

Teste de conceito para desenvolvedores juniors – Analista de documentação

Candidato: Henrique Barbosa de Lima Dias

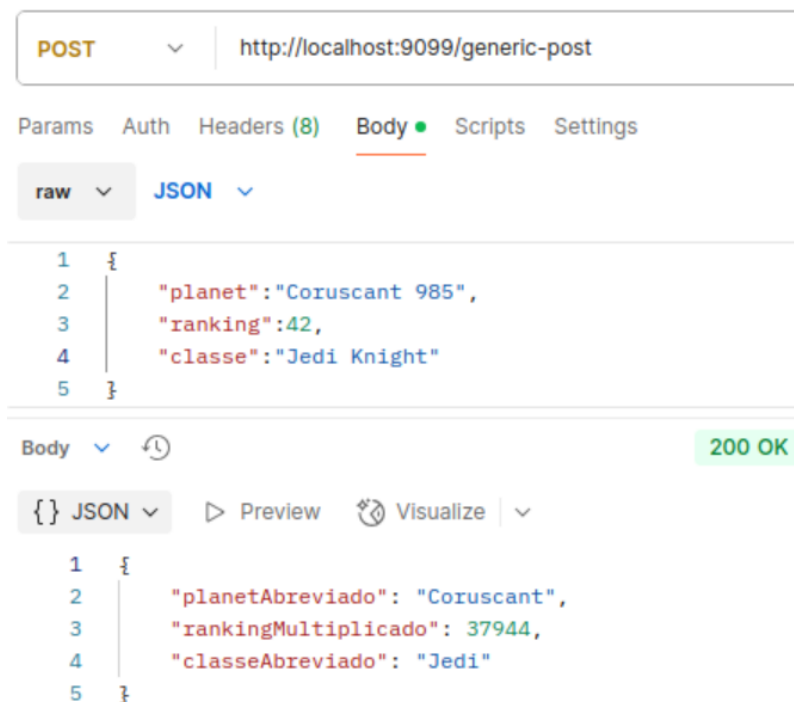
1. Subindo a aplicação

Durante o processo de subir a aplicação, identifiquei alguns problemas que foram resolvidos da seguinte forma:

- Configuração do *JAVA_HOME*:
Foi necessário ajustar a variável de ambiente *JAVA_HOME* para apontar corretamente para a instalação do Java.
- Ausência do *PostgreSQL*:
Inicialmente, o *PostgreSQL* não estava instalado, o que resultava em erros indicando que não havia nada na porta correspondente. Esse problema era decorrente de dependências desatualizadas no *Ubuntu*. Após identificar o problema, o *PostgreSQL* foi instalado corretamente.
- Credenciais do usuário *postgres*:
Ao tentar autenticar no *PostgreSQL*, a credencial do usuário *postgres* não estava funcionando. Resolvi o problema alterando a senha, que estava vazia, para *postgres*.
- Ausência da base de dados *padawan*:
A aplicação apontou a ausência da base de dados necessária chamada *padawan*. Para resolver, foi necessário criá-la manualmente.

Após resolver esses problemas, a aplicação foi iniciada com sucesso.

2. Utilizando o *Postman*, ao fazer um *POST* em <http://localhost:9099/generic-post> passando o *body* especificado, foi retornado: "planetAbreviado": "Coruscant", "rankingMultiplicado":37944, "classeAbreviado":"Jedi". Isso pode ser visualizado na imagem a seguir:



3. Ao fazer uma mudança de *ranking* para 0, a aplicação apresenta erro porque nela há um processo que realiza uma divisão utilizando o resultado de (*ranking* * 2) como divisor. A divisão por zero resulta em uma indeterminação matemática. O cálculo presente no processo é mostrado na imagem a seguir:

```
@Service
public class GenericoService {

    public ProcessDTO process(GenericoDTO generico){
        ProcessDTO processDTO = new ProcessDTO();
        processDTO.setClasseAbreviado(generico.getClasse().split(" ")[0]);
        processDTO.setPlanetAbreviado(generico.getPlanet().split(" ")[0]);
        Integer divisor = generico.getRanking() * 2;
        processDTO.setRankingMultiplicado(generico.getRanking()*124*612/divisor);
        return processDTO;
    }
}
```

4. É possível mudar a porta onde a aplicação sobe alterando o "server.port" no arquivo "application.properties". Nesse caso, fiz a alteração da porta (9099) para a porta (1089) e verifiquei se a aplicação subiu na porta alterada através do comando "netstat -lnpt" no *bash*. Isso pode ser visualizado nas imagens a seguir:

```
src > main > resources > application.properties
```

```
1 server.port=1089
2 spring.datasource.url=@spring.datasource.url@
3 spring.datasource.username=@spring.datasource.username@
4 spring.datasource.password=@spring.datasource.password@
5 spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDialect
6 spring.jpa.hibernate.ddl-auto=update
7
```

Active Internet connections (only servers)

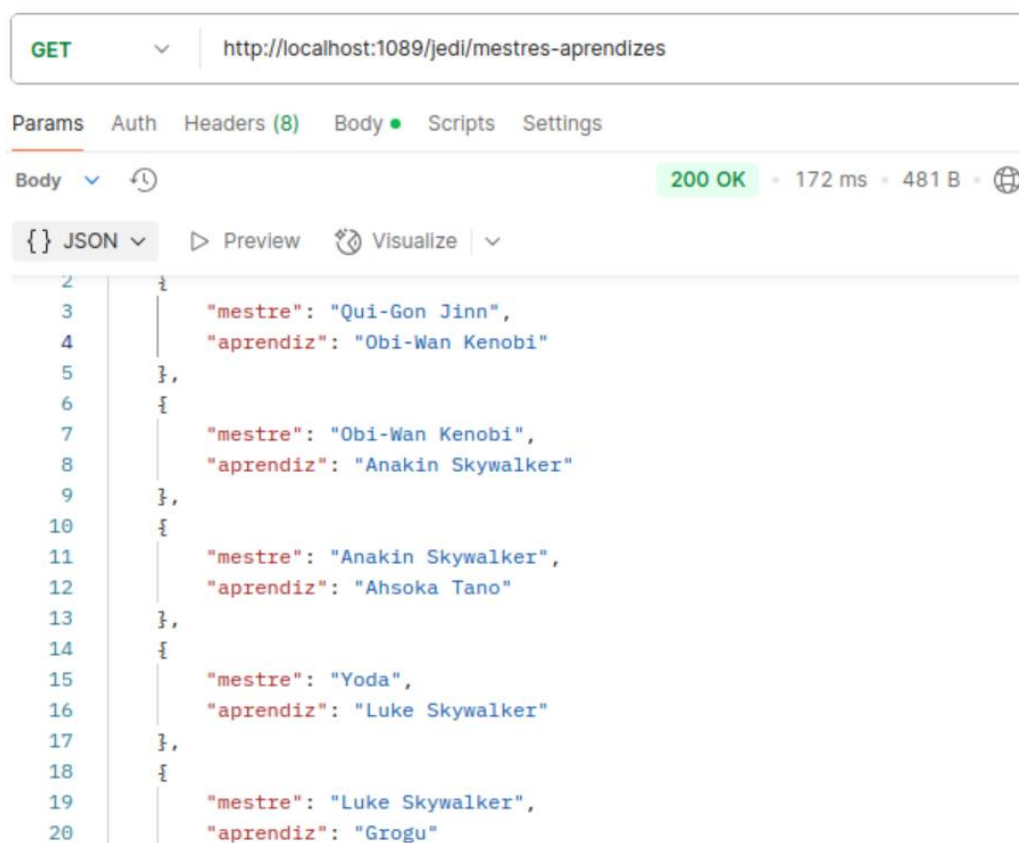
Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State
PID/Program name					
tcp	0	0	127.0.0.54:53	0.0.0.0:*	LISTEN
489/systemd-resolve					
tcp	0	0	127.0.0.1:5432	0.0.0.0:*	LISTEN
11894/postgres					
tcp	0	0	127.0.0.53:53	0.0.0.0:*	LISTEN
489/systemd-resolve					
tcp	0	0	127.0.0.1:631	0.0.0.0:*	LISTEN
1097/cupsd					
tcp6	0	0	:::1089	:::*	LISTEN
48676/java					
tcp6	0	0	:::1:631	:::*	LISTEN
1097/cupsd					
tcp6	0	0	:::15611	:::*	LISTEN
47653/postman --no-					

5. Criei um novo *endpoint* com meu nome e minhas *skills* sem alterar o *endpoint* do Darth Vader. A imagem a seguir mostra o resultado:

The screenshot shows a REST client interface with a GET request to `http://localhost:1089/henrique/skills`. The response is in JSON format, showing a nested object with a key "Henrique" containing an array of skills.

```
{
  "Henrique": [
    "banco de dados",
    "python",
    "linux",
    "c++",
    "ingles"
  ]
}
```

6. Criei uma classe intitulada “Jedi”. Nela há informações sobre Jedis com atributos nome, status (“padawan” / “jedi” / “mestre jedi”) e mentor, com métodos *get* e *set* para cada atributo. Pode ser visualizado na aplicação.
 7. Adicionei à classe “Jedi” o atributo “midichlorians” (indicador de força), que por sua vez também possui métodos *get* e *set*. Pode ser visualizado na aplicação.
 8. As consultas foram implementadas utilizando *Native SQL*, e os *endpoints* correspondentes foram criados para atender às necessidades requisitadas.
1. Listar todos os mestres e seus aprendizes:
 - Consulta criada para buscar mestres e os respectivos aprendizes com base no relacionamento *mentor_id*.
 - *Endpoint* criado: GET /jedi/mestres-aprendizes.

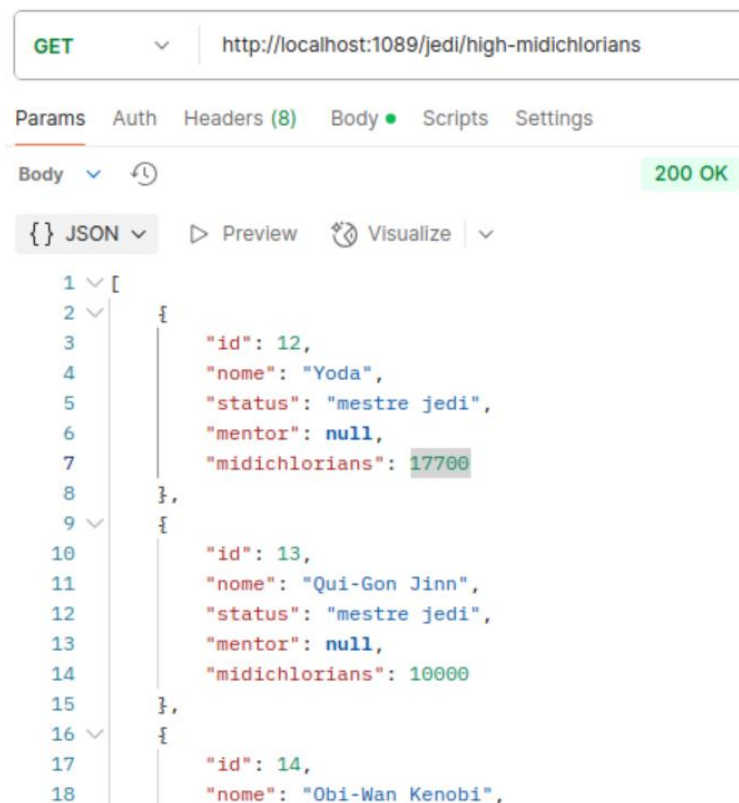


```
GET http://localhost:1089/jedi/mestres-aprendizes

Params Auth Headers (8) Body ● Scripts Settings
Body 200 OK • 172 ms • 481 B •
{} JSON Preview Visualize

[
  {
    "mestre": "Qui-Gon Jinn",
    "aprendiz": "Obi-Wan Kenobi"
  },
  {
    "mestre": "Obi-Wan Kenobi",
    "aprendiz": "Anakin Skywalker"
  },
  {
    "mestre": "Anakin Skywalker",
    "aprendiz": "Ahsoka Tano"
  },
  {
    "mestre": "Yoda",
    "aprendiz": "Luke Skywalker"
  },
  {
    "mestre": "Luke Skywalker",
    "aprendiz": "Grogu"
  }
]
```

2. Listar todos Jedis cujo midichlorians sejam acima de 9000:
- Endpoint criado: GET /jedi/high-midichlorians.

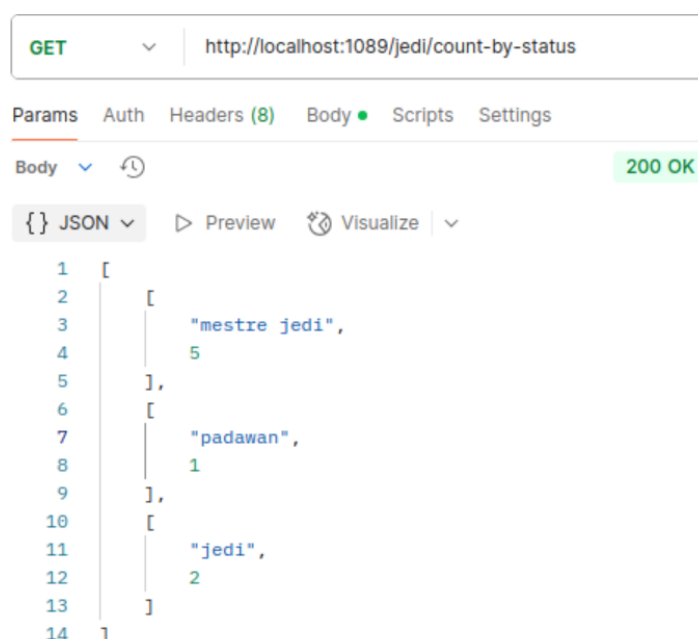


The screenshot shows a REST client interface with a GET request to `http://localhost:1089/jedi/high-midichlorians`. The response status is `200 OK`. The response body is a JSON array containing three Jedi objects. The first object is Yoda with 17700 midichlorians, the second is Qui-Gon Jinn with 10000 midichlorians, and the third is Obi-Wan Kenobi.

```
1 [
2   {
3     "id": 12,
4     "nome": "Yoda",
5     "status": "mestre jedi",
6     "mentor": null,
7     "midichlorians": 17700
8   },
9   {
10    "id": 13,
11    "nome": "Qui-Gon Jinn",
12    "status": "mestre jedi",
13    "mentor": null,
14    "midichlorians": 10000
15  },
16  {
17    "id": 14,
18    "nome": "Obi-Wan Kenobi",
```

3. Listar por categoria, quantos são os Jedis:

- Consulta criada para contar Jedis agrupados por categoria (status).
- Endpoint criado: GET /jedi/count-by-status.



The screenshot shows a REST client interface with a GET request to `http://localhost:1089/jedi/count-by-status`. The response status is `200 OK`. The response body is a JSON array containing three objects, each representing a status and its count: "mestre jedi" with a count of 5, "padawan" with a count of 1, and "jedi" with a count of 2.

```
1 [
2   [
3     "mestre jedi",
4     5
5   ],
6   [
7     "padawan",
8     1
9   ],
10  [
11    "jedi",
12    2
13  ]
14 ]
```