Assignment 3

Trivia Question Web Service

Submit a single zip file called **assignment3.zip**.
This assignment has 100 marks.
You should read the marking scheme posted on cuLearn for details.

Assignment Background

In this assignment, you will implement a server that provides an API for clients to access trivia questions. Your service will operate in a similar way to Open Trivia DB, by allowing clients to request subsets of the available questions based on difficulty and category. Note that your server must support the specified API precisely. It should also be programmed in a way that the server will not crash if incorrect input is provided (e.g., category names instead of integer IDs).

To start, download the A3-resources.zip file from cuLearn. This zip contains a 'questions' directory, which contains a .json file for each question that will be included in your database. It also includes a 'public' directory, which contains HTML/Javascript that your server can host to allow you to easily query your database while testing. Alternatively, you can use the Postman tool to issue queries to your server.

Before starting your design for the assignment, you are encouraged to read through the entire specification. It is possible that later parts will inform your design decisions and, by preparing in advance, you can avoid having to significantly refactor your code as you progress through the assignment.

The Basic Trivia API

GET /questions: When a client wishes to retrieve questions from your server, they will make a GET request for the /questions resource. Your server must support the following for these requests:

1. The server will respond with a JSON response containing the following key/values:

   status: an integer value of either 0 (success), 1 (no results or not enough questions to fulfill request), 2 (invalid token, see 'Adding Sessions' section)

   results: an array of question objects, where each question contains all data stored in that question's data file. This does not need to be sent if status is 1 or 2.

2. If no query parameters are specified, the default server behaviour will be to return 10 randomly selected questions.
3. The client should be able to supply appropriate query parameters, in which case the server should respond with the correct number of questions that match the specified category/difficulty. Note that a client may specify any number of these parameters. For example, they may only specify a desired category. If a query parameter is not specified, the default behaviour for the parameter, described above, should be used. The query parameters your server must support are:
    a. limit: integer – the number of desired questions
    b. difficulty: integer – the desired difficulty ID
    c. category: integer – the desired category ID
    d. token: string – the session ID (see 'Adding Sessions' section)

Adding Sessions to The Trivia API

For this problem, you will add the concept of sessions to your trivia API. This will allow a client to generate a unique session ID and ensure that when the user specifies their session ID as a query parameter, the same question is never served up to them more than once. All your session data should be stored on the filesystem to ensure that many sessions can be stored at once. While it is inefficient, you should use synchronized read/write operations for these files. It is unsafe to call fs.writeFile multiple times without waiting for the callback function to be executed. If you do so, you may end up with corrupted data. We will address this problem once we have covered databases. The session API should support the following.

POST /sessions – This must create a new session on the server. Generate a unique ID, initialize a blank session (i.e., one that has not requested any questions yet), and respond to the request with status code 201 (created) and a body containing the newly created session ID.

GET /sessions – Respond with status code 200 and an array of string values containing each active session ID the server currently has. Note that, in a real application, you would not want this list to be publicly available for anybody to request. However, it is useful for testing purposes.

DELETE /sessions/:sessionid – This will delete the session with ID equal to the :sessionid parameter, if it exists. The server should respond with a status code of 200 (success) or 404 (session did not exist).

Finally, GET requests for the /questions resource must allow a client to specify a token value in the query string. This token value will represent the unique session ID for a session on the server. If the token is valid (i.e., the session exists), the results included in the response should not contain any questions that the requesting session has already received. If the token is invalid or there are not enough questions to fulfill the request, the server should respond with JSON data indicating the correct status, as outlined in the Basic Trivia API section.

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, instructions for how to run/use your system, and an explanation of any additional features that you implemented for the assignment.**

Recap

Your zip file should contain all the resources required for your assignment to run. The TA must be able to run your server and use the system by following instructions in the README.txt file.
Submit your **assignment3.zip** file to cuLearn.
Make sure you download the zip after submitting and verify the file contents.
