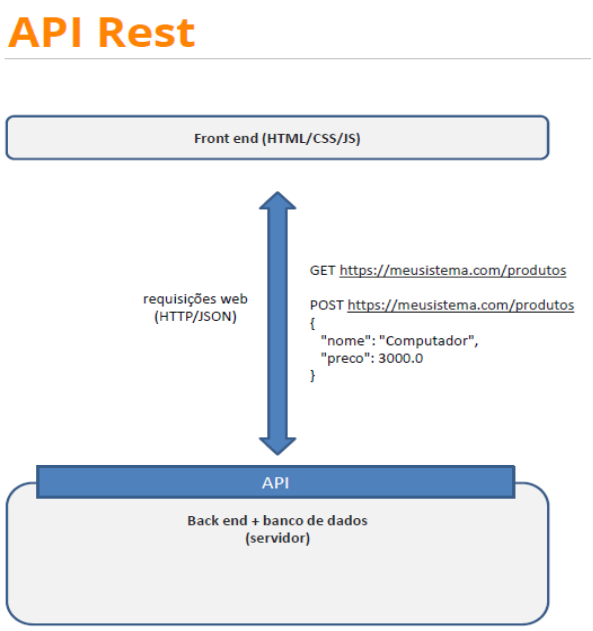
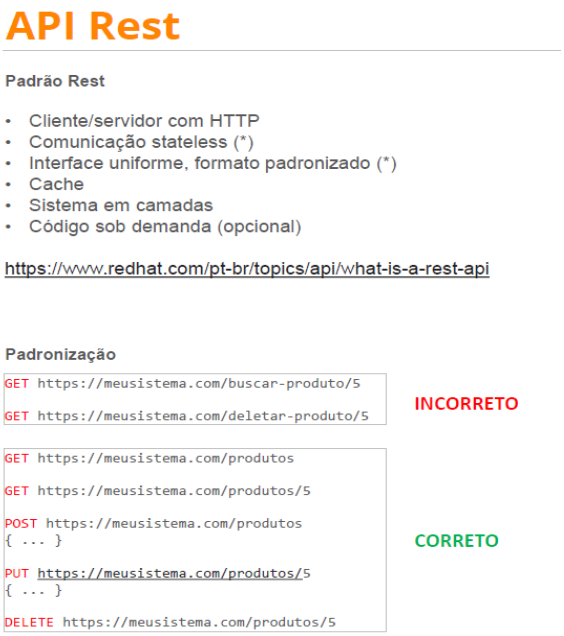
Minhas anotações do projeto:

API Application Interface - é o que fica exposto para WEB. É a porta de entrada do seu back-end.



REST - é um padrão que especifica como uma API deve ser implementada para ser considerada REST. Seguindo os padrões do REST você terá uma API REST:



1- Tem que ser uma aplicação cliente/servidor;
2- Comunicação é stateless (não guarda status, ou seja, o resultado da requisição não depende de algo que o sistema deva guardar para retornar o resultado. Ex. o retorno não depende de algo que precisa estar armazenado previamente na sessão do usuário para retornar)
3-Interface uniforme com formato padronizado utilizando os verbos http (utilização dos verbos http para realização das operações)
GET - obter
POST - gravar
PUT - alterar (verbo idempotente significa que uma requisição realizada 1 vez ou 10 vezes, o resultado será o mesmo. Ex. alterar um endereço, é sempre o mesmo endereço então é idempotente. Agora no nosso sistema ao alterar a posição dos jogos não é idempotente, porque a cada mudança de posição gera outro resultado, neste caso o verbo a ser utilizado será o POST).
DELETE - deletar

4 - Utilização de Cache
5 - Sistema em camadas
6 - Código sob demanda (opcional)

Padrão camadas

. Controladores REST (Os controladores são as portas de entrada/interface = API);
. Camada de Serviços (realiza as transações);
. Camada de Acesso a Dados (realiza as transações) / Repositórios;

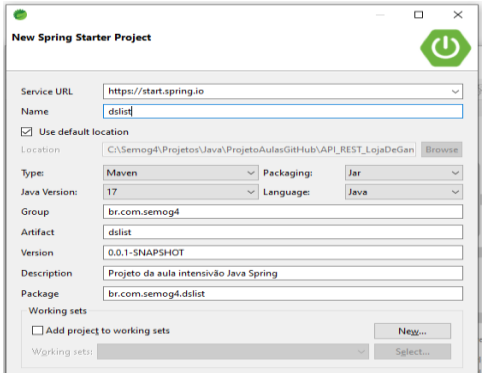
Métodos / Anotações:

1. BeanUtils.copyProperties(entityGame, this) : permite copiar todos os valores dos atributos de "entityGame" para this. Isso só será possível se os dois objetos possuírem os mesmos atributos/tipos. A classe de destino também deverá possuir todos os get e set referente aos atributos.
2. @Transaction(readOnly=true): comum ser definido nos métodos referentes as classes services. Utilizado para garantir que ocorra a transação desejada. Neste caso estamos informando ao banco de dados que a transação é somente de leitura e não de escrita, tornando o processo mais rápido.
3. @RequestBody: anotado como parâmetro no método service que irá tratar o body da requisição, ou seja, irá atribuir os valores indicados no body para a classe indicada no método (de x para). A classe ReplacementDTO deverá possuir os métodos get e set correspondente a cada atributo do body (tem que ter o mesmo nome).

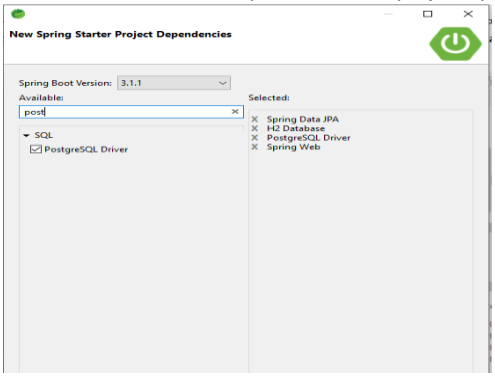
Configuração do projeto:

Criando o projeto utilizando a ferramenta Spring Tool Suíte - STS (pasta: API_REST_LojaDeGames)

- ✓ Selecione a opção New Spring Starter Project



- ✓ Selecione as dependências do projeto: pom.xml



✓ O projeto possui uma tabela de relacionamento que define a posição de cada jogo e sua categoria, onde esta tabela foi representada pela classe Belonging. Neste caso criamos a classe BelongingPK, já que no Repository só podemos ter um identificador como chave.

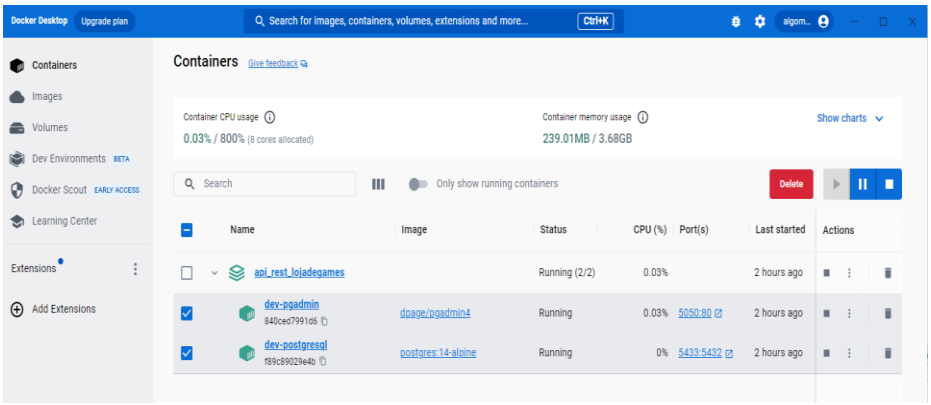
✓ Preparando a aplicação para rodar no banco de PostgreSql:

1. Podemos executar a aplicação de duas formas:
    a. Instalando o Postgresql e o cliente pgAdmin na máquina local junto com a aplicação.
    b. Instalando através do Docker o PostgreSql e o PgAdmin em containers e executando a aplicação local.

2. Iremos utilizar a opção om o Docker Compose:
    a. Baixar o script do docker-compose.yml (o script possui a configuração para criar os dois containeres: Servidor do Postgresql e do PgAdmin). O parâmetro volumes do arquivo docker-compose.yml cria uma pasta data, onde é armazenado o status e as informações atuais do banco, logo ao desligar a máquina, as informações não serão perdidas junto com o container. Ao levantar o container novamente, as informações do banco serão recuperadas através da pasta data.

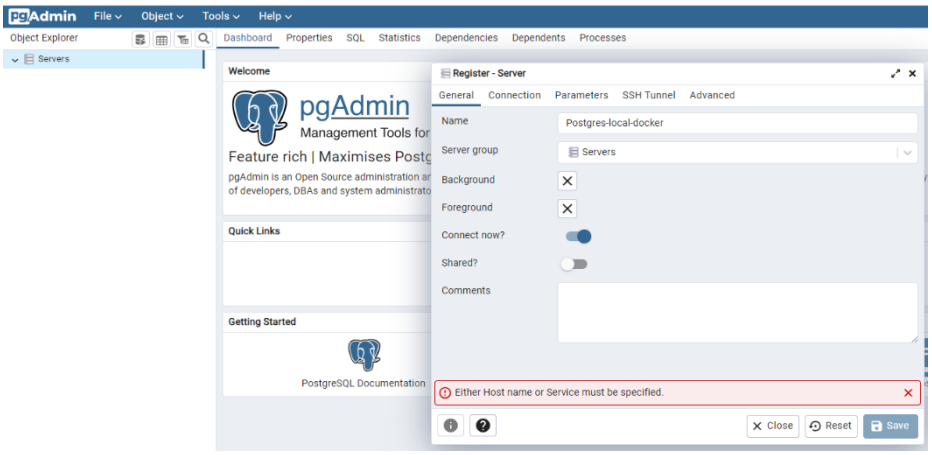
    b. Executar a aplicação Docker Desktop. É possível visualizar os dois containers inicializados: dev-pgadmin e dev-postgresql (estes nomes são os nomes definidos no docker-compose.yml):

        i. Como o arquivo docker-compose.yml já foi reconhecido pelo Docker, podemos inicializar os containers pela própria interface no botão Actions;
        ii. Na primeira vez foi necessário subir a aplicação docker e executar no powerShell como administrador o comando: docker-compose up -d. Executando assim o script docker-compose.yml

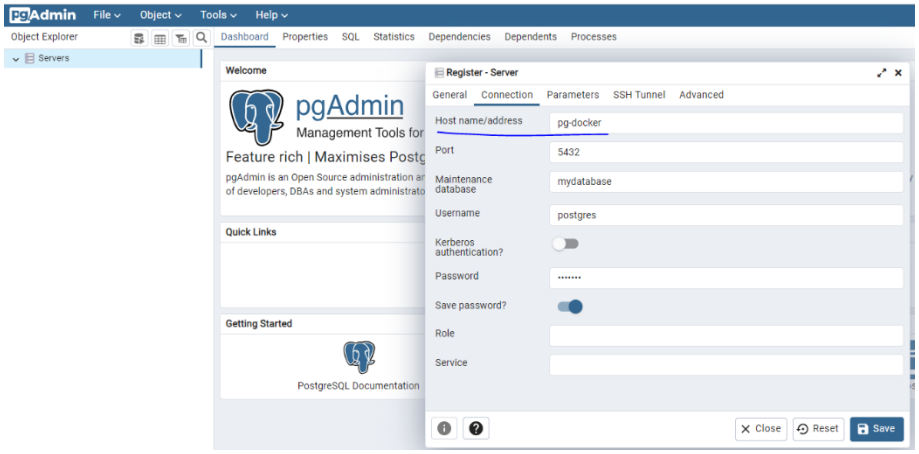


    c. Sem a interface do Docker podemos confirir se os containers estão executando através do comando: docker ps
    d. Executando o PgAdmin através do navegador: http://localhost:5050
    e. Logue com o usuário e senha do PgAdmin definido no script docker-compose.yml

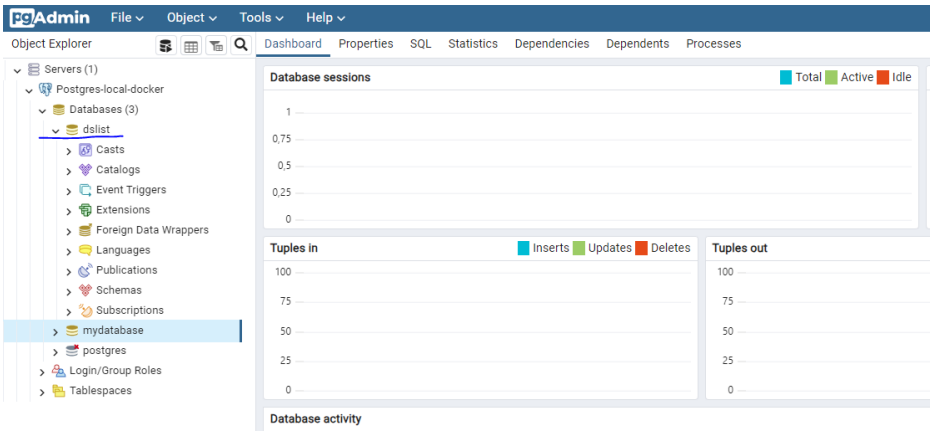
f. Configurando o Server local no Postgresql através do PgAdmin:



g. Através das informações que estão no docker-compose.yml configurar a Connection. O host name, é o nome do container do Postgresql que está no script.

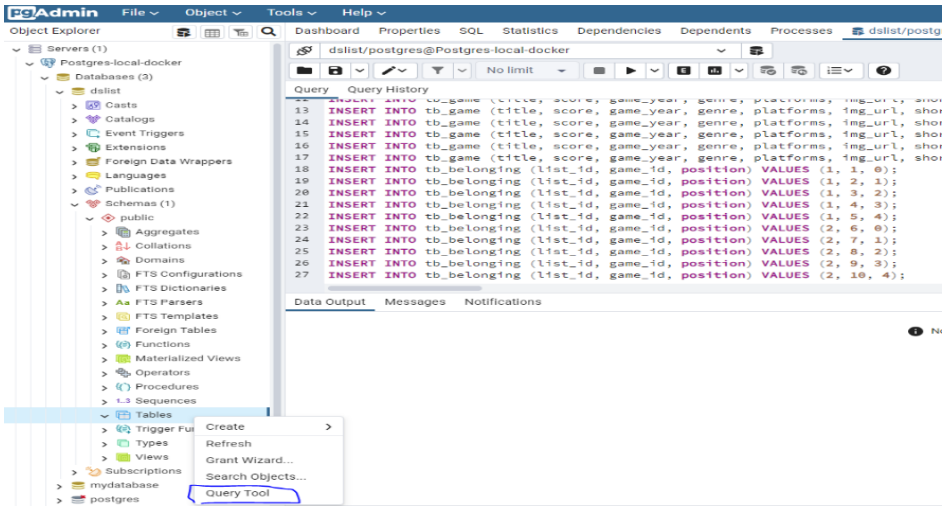


h. Criando o banco de dados dslist conforme abaixo:



i. Para popular o banco dslist no Postgresql, vamos configurar o arquivo application-hom.properties conforme as instruções no arquivo application-hom.properties. Ao inicializar a aplicação será gerado um script com o nome create.sql com todas as informações necessárias para rodar no banco de dados. Após a criação do arquivo create.sql as configurações realizadas no arquivo application-hom.properties devem ser comentadas.

j. Abra o arquivo create.sql e execute no banco dslist conforme abaixo:



ATENÇÃO: OPCIONAL NO TREINAMENTO!!

Os serviços de hospedagem de back end com banco de dados não estão mais oferecendo planos gratuitos. Assim, você pode apenas assistir essa parte a aula para conhecimento, para entender como funciona o processo de implantação. Neste momento, foque em deixar seu projeto caprichado no Github, com um bom Readme, conforme mostramos na aula.

Pré-requisitos

- Conta no Railway
- Conta no Github com mais de 90 dias
- Projeto Spring Boot salvo no seu Github
- Script SQL para criação e seed da base de dados
- Aplicativo de gestão de banco instalado (pgAdmin ou DBeaver)

Passos Railway

1. Prover um servidor de banco de dados

2. Criar a base de dados e seed

3. Criar uma aplicação Railway vinculada a um repositório Github

4. Configurar variáveis de ambiente

```
APP_PROFILE
DB_URL (Formato: jdbc:postgresql://host:porta/nomedabase)
DB_USERNAME
DB_PASSWORD
CORS_ORIGINS
```

5. Configurar o domínio público para a aplicação

6. Testar app no Postman

7. Testar a esteira de CI/CD

✓ Como instalar o Docker:

https://www.docker.com/products/docker-desktop/

https://docs.docker.com/desktop/install/windows-install/

https://youtu.be/trto4i0Olwg