**TRIVIA GAME API**

The main goal is to create a simple API and multiplayer server for a Trivia game.

Please review a [live example of this game](http://projects.fgfactory.com/trivia_multiplayer/) (to play with “opponent” just open this link twice in the different browser tabs or windows and leave the default root API url - [http://fgftestapi.somee.com](http://fgftestapi.somee.com/)).

**TECHNOLOGIES STACK:**

ASP.NET Core 3.\*

SignalR

MS SQL DB / Mongo DB (preferred)

Entity Framework

**MAIN MODELS**

(this structure can be modified if needed, but should work properly with client game application)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Player** | **Category** | **Question** | **Answer** | **GameplayRoom** |
| Id  Name  Score  LastGameDate  IsGameOrganizer  ConnectionId  CharacterColor | Id  Name  Questions | Id  Text  Answers  Category | Id  Text IsCorrect | Id  MaxPlayers  Players |

If database does not exist, it must be created on application start and **seeded with test data**:

* Players - about 100 players with random score in range 0 - 700 and random last game date during past two months)
* Categories - any 10 categories
* Questions - 50 sample questions for every category (like “This is the 7 question for the category ‘Art’”)
* Answers - 4 answers for every question (only one answer can be marked as correct)

**CORS POLICIES**

API and SignalR hub should be accessible for the limited list of domains (according to the client application hosting specific).

**API ENDPOINTS**

Such endpoints should be accessible

|  |  |  |
| --- | --- | --- |
| **Method** | **Endpoint** | **Description** |
| GET | {host}/api/Categories | Returns the list of all existing categories |
| GET | {host}/api/Questions/By\_Category/{categoryId} | Returns random question from specified category (answers array included) |
| GET | {host}/api/Players/leaderboard/{daysPeriod} | Returns the list of all players, who played a game for the last time in a specified period (sorted descending). For example, if daysPeriod = 30, all the users who played the game during the last month, should be sorted and returned. |

**MULTIPLAYER**

Multiplayer hub must be implemented using SignalR technology. Client game application uses @microsoft/signalr package for integration.

Client game can invoke such methods of the hub:

|  |  |  |
| --- | --- | --- |
| **Method** | **Arguments** | **Description** |
| Join | characterColor: string | Is called when user is ready to play. Selected character color is passed as an argument.  If no user with such connection Id exists in the database - a new player with random name is created.  If nobody is waiting to play - a new room should be created. After that the user is waiting for someone to play with.  If someone has already created a room and is waiting for his opponent - player joins his room and the game is started.  When the second player appears in the room, hub must inform both of them about the opponent's data (“OpponentJoined”) and tell them that they can play (“CanPlay”). See detailed description below. |
| Send | jsonData: string | This method is used for data exchange between players. The hub’s job is to get JSON data string and to send it to the second player in the room (method “Send”). |
| Leave | - | Client invokes this method when player decides to complete the game, or it can be called on the hub side when player becomes disconnected.  Hub must send notification (“OpponentLeave”) to the opponent of the player who left and throw both of them out from the game room. This room can be deleted from the DB. |

Hub invokes such methods of the game client:

|  |  |  |
| --- | --- | --- |
| **Method** | **Arguments** | **Description** |
| OpponentJoined | -Name: string  -CharacterColor: string  -IsGameOrganizer:boolean | Is sent to both players when the second player appears in the room.  CharacterColor - a string to inform which character was selected by the player  IsGameOrganizer - true is assigned to the room creator or game initiator (in case of replay) |
| CanPlay | - | Is sent to both players when the second player appears in the room. |
| OpponentLeave | - | Hub sends this method to the opponent of the player who left, when player decides to complete the game, or when player becomes disconnected.  Both players become thrown out from the game room and this room is deleted from the DB. |
| Send | jsonData: string | The hub takes JSON data string and sends it to the second player in the room. |

**TEST INSTRUCTIONS**

[This game client](http://projects.fgfactory.com/trivia_multiplayer/) can be used to test integration (just specify your hosted API url in the prompt).

If some client-side investigations or modifications are required - please use [this source code](https://drive.google.com/drive/folders/1vtRdVSawPbA0FHuMNdkYWdluoxFUVvsS?usp=sharing) to create your custom build of the game.

To **launch client-side project**, run:

* npm install
* npm run dev - to run the game in the development mode (build appears in the dist folder and can be launched with any http server)
* npm run build - to create the production build (build appears in the dist folder and can be launched with any http server)

**Server communication** settings are here: src\Helpers\GameSettings.ts

**API integration** was made here: src\Server\ApiServer.ts

**SignalR hub** integration is here: src\Server\MultiplayerServer.ts

**Multiplayer actions** are processed with a state machine, which is implemented in this folder: src\state

**TEST TASK RESULT PRESENTATION**

Please provide source code of the project as a link to the public git repository with deployment instructions in README.md.

If custom game build was made - it should be provided either.

Also, projects can be hosted ([free ASP hosting](https://somee.com/FreeAspNetHosting.aspx) (MS SQL only), [free hosting for game client](https://www.000webhost.com/)) and the link provided.