Assignment 5

Sessions and User Profiles

Submit a single zip file called **assignment5.zip**. **Your submission MUST contain a package.json file that allows your assignment to be built using `npm install`. You should not include the node_modules folder or database files in your submission. You may assume that the TA will have the MongoDB daemon running and that the TA will have run `database-initializer.js` before running your server.**
This assignment has 100 marks. Read the marking scheme posted on cuLearn for details.

Assignment Background

In this assignment, you will add sessions and user profiles to an existing trivia quiz server. You should use the express-sessions module for session data and the connect-mongodb-session module as a session store.

To get started, download the `A5-BaseCode.zip` file from cuLearn. This zip file includes a working Express-based trivia quiz server. You can install the dependencies by running `npm install` in the directory with the `package.json` file. The zip file also includes a `database-initializer.js` file that, when executed, will reset the `quiztracker` database in the running Mongo instance and re-build the `questions` and `users` collection to their default state. The database uses the two Mongoose schemas that are also included – `QuestionModel.js` and `UserModel.js`. The `users` collection will have 10 users. You can find the user's names by querying the database or looking in the `database-initializer.js` file. Each user's password is also the same as their username. Read through the provided code to familiarize yourself with how the server works. A basic outline of the server's current functionality is described below.

GET requests for the `/quiz` route return an HTML page containing 10 randomly selected questions. The answers for the questions are not shuffled, so the first option shown for each question will always be the correct answer. To speed your testing, the client-side Javascript loaded on this quiz page automatically selects random answers when the page loads.

When the user selects "Submit" at the bottom of the quiz page, the answers are sent to the server using a POST XMLHttpRequest to the `/quiz` route. The server calculates the number of correct answers and sends a JSON response back to the client indicating:

1. `url`: the URL the browser should be redirected to
2. `correct`: the number of correct answers

The client-side Javascript then displays an alert with the number of correct answers and redirects the browser to the URL supplied by the server.

The remainder of the assignment will involve adding additional functionality into this code base. You should not need to modify the client-side Javascript, but are free to do so if you want to.

Assignment Requirements

Each page on the quiz tracker site should have a header that contains the following:

1. A link to the home page (`/`)
2. A link to a page for starting a new quiz (`/quiz`)
3. A link to a page which lists all users (`/users`)
4. If the user is currently logged in, a link to their profile (`/users/:userID`)
5. If the user is not currently logged in, a log in form where they can enter a username/password combination before clicking a Login button

When a user tries to log in, the server should validate their username and password by searching for matching entries in the database. You can assume that usernames will be unique within the database. If the credentials are correct, that user should be logged in to the system and should continue to be classified as logged in until they choose to log out or until the cookie associated with their session expires (you can decide on the expiry method). Additionally, once logged in, the user should be redirected to their profile page. On the other hand, if the credentials are incorrect, the user should be redirected to the homepage.

A GET request for the `/users` route should return an HTML page containing a list of all users in the server's database that have their privacy value set to "Off" or "false". The entry for each user in this list should contain a link to that user's profile with the link text showing their username. Users who have set their privacy value to "On" or "true" should not appear in the list.

A GET request to the parameterized route `/users/:userID` should behave as described below:

1. If the requesting user is logged in and requesting their own profile, then the page will:
    a. Provide a method for the user to toggle their privacy setting (e.g., two radio buttons or a drop-down list).
    b. Provide a method for the user to 'save' the changes to their privacy setting, which should send the selected privacy value to the server and update the database. The user should remain on their profile after this is completed.
    c. List the user's total quizzes completed and average quiz score
    d. Provide a "Log Out" button that, when clicked, logs the user out of the system and redirects them to the home page.
2. If the requesting user is not logged-in or is requesting some other user's profile, then:
    a. If the profile being requested is set to be private, you should respond with an HTTP status code of 403, indicating that the profile cannot be accessed.

b. If the profile being requested is not set to be private, then the page should show the total quizzes and average quiz score of the requested user.

When a POST request is made by a logged-in user to the `/quiz` route to complete a quiz, that user's profile should be updated and saved. The user should then be redirected to their own profile. If a user who is not logged-in to the system makes a POST request to `/quiz`, no changes need to be made to any users and the user should be redirected to the home page.

Code Quality and Documentation

Your code should be well-written and easy to understand. This includes providing clear documentation explaining the purpose and function of pieces of your code. You should use good variable/function names that make your code easier to read. You should do your best to avoid unnecessary computation/communication and ensure that your code runs smoothly throughout operation. **You must also include a README.txt file that explains any design decisions that you made and precise instructions for how to run/use your system.**

Recap

Your zip file should contain all the resources required for your assignment to run. Your submission MUST contain a package.json file that allows your assignment to be built using `npm install`. You should not include the node_modules folder or MongoDB database files in your submission. The TA must be able to run your server and use the system by following the instructions in your README.txt file. You may assume that the TA will have the MongoDB daemon running and will have run `database-initializer.js` before running your server.
Submit your **assignment5.zip** file to cuLearn.
Make sure you download the zip after submitting, verify the file contents, and verify that your instructions are sufficient for installing/running your system.
