

Descrição da atividade: codificar um serviço para persistir nomes de times de futebol e datas de partidas.

Data de entrega: presencialmente na aula de 23/maio (terça-feira).

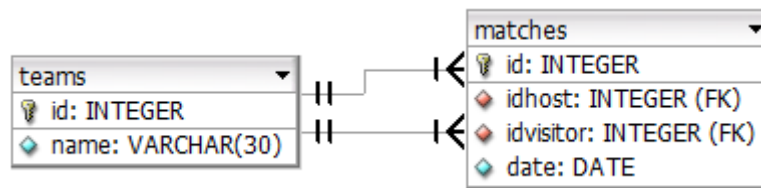
Forma de entrega: individual.

Objetivos:

- TypeORM;
- Rotas.

Requisitos funcionais:

- Utilize o seguinte modelo de dados:



- A aplicação deverá ter as seguintes rotas:

- HTTP GET localhost:3004/team: retorna um array com os JSON de todos os times ordenados em ordem ascendente de nome. Exemplo de resultado para a requisição <http://localhost:3004/team>:

```
[
  {
    "id": 4,
    "name": "Botafogo"
  },
  {
    "id": 3,
    "name": "Flamengo"
  },
  {
    "id": 8,
    "name": "Fluminense"
  },
  {
    "id": 6,
    "name": "Juventude"
  },
  {
    "id": 1,
    "name": "Palmeiras"
  },
  {
    "id": 5,
    "name": "Santos"
  },
  {
    "id": 7,
    "name": "São Gonçalo"
  },
  {
    "id": 2,
    "name": "São Paulo"
  }
]
```

- HTTP GET localhost:3004/team/**termo**: retorna um array com os JSON de todos os times que possuem no nome o **termo** fornecido. O resultado deve ser ordenado em ordem ascendente de nome. Exemplo de resultado para a requisição http://localhost:3004/team/**são**:

```
[
  {
    "id": 7,
    "name": "São Gonçalo"
  },
  {
    "id": 2,
    "name": "São Paulo"
  }
]
```

- HTTP POST localhost:3004/team: persiste um registro na tabela teams do SGBD. Exemplos de requisições:

Parâmetro enviado pelo body da requisição:	Resposta do servidor:
{ "name": "Fluminense" }	{ "name": "Fluminense", "id": 8 }
Ao tentar repetir o nome do time:	Resposta do servidor:
{ "name": "Fluminense" }	{ "error": "O nome já existe" }

- HTTP PUT localhost:3004/team: atualiza o nome do time. Exemplos de requisições:

Parâmetro enviado pelo body da requisição:	Resposta do servidor:
{ "id": "5", "name": "Ponte Preta" }	{ "id": 5, "name": "Ponte Preta" }
Ao tentar usar um nome repetido:	Resposta do servidor:
{ "id": "5", "name": "Flamengo" }	{ "error": "O nome já existe" }

- HTTP DELETE localhost:3004/team: deleta um time da tabela teams do SGBD. Exemplos de requisições:

Parâmetro enviado pelo body da requisição:	Resposta do servidor:
{ "id": "5" }	{ "raw": [], "affected": 1 }
Ao tentar excluir um registro que não existe:	Resposta do servidor:
{ "id": "5" }	{ "raw": [], "affected": 0 }

- HTTP POST localhost:3004/match: persiste um registro na tabela matches do SGBD. Exemplo de requisição:

Parâmetro enviado pelo body da requisição:	Resposta do servidor:
{ "idhost": "1", "idvisitor": "7", }	{ "host": { "id": 1, }

```
{
  "date": "2023-04-29"
}
```

```
    "name": "Palmeiras"
  },
  "visitor": {
    "id": 7,
    "name": "São Gonçalo"
  },
  "date": "2023-04-29",
  "id": 1
}
```

- HTTP GET localhost:3004/match: retorna os registros da tabela matches do SGBD. O resultado é apresentado em ordem decrescente de data. O resultado pode ser paginado usando os parâmetros limit e offset. Exemplo de requisição:

Parâmetro enviado pelo body da requisição:

```
{
  "limit": "",
  "offset": ""
}
```

Resposta do servidor:

```
[
  {
    "id": 1,
    "date": "2023-04-29",
    "host": {
      "id": 1,
      "name": "Palmeiras"
    },
    "visitor": {
      "id": 7,
      "name": "São Gonçalo"
    }
  },
  {
    "id": 2,
    "date": "2023-04-22",
    "host": {
      "id": 8,
      "name": "Fluminense"
    },
    "visitor": {
      "id": 3,
      "name": "Flamengo"
    }
  }
]
```

- HTTP GET localhost:3004/match/:id: retorna os registros da tabela matches que possuem o time como visitante ou mandante. O resultado é apresentado em ordem decrescente de data. Exemplo de resposta para a requisição http://localhost:3004/team/3. No resultado estão os jogos que possuem idhost = 3 ou idvisitor=3:

```
[
  {
    "id": 3,
    "date": "2023-05-02",
    "host": {
      "id": 3,
      "name": "Flamengo"
    },
    "visitor": {
      "id": 4,
      "name": "Botafogo"
    }
  }
]
```

```

    }
  },
  {
    "id": 2,
    "date": "2023-04-22",
    "host": {
      "id": 8,
      "name": "Fluminense"
    },
    "visitor": {
      "id": 3,
      "name": "Flamengo"
    }
  }
]

```

- HTTP PUT localhost:3004/match: atualiza os dados de um jogo na tabela matches do SGBD. Exemplos de requisições:

Parâmetro enviado pelo body da requisição:	Resposta do servidor:
<pre> { "id": 3, "idhost": "11", "idvisitor": "5", "date": "2023-05-05" } </pre>	<pre> { "error": "Mandante desconhecido" } </pre>
<pre> { "id": 3, "idhost": "4", "idvisitor": "5", "date": "2023-05-05" } </pre>	<pre> { "error": "Visitante desconhecido" } </pre>
<pre> { "id": 3, "idhost": "2", "idvisitor": "6", "date": "2023-05-05" } </pre>	<pre> { "id": 3, "date": "2023-05-05", "host": { "id": 2, "name": "São Paulo" }, "visitor": { "id": 6, "name": "Juventude" } } </pre>

- HTTP DELETE localhost:3004/match: deleta um jogo da tabela matches do SGBD. Exemplos de requisições:

Parâmetro enviado pelo body da requisição:	Resposta do servidor:
<pre> { "id": "3" } </pre>	<pre> { "raw": [], "affected": 1 } </pre>
<pre> { "id": "3" } </pre>	<pre> { "raw": [], "affected": 0 } </pre>

Requisitos não funcionais:

- a. A aplicação deverá ser codificada usando Node TypeScript e Express;
- b. A aplicação deverá rodar na porta 3004;
- c. A aplicação deverá usar migrations para criar as tabelas;
- d. As tabelas deverão ser mantidas no SGBD PostgreSQL da cloud ElephantSQL (<https://www.elephantsql.com>);
- e. A aplicação deverá estar no cloud Render (<https://render.com>).

Dicas:

- Adicione o pacote cors (Cross-origin Resource Sharing) para o servidor aceitar requisições de outros domínios (<https://www.npmjs.com/package/cors>) e a dependência de desenvolvimento @types/cors.

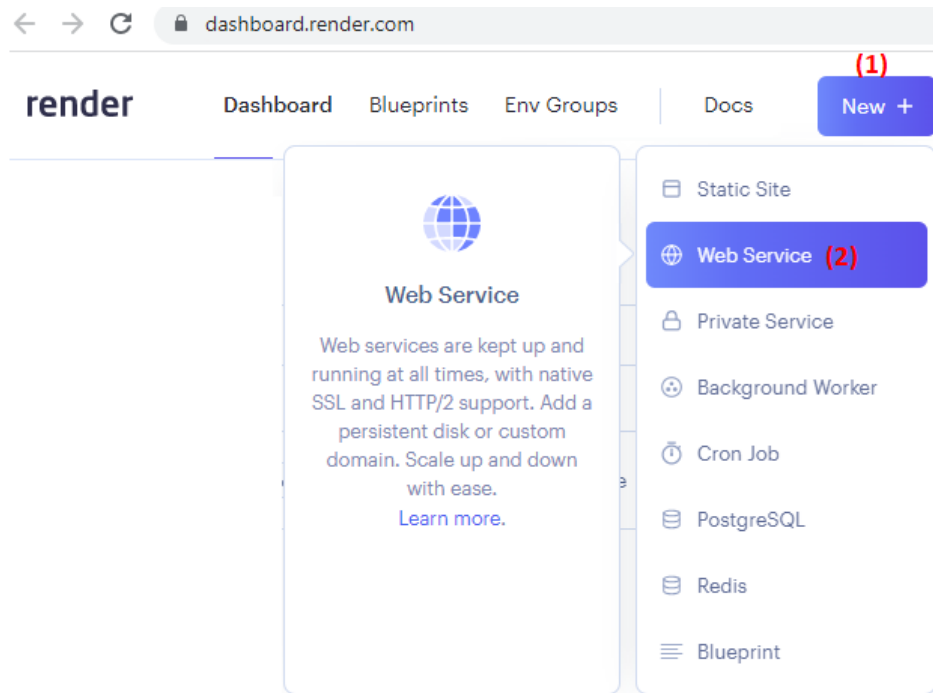
Adicione as instruções sinalizadas em amarelo no módulo de criação do seu aplicativo (arquivo index.ts):

```
import * as express from "express";
import {Request} from "express";
import * as dotenv from 'dotenv';
import * as cors from 'cors';
dotenv.config();

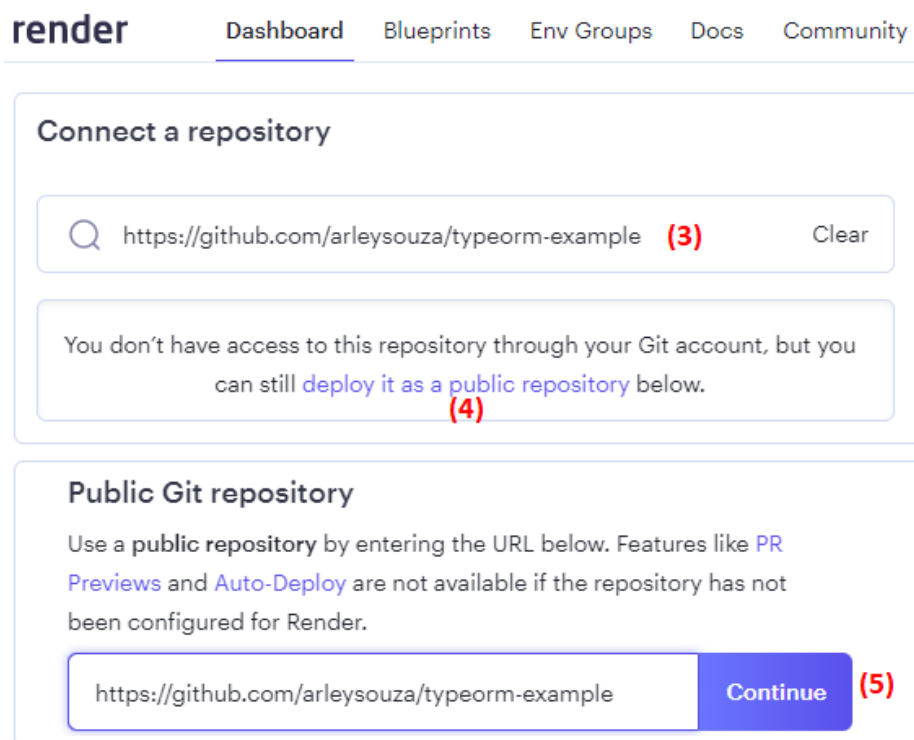
const PORT = process.env.PORT || 3000;

const app = express();
app.use(express.json());
app.use(cors<Request>()); //aceitar requisições de outros domínios
app.listen(PORT, () => console.log(`Rodando na porta ${PORT}`));
```

- Passos para fazer o deploy da aplicação no Render:
 - 1) O primeiro passo é subir a aplicação no GitHub – o repositório pode ser público ou privado;
 - 2) Acesse o Dashboard do Render(<https://dashboard.render.com>) para criar uma instância do tipo **web service**. No caso de dúvidas veja o vídeo <https://www.youtube.com/watch?v=bnCOyGaSe84>;



- 3) Na tela seguinte forneça a URL do repositório no GitHub. Em repositórios privados você será direcionado para uma tela do GitHub para autorizar;



- 4) Na tela seguinte forneça os parâmetros para subir a aplicação na cloud. Pontos de atenção:

- Forneça a branch do seu repositório no GitHub;
- Forneça o comando `npm install` para instalar as dependências do seu projeto;
- Forneça o comando para subir a aplicação. Neste exemplo usaremos o `ts-node`;


No final da tela clique no botão [Create Web Service](#) para iniciar o deploy da aplicação.


Name	<input type="text" value="exemplo"/>	
Region	<input type="text" value="Oregon (US West)"/>	
Branch	<input type="text" value="master"/>	
Root Directory	<input type="text" value="."/>	
Runtime	<input type="text" value="Node"/>	
Build Command	<input type="text" value="."/>	<input type="text" value="\$ npm install"/>
Start Command	<input type="text" value="."/>	<input type="text" value="\$ ts-node src/index.ts"/>

- 5) Na tela seguinte será exibida a URL da aplicação no Render e o progresso do deploy numa janela de console.

render Dashboard Blueprints

WEB SERVICE

 **exemplo** Node Free

<https://exemplo-d2zq.onrender.com> 

```

28 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities

=> Generating container image from build. This may take a few minutes...
=> Uploading build...
=> Build uploaded in 23s
=> Build successful 🎉
=> Deploying...
=> Starting service with 'ts-node src/index.ts'
  
```