Analysis and Design Report

Developing for the internet

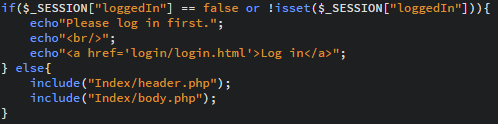
# Planning

I have used the latest version of Bootstrap to style my site.

Before beginning task A, I decided to build a skeleton site. This consisted of:

* Login page
* Login script
* Index page

The index page consisted of a header and a body. I initially included a footer but found no use for it.



The login page and script are more or less the same as the one used for my HitTastic! site previously built, they simply disallow access and provide a link to the login page if the user is not logged in. Every page requires the user to be logged in, as certain session variables need to be set for the site to function.

# Section A

Allow a user to add a new venue.

For this task, I would need to create a class for any venue objects that I would be using. This class would need a constructor, getter methods and both a function to update a venue, as well as a function to add a new venue to the database.

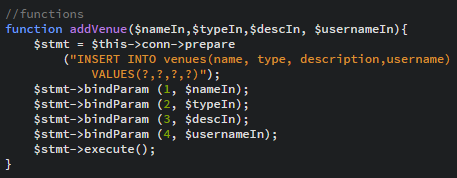
I would need an interface for the user to input three pieces of information: name, type and description. The id would be automatically generated and assigned upon creation, username would be automatically assigned, and the recommended column would be updated automatically as reviews are added. Finally, a script would be needed to create a venue object and add it to the database.

Page(s) necessary for task:

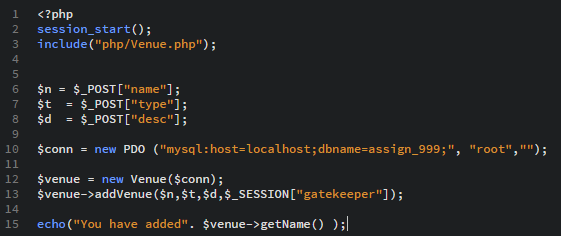
* Venue.php
* addVenue.php
* AddVenueScript.php

Before implementing anything, I created the venue class. It contained a constructor, various getter methods and two functions, addVenue() and updateVenue().

addVenue() takes in the four variables and uses SQL to insert them on a new row into the venues table. The id is assigned automatically. I used a prepared statement for this, to protect against possible SQL injection, and the compromise of my database.



addVenue.php contains a form with a POST method, linking to AddVenueScript.php. This script is shown below. I chose post for this data as browser URL limits apply, and the description has the potential to be a lengthy string. It assigns the users three inputs to variables, creates a new connection to the database and creates a new venue object. It then uses the addVenue() function to input the three values, plus the user’s name into the database.



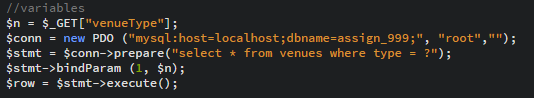
# Section B

Allow a user to search for a venue by type.

To achieve this, I would need to create a search interface for the user and a script to search for and present search results. For this task I used:

* searchResults.php
* searchByType.php

searchByType.php holds a form, containing a search box and a submit button. It uses the GET method to pass the search data to searchResults.php, as it is not sensitive data. searchResults.php takes the variable and uses a prepared statement to retrieve matching values from the database.



Using a while loop, the search then proceeds to echo out each attribute of the results along with an apprpriate descriptor.

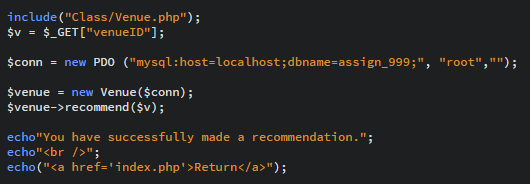


# Section C

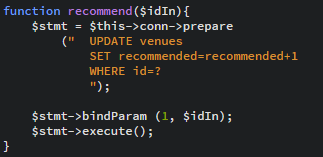
Allow a user to recommend a venue.

For this task, I needed to add a new function to my venue class, and a script page to allow the user to recommend a venue. Page(s) necessary for task:

* recommendVenue.php



The script above is from recommendVenue.php. This uses GET to retrieve the venueID, as it is not sensitive data. It creates a new connection and a venue object, and calls the recommend function, parsing the venue ID along with it.



Back inside of the Venue class the recommend() function is called. Using a prepared statement, it identifies which venue to update by comparing the venue ID parsed in, and increments it.

# Section D

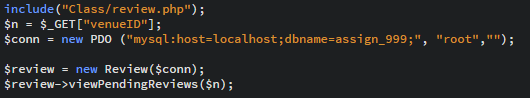
Allow a user to view all reviews for a given venue.

This task extends on the search results page previously built. Each search result returns three links, one to recommend the venue, to review the venue and to view all existing reviews for the venue. The view reviews link will send the user to another page, where the reviews are displayed in a list. I would also need to implement a review class, along with necessary functions.

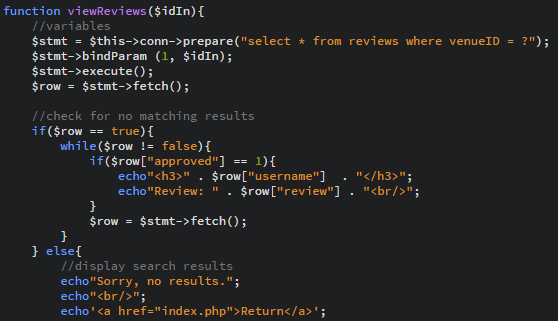
Page(s) necessary for task:

* viewReviews.php
* Reviews.php

The viewReviews.php page contains the following script. It retrieves the venue ID through the GET method and creates a new database connection. A new review item is then created, and the viewReviews function is called., parsing in the venue ID.



From there, a prepared statement is used to search the reviews table for all reviews matching the parsed in venue ID. An if statement is used to check if there are any results. If there are, then a while loop runs, displaying each result in a list.



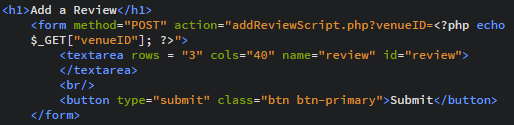
# Section E

Allow a user to review a venue.

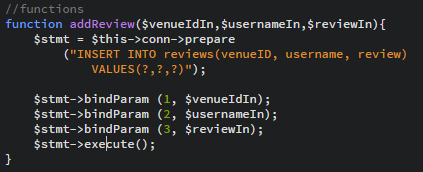
One of the links from the searchResults.php search results link to a fresh page, parsing in the venue ID as a URL query. I will need an interface for the user to input their review, as well as a script to process it. A new function in the review class will be made. Page(s) necessary for task:

* addVenue.php
* addVenueScript.php

addVenue.php contains a form, using the GET method to pass the venue ID onto the script. The review itself is sent across via POST, due to browser URL character limits. Once the user presses submit, the addVenueScript page is loaded.



The script takes both values, creates a new database connection and review object and parses the venue ID, the user’s username and the review to the addReview function. From there, it acts extremely similiarly to the addVenue function from earlier. Using a prepared statement, it inserts the three values into the database, with a unique ID being automatically generated upon creation.



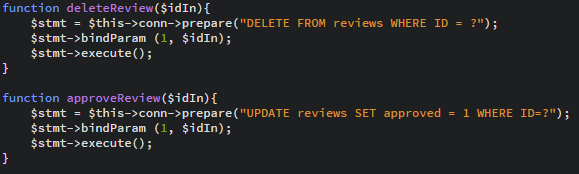
# Section F

Extend the reviews system so that all reviews have to be approved by administrators.

For this task, I needed to create a new page for administrators only. They would need to be able to view and approve all pending reviews, as well as delete them. Two new scripts for both actions would be required, and I would need to add two new functions to Review.php, to add and delete reviews. Page(s) necessary for task:

* adminApprove.php
* deleted.php
* approved.php

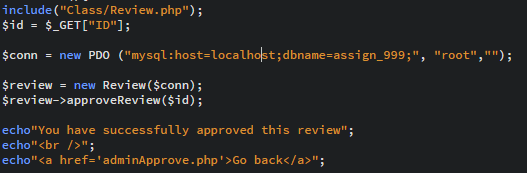
To begin with, I added the two functions I would require in Review.php. They are both extremely similar in logic. They take the ID in and perform a query; either to remove the entry entirely or set the ‘approved’ attribute to 1. Both are performed in prepared statements to prevent SQL injection.



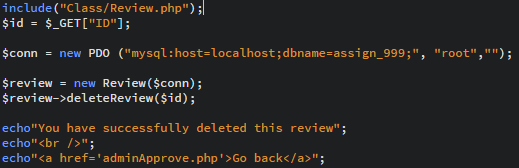
I decided that creating a third function, viewPendingReviews() would be necessary. It is extremely similar to pendingReviews(), however it selects all reviews that are not approved. It then cycles through every row matching the criteria, and creates two buttons for each: approve and delete. These link to approved.php and deleted.php.



approved.php gets the ID, created a new connection and a new Review object. It then calls the approveReview() function mentioned previously and displays a message to the user.



deleted.php works in an identical way, getting the ID, setting a connection and calling the deleteReview() function.



# Task G

Implement the majority of tasks using appropriate object-oriented PHP.

Wherever I could, I opted to use object-oriented programming. Modularity and reusability are key in larger projects, can save a lot of time, and can make code more readable. Page(s) necessary for task:

* Review.php
* Venue.php

# Task H

Implement an AJAX interface.

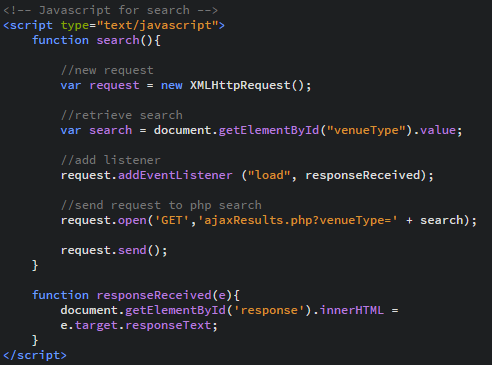
For this task, I would need to add a Javascript request to a PHP script to retrieve and display information automatically without reload. I edited my previous searchResults.php, and re-saved it as ajaxResults.php. Page(s) necessary for task:

* ajaxResults.php
* body.php

I started by creating my search bar; it would not contain a search button as it is responsive and contains a div below it, which will contain the response from my Javascript. The ‘onkeyup’ value runs my search() function every time a key is lifted and is therefore responsive to user input.



The next step was the create my Javascript. It contains two functions; search(), and responseReceived(). The search() function makes a new XML Http request, retrieves the value currently stored in the search bar, and adds a listener for when it gets a response. The function then sends the request to ajaxResults.php along with a query string containing the search contents. When the server responds, the responseRecieved() function is called, and displays the response on the page. This is then outputted in the div tags from earlier. The script is shown below.



Finally, this is my ajaxResults.php page. It uses GET to retrieve the search contents, and conducts a query to using LIKE, which will retrieve all results containing even only partially the search contents. This is done in a prepared statement for security reasons. The results are then echo’d back in the exact same way as they were in searchResults.php.

