**School of Media Arts and Technology**

BSc (Hons) **Software Engineering**

**Academic Year 2019-2020**

**STEFAN SHKENDEROV**

**Points Of Interest Report**

Assignment 1

**Developing for the Internet**

**Tutor: Nick Whitelegg DATE 04.02.2020**

Grade D:  
Task a)

For task a) it is required that the user can add a new Point of Interest. For this part, two scripts are created in the folder “addPoi”. The first file is called addpoiform.php and contains the form that is to be displayed to the user when adding a new POI. It has text box inputs for the name, type, country, region, longitude, latitude and description of the POI. Everything except the coordinates is required to be filled in, with the coordinates being of type=“number”. When pressing the submit button, the data from the text fields is sent to the other script in this folder – addpoi.php via a POST request. In that file, the data is fetched and set into php variables. In it are checks for if the longitude and latitude are set (Because they are not required to be entered) via the is\_numeric function. If they are not entered (which means they are are not numeric), the values are set to null. A new poi object is created and populated with the parsed data, which is now stored in the php variables. The username column, which gives information of which user added the POI is entered is set in the constructor to the session gatekeeper value. and it is parsed via the addPoi function to the PoiDAO *(see grade A for more information on the objects).* After that, a confirmation message is echoed as well as a link to return to the main page.

## Task b):

For task B there is a text field and a button in the index page, which when pressed displays the POIs in the region entered by calling the searchByRegion function in the PoiDAO (see grade A). This returns an array of POIs with the region entered in the text box. The results display the fields of the poi object, i.e. the region, name, description etc. via a foreach loop, looping trough the returned array, surrounded by an echoed div, which is used for formatting the data so it can be displayed in a readable way. Under every result there is a link that lets the user recommend the POI (see task C) and a link to see all of the reviews (see task D). An if – else block is used to check if the returned array is empty. If it is, a message is displayed to the user. Task B is implemented via Slim and Ajax, for more information see grade A.

## Task c):

Task C is allowing the user to recommend a POI. It is implemented via a link under every search result, sending a get request to the recommend.php file with the id attribute of the selected POI. The recommend.php file fetches the id, finds the POI object with that id via the PoiDAO findpoibyid($id) function and after that the recommend() function is called on the poi found (that function is contained in the POI object). This function adds 1 to the recommended attribute of the object. Then the updatePoi function of the PoiDAO is called, which in this case means adding 1 to the recommended field for the selected poi in the database.

# Grade C:

## Task d):

For the implementation of the Review functionality there is an echoed link below every search result which takes the user to the showreviews.php script. It uses a get request which contains the id of the selected poi. That script fetches the id and calls the findReviewsByPoiId method from the PoiDAO. This returns an array of Review objects (see grade A). Via a foreach loop for each review in the array the printreview method is called, which echoes the review text. If there are no reviews for that POI, i.e. the findReviewsByPoiId method returns null, a message is echoed out with a link to return to the index page.

## Task e):

For this task, it is required that the user can submit a review. After searching for POIs by region and pressing the “see reviews” link below on of the results, you are taken to the showreviews.php script. The id of the POI is parsed via a get request in the link of the search results. In the showreviews.php script there is a form with a text field, which contains the review text as input and a hidden input which is the POI ID. The form is sent via a POST request to the script addreview.php. This script fetches the entered data form the request, creates a new Review object with it and calls the addReview function of the poiDao (See grade A). Then it redirects back to the list of reviews page with a get request containing the id of the POI. This task is altered to work with Slim (see grade A).

For the task of guarding against injection and cross-site-scripting all of the SQL queries are using prepare statements, which guards against SQL injection. All of the parsed in values via requests are using the htmlentities function when fetched from a script, which encodes HTML-specific characters, preventing from executing harmful scripts, guarding against Cross-site scripting attacks. All of the links are tested and working.

The website uses a mobile-first layout. The styling includes creating a header and footer. The photos used, including for the body of the pages are downloaded from unisplash.com, a site for stock photos which are free to be used. The colour scheme emphasizes on the background picture and the main colours in it. For text, white is used and for accents like :hover and :focus events, a blue colour with the code of #0067bb is used. The footer is black to differentiate from the rest of the page and the navigation bar is transparent, with a white border. For the purpose of the responsive layout, the links in the navigation bar are hidden and replaced with a burger when the width reaches certain size. When clicking the burger, the links are displayed on the side. This is done via a JS event listener on the burger and it’s intended to be used when the site is opened from a smaller device.

# Login System

When entering a website, the user is prompted to log in or register. Both the login and register functions are html files with forms, sending a post request to their corresponding script files – register.php and login.php. The login script checks the entered data for matching users via the login function of the userDAO. This function returns a username in case the password is correct, if not – it returns null. In the login script there is an if statement that checks if the returned username is null and if it is, it displays an error message with a link back to the login.html form. If the returned username is not null, the login is successful, the session gatekeeper is set to the returned username and the user is redirected to the index page.

When the user wants to register, he fills in a form in the register.html with the username, password and repeated password. This form is sent via a POST request to the register.php file, where it first fetches the text fields, then checks if the two passwords match. If they don’t, an error message is displayed to the user with a link redirecting back to the registration form. If the passwords match, a new User object is created and added to the database via the addUser function of the userDAO. Then a message is displayed with a link to the index page.

*The following functionality applies only when users are logged in, i.e. does not apply to the login and register pages.*

In the navigation bar of every page there is a message greeting the user which gets the username from the session gatekeeper variable. Next to it is a log out button, which is actually the submit button of a form that uses a post request to the index.php file, parsing a hidden input, which is an action string named “logout” with the value of 1. This form is used in the footers of the pages as well. In the index.php file the logout variable is created with the initial value of 0 and on the next line the post request is fetched, so it changes the value if the log out button has been pressed. The log out button always submits the form to the index.php page. In the index.php page after fetching the logout action string, an if statement is created checking if it has the value of 1. If that is true, the user wants to log out, so the session is destroyed and the user is redirected to the login.html page.

Another check that is made on every page (except the login and register pages) is the if statement that checks if the session gatekeeper is set. If it is not, meaning the user is not logged in, the page redirects to the login.html page.

# Grade B:

## Task f):

The project has implemented the admin functionality, which allows an admin to delete or approve reviews. Before the admin has deleted or approved a review, the review is not shown when the user checks the other reviews of a POI.

A dedicated admin panel page is created, which contains all of the reviews pending approval. Those reviews are displayed via the findNotApproved() function of the poiDAO, which returns an array of reviews, which have the approved =0 attribute. If there are none, i.e. the returned array is empty, a message is echoed that there are no pending reviews. If there are pending reviews, they are displayed via a foreach loop. In that loop every review text is displayed via the printreview function of the review object. Other than that, a heading appears on top of every review, displaying the POI that this review is written about. This is done in the same foreach loop by first getting the POI ID foreign key value from the review via the getter in the review object, then calling the findPoiById method of the poidao with the argument of that value, so the POI can be found and stored in a variable, then the getName function is called on that POI so that the admin knows which review is for which POI. Under every pending reviews, there are 2 links, parsing the id of the review (it is stored in a variable which is set by the getId function of the review object) via a GET request to the same file as well as an action string which is “del” for deleting the review and “app” for approving the review. This is done without a form, but by including the parsed data in the URL. The link is to the same adminpanel.php file, which in the beginning fetches the data from the get request. Then the actionstring value is checked via if statements, and if it is set to “del”, then the deleteReviewById function of the poi DAO is called, and if it is set to “app”, then the approveReview function of the POI DAO is called. The latter sets the approved value in the database to 1.

*\*The findReviewsByPoiId function of the POI DAO (used when displaying reviews to the user after searching for POIs by region) is edited so that it only returns reviews, approved by an admin.*

The admin panel can only be accessed by a user with the admin value in the database set to 1. When that is the case, a link to the panel is shown in the navigation bar. This is done by including an if statement in the navbar code, adding a new element when the user is an admin. The condition of the if statement is that the isAdminByName method of the userDAO returns one with the argument of the session gatekeeper. This check is done for all of the pages including in the adminpanel.php script itself, in case the user types in manually the URL of the admin panel. If not logged in as an admin, the user is redirected to the login.html form. An exception for this check is for the login/register scripts and html files.

The user friendliness of the site is achieved by not requiring the user to use IDs (for example when he wants to see the reviews on a POI), the flow of the website is easy to understand with the functionality clearly understandable and ensuring that the user will always be able to switch between different functions of the site without having to search for links or go back to previous pages. All of the text fields have implemented checks for entering data that is not present in the database and entering empty fields and the error dialogues that are displayed have clear messages to what happened, without any error codes.

# Grade A

## Task g):

Every script of this project makes use of object-orientation, with all of the queries being handled by Data Access Objects and the Points Of Interest, Reviews and Users being represented as objects as well.

The POI Object has the attributes of the POI table of the database – id, name, type, country, region, longitude, latitude, description, recommended and username (the user who added the POI).

The methods of the POI object are:

**Constructor:** Upon creating the object, the constructor sets the values to all of the attributes, except the id, which is unknown until the entry of that object in the database. When called, the constructor sets the Id to a value of null.

**Setters:** The only setter required is for the id, which is called by the DAO after entering the object in the database.

**Getters:** Every attribute has a getter method.

**Recommend():** A function which adds 1 to the value of the recommended field. Used when the user wants to recommend the POI.

**PrintDetails():** A toString type function which echoes all of the object’s attributes. Used when a uses searches for POIs.

The Review object has the attributes: id, review, approved (=1 if an admin approved the review, if not =0) and poi\_id(the foreign key linking to the POI for which that review is written).

The methods of the Review object are similar to the POI object in the sense that the constructor sets everything except the id, which has a setter to be used once the Review is entered in the database. All attributes have getters, with the approved attribute having a setter as well, which is used when an admin approves a review. Like the POI object, it has a printReview function, which echoes out the review attribute. This is used when a user sees the reviews for a selected POI.

The third object is the User object, which similar to the previous two, has the attributes of the fields for the poi\_users table, the constructor does not set the id upon creation of the object, but does it in a setter. All the attributes have getters. No other functions were found to be needed for this object.

The PoiDAO object is a data access object which is used for handling all SQL operations between the page scripts and the POI and Review objects. The two objects share the same DAO because they are have a one-to-many relationship in the database, this means that there are operations using both the Review and POI objects, for example when displaying the reviews for a POI, the POI name is good to be displayed so that the site is more user friendly.

Attributes:

**poi\_table:** the name of the database table for points of interest.

**review\_table:** the name of the database table for reviews.

**conn:** the PDO object for connecting to the database.

Methods:

**Constructor:** The three attributes are set to the argument values parsed in.

**FindPoiById($id)**: This method sends an SQL query to the POI table (using the poi\_table attribute), selecting every entry with the id, parsed in the method. With the returned rows a new POI object is created and the setter for the ID is called after the constructor. For this purpose no loops are necessary because in the database no two entries can have matching id’s, so the query only returns one database entry.

**SearchByRegion($region):** This method sends an SQL query to the database similar to the findpoibyid function, but instead of the ID the Where condition is the region column of the database. After the rows are returned, an array of POI objects is created, and each returned row is first set into a POI object (with the id set after the initial creation, see POI object), then it is put into the array, which all is done via a while loop. After that the method returns an array of POI Objects.

**addPOI(POI &$poi):** This method adds a POI into the database via an SQL query, after which it sets the parsed in poi’s ID to the id of the new entry in the database via the setter method of the poi object and the lastinsertId function. Then, this object with the set id is returned, so that the new id can be known in the program if needed. The ampersand of the parsed in poi is needed so that the id of the object is changed in the script that calls this function as well.

**updatePOI(POI &$poi):** This method is contains an SQL Update query to the poi table changing the columns of the POI in the database with the same Id as the parsed in POI’s attributes.

**findReviewsByPoiID($id):** Method, similar to findpoibyid in terms of syntax, with the difference that it returns an array of review objects. It uses a select statement, searching via the foreign key. It is used when a user wants to see all reviews for a POI. For the user to be able to see a review,it has to be approved by admin and for this reason another condition is added (WHERE approved=1) to the select statement.

**findReviewsById($id), updateReview(Review &$review), addReview:**

Methods, similar to the findPoiById, updatePoi and addPOI with the exception that they use the select statements on the poi\_reviews table and the results are populated in Review objects.

**findNotApproved():** method that finds all reviews with the approved column=0. The results are populated in an array of Review objects similar to the findpoibyid method.

**approveReviewById($id):** Method that sends an SQL query to the review table, setting the approved column to one where the id parsed in is equal to a row in the database.

**deleteReviewById($id)**: Uses the delete query on the row with the Id that is parsed in the method.

The **User** class is used to store data for the poi\_users table. It has attributes of id, username, password and isAdmin, which are the table’s columns. It only has the getter methods and a setter method for the id, like the POI and Review classes.

The **UserDAO** class is similar to the PoiDAO class in it’s attributes, constructor, and findById, SearchByUsername(similar to searchByRegion), addUser and UpdateUser methods.

The **login($username,$password)** sends a SELECT query checking if a user with those credentials exists and if it does, it returns the username, so that it can be directly set to the session gatekeeper value(see Login System).

The **isAdminByName($user)** method sends a SELECT query to the users table with a WHERE condition on the username and returns the isAdmin column of the matching row. This is needed to check if the adminPanel link is accessible to the user.

All SQL queries in the DAO objects are surrounded by a try-catch block, for debugging any possible PDO exceptions thrown. The catch echoes the exception thrown. This functionality is enabled in the page scripts after initializing the connection variable (the PDO Object) via the setAttribute function with arguments of PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION.

## Task h:

For the search and review functionality a new php script is created, called searchandreview.php, which uses slim. It contains two endpoints.

The /search endpoint is intended for use when the user searches for points for interest. It uses a get request, parsing the region value from the main screen. In that endpoint calls the searchByRegion method of the PoiDAO with the argument of the parsed region. The returned array of POI objects is then parsed to the searchresults.phtml where the results are formatted in a readable way. That page is rendered to the index page where the results are shown (see Task B for how the formatting is implemented).

The /review endpoint uses a POST request to submit a review written by the user in the database. It gets the parsed data, creates a new Review object with it and calls the PoiDAO method addReview. After that, a confirmation message which is in the submitreview.phtml is rendered and returned from the slim endpoint.

Both endpoints use the use() statement to be able to access the global variables of $poiDAO and $view. The first is used for the functions that interact with the database, the second is used for the render() statement.

For the AJAX front end, two JavaScript files are created, one for the index page, where the user searches for POIs by region, and one for the shoreviews page, where the user can submit a review. The first one, named ajax.js, is onloaded on the body of the index page, adding an Event Listener to the search button, which when clicked will invoke the ajaxrequest() function. This function first creates an HttpRequest variable, then reads from the text box where the user types in the region. An if statement is included, for when the button was pressed when the text box is empty. In that case, an alert pops up, prompting the user to type in a region.

If the text box is not empty, an event listener to the httpRequest variable is made, specifying the callback function. Then, via the open() statement, the connection is opened, specifying the Get request and the url, which is /search/+a, where a is the variable for the entered region. Then the request is sent. The response function which is invoked fills the rendered data in the response div of the home page via the responseText command. The browser then scrolls to that div for better user experience.

The other file, handling the review submission is similar with the exception of using a POST request. For that reason, the parsed values of the review text and the id of the POI that is reviewed is put into a FormData object and then is sent to the /review endpoint of the slim. Both values are sent from a form in the showreviews.php page, with the id being a hidden input field. This value is the same that was parsed from the index page to the showreviews page when the user clicks on the “View reviews of this POI” link (see task E). The response is rendered into the “responsedivreview” div, which is hidden until a review is submitted. Then, using the innerHTML and style.display, the div is populated with the data and then shown to the user.