Bases Utilizadas

Para cadastrar as jogadas, será usada a base Jogo. O banco de dados utilizado é o MySQL.
A base é usada como Classe para o desenvolvimento da aplicação.

Jogo

Armazena os dados das partidas

```
Id int NOT NULL AUTO_INCREMENT
Placar int NOT NULL
PontuacaoMinimaTemporada int NOT NULL
PontuacaoMaximaTemporada int NOT NULL
QuantidadeRecordMinimoQuebrado int NOT NULL
QuantidadeRecordMaximoQuebrado int NOT NULL
```

Estrutura do Projeto

Aplicação principal, desenvolvida em Angular utilizando a IDE VSCode. Para o design das interfaces, foi utilizado Angular Material. O projeto foi desenvolvido com a seguinte estrutura:

front-end (\frontend)

```
ng new frontend
```

Estrutura do frontend

- components – foi criado um componente para as ações Criar (create) e Ler (read)

```
ng generate component classe-acao
Ex.: ng generate component jogo-read
```

- views - Componente para "Apresentação" da entidade: opção para um novo registro da entidade e a leitura (read) na mesma.
- models - Modelo da Entidade Jogo.
- services - Para a entidade, foi criado um serviço para comunicação com a API: getAll e Post.

```
ng generate service entidade
Ex.: ng generate service jogo
```

A comunicação com a API é feita através dos caminhos:

```
baseURL = `${environment.mainUrlAPI}jogo`
mainUrlAPI: 'http://localhost:5000/'
Ex.:
getAll(): Observable<Jogo[]> {
    return this.http.get<Jogo[]>(this.baseURL).pipe(
    map((obj) => obj),
    catchError((e) => this.errorHandler(e))
);
```

Rota

```
http://localhost:4200/jogos
```

back-end (\backend)

API desenvolvida em .Net Core para comunicação entre o Banco de Dados MySQL e a aplicação principal.

```
dotnet new webapi
```

Estrutura do backend

- data
  - DataContext: através dele podemos acessar os métodos create, read e outros métodos de interação com o banco de dados.
  - IRepository: interface que contém a assinatura de métodos padrões (Add, Update, SaveChanges) referentes a cada modelo.
  - Repositoy: classe que implementa a interface IRepository.
- Controllers: responsável por responder as requisições em nossa API.
- models: modelo da aplicação, que seriam as referências às tabelas que temos no banco de dados.
- Migrations: guarda informações das migrações que são feitas. Através do comando abaixo, EF Core "liga" as informações contidas na pasta models com as contidas no DbContext e cria um esquema da nossa base de dados: banco e tabelas, criando um histórico dentro desta pasta. Com o próximo comando, o EF cria/atualiza o banco de dados a partir da migração.

```
dotnet ef migrations add Nome-Migration
dotnet ef database update
```

- Services: contém Classes com serviços específicos.
  - CalculoMinimoTemporadaServico: atualiza o placar mínimo, caso o novo placar seja menor que o mínimo já existente.
  - AtualizaQuebraRecordMinimoServico: atualiza a quantidade de quebras de recorde mínimo caso o placar mínimo tenha sido atualizado.
  - CalculoMaximoTemporadaServico: atualiza o placar máximo, caso o novo placar seja maior que o máximo já existente.
  - AtualizaQuebraRecordMaximoServico: atualiza a quantidade de quebras de recorde máximo caso o placar máximo tenha sido atualizado.

Depois de pronta, para testar a API, foi utilizado o Postman. O objetivo é fazer requisições HTTP e avaliar se as repostas (retornos) foram dentro do esperado.

Executando a aplicação no VSCode

Para que a aplicação seja executada, deve-se abrir o terminal no VSCode e executar os seguintes comandos: (os passos a seguir, devem ser executados apenas na primeira vez ou somente quando necessário)

- Passo 1: dentro de \backend: (executar somente uma vez, para a criação da base de dados)

```
dotnet ef database update
```

- Passo 2: dentro de \frontend: O próximo passo faz-se necessário somente caso necessite baixar/atualizar alguma dependência do projeto. Antes da primeira vez que for rodar a aplicação, deve-se baixar todas as dependências do projeto, executando o procedimento abaixo.

```
npm update
```

(os passos a seguir, devem ser executados após os passos anteriores)

- Passo 3: dentro de \backend:

```
dotnet watch run
```

- Passo 4: dentro de \frontend:

```
npm start
```

Se não apresentar nenhum erro, pode-se acessar a aplicação pelo navegador, através do endereço http://localhost:4200

Retornos

Retorno das requisições feitas através do Postman

Obter Jogo

- GetAll: obtém todos as jogadas/pontuações cadastradas

```
url = http://localhost:5000/jogo
method = GET
{
        "id":
        "placar":
        "pontuacaoMinimaTemporada":
        "pontuacaoMaximaTemporada":
        "quantidadeRecordMinimoQuebrado":
        "quantidadeRecordMaximoQuebrado":
    }
```