Even if no user input.)

  Deduction Warning: Do not use JavaScript for POST requests. Use a form.

  For example, uploading an image or deleting an image meets the guidelines for a POST request. Remember to style the "submit" button to leverage existing design patterns. Delete should likely be a menu item, not a "submit" style button.

  Hint: You may use hidden inputs if necessary.

- If you require user input for a GET request, you may use a `<form>`.

  For example, search. Search requires user input and is typically meets the guidelines for a GET request.

  Hint: You may use hidden inputs if necessary.

  Deduction Warning: If you do not require user input for a GET request, do not use a form with hidden inputs. Use query string parameters.

4. Database

- You may only have 1 database; you may not create more than one database.
  - Your database may have as many tables as you like.
  - There is no requirement for a minimum number of tables.
- You should plan your database before you implement anything.

  Tip: Planning your database is the key to success for this project. A bad database design can make the basic tasks of this project extremely difficult.

- Your plan should include a schema for each table in your database with constraints.
- Your database is required to implement a **many to many** relationship.

  Your two main entities, images and tags, should be in a **many to many** relationship. One tag can have many images and one image can have multiple tags.

- You are required to use SQL join clauses when needed gather data from multiple tables.

  Deduction Warning: Algorithmically implemented search in PHP is prohibited; you may not retrieve records from the database and then use a loop in PHP to 'search' through the database. Instead, write a single SQL query to achieve the same result.

- All database tables must have a primary key called `id`.
- All foreign keys should follow the convention of *singular table noun* `_id`.

- You are required to create/initialize your database using SQL.
    - Initialize your database by using an initialization script (**secure/init.sql**).
    - **Do not commit and push any .sqlite or .db files.**
    - You may ignore the **.checksum** file created by `open_or_init_sqlite_db()`.
    - You are **not permitted** to use **DB Browser for SQLite** for creating or modifying your database for this assignment; you must use init.sql.
    - You may use DB Browser for SQLite to *view* your development database as well as *test* your SQL queries.
    - All seed data must reside in **secure/init.sql**. Do not use DB Browser for SQLite to "create" seed data.
- Rigorous SQL injection prevention is required for this assignment. Use prepared statements with parameter markers.
- You must connect and interact with the database using PHP's PDO extension. No other methods/libraries are permitted.

5. File Uploads

- All file uploads should be placed in a sub-folder under the **uploads** folder.
- All *uploads* sub-folders should have the same name as their corresponding database table.

  For example, if you have a table called *images* then you should have an folder called *uploads/images*.
- All files stored in each *uploads* sub-folder are **required to be named as the primary key** of the corresponding database record.

  For example, for image with `id` of 1, you should store that as *uploads/images/1.jpg*.

  Hint: Store the file extension as a *field* for each record to support multiple image formats (jpg, png, gif).
- Do not commit any uploaded images (except seed images) to your Git repository.
- **Do not store file uploads in the *images* folder;** store file uploads in the *uploads* folder.

  Never mix static site files with user data.

6. Seed Data (Populated Database + Images)

- You should provide initial *seed* data for your website.
    - You should have at least 10 images.
    - You should have at least 5 tags.
    - At least 3 tags must applied to at least 1 image.
    - At least 8 images need to have a tag.
    - At least 3 images need to have *multiple* tags.

- All seed data should be SQL queries in your database initialization script (**secure/init.sql**).

  You are not permitted to add seed data to your development database using DB Browser for SQLite.

- All seed image files should be stored in a sub-folder under the **uploads** folder.
  - The sub-folder should be named the same as the corresponding database table.
  - All seed images should named to correspond to their respective database record's primary key.
  - **Do not commit files in the uploads folder that are not seed files.**
- Your seed data and uploads should be under 10MB.

7. Best Practices

- Your website's implementation should follow best practices (see Project 1, Project 2, and class notes).

  This includes making efficient use of templates (PHP includes) and user-defined functions.

- Filter Input
  - You should thoroughly filter all input.
  - Use parameter markers in your SQL queries
- Escape Output
  - Make sure you escape any untrusted inputs.
- You should rigorously and thoroughly test your website and forms to make sure there are no errors and everything works as intended.

8. Scope

- There is no minimum/maximum number of pages you need to implement.
- You do not need to implement a responsive design with media queries.
- You must filter input and escape output to protect your database and your users.
- Your forms should provide corrective feedback.
- Your forms do not need to be sticky.

Milestone 1: Design, Plan, and Draft Website

In Milestone 1, you'll design and plan your website and database. You'll also generate *seed* data and build a draft of your website.

Complete the **Milestone 1** section of the design journey: **documents/design-journey.md**.

1. Design

Design your Project 3 website. When designing take care to meet the gallery requirements above. Design is substantial component of Project 3. Take it seriously. You should strive to produce a well designed site that you would be proud to include in your portfolio.

Feel free to annotate your sketches with notes about how the behavior or design changes based on certain conditions. For example, "only show tags for this image..."

There is no minimum number of sketches. Include as many as necessary to fully plan out your website's design.

2. HTTP Requests

Your gallery will need a quite a few HTTP requests to function. Plan out your requests in the design journey. Use your final design to determine every request you need, what that request does, what type of HTTP request it is (GET or POST), and what parameters you will pass along in your request.

Carefully follow the request requirements above.

3. Plan your Database Schema & Queries

Plan out your database schema.

Take the time to carefully plan your database. The extra care you put into planning your database will save you time in later milestones with less complex SQL queries. Trust me, **poorly planned databases can lead to some pretty impossible SQL queries**.

Think about the types of SQL queries you need to implement your design.

You may feel totally lost here. You may feel like there are so many requirements and you don't know where to begin. This is normal. It's difficult learning how to take the tools from your *web programmer's toolbox* and use those to implement something new. Take it one step at a time. Think about how you might write a query for each of the requirements above. After you go through this process, you'll starting feeling a lot more confident about this!

4. Plan PHP Code

Plan out the structure of your website.

Tip: Do not have a PHP page for each table in your database. You'll often discover that the way you present information to your user is different than the way you store that information in the database.

Plan out your pseudocode for each page. Each page's pseudocode should indicate where your planned database queries will execute. They should also take care to *filter input, and escape output*.

For example, you might want to do something like this:

```
if the user uploaded image (SELECT user_id from images WHERE id = 4) then
  show all tags (SELECT * from tags WHERE image_id = 4) to the user so they can delete them if
they want

  if user clicks on delete link (image.php?action=delete_tag&tag_id=10&image_id=4), then
    (DELETE tag_id = 5 for image_id = 4 in image_tags table)
    (double check during filter input that current user "owns" the image before permitting
deletion of tag)
  end
end
```

Note: Observe how I'm using query string parameters to pass information back to the *server* in order to complete certain requests.

It's okay if your plan doesn't exactly match your final code, although it should be close. **The point of this exercise is to have you thoroughly think through how you will implement this before you start coding.** It's a lot easier to starting coding once you've thought through how you will code it. I really want to help you develop this habit of planning first, and coding second. It really does save you time in the long run and you'll often be expected to do this for most programming related jobs.

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

5. Database Creation & Seed Data

Create your database based on your plan and populate it with initial seed data. Take care to meet the "Seed Data" requirements above.

You are not permitted to create or populate your database with *DB Browser for SQLite*. For this project you will need to write the SQL queries to create the tables and insert the seed data in the **secure/init.sql** file. You will need to look up the reference documentation for creating tables in SQL. We are not covering this in lecture or labs because we want you to have practice with SQL reference documentation.

We are using this script approach (init.sql) so that once Project 3 is over we can help you *deploy* your website to an actual live production server. Recall that production servers do not usually support SQLite. Instead they use MySQL or PostgreSQL. In order to *deploy* your website you'll need to have an SQL script ready in order to create the database on actual web server.

You will also need to upload seed image files for your seed data. Follow the requirements in "Seed Data" above.

6. Draft Website

Start coding up your website. You should try to get as much of the HTML and CSS done as you can. That way you can focus on the PHP and SQL in future milestones. **It's okay if you don't get very far yet. Just get it started.**

Milestone 2: Gallery

In Milestone 2, you'll begin to implement the gallery.

1. Implementation Requirements

- The user should be able to view all images in a gallery.
- The user should be able to view a single image and it's details.
- The user should be able to upload an image.
- Your tags should be mostly there, but it's okay if some of it doesn't yet work.

2. Connecting to the Database

Since we're no longer using *DB Browser for SQLite*, you'll need handle creating your database differently to support deploying our website after you've finished this Project.

You will need to connect to your database using the `$db = open_or_init_sqlite_db('secure/gallery.sqlite', 'secure/init.sql');` function provided for you in **includes/init.php**. This function checks to see if your SQLite database exists, if it does it connects to it. If it does not exists (i.e. there is no gallery.sqlite file) then this function creates the database initializing it with your **secure/init.sql** script. If you want to delete your database, just delete the **gallery.sqlite** file and the next time you refresh a page in your web browser, your database will be recreated!

IMPORTANT! If you change **secure/init.sql** you **must delete *secure/gallery.sqlite*** to regenerate the database. Many students fail to do this before their final submission only to submit broken init.sql scripts. The database then fails to create for the grader and we pretty much have to give you a 0 because your Project 3 won't work without a database. **CHECK CHECK CHECK your init.sql script by deleting gallery.sqlite and visiting index.php in your browser BEFORE you submit!**

Warning: Do not run the PHP Server and DB Browser for SQLite at the same time! This can cause your database to become corrupted. If you need to view the database, stop the PHP server and then open DB Browser. When you are done with DB Browser you must exit it before you restart the PHP server.

Tip: If your database does become corrupted and you need a new one, simply delete the **.sqlite** file from the secure folder. The next time you connect to your database from your PHP code, your database will be recreated from **init.sql**.

Final Submission: Complete & Polished Website

You will finish coding the website that you planned in Milestone 1 and started implementing in Milestone 2. You should now implement everything necessary to meet the full requirements of the assignment. **You should spend most of your time polishing your design.** Your design should be professional, be aesthetically pleasing, employ visual design principles, and leverage existing design patterns to improve usability; your design should be worthy of your portfolio.

Complete the **Final Submission** section of the design journey: **documents/design-journey.md**.

Test your final web site thoroughly, especially your forms. We will try to break your web page during grading. We will also try to inject HTML & SQL during grading.

IMPORTANT! TEST YOUR DATABASE! Delete gallery.sqlite and visit index.php and double check that your database was successfully re-created from init.sql. No database created for the graders = 0.