Overview:

The project aims to implement the requirements mentioned in the document by implementing the extended game version of rock-paper-scissors.

As technologies for the implementation, it has been used as following:

- .Net Core 3.1 for API development
- MSSQL database for storage system
- EF Core as an ORM to connect to the storage systems
- Swagger for API interaction and possible documentation
- Fluent Validation for input validation
- Docker for containerization support

The projects' structure fallows clean architecture with elements from design driven development.

To achieve this CQRS and Mediator patters have been used.

High level architecture:

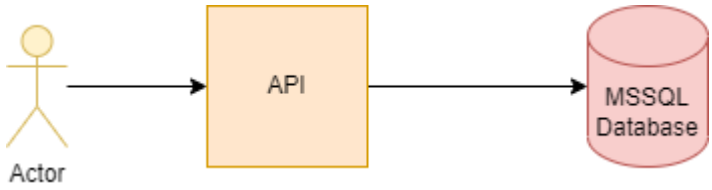

Fig 1.0 High Level Diagram

Useful Links:

The github repository for the code source:

https://github.com/indrefi/Rock_Paper_Scissors_Lizard_Spock_Game

A docker image of the api on the public docker hub:

https://hub.docker.com/r/ionutindre/rock_paper_scissors_lizard_spock_game

API Structure:

The API is structured as following:

Presentation -> Web API

Core -> Application + Domain

Infrastructure -> Infrastructure + Persistence + Database Project

Tests -> Unit Tests

Data Flow :

- The data flow starts in the web API project.
- When a specific endpoint is called a specific query or command is instantiated and called
- The command and query data are validated using query validators defined on the same folder level as the queries or commands.
- For each query or command there is an interface and an implementation for that specific request.
- In the Application project this can be identified in the **UseCases** folder. (See Fig. 2.0)
- The handler implementation resided in the Persistence project along the EF Core Configurations. (See Fig 3.0)

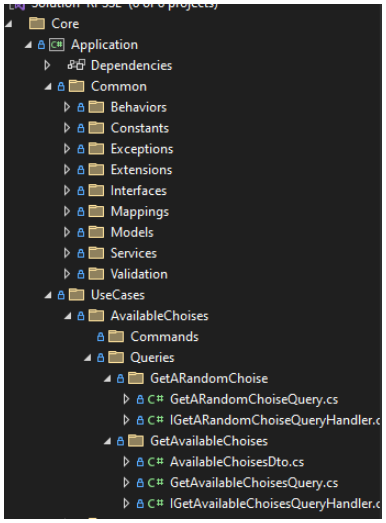

Fig 2.0 Application Project and Use Cases

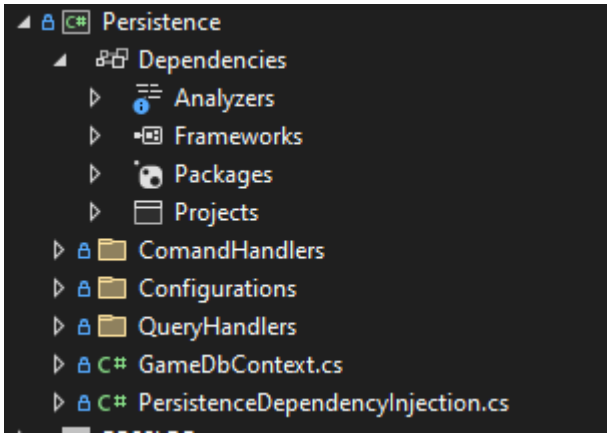

Fig 3.0 Persistence Project

- For each project there is a dependency injection extension which is called in the end in the startup project to glue all together.
- Note: When adding a new service it will be needed to register it to the dependency injection extension in the persistence layer (see also mediatR service registration)

Setup:

1. Database

The prerequisite will be to have installed an MSSQL server installed on the system.

After this step is completed to setup the database you can do this by pressing **Publish** button from the Database project in Visual Studio.

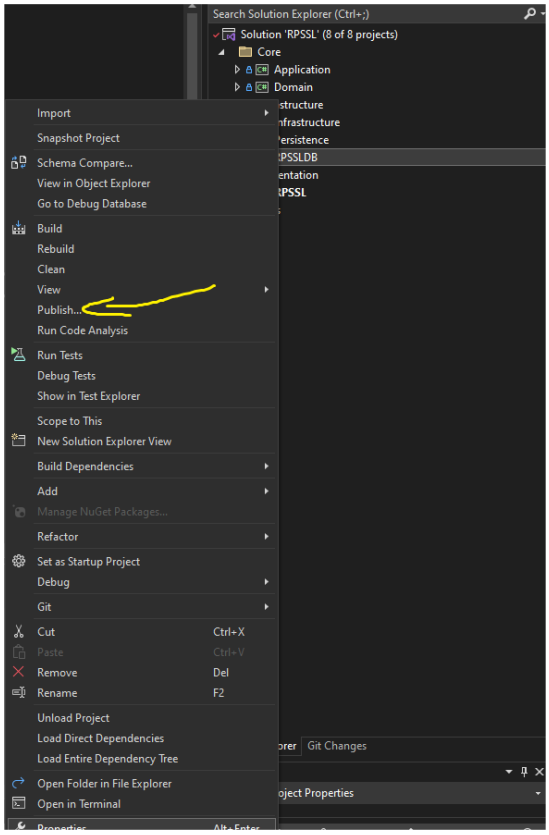

Fig 4.0 Publish Database

After that you will have to select the SQL Sever to which you want to run the publish against.

For Database name use RPSSLDB as can be seen in the picture below.

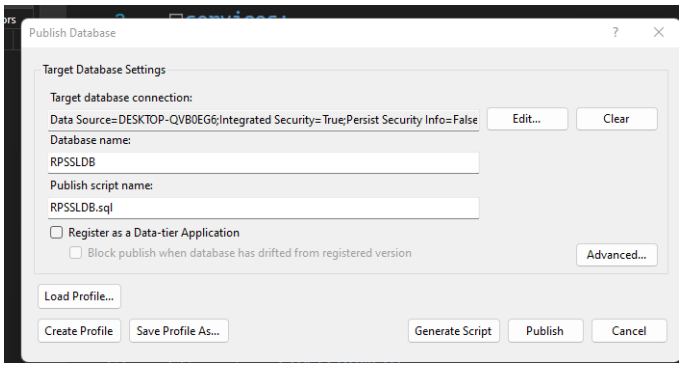

Fig 5.0 Publish Database settings

If you don't want to do it from Visual Studio there is a script in **/docs** folder under the GitHub Repository called **deployment_query.sql.** by running this one against MSSQL Server will create the schema and add the seed data.

A third option can be to use a docker image for mssql, connect to it and run the script manually there.

Get image:

docker pull mcr.microsoft.com/mssql/server

Run container:

docker run -e "ACCEPT_EULA=Y" -e "MSSQL_SA_PASSWORD=yourStrong(!)Password" -e "MSSQL_PID=Express" -p 1433:1433 -d mcr.microsoft.com/mssql/server:2019-latest

Connect to container manually:

docker exec -it <container_id|container_name> /opt/mssql-tools/bin/sqlcmd -S localhost -U sa -P <your_password>

2. Setup and run the API

The interaction with API endpoint can be done in an user friendly manner by just starting the api and going to the root address. **https://localhost:5001/index.html**

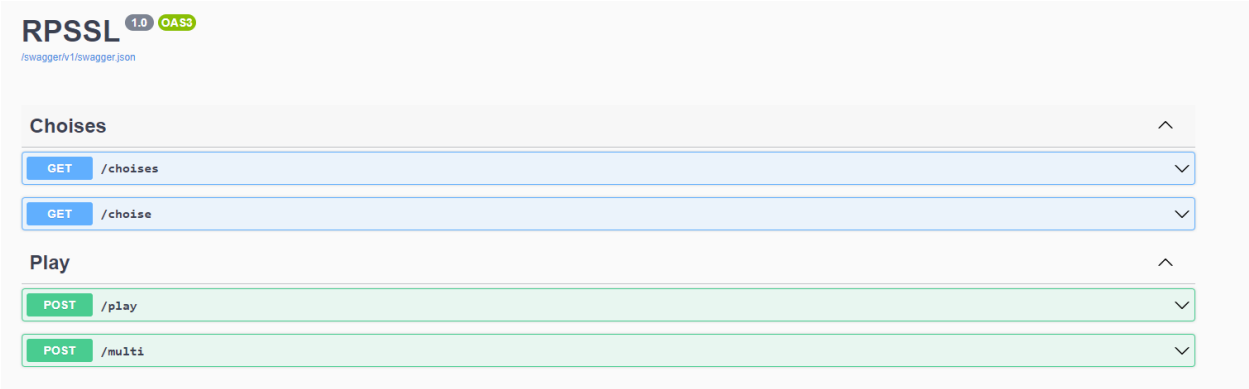

Fig. 6.0 Swagger Endpoints

Here are the necessary endpoints that provide the ability to fetch available choices or a random calculated chose.

Note:

If you want to build the docker image for the API locally the following command can be used. Before running it you will have to be in the root directory where src folder is.

```
docker build -f "src\RPSSL\RPSSL\Dockerfile" --force-rm -t rpssl:dev
```