Points of Interest Website Design and Analysis

By Matthew Dear

Task A: Allow a user to add a new POI. (Point of Interest)

This task will require a number of pages login.php, index.php, functions.php, addPoiForm.php, and addPoi.php. The login.php script is needed to identify the user when they add a POI, this will be done using the gatekeeper session variable. The index.php page is needed to join everything together. The functions.php script is used to add links, back button, footer, and page header to all pages. The addPoiForm.php display's a form for the user to enter the new POI details and sends them to the addPoi.php page using a POST request. A POST request has been used because the information sent will be added to the database this improves security and protects the quality of your data.

functions.php

The functions.php script will contain five functions in total and all of these functions have been placed in this file so that they can be called from any page, this allows for code re use and avoids repetition. The first being the title function that will take one argument a user type. This will be a 1 if the user is an admin and a 0 in all other situations. It will output the logo and page title before calling the links function and passing in the user type as an argument. The links function will then use an if statement to check if the input user type is equal to 1. If this is the case it will create an associative array containing all the admin links. It will then use a foreach loop to iterate though the array and output the links to the page. If the value is not equal to one it will follow the same process but, the awaiting approval link will not be included. The back button function will be used to output a HTML button to the page that uses the JavaScript history function to return you to the previous page in your browser history. The database connection function will create a new PDO connection object for the database and then set the error modes to allow for better error reporting. Once set the PDO object will be returned to the requesting page or script. This entire process will be surrounded by a try catch block to stop any connection issues from completely breaking the site. The final function called footer will be used to echo out the HTML footer for the website.

login.php

This page will start with the built in PHP session start function that will either start a new session or resume one that already exists for that user. It then uses the built in PHP include function to link it to the functions and userDAO scripts to allow their functionality to be used. It will then call the custom title function from the functions script passing in the contents of the isadmin session variable if it has been set otherwise, it will pass in 0 by default to make sure that users not logged in still get a navigation menu. Then using an if statement the page will check if the gatekeeper session variable has been set, if it has the uses name will then be echoed out to the page. The custom back button function is then called completing the header section of the page. The username and password that have been sent to the page from the login form page using a POST request are then store in the un and pw variables. A try catch block is then opened and the custom database connection function is called and the returned PDO object is stored in the conn variable. An if statement is then used to validate the username and password from earlier, custom regular expressions are used to validate the inputs and defend against cross site scripting attacks. If the inputs are not valid an error will be echoed out to the page. If the inputs are valid then a new usersDAO is created passing in two arguments the conn variable and the hard coded table poi_users. The returned usersDAO is then stored in the variable usersDAO. Once created the verify login function will be called on it passing in two arguments the un and pw variables and the returned usersDTO will be stored in the variable usersDTO. Next the gatekeeper session variable will be set to the value of the un variable and the isadmin session variable will be set using the isadmin attribute of the usersDTO variable. Then the built in PHP header function will be used to return the user to the index page. Finally, after closing the try catch block the custom footer function will be called.

index.php

The logic of this page works the same way as the login page up until the end of the header section. The only difference is the included scripts for the index page are functions and poiDAO. A try catch block is then opened and the custom database connection function is called and the returned PDO object is stored in the conn variable. Then a new poiDAO is created passing in two arguments the conn variable and the hard coded table name pointsofinterest. The returned poiDAO is then stored in the variable poiDAO. Once created the find regions function will be called on it and the returned array will be stored in the variable pois. A foreach loop will then iterate thought the returned indexed array to output the regions in the database to the page. Finally, after closing the try catch block the custom footer function will be called.

addPoiForm.php

The logic of this page works the same way as the login page up until the end of the header section. The only difference is the included script for the addPoiForm page is functions. A form is then echoed out to the page with a text input for the POI name, description, type, and region. Then a select box is populated with a list of countries and an input of type submit is added and the form is closed. Finally, after closing the try catch block the custom footer function will be called.

addPoi.php

The logic of this page works the same way as the login page up until the end of the header section. A try catch block is then opened and the custom database connection function is called and the returned PDO object is stored in the conn variable. Then the name, type, country, region, and description that have been sent by the addPoiForm page using a POST request are then store in the variables with the same name. A username variable is then created to store the value of the gatekeeper session variable. An if statement is then used to validate all variables using custom regular expressions. If the inputs are not valid an error will be echoed out to the page. If the inputs are valid then a new poiDTO is created passing eight arguments the name, type, country, region, description, and username variables. The returned poiDTO is then stored in the variable poiDTO. Then a new poiDAO is created by passing in two arguments the conn variable and the hard coded table "pointsofinterest". The returned poiDAO is then stored in the variable poiDAO. Once created the add function will be called on poiDAO and the poiDTO will be passed in as the only argument, the returned poiDTO will then be stored in the returnedPoiDto variable. The returnedPoiDto will then be used to output the new POI to the page. Finally, after closing the try catch block the custom footer function will be called.

Task B: Allow a user to search for a POI by region.

Task b will use the index.php and functions.php from task a but, will also require a regionResult.php page. This page will be used to display a list of POI's in a particular region to the end user. To do this the index.php page will send the region to the regionResult.php page using a GET request. A GET request is used as we are not editing the database and GET requests require less code than a POST request.

regionResults.php

The logic of this page works the same way as the index page up until the end of the header section. A try catch block is then opened and the custom database connection function is called and the returned PDO object is stored in the conn variable. Then the region that was sent by the index page using a GET request is store in a variables with the same name. An if statement is then used to validate the region variable using a custom regular expression. If the region is not valid an error will be echoed out to the page. If it is then a new poiDAO is created passing the conn variable and the hard coded table name "pointsofinterest" as arguments it is then stored in the poiDAO variable. The find by region function is then called on poiDAO and the region is passed in as its only argument, the returned array will be stored in the variable pois. An if statement is then used to validate that pois in not null. If pois is not valid an error will be echoed out to the page. If pois is valid then a foreach loop will iterate thought it outputting the POI's to the page. Then for each POI a form is echoed out to the page with two hidden text inputs one being POI ID the other the POI region and a submit input before closing the form. This for will allow a user to recommend the POI. Then two query string links our output the first to the reviewResults page with the POI ID allowing users to see reviews for that POI, and the second to the addReviewForm page with the POI ID and name allowing users to add a review for that POI. Finally, after closing the try catch block the custom footer function will be called.

Task C: Allow a user to recommend a POI.

Along with the use of regionResults.php and functions.php from task b, task c will also require an addRecommendation.php page. This page will be used to update the number of recommendations for a given POI in the database. To do this it will be sent the POI ID from the region results page using a POST request.

addRecommendation.php

The logic of this page works the same way as the index page up until the end of the header section. A try catch block is then opened and the custom database connection function is called and the returned PDO object is stored in the conn variable. Then the POI ID and region sent by the regionResults page using a POST request are stored in variables with the same names. An if statement is then used to validate the POI ID and region variables using a custom regular expression. If either are not valid an error will be echoed out to the page. If they are valid then a new poiDAO is created passing the conn variable and the hard coded table name "pointsofinterest" as arguments it is then stored in the poiDAO variable. The add recommendation function is then called on poiDAO and the POI ID and region are passed in as arguments. Finally, after closing the try catch block the custom footer function will be called.

Task D: Allow a user to view all reviews for a given POI.

Overview

This task will require a new reviewResults.php page and will be used alongside the regionResults.php and functions.php pages from task b. The reviewResults.php page will be used to display a list of reviews for a given POI to the end user. To achieve this the regionResults.php page will send the POI ID to the reviewResults.php page using a GET request.

reviewResults.php

The logic of this page works the same way as the login page up until the end of the header section. The only difference is the included scripts for the reviewResults page are functions, reviewsDAO and poiDAO. A try catch block is then opened and the custom database connection function is called and the returned PDO object is stored in the conn variable. Then the POI ID sent by the regionResults page using a GET request is stored in a variable with the same name. An if statement is then used to validate the POI ID using a custom regular expression, if it is not valid an error will be echoed out to the page. If it is valid then a new reviewsDAO is created passing the conn variable and the hard coded table name "poi_reviews" as arguments it is then stored in the reviewsDAO variable. The findByPoiIdAndApproved function is then called on reviewsDAO and the POI ID is passed in as an argument, the returned array will be stored in the variable reviews. Then a new poiDAO is created passing the conn variable and the hard coded table name "poi_reviews" as arguments it is then stored in the poiDAO variable. The findByid function is then called on poiDAO and the POI ID is passed in as an argument, the returned POI will then be stored in the variable poi. An if statement is then used to validate that poi is not null, if poi is not valid an error will be echoed out to the page. Then an else if statement is used to validate that reviews is not null if reviews is not valid an error will be echoed out to the page. If both are valid then a foreach loop will iterate thought the indexed array outputting the reviews to the page. Finally, after closing the try catch block the custom footer function will be called.

Task E: Allow a user to review a POI.

Task e will use the regionResults.php and functions.php pages from task b but, will require the addition of an addReviewForm.php and addReview.php page. The addReview.php page display's a form for the user to enter the new review for a POI. This information is then sent to the addReview.php page using a POST request.

addReviewForm.php

The logic of this page works the same way as the login page up until the end of the header section. The only difference is the included scripts for the reviewResults page are functions and reviewsDAO. A try catch block is then opened and the custom database connection function is called and the returned PDO object is stored in the conn variable. Then the POI ID and review sent by the regionResults page using a POST request are stored in variables with the same names. An if statement is then used to validate the POI ID and review variables using a custom regular expression, if either are not valid an error will be echoed out to the page. A form is then echoed out to the page with a text input for the review, a hidden input for the POI ID, and a submit input and then the form is closed. Finally, after closing the try catch block the custom footer function will be called.

addReview.php

The logic of this page works the same way as the login page up until the end of the header section, the only difference is the only included script for the addReview page is functions. A try catch block is then opened and the custom database connection function is called and the returned PDO object is stored in the conn variable. Then the POI ID and POI name sent by the addReviewForm page using a GET request are then store in the variables with the same name. An if statement is then used to validate all variables using custom regular expressions. If the inputs are not valid an error will be echoed out to the page. If the inputs are valid then a new reviewsDTO is created passing null, POI ID variable, review variable, and 0 (this sets the review to unapproved) as the arguments, the returned reviewsDTO is then stored in the variable reviewsDTO. Then a new reviewsDAO is created by passing in two arguments the conn variable and the hard coded table "poi_reviews", the returned reviewsDAO is then stored in the variable reviewsDAO. Once created the addReview function will be called on reviewsDAO and the reviewsDTO will be passed in as the only argument, the returned reviewsDTO will then be stored in the returnedreviewDto variable. The returnedreviewDto will then be used to output the new review to the page. Finally, after closing the try catch block the custom footer function will be called.

Task F: Make it so all new reviews must be approved by an admin.

For task will require two new pages reviewResultsAdmin.php and approveReview.php. These will be used alongside index.php and functions.php from task a. The reviewResultsAdmin.php page will be used to display a list of reviews pending approval to an admin user. The approveReview.php page will be used to update the approved status of a given review in the database, to do this it will be sent the review ID from the reviewResultsAdmin.php page using a POST request.

reviewResultsAdmin.php

The logic of this page works the same way as the login page up until the end of the header section. The only difference is the included scripts for the reviewResultsAdmin page are functions, reviewsDAO and poiDAO. A try catch block is then opened and the custom database connection function is called and the returned PDO object is stored in the conn variable. Then a new reviewsDAO is created by passing in two arguments the conn variable and the hard coded table "poi_reviews", the returned reviewsDAO is then stored in the variable reviewsDAO. Once created the findByUnapproved function will be called on reviewsDAO, the returned array will then be stored in the reviews variable. An if statement is then used to check that the $reviews variable is not null, if it is then an error is echoed out to the page. If reviews is valid then a foreach loop will iterate thought the indexed array outputting the reviews to the page. Then for each review a form is echoed out to the page with one hidden text input containing the review ID and a submit input before closing the form. This form will allow a review to be approved. Finally, after closing the try catch block the custom footer function will be called.

approveReview.php

The logic of this page works the same way as the login page up until the end of the header section. The only difference is the included scripts for the approveReview page are functions and reviewsDAO. A try catch block is then opened and the custom database connection function is called and the returned PDO object is stored in the conn variable. Then the review ID sent by the reviewResultsAdmin page using a POST request is then store in a variables with the same name. An if statement is then used to validate the review id using a custom regular expressions. If the input is not valid an error will be echoed out to the page. Then a new reviewsDAO is created by passing in two arguments the conn variable and the hard coded table "poi_reviews", the returned reviewsDAO is then stored in the variable reviewsDAO. Once created the approveReview function will be called on reviewsDAO and the review ID will be passed in as the only argument. Finally, after closing the try catch block the custom footer function will be called.

Task G: Implement the site using Object Oriented PHP.

To achieve task g I will need to add number of PHP scripts: poiDTO.php, poiDAO.php, reviewsDTO.php, reviewsDAO.php, usersDTO.php, and usersDAO.php. These scripts will be linked to other pages using the include function and all DAO's will be linked to their corresponding DTO's using the same include function.

poiDTO.php, reviewsDTO.php, and usersDTO.php

These scripts are all extremely similar and the only differences are the number and names of private attributes, below I will talk about the poiDTO.php script. This script will have 8 private attributes: id, name, type, country, region, description, recommended, and username. It will have a public constructor function that takes 8 arguments and stores them in there corresponding attributes. There will be public getter functions for each attribute, that return the value contained in the corresponding attribute to the requesting page or script. There will also be public setter functions for each attribute, these will each take one argument from a page or script and update the attribute to the input value. Finally, this script will include a display function, this will be used for diagnostic purposes and will echo out all of the attributes of a specific instance of this object and return it to the requesting page.

poiDAO.php

This script will use the built in PHP include function to link it to the poiDTO.php script. It will also have 2 private attributes: table and conn. Table to store the database table name and conn to store the PDO connection object that are passed though as arguments to the construct function. There will be a public `__construct($conn, $table)` function that will allow a new poiDTO object to be created, it will take in two arguments a PDO connection object and a table name. These inputs will then be stored in the corresponding attributes. There will also be a public `findRegions()` function that runs a prepared SQL select statement on the database using the conn and table variables allowing it to find all the regions currently in the database. (Prepared SQL statements have been used in all DAO's to guard against SQL injection attacks.) A while loop will then iterate thought the returned rows and add them to an indexed array that will then be returned to the requesting page. It will also include a public `findById($poiId)` function that takes one argument a POI ID. It will then run a prepared SQL select statement on the database using the conn, table and input poiId variables allowing it to pull out a specific database row. That row will then be used to create a new poiDTO object that can be returned to the requesting page. There will be a public `addRecommendation($poiId, $region)` function that takes two arguments a POI ID and a region. It will then run a prepared SQL update statement on the database using the conn, table and input poiId variables to add one to the recommended value in the database. Then using the built in PHP header function the user will be returned to the regionResults.php page, the input region variable will be used to complete the query string sending the user back to the page they came from. The last function this script will contain will be a public `add(poiDTO &$poiObj)` function that takes one argument a poiDTO. It will then execute a prepared SQL insert statement on the database using the conn, table, and input poiObj variables, populating the table columns with the POI objects attributes. The POI object will then have its ID attribute set to the ID from the database before being retuned to the requesting page.

reviewsDAO.php

This script will use the built in PHP include function to link it to the reviewsDTO.php script. It will also have two private attributes and a public `__construct($conn, $table)` function that will work in the same way as the one in the poiDAO.php script. There will be a public `findByPoiIdAndApproved($poiId)` function that takes one argument a POI ID. It will then run a prepared SQL select statement on the database using the conn, table, and input poiId variables, allowing it to find all reviews that have been approved for that POI. A while loop will then iterate thought the returned rows and add them to an indexed array that will then be returned to the requesting page. There will also be a public `findByUnapproved()` function that will run a prepared SQL select statement on the database using the conn and table variables, allowing it to find all reviews pending approval by an admin. A while loop will then iterate thought the returned rows and add them to an indexed array that will then be returned to the requesting page. There will be a public `addReview(reviewsDTO &$reviewObj)` function that takes one argument a reviewDTO. It will then run a prepared SQL insert statement on the database using the conn, table, and input reviewObj variables, populating the table columns with the review objects attributes. The review object will then have its ID attribute set to the ID from the database before being retuned to the requesting page. The last function this script will contain will be a public `approveReview($reviewId)` function that takes one argument a review ID. It will then execute a prepared SQL update statement on the database using the conn, table, and input reviewId variables, to approve that specific review. Once completed the built in PHP header function will return the user to the reviewResultsAdmin.php page.

usersDAO.php

This script will use the built in PHP include function to link it to the usersDTO.php script. It will also have two private attributes and a public `__construct($conn, $table)` function that will work in the same way as the one in the poiDAO.php script. It will also contain a public `verifyLogin($un, $pw)` function that takes two arguments username and password. It will run a prepared SQL select statement on the database using the conn and table variables. It will then use the un and pw variables to retrieve the matching database record. It will then use the fetched row to create a new usersDTO object and this will be returned to the requesting page.