Assignment 4

MongoDB and Sessions

Submit a single zip file called **assignment4.zip**. **You should not submit your node_modules folder or database folder. Instead, you should submit the resources required to install your dependencies using NPM.** The TA will run the provided database-initializer.js file before grading your assignment. This assignment has 100 marks. You should read the marking scheme posted on cuLearn for details.

Assignment Background

In this assignment, you will use a database to store user profile and order information. Additionally, you will add session support to the application so that users may log in, place orders, and view a history of their orders. Users will also have the option of setting their own profile to private/public, which will limit who can view their order history. **You MUST use MongoDB to store the user profile, order data, and session data for this assignment. You may choose to use the MongoDB module or Mongoose for database manipulation within Node.js.**

To start this assignment, download the `A4-Data.zip` file from cuLearn. This zip file contains several resources that will be used for the assignment. The first of these resources is the `database-initializer.js` file, which you can use to create the initial database for the assignment (or re-create it if you break the database during testing). Initialize your project using `npm init`. Ensure the MongoDB Node.js driver is installed in your assignment directory by running `npm install mongodb`. Run the `database-initializer.js` file using Node.js while the MongoDB daemon is also running on your computer, and it will initialize a database called '`a4`' with a collection called '`users`' containing ten initial user profiles for your server. Each user document contains a username, a password (which is the same as the username, for convenience), and a privacy field that will initially be set to false. You may modify the provided `database-initializer.js` file to perform any additional initialization your server requires. If you do modify the file, ensure that you include a copy with your assignment submission.

The `A4-Data.zip` file also contains a public folder with order form resources. This includes a working `orderform.html` and `orderform.js` file that will allow a client to select a restaurant and submit an order to the server using an XMLHttpRequest. Note

that you may need to modify the order form resources, or create a similar version using a template engine, to meet the navigation header requirements of the assignment. You may use your own version of similar resources, if desired. For this assignment, it is acceptable to store the data for the three restaurants in the client-side Javascript. However, you can also choose to serve the restaurant data from the server, as was done in the previous assignment.

Navigation Header (12 marks)

Each page on your web application should have a navigation header. The information contained in this header will vary depending on the current session state of the client.

If the client is logged in as a user of the application, the header must include links to the home page (/), the users page (/users), the order form, and the user's profile. Additionally, the header should provide a way for the user to log out of the application (e.g., a link or button).

If the client is not logged in, the header must include links to the home page (/), the users page (/users), and the registration page. Additionally, the header should include a way for the user to log in to the application (e.g., a login form in the header, or a link to a login page). The link to the order page should NOT be shown unless the client is logged in.

User Registration (16 marks)

Your server must provide a page to allow users to register through a registration form. To register, the user must provide a username/password and click the register button. All new users should start with a public profile (e.g., privacy = false) and no order history. Duplicate usernames should not be allowed. If the registration is successful, the user should be redirected to their own profile page and be considered 'logged in' to the application. If the registration is unsuccessful (e.g., due to duplicate username), an error message should be displayed, and the user should remain on the registration form.

User Directory (12 marks)

Your server must provide a route to handle GET requests for the URL `/users`, which will allow clients to search for users. This route must support the query parameter `name`, which will be a string value. The server must respond with an HTML page containing links to all non-private user profiles that match the query. The text of each link should be the username of that user. A profile will match the query if the username of that user

contains the given `name` query parameter value. This search should be done in a case-insensitive manner (e.g., et, ETE, and PetE, all would match the username pete). Note that private user profiles should NOT show up in the search results.

User Profile Page (20 marks)

Your server must provide a route to handle GET requests for the parameterized URL /users/:*userID*, which will allow the client to view a specific user's profile page (note: it is recommended that you use the MongoDB generated IDs for each user, but you can use your own IDs if you want). There are several rules that will determine how a request for this route should be handled:

1. If the requested profile is set to private and the requesting client is NOT logged in as the owner of the profile, then a 404 or 403 status and error message should be sent in response. That is to say, the only person that can view a private profile is the owner of that profile. The exact status code you use is a design decision. Sending 403 would give more information to the requesting client, as they would be able to tell the profile does exist but is set to private. Sending a 404 would provide less information.
2. If the requested user's profile is NOT set to private, or the requesting client is logged in as the owner of the profile, then the resulting HTML page should show the user's username and a list of their order history. Each entry in this order history list should be a link to that specific order's page (i.e., /orders/*thatOrdersID*). You can use the order ID for the text of the link.
3. Additionally, if the user is logged in and viewing their own profile, the page must provide a way for them to update their privacy value and save the changes to the server. Note that ONLY logged in users should be able to change their own profile. If a user is not logged in or is trying to change the profile of another user, the server should respond with 403 or 404 status and error message.

Order Summary Page (15 marks)

Your server must provide a route to handle GET requests for the parameterized URL /orders/*:orderID*, which will allow the client to view a specific order. The requested order should only be shown if:

1) The user who placed the order is NOT set to private.

OR

2) The requesting client is logged in as the user who placed the order.

If neither of the above are true, the server must respond with a 403 or 404 status code and error message.

The contents of the order summary page must include:

1. The username of the user who placed the order
2. The restaurant's name
3. A list of item names and their quantities.
4. The subtotal, tax, delivery fee, and total of the order

Order Form and Submission (10 marks)

Your server must provide a way to access the order form and submit orders. The URL you use is up to you but must match the one you provide in the navigation header. The order form should only be viewable by users who are logged in. You can serve the order form resources provides in the `A4-Data.zip` file or use your own version. The minimum requirements the order form must support are:

1. Contains the proper navigation header. The easiest way to achieve this is likely to duplicate the orderform.html contents using a template engine file.
2. Allow the client to select one of the three restaurants we have been working with throughout the course. This data can be stored in the client-side Javascript or requested from the server (your choice).
3. Allow the client to add items to an order and view the order summary.
4. Allow the client to send the order to the server.

The provided order form resources will send the order using a POST request to the URL */orders*. An example of the format of the data sent to the server is included below (you can change the structure as you see fit):

```
{
        restaurantID: 0,
        restaurantName: "Aragorn's Orc BBQ",
        subtotal: 31.5,
        total: 39.65,
        fee: 5,
        tax: 3.15,
        order: {
            '2': { quantity: 2, name: "Sauron's Lava Soup" },
            '3': { quantity: 4, name: "Eowyn's (In)Famous Stew" },
            '4': { quantity: 1, name: 'The 9 rings of men.' }
// '2', '3', and '4' are the item IDs in the menu, though you may not need them
        }
}
```

Your server must accept the order submission request and process it in whatever way is required to meet the rest of the requirements of the assignment (e.g., by adding the order information to the database).

Code Quality and Documentation (15 marks)

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 and ensure that your code runs smoothly throughout operation. **You should also include a README.txt file that explains any design decisions that you made, precise instructions detailing how to run your server, as well as any additional instructions that may be helpful to the TA.**

Recap

Your zip file should contain all the resources required for your assignment to be installed and run by the TA. **You should not submit your node_modules folder or your MongoDB database folder.** Submit your **assignment4.zip** file to cuLearn. Make sure you download the zip after submitting, verify the file contents, and ensure that your installation instructions are sufficient.
