Assignment 2

Real-time communication using Socket.io

Submit a single zip file called **assignment2.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 the server and client for a real-time multiplayer trivia game. The server will be responsible for serving up all necessary resources (HTML, Javascript, etc.) and coordinating the communication between multiple clients using Socket.io. The client will be responsible for displaying questions, allowing the user to answer questions, and displaying the scores of players in the game.

Before starting your design for the assignment, you are encouraged to read through the entire specification. It is possible that later questions 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.

Basic Trivia Game Rules

The rules below outline the basic requirements of the trivia game. It will be your responsibility to ensure that the server and clients follow these rules.

1. The first round of trivia should start when the first user joins. If all the players leave at any point (i.e., there are no users connected), the current round should be aborted and a new round should start when the next person joins.
2. Each round of trivia will consist of five multiple choice questions.
3. At the start of each round, every player's score should be set to 0.
4. For each question, each player should only be able to select their answer once. To avoid cheating, additional answers should be disabled or ignored.
5. If a player's answer is incorrect, that player should lose 100 points.
6. If a player's answer is correct, that player should gain 100 points.
7. The round should only proceed to the next question after all connected players have answered.
8. When a round is over, all players should be shown the name(s) of the winner(s) and a new round should start.
9. If a player joins during the middle of a round, they can start playing immediately.

Server Requirements

1) The server must serve up any resources that your clients will require (e.g., join game page, game play page, client-side Javascript, etc.).
2) The server should use the 'request' module to retrieve new questions from Open Trivia DB at the start of each new round. For testing purposes, you may want to start by hard-coding question data into your server code and add this functionality later.
3) The server must support any number of players in the game.
4) The server should gracefully handle invalid requests. For example, if a request is made for a resource that doesn't exist, the server should send a 404 response. The server should not crash when an invalid request is made.
5) The server should start a new round when the first player joins. If all players leave, the server should stop the current round and start a new round when the next player joins.

Client Requirements

At minimum, the client should:

1) Allow the player to enter their name and choose to join the game.
2) Allow the player to see multiple choice questions and select answers.
3) Show the player their score.
4) Show the player the names of other players in the game and their scores.
5) Update the display whenever the game state changes (e.g., when a player's score increases, the current question is changed, etc.).
6) Allow each client to see the status of others in the game (whether they have answered or not). This can be done by changing the name color of players, highlighting, etc.

Add-On Features

In addition to the minimum feature set described above, select and implement some extensions from the following list. This section is worth twenty-five points and each of the given extensions are allocated a certain number of points. If you implement the given extension perfectly, you will get the full marks allocated to it. You may implement as many of these add-ons as you want. The maximum marks you can receive for this part is thirty-five, which can allow you to make up for small mistakes on earlier parts or minor issues in your add-on implementation(s). The maximum mark you can receive on the assignment will still be 100.

1) Add a statistics page that shows information about each player, such as average score, number of games played, number of wins, etc. (10 marks). Also store the player statistics to a file so that the statistics are remembered even if the server is restarted (5 marks).
2) Add chat functionality to the game. Players should be able to send messages and view all messages sent by others. Each message should display the name of the

player who sent that message (5 marks). Add private messaging, so a user can send a message to a selected player without the other players seeing (10 marks).

3) Add an administrator page which allows control over the game. The administrator page should allow the person viewing that page to restart the game, advance to the next question, and remove players from the game (10 marks).
4) Add a timer to the game so that players only have 30 seconds to answer each question (5 marks). Additionally, modify the game to give points relative to how much time the user takes to answer the question (5 marks). The exact scoring scheme is left to your discretion, but for example, you could give 100 points for <5 seconds, 75 points for 5-10 seconds, 50 points for 10-20 seconds, and 25 points for 20-30 seconds. Add hints to the game by using a timer to remove/disable incorrect answers over time as the time limit approaches (10 marks). If you correctly implement all three of these features, they would be worth 20 marks total.
5) Allow multiple games to be run simultaneously. Add functionality to the client so that a player can choose to create a game instead of join. When a game is created, the player should be given a randomly generated code representing their game name. You should ensure there are no duplicate codes. Players who wish to join a game will have to supply a valid code (25 marks).

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 add-on 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 **assignment2.zip** file to cuLearn.
Make sure you download the zip after submitting and verify the file contents.