Teste de Conceito Dev Jr - Pacto

LUCAS DE ALMEIDA BORGES - (62) 98309-7506

1. **Yoda: Como primeira parte de seu treinamento, capaz você deverá ser para subir a aplicação;**

Para subir a aplicação, inicialmente, identifiquei a ausência da biblioteca `org.json`, que causava um erro de execução. Resolvi o problema adicionando a seguinte dependência ao arquivo `pom.xml`:

```
<dependency>
    <groupId>org.json</groupId>
    <artifactId>json</artifactId>
    <version>20250107</version>
</dependency>
```

Além disso, o arquivo de configuração `application.properties` já continha os campos necessários para a configuração do banco de dados, mas foi necessário preencher com a URL, usuário e senha do banco que configurei localmente. Usei os seguintes valores:

```
spring.datasource.url=jdbc:postgresql://localhost:5432/pada
wan
spring.datasource.username=postgres
spring.datasource.password=lucas
```

Com essas alterações, a aplicação foi iniciada com sucesso no ambiente local, garantindo o funcionamento correto.

---

2. **Qual será o retorno se eu fizer um POST em http://localhost:9099/generic-post passando o seguinte body?**

```
{
   "planet": "Coruscant 985",
   "ranking": 42,
   "classe": "Jedi Knight"
}
```

Utilizando o Postman para realizar uma requisição POST com o body acima, o retorno será:

```
{
    "planetAbreviado": "Coruscant",
    "rankingMultiplicado": 37944,
    "classeAbreviado": "Jedi"
}
```

---

3. **Se mudarmos o campo “ranking” para 0, a aplicação dá um erro. Por quê?**

Após realizar o debug, percebi que na classe `GenericoService` é criada uma variável chamada `divisor`, que recebe o valor de `ranking` multiplicado por 2. Na linha seguinte, ao tentar definir o valor de `rankingMultiplicado`, a aplicação realiza o cálculo:

```
generico.getRanking() * 124 * 612 / divisor
```

Quando o valor de `ranking` é `0`, o cálculo resulta em uma divisão por zero, o que gera uma Exception, pois na matemática não é possível dividir um número por zero, e na programação isso causa uma exceção.

Essa exceção é capturada no método `process` da classe `GenericoController`, que retorna a seguinte resposta:

```
return new ResponseEntity("erro ao processar o generico", H
ttpStatus.INTERNAL_SERVER_ERROR);
```

Como resultado, o cliente recebe a mensagem **"erro ao processar o generico"** com o status HTTP **500 (Internal Server Error)**.

---

4. **Como podemos mudar a porta onde a aplicação sobe?**

No arquivo `application.properties`, estão definidas as propriedades de conexão com o banco de dados e da aplicação, como mostrado abaixo:

```
server.port=9099
spring.datasource.url=jdbc:postgresql://localhost:5432/pada
wan
spring.datasource.username=postgres
spring.datasource.password=lucas
spring.jpa.properties.hibernate.dialect=org.hibernate.diale
ct.PostgreSQLDialect
spring.jpa.hibernate.ddl-auto=update
```

Para alterar a porta onde a aplicação será executada, basta modificar o valor da propriedade:

```
server.port=9099
```

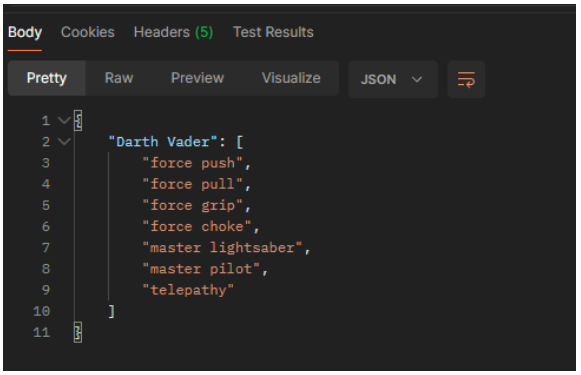
---

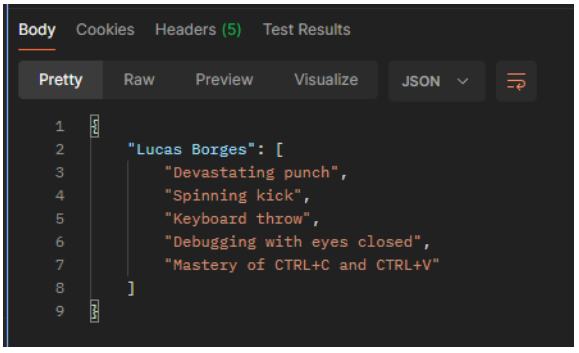
5. **Agora você vai ter que programar. Massa, hein?! Quando acessamos o endpoint http://localhost:9099/vader/skills, a aplicação retorna uma lista de habilidades do Darth Vader. Faça com que o sistema tenha um endpoint igual a esse, só que com seu nome e suas habilidades. Por exemplo, o meu seria http://localhost:9099/joaoalcides/skills. Detalhe muito importante: ao fazer isso, você não pode impactar o do Darth Vader. Envie para mim seu usuário do github, vou te adicionar nesse projeto para que você possa subir uma branch com suas alterações.**

Para criar um endpoint com as minhas skills sem mexer no que já existe para o Darth Vader, eu criei duas novas classes: `LucasController` e `LucasService`.

Na `LucasController`, implementei o endpoint `/lucas/skills` para retornar minhas habilidades, e na `LucasService`, coloquei uma lista com as skills. Dessa forma, tudo ficou separado e organizado, sem interferir no funcionamento do endpoint `/vader/skills`.

Segue a imagem com as skills que coloquei no meu endpoint.





---

6. **Mestre Windu explica que todo Jedi já foi um Padawan, então necessário é definir nome, status (padawan/jedi/mestre jedi) e um mentor para cada Jedi;**

Para atender à explicação do Mestre Windu, criei as seguintes classes:

JediEntity

- Utilizei as anotações do **JPA** para mapear a entidade ao banco.
- Atributos:
  - `id`: Identificador único.
  - `name`, `status`, `mentor` e `midichlorians`: Representam as colunas na tabela.

JediDTO

- Utilizei **Bean Validation** para garantir a integridade dos dados.
- Atributos:
  - `name`: Não pode ser nulo ou vazio e deve ter pelo menos 3 caracteres.
  - `status`: Deve ser "padawan", "jedi" ou "mestre jedi".

- `mentor`: Não pode ser nulo ou vazio e deve ter pelo menos 3 caracteres.
- `midichlorians`: Deve ser maior ou igual a 1 e não pode ser nulo.

JediService

- Métodos implementados:
  - **Adicionar Jedi**: Converte o `JediDTO` em `JediEntity` e salva no banco.
  - **Listar Jedis**: Retorna todos os Jedis cadastrados.
  - **Atualizar Jedi**: Atualiza os dados de um Jedi existente.
  - **Remover Jedi**: Exclui um Jedi pelo ID.

JediController

- `/jedi/add`: Valida os dados e adiciona ao banco. Em caso de erro, retorna mensagens detalhadas.
- `/jedi/list`: Retorna a lista de todos os Jedis cadastrados.
- `/jedi/{id}`: Permite buscar, atualizar ou deletar um Jedi específico.

Dependências Utilizadas

Adicionadas ao arquivo `pom.xml` para suporte às validações:

```
xml
CopiarEditar
<dependency>
    <groupId>org.hibernate.validator</groupId>
    <artifactId>hibernate-validator</artifactId>
    <version>6.1.7.Final</version>
</dependency>

<dependency>
    <groupId>jakarta.annotation</groupId>
    <artifactId>jakarta.annotation-api</artifactId>
    <version>1.3.5</version>
</dependency>

<dependency>
```

```
    <groupId>org.glassfish</groupId>
    <artifactId>jakarta.el</artifactId>
    <version>3.0.3</version>
</dependency>
```

---

7. **Qui-Gon** ao fazer contato, estando no mundo espiritual da Força, nos diz que é possível mensurar a quantidade de midichlorians em cada Jedi, por isso, você deverá informar isso lá;

Para atender a esse requisito, alterei as seguintes classes:

- **JediEntity**:
  - Adicionei o atributo `midichlorians` na entidade para representá-lo como uma coluna no banco de dados.
  - Usei a anotação `@Column(nullable = false)` para garantir que esse valor seja obrigatório no banco.
- **JediDTO**:
  - Adicionei o atributo `midichlorians` e apliquei validações com **Bean Validation**:
    - `@NotNull`: Garante que o valor seja obrigatório.
    - `@Min(1)`: Garante que o valor seja maior ou igual a 1.
- **JediService**:
  - Adaptei o método de cadastro para armazenar o valor de `midichlorians` no banco ao criar um Jedi.
  - O método de listagem foi ajustado para incluir o atributo no retorno.
- **JediController**:
  - Atualizei o endpoint `/jedi/add`:
    - Valida o valor de `midichlorians` ao receber os dados do cliente.
    - Retorna mensagens de erro detalhadas em caso de validação falhar.
  - Atualizei o endpoint `/jedi/list`:

- Agora exibe a quantidade de `midichlorians` cadastrada para cada Jedi.

---

8. **Anakin Skywalker antes de se tornar o Darth Vader, deixou para você a última missão, após isso, estará apto a se tornar um Jedi. Essa missão é a de criar, com Native SQL as seguintes consultas, com seus respectivos endpoints:**

   a. **Listar todos os mestres Jedis e seus aprendizes;**

   Para retornar todos os mestres Jedi e seus aprendizes, implementei um SQL que seleciona o mentor de quem possui o status **"mestre jedi"**. Utilizei um **JOIN** entre as colunas `name` (do mestre) e `mentor` (do aprendiz) para relacionar os mestres com seus respectivos aprendizes. A ordenação foi feita pelo nome do mestre para facilitar a leitura dos resultados.

```
SELECT m.name AS mentor, p.name AS apprentice
            FROM jedis AS m
            JOIN jedis AS p
            ON m.name = p.mentor
            WHERE m.status = 'mestre jedi'
            ORDER BY m.name
```

   Após criar a consulta SQL nativa, implementei as funções necessárias nas seguintes camadas:

   - **Repository:** Criei o método com a consulta SQL usando a anotação `@Query`.
   - **Service:** Adicionei a lógica para processar e retornar os resultados formatados.
   - **Controller:** Criei o endpoint responsável por expor os dados da consulta para a API.

   b. Listar todos Jedis cujo midichlorians sejam acima de 9000;

Para listar todos os Jedis cujo número de **midichlorians** seja acima de 9000, segui os mesmos passos descritos anteriormente. Porém com o SQL a seguir.

```
SELECT * FROM jedis
WHERE midichlorians > 9000
```

c. Listar por categoria, quantos são os Jedis;

Para listar todos os Jedis cujo número de **midichlorians** seja acima de 9000, segui os mesmos passos descritos anteriormente. Porém com o SQL a seguir."SELECT status, COUNT(*) AS total FROM jedis GROUP BY status"

```
SELECT status, COUNT(*) AS total
FROM jedis GROUP BY status
```