****

**PLAYSTUDIOS TECHNICAL CHALLENGE**

**Step by step instructions to run PSQuest.API solution**

## **Background:**

* This Web API project for Player Quests progression is built using ASP.Net Core 5.0 and Microsoft SQL Server v2015 with some handy Nuget packages such as AutoMapper, FluentValidation, EntityFrameworkCore, EntityFrameworkCore.SqlServer, Swashbuckle.AspNetCore.Swagger etc.
* The IDE used to develop this API project is Microsoft Visual Studio 2019 but it can also work with Visual Studio Code.
* The structure of this solution contains mainly 4 projects :
  + **PSQuest.API**: It is an ASP.Net Core 5.0 Web API project where the controllers of the APIs reside. It is an entry point for the clients to enter and call the API endpoints.
  + **PSQuest.Core:** As the name suggests, the core business logic of handling Player Quest progression is written in various classes of this Class Library project.
  + **PSQuest.Data:** It is a Class Library project containing the Models, Entities, Migrations and Data transfer objects. This project is responsible to connect with the Database named “**PSQuest”**. The connection to the database is done through EntityFrameworkCore.
  + **PSQuest.Tests:** This is a XUnit test project contains all the Unit test cases to test the API endpoints and the core business logic.

## **Prerequisites**

* Microsoft .Net Framework 5.0
* Microsoft Visual Studio 2019 or Visual Studio Code
* Microsoft SQL Server V15+
* Execute the DB schema **PSQuest\_Schema.sql** located under Documents folder

## **Further instructions to run the solution:**

* To run this API project, first you will need to run the database schema attached in the “**Documents**” folder. The file named “**PSQuest\_Schema.sql**” contains the schema of the tables used in this API project.
* After running the schema on your server, you will need to change the connection string with your server name and the user credentials on your server in the AppSettings section stored in the variable “SqlConnectionString” in the **appsettings.json** file. The path to “appsettings.json” file is ***Solution -> src -> PSQuest.API -> appsettings.json***
* You are advised to keep the database name as “**PSQuests**” or if you keep a different name then make sure you change the database name in the connection string above in the property “SqlConnectionString” in the **appsettings.json** file.
* Once the database connection string is updated as per above steps, you can look for “**questConfig.json**” file located in ***PSQuest.Core -> Common -> questConfig.json*** Path.
* The “questConfig.json” file contains all the Quest configurations and the properties used in the JSON configuration file helps to support the Player Quest progression endpoint in the API project. There is further explanation for each properties used in this JSON configuration file in the documents folder in the file named “***Quest Configuration JSON-Explained.docx***”.
* The path to Quest configuration is also stored in the AppSettings section stored in the variable “QuestConfigFilePath” in the **appsettings.json** file.
* You can make changes in the Quest configuration file as per your needs and change status of any quest to Active to Inactive. You can add n number of milestones in each quest.
* The database contains Players table where the basic information of each player is stored who will be participating the Game Quests. Any player not found in the database table will not be given access to participate in the quest.
* A controller is added in the “**PSQuest.API**” project and it contains the below endpoints
  + *api/progress* endpoint to serve the quest progress requests
  + *api/state* endpoint to server the player quest state.
* There are 3 services created in the “**PSQuest.Core**” project for specific purpose like:
  + **QuestConfigService**: Responsible to load the JSON configurations of all quests and determine the Active Quest (Only 1 active quest at any given time). This service is called by below two services as and when needed.
  + **QuestService:** Responsible to handle player quest progression and return the response to *api/progress* endpoint. This service also handles the Player Quest state with the endpoint *api/state*
* Once the database and the quest configuration JSON file is set up, you are all ready to run the API project by debugging the “PSQuest.API” project and test the endpoints with Swagger or Postman tools. The swagger is configured in the Startup.cs class to test the API endpoints.
* The *api/progress* request must contain PlayerId (string), PlayerLevel (number) and ChipAmountBet (number) in the request object. I have assumed that the PlayerId and PlayerLevel passed in the API request is legitimate and managed by the client on their side. The PlayerId passed in the API request is eligible to participate in the quest progress.
* The same assumption about a PlayerId mentioned above applies to the other endpoint *api/state* for maintaining player quest state.