**Web Application Development (QHO540)**

**TCA**

**Scenario**

The tourist department of a small mountainous island would like to develop a site which allows tourists to look up mountain climbs on the island. The site should also allow users to “like” a given climb, and review given mountain climbs, by writing comments on the beauty of the scenery, the difficulty of the path, and so on.

You have 2 hours to complete this TCA (2.5 if eligible for extra time).

**What reference material can I use?**

You **can** use the ***module’s lecture notes***, your ***assignment work*** (either original or resit) and ***your class work***.

**You cannot use *third-party websites*, the *mock TCA answers*, or the *answers to any previous TCA (mock or real).* Use of any of these will be considered as cheating, and if there is sufficient evidence that you have used any of these prohibited sources, your work will be referred to an academic misconduct panel.**

**Database tables**

The following database tables have been set up in the provided SQLite tourist.db file.

***mt\_users - records registered users***

|  |  |  |
| --- | --- | --- |
| **Column** | **Type** | **Role** |
| ID | INTEGER | a unique identifier for each record in the table (i.e. the primary key) |
| username | TEXT | Username |
| password | TEXT | Password |
| admin | INTEGER | Admin status (0 for non-admins, 1 for admins) |

***mt\_mountains - details of all mountains***

|  |  |  |
| --- | --- | --- |
| **Column** | **Type** | **Role** |
| ID | INTEGER | a unique identifier for each record in the table (i.e. the primary key) |
| name | TEXT | The name of the mountain |
| difficulty | TEXT | The difficulty (easy, medium, hard) |
| height | REAL | The height of the mountain (metres) |
| distance | REAL | The distance of the route up the mountain (kilometres) |
| likes | INTEGER | Number of likes for this climb |

***mt\_reviews - user reviews***

|  |  |  |
| --- | --- | --- |
| **Column** | **Data type** | **Role** |
| ID | INTEGER, PRIMARY KEY | a unique ID for each review |
| review | TEXT | the review |
| username | TEXT | the username of the user who made this review |
| mountain\_ID | INTEGER | the ID of the mountain that the review relates to |
| approved | INTEGER | is the review approved yet? (0=no 1=yes) |

**Default data**

The provided SQL file creates a database with sthe following data.

*mt\_users*

|  |  |  |
| --- | --- | --- |
| ***username*** | ***password*** | ***admin*** |
| admin | admin | 1 |
| tim | tim123 | 0 |
| kate | kate123 | 0 |

*mt\_mountains*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ***ID*** | ***name*** | ***difficulty*** | ***height*** | ***distance*** | ***likes*** |
| 1 | Mount Weatherstone | medium | 2500 | 8 | 0 |
| 2 | Ice Glaze Peak | hard | 3500 | 12 | 0 |
| 3 | Mount Tharg | medium | 2600 | 10 | 0 |
| 4 | Mount Wight | medium | 1800 | 7 | 0 |
| 5 | Glacier Ridge | hard | 3400 | 13 | 0 |
| 6 | Green Mountain | easy | 1300 | 5 | 0 |

**Pre-prepared scripts**

A ZIP file of pre-prepared, partly-complete scripts is available on SOL. Your task is to complete as many of the scripts as possible according to the instructions below.

|  |  |
| --- | --- |
| **Script / HTML page** | **Role** |
| index.html | An HTML page containing various forms: a form to allow the users to search for climbs by difficulty level, an “add climb” form, and a login form. |
| client.js | A front-end JavaScript file, containing code which will communicate with the server via AJAX and interpret the response returned |
| server.js | A Node and Express based server which interacts with the database |

**Tasks**

There are 18 tasks, detailed below.

Tasks 1-14 are also indicated as comments in the relevant files. Please complete in order, as they are arranged in approximate order of difficulty and some later tasks depend on earlier ones**.**

**If you skip tasks, you will lose credit appropriate to the complexity of the task.**

**Please do NOT add unnecessary code which is irrelevant to the question. This could result in a reduction of your grade as it can be a sign of a lack of understanding.**

**There is no need to re-write the code to use routes, controllers and DAOs. You will not be given extra marks for doing this. Please focus on completing the code to meet the requirements in the simplest possible robust, bug-free way.**

1. In client.js, complete the indicated getElementById() call so that the code handles click events from the Search button on the “search for climbs” form on index.html.

2. In client.js, complete the statement to read in the difficulty level from the form.

3. In client.js, complete the fetch API call so that it sends the user’s chosen difficulty level to the “climbSearch” route in server.js. You will need to look at the server.js code to complete this successfully.

4. In server.js, complete the “climbSearch” route so that it finds all climbs of the user’s chosen difficulty level.

5. In server.js, complete the “climbSearch” route so that it sends back the matching climbs as JSON.

***If you get this far, you will achieve 15% (F3).***

6. Complete client.js so that it parses the JSON returned from the “climbSearch” route on the server, and outputs it in HTML to the ‘climbResults’ <div> in index.html. It must be formatted as shown in the following example:

Name – Mount Tharg

Difficulty - Medium

Height – 2600m

Distance - 10km

***If you get this far, you will achieve 35% (F1).***

7. In server.js complete the “like” route so that it uses an SQL UPDATE statement to increase the number of likes by one.

8. Update your answer to question 6 so that there is a “Like” button for each matching result. When the button is clicked, an appropriate HTTP request must be sent to the “like” route, to allow the selected mountain climb to be liked.

***If you get this far, you will achieve 45% (D2).***

9. Complete the “add climb” functionality on the client.

In client.js, you need to:

* complete the indicated getElementById() call so that the code handles click events from the Submit button on the “add mountain climb” form in index.html.
* complete the JavaScript code to read in the details (mountain name, difficulty level, height, distance) entered by the user in the “add mountain climb” form on index.html;
* complete the JavaScript code to send all four items of data to the “add mountain climb” route in server.js as an AJAX POST request. Data should be sent within the request body.

10. In server.js, complete the “add climb” route so that it reads the details sent to it from the AJAX code which you completed in the previous question and creates a new record in the *mt\_mountains* table containing the details entered. **Ensure that the server is set up to read POST data from the request body - you will need to make an alteration to the server to enable this!**

***If you get this far, you will achieve 55% (C2).***

11. Change the “add climb” route so that it sends back an appropriate HTTP status code if the mountain height or route distance is 0 or less.

***Hint:*** Use the || operator for “or”. Here is an example or an “if” statement with an “or” operator:

if(username == "" || password == "") {

alert("Please enter a username and password!");

}

12. Modify your answer to Question 9 in client.js to handle non-200 HTTP status codes returned from your question 11 answer. Ensure that the errors are displayed to the user in a user-friendly way in the ‘addClimbStatus’ <div>.

***If you get this far, you will achieve 58% (C1).***

13. Complete the login functionality on the client. In client.js, you need to:

* complete the indicated getElementById() call so that the code handles click events from the Submit button on the login form in index.html.
* complete the JavaScript code to read in the details entered by the user in the login form on index.html;
* complete the JavaScript code to send the username and password to the login route in server.js as an AJAX POST request. Data should be sent within the request body.

***No additional credit for this, as it is very similar to question 9.***

14. Complete the login route on the server, so that it reads the username and password from the client, checks them against the database, and sets the **session** appropriately (you need to setup session variables on the server to do this). If the login details are incorrect, this must be communicated to the client appropriately, and the client must then display a user-friendly error message in the ‘loginStatus’ <div>.

15. Change the “like” route so that it is only accessible to logged-in users. Also, change the “like” functionality on the client so that an error message is displayed if a user tries to like a climb when they are not logged in.

***If you get this far, you will achieve 65% (B2).***

16. Enhance your answer to question 14, so that the login system takes into account whether a user is an administrator. (Administrators are users for which the ‘admin’ column in the *mt\_users* table is equal to 1).

***If you get this far, you will achieve 74% (A4).***

17. Change the server so that the “add climb” route is only available to administrators.

***If you get this far, you will achieve 83% (A3).***

18. Enhance your system to allow users to add reviews of a climb. There should be a “Review” button alongside each climb in the search results, and a review box. Also, you should add a new route to your server to add the review to the “*mt\_reviews*” table. This route should only be accessible to logged-in users, and if a user tries to review a route when they are not logged in, an error message should be displayed on the client.

***As completion of Question 18 results in a high A, some thought is required!***

***Completion of questions 1-18 with no errors will receive 100% (A1). If you complete 1-17, and partly implement 18, you may receive 92% (A2) depending on the amount of work done.***

***Best Wishes.***