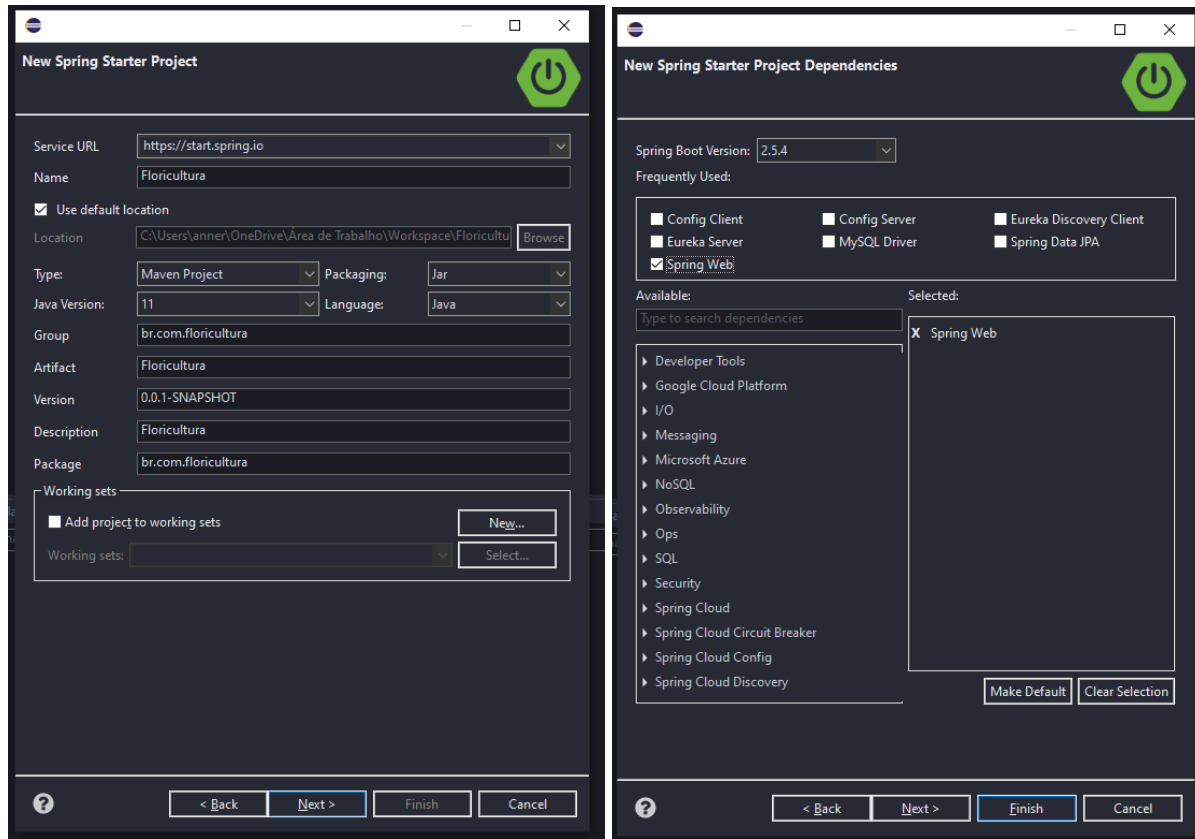


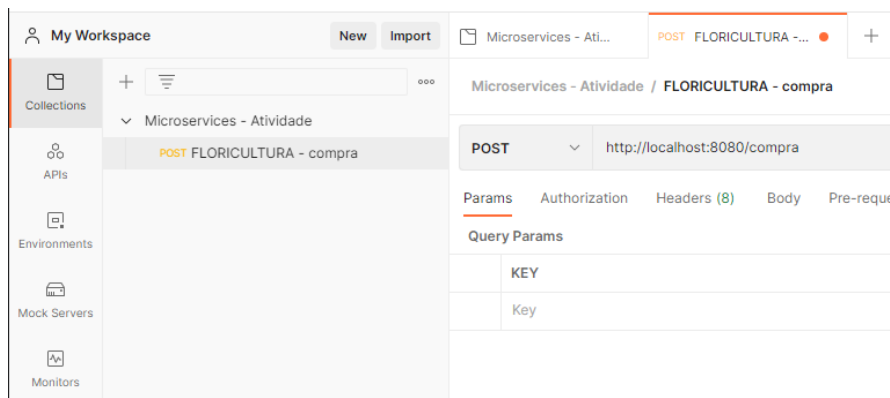
Spring Cloud Gateway

- Criação do micro serviço Floricultura usando Spring Boot
- Iniciar criando um novo projeto spring e configurando o nome, grupo, artefato, descrição e pacote e adicionar as dependências:



- Primeiro Micro Serviço: **Floricultura**

A primeira a se fazer é abrir o Postman, que é uma aplicação que nos permite montar requisições sem precisarmos de uma interface gráfica. Vamos criar uma Collection e adicionar os requests dentro dessa collection. Depois vamos criar um Request chamado de “FLORICULTURA - compra” e direcionar esse request ao endereço “<http://localhost:8080/compra>”. Essa requisição responderá a nossa classe Controller, em nosso código.



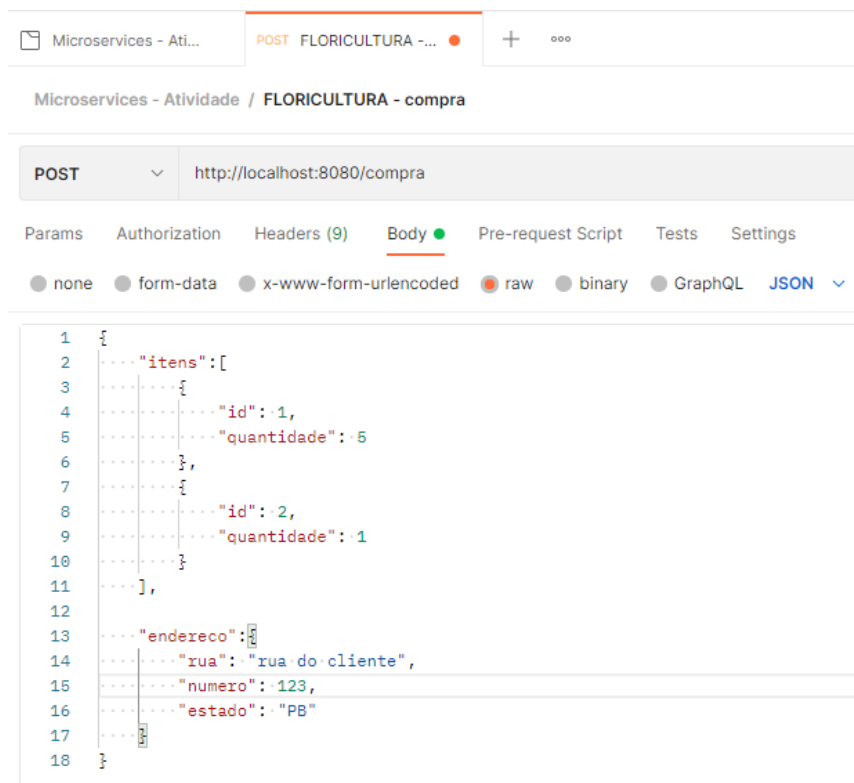
```
J FloriculturaApplication.java J CompraController.java X
1 package br.com.floricultura.controller;
2
3 import org.springframework.web.bind.annotation.RequestBody;
4
5
6
7
8
9
10 @RestController
11 @RequestMapping("/compra")
12 public class CompraController {
13
14     @RequestMapping(method = RequestMethod.POST)
15     public void realizaCompra(@RequestBody CompraDTO compra) {
16
17     }
18 }
19
```

Essa classe vem com a anotação de `@RestController` é para marcar que o controlador está fornecendo serviços REST com o tipo de resposta JSON. A anotação `@RequestMapping("/compra")` que está enviando o request para o nosso POST na porta 8080, no software postman.

```
Floricultura - FloriculturaApplication [Spring Boot App] C:\Program Files\Java\jdk13.0.8-win_x64\bin\javaw.exe (20 de set de 2021 14:51:20)
=====
:: Spring Boot ::
(v2.5.4)

2021-09-20 14:51:21.938 INFO 10708 --- [main] b.c.f.FloriculturaApplication : Starting FloriculturaApplication using Java 13.0.8 on D
2021-09-20 14:51:21.941 INFO 10708 --- [main] b.c.f.FloriculturaApplication : No active profile set, falling back to default profiles
2021-09-20 14:51:22.874 INFO 10708 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port(s): 8080 (http)
2021-09-20 14:51:22.883 INFO 10708 --- [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2021-09-20 14:51:22.883 INFO 10708 --- [main] org.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.52]
2021-09-20 14:51:22.956 INFO 10708 --- [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2021-09-20 14:51:22.956 INFO 10708 --- [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in
2021-09-20 14:51:23.284 INFO 10708 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port(s): 8080 (http) with context pat
2021-09-20 14:51:23.291 INFO 10708 --- [main] b.c.f.FloriculturaApplication : Started FloriculturaApplication in 1.684 seconds (JVM r
```

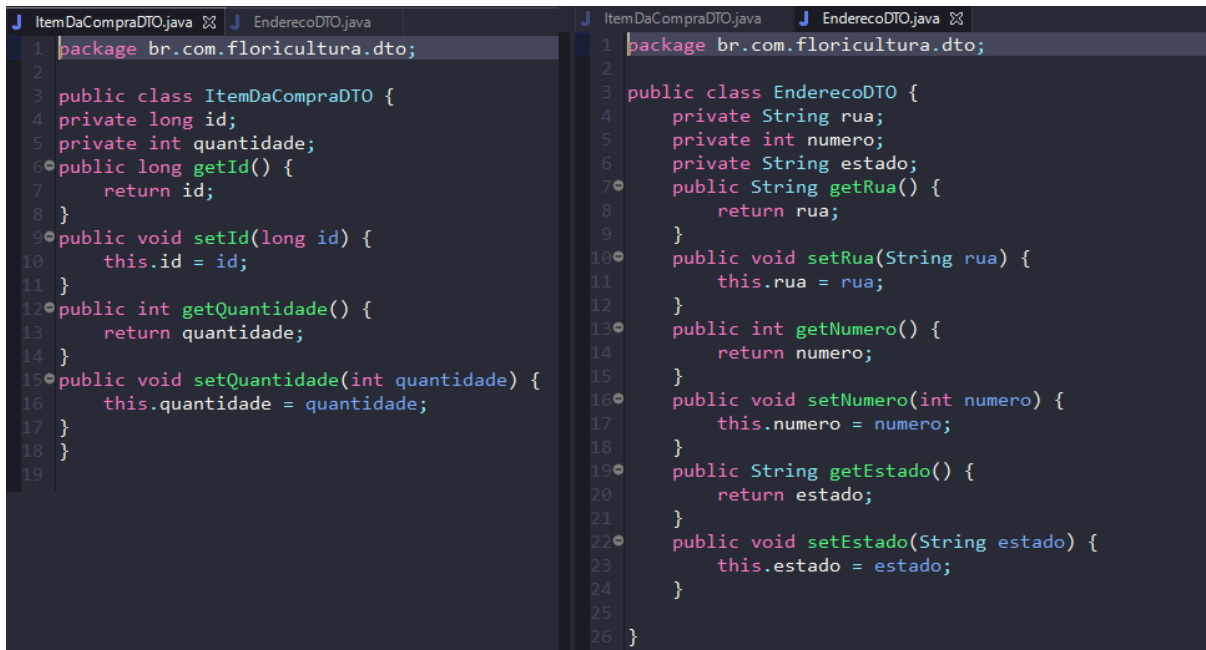
Iremos preencher a requisição do postman com os dados da compra, para que possa ser enviado a resposta de volta para a API. Colocamos Itens (id e quantidade) e endereço (rua, número e estado do cliente).



Após preencher o Postman, iremos criar a classe `CompraDTO`, usando o padrão Objeto de Transferência de Dados, que agrega e encapsula dados para transferência, no caso os dados serão os atributos **itens** e **endereço**.

```
FloriculturaApplication.java  CompraController.java  CompraDTO.java
1 package br.com.floricultura.dto;
2
3 import java.util.List;
4
5 public class CompraDTO {
6     private List<ItemDaCompraDTO> itens;
7     private EnderecoDTO endereco;
8     public List<ItemDaCompraDTO> getItens() {
9         return itens;
10    }
11    public void setItens(List<ItemDaCompraDTO> itens) {
12        this.itens = itens;
13    }
14    public EnderecoDTO getEndereco() {
15        return endereco;
16    }
17    public void setEndereco(EnderecoDTO endereco) {
18        this.endereco = endereco;
19    }
20 }
21
```

Após criar o CompraDTO, precisaremos criar as classes ItemDaCompraDTO e EnderecoDTO, que vão comportar os atributos de cada um.

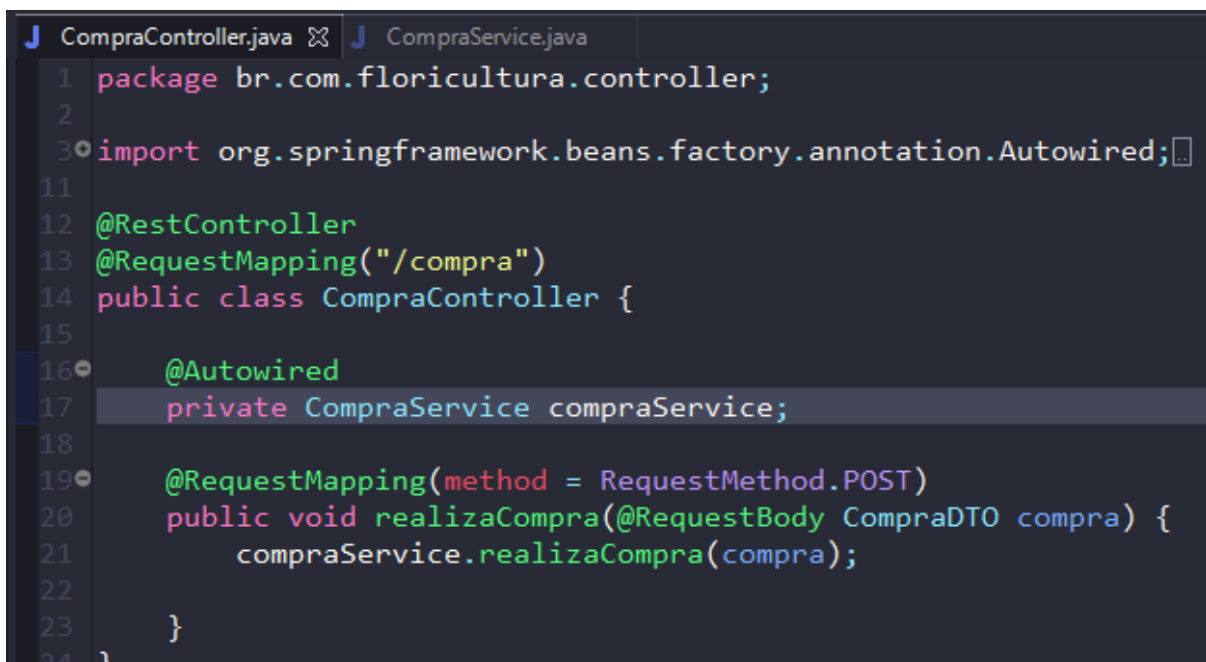


```
ItemDaCompraDTO.java
1 package br.com.floricultura.dto;
2
3 public class ItemDaCompraDTO {
4     private long id;
5     private int quantidade;
6     public long getId() {
7         return id;
8     }
9     public void setId(long id) {
10        this.id = id;
11    }
12    public int getQuantidade() {
13        return quantidade;
14    }
15    public void setQuantidade(int quantidade) {
16        this.quantidade = quantidade;
17    }
18 }
19

EnderecoDTO.java
1 package br.com.floricultura.dto;
2
3 public class EnderecoDTO {
4     private String rua;
5     private int numero;
6     private String estado;
7     public String getRua() {
8         return rua;
9     }
10    public void setRua(String rua) {
11        this.rua = rua;
12    }
13    public int getNumero() {
14        return numero;
15    }
16    public void setNumero(int numero) {
17        this.numero = numero;
18    }
19    public String getEstado() {
20        return estado;
21    }
22    public void setEstado(String estado) {
23        this.estado = estado;
24    }
25 }
26 }
```

Após essa implementação, o nosso primeiro microservice está pronto e recebendo os dados da compra, agora iremos montar o segundo micro serviço para processar essa compra.

Nós já temos a requisição pronta, feita pelo usuário, com os produtos no Postman, Já enviamos e recebemos essa requisição. Agora nós precisamos processar essa compra, que normalmente é feita por um serviço. Iremos criar a classe "CompraService" e injetá-la no CompraController com @Autowired.



```
CompraController.java
1 package br.com.floricultura.controller;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4
5 @RestController
6 @RequestMapping("/compra")
7 public class CompraController {
8
9     @Autowired
10    private CompraService compraService;
11
12    @RequestMapping(method = RequestMethod.POST)
13    public void realizaCompra(@RequestBody CompraDTO compra) {
14        compraService.realizaCompra(compra);
15    }
16 }
17

CompraService.java
1 package br.com.floricultura.service;
2
3 import org.springframework.stereotype.Service;
4
5 @Service
6 public class CompraService {
7
8     public void realizaCompra(CompraDTO compra) {
9         // Implementação do serviço
10    }
11 }
```

```

1 package br.com.floricultura.service;
2
3 import org.springframework.http.HttpMethod;
4 import org.springframework.http.ResponseEntity;
5 import org.springframework.web.client.RestTemplate;
6
7 import br.com.floricultura.dto.CompraDTO;
8 import br.com.floricultura.dto.InfoFornecedorDTO;
9
10 public class CompraService {
11
12     public void realizaCompra(CompraDTO compra) {
13         RestTemplate client = new RestTemplate();
14         ResponseEntity<InfoFornecedorDTO> exchange =
15             client.exchange("http://localhost:8081/info/"+compra.getEndereco().getEstado(),
16                             HttpMethod.GET, null, InfoFornecedorDTO.class);
17
18         System.out.println(exchange.getBody().getEndereco());
19     }
20 }
21
22 }

```

Esse compraService será usado pelo CompraController, e vai delegar o processamento dessa requisição, então ele vai delegar para o método compraService.realizaCompra(compra), passando os dados que estão no DTO.

No CompraService, vamos processar a compra em si. Esse processamento será mais longo, mas a primeira implementação que precisamos fazer é a comunicação entre o **micro serviço da loja e o do fornecedor**, então nós precisamos, além dessas informações que nós já temos, que são a lista de item e o endereço de destino, precisamos do endereço de origem, que é o fornecedor.

- Segundo micro serviço: Fornecedor da floricultura.

Vamos criar um novo projeto Spring Starter Web, que será o nosso fornecedor. Preenchemos as configurações iniciais do projeto e adicionamos as dependências: Precisamos do "Spring Web Starter", que já colocamos na loja, só que, nesse caso, vamos fazer a comunicação com o banco de dados, então precisamos do "Spring Data JPA" e do "MySQL Driver", que é o mesmo usado para o MariaDB.

A implementação do fornecedor começa pela configuração dele, então vamos criar uma classe Controller com anotações `@RestController` e `@RequestMapping("/info")`, direcionando a requisição `"/info"`, para o `InfoController`. Temos que configurar também a porta do servidor para `"localhost:8081"` no `"application.yml"`.

No InfoController, vamos criar um método que vai retornar as informações. Esse método vai se chamar `getInfoPorEstado(String estado)`, esse é aquele endpoint `"/estado"`. Era `"/info/estado"`, estado é desse `getInfoPorEstado`, `@RequestMapping("/estado")`.

E é disso que precisamos para a requisição `"localhost:8081/info/estado"`. Mas precisamos de alguém que vá ao banco de dados e pegue as informações do fornecedor. Vamos criar também um serviço chamado de `private InfoService infoService`, e vamos injetar ele com `@Autowired`.

```
Y application.yml  J InfoController.java  X
1 package br.com.floricultura.fornecedor.controller;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4
5
6 @RestController
7 @RequestMapping("/info")
8 public class InfoController {
9
10     @Autowired
11     private InfoService infoService;
12
13     @RequestMapping("/{estado}")
14     public InfoFornecedor getInfoPorEstado(String estado) {
15         return infoService.getInfoPorEstado(estado);
16     }
17 }
18
19 }
```

Vamos criar esse serviço dentro do pacote `"br.com.floricultura.service"`. Anotar ele como `@Service` e criar um método dentro dele. Então quando o fornecedor receber a requisição `getInfoPorEstado`, ele vai delegar para o serviço e vai passar `infoService.getInfoPorEstado(estado)`, passando o estado.

```
Y application.yml  J InfoController.java  J InfoService.java  X
1 package br.com.floricultura.fornecedor.service;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4
5
6 @Service
7 public class InfoService {
8
9     @Autowired
10     private InfoRepository infoRepository;
11
12     public InfoFornecedor getInfoPorEstado(String estado) {
13         return infoRepository.findByEstado(estado);
14     }
15 }
16
17 }
```

O InfoController não tem responsabilidade de ir no banco de dados, ele vai delegar isso para a camada de serviço. Então o que vamos fazer nesse método de serviço do getInfoPorEstado é acessar o banco de dados. Quem vai fazer isso será um repositório, então é uma outra classe que irá fazer isso.

```
Y application.yml  J InfoController.java  J InfoService.java  J InfoRepository.java
1 package br.com.floricultura.fornecedor.service;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4
5
6
7
8 @Service
9 public class InfoService {
10
11     @Autowired
12     private InfoRepository infoRepository;
13
14     public InfoFornecedor getInfoPorEstado(String estado) {
15
16         return infoRepository.findByEstado(estado);
17
18     }
19
20 }
```

Vamos injetar `private InfoRepository infoRepository;` e vamos implementar esse repositório, que é uma interface. Anota ele como `@Repository`, `extends CrudRepository<>` para ganhar métodos do `CrudRepository`. Vai ter um CRUD de `<InfoFornecedor>`, de informações do fornecedor, que tem como id um atributo `<InfoFornecedor, Long>`.

```
Y application.yml  J InfoController.java  J InfoService.java  J InfoRepository.java
1 package br.com.floricultura.fornecedor.service;
2
3 import org.springframework.data.repository.CrudRepository;
4
5
6
7
8 @Repository
9 public interface InfoRepository extends CrudRepository<InfoFornecedor, Long>{
10
11     InfoFornecedor findByEstado(String estado);
12
13 }
14
```

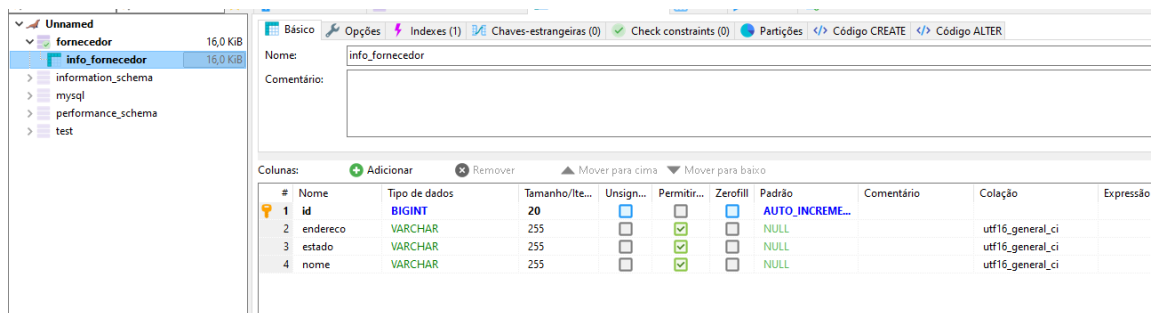

Vamos criar a classe InfoFornecedor dentro do pacote "br.com.aura.floricultura.fornecedor.model". Vamos fazer a anotação @Entity e adicionar os atributos. Serão eles: **private Long id**; vamos colocar um @Id e um @GeneratedValue(strategy = GenerationType.IDENTITY), para que esse valor seja gerado automaticamente. Vamos adicionar também **private String nome**, **private String estado**, **private String endereco** e criar os getters e setters. A classe InfoFornecedor, será usada pelo hibernate para fazer o mapeamento de objeto relacional. Ela já contém o que precisamos guardar no banco, que é o id, a coluna para o nome, estado e endereço.

```
J InfoFornecedor.java
1 package br.com.floricultura.fornecedor.model;
2
3 import javax.persistence.Entity;
4
5 @Entity
6 public class InfoFornecedor {
7     @Id
8     @GeneratedValue(strategy = GenerationType.IDENTITY)
9
10    private Long id;
11    private String nome;
12    private String estado;
13    private String endereco;
14
15    public Long getId() {
16        return id;
17    }
18    public void setId(Long id) {
19        this.id = id;
20    }
21    public String getNome() {
22        return nome;
23    }
24    public void setNome(String nome) {
25        this.nome = nome;
26    }
27    public String getEstado() {
28        return estado;
29    }
30    public void setEstado(String estado) {
31        this.estado = estado;
32    }
33    public String getEndereco() {
34        return endereco;
35    }
36    public void setEndereco(String endereco) {
37        this.endereco = endereco;
38    }
39
40 }
41
42
43 }
```

Para conectar e configurar o banco de dados, vamos no "application.yml" e vamos colocar a configuração com o banco. Para isso, temos que pegar essas propriedades e colocar dentro do nosso YML.

```
1 server:
2   port: 8081
3
4 spring:
5   datasource:
6     password: 12345
7     username: root
8     url: jdbc:mysql://localhost:3306/fornecedor
9   jpa:
10     hibernate:
11       ddl-auto: create
12
```

Agora vamos criar o banco de dados para que essa informação seja armazenada.



Criar no Postman um GET para pegar as informações do fornecedor.



- Adicionando o Service Discovery com Eureka

Iniciamos um novo projeto de spring, adicionando a dependência Eureka Server

The screenshot shows the 'New Spring Starter Project' wizard in an IDE. The 'New Spring Starter Project Dependencies' tab is active, displaying a list of frequently used dependencies. The 'Available' list on the right includes various Spring Cloud dependencies, and 'Eureka Server' is selected. The 'Selected' list on the right shows 'Eureka Server' as the chosen dependency. The 'Frequently Used' section includes checkboxes for 'Config Client', 'Config Server', 'Eureka Discovery Client', 'Eureka Server', 'MySQL Driver', and 'Spring Data JPA'. The 'Available' list includes 'Developer Tools', 'Google Cloud Platform', 'I/O', 'Messaging', 'Microsoft Azure', 'NoSQL', 'Observability', 'Ops', 'SQL', 'Security', 'Spring Cloud', 'Spring Cloud Circuit Breaker', 'Spring Cloud Config', and 'Spring Cloud Discovery'. The 'Selected' list shows 'Eureka Server' as the chosen dependency. The 'Frequently Used' section includes checkboxes for 'Config Client', 'Config Server', 'Eureka Discovery Client', 'Eureka Server', 'MySQL Driver', and 'Spring Data JPA'. The 'Available' list includes 'Developer Tools', 'Google Cloud Platform', 'I/O', 'Messaging', 'Microsoft Azure', 'NoSQL', 'Observability', 'Ops', 'SQL', 'Security', 'Spring Cloud', 'Spring Cloud Circuit Breaker', 'Spring Cloud Config', and 'Spring Cloud Discovery'. The 'Selected' list shows 'Eureka Server' as the chosen dependency.

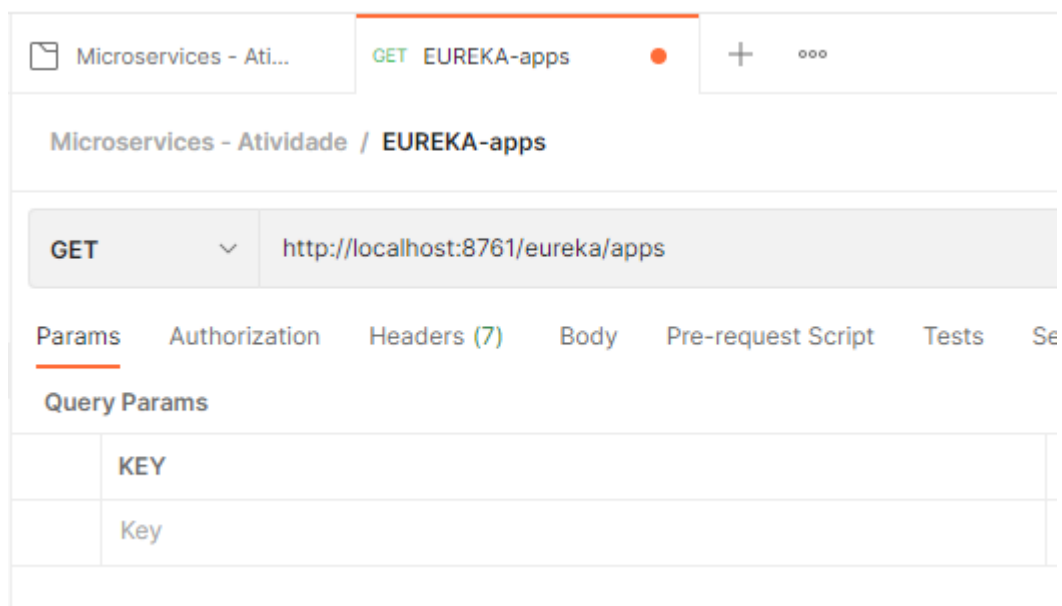
Não há muita implementação nessa parte, é apenas configuração do YML, temos que mudar a porta do servidor para: 8761, que é a padrão do eureka.

```
Y application.yml
1
2 server:
3   port: 8761
4
5 eureka:
6   client:
7     fetch-registry: false
8     register-with-eureka: false
9
```

E lembrar de colocar na classe application a anotação `@EnableEurekaServer`

```
Y application.yml  J EurekaServerApplication.java  ⌕
1 package br.com.fornecedor.eureka;
2
3 import org.springframework.boot.SpringApplication;
4
5
6
7 @SpringBootApplication
8 @EnableEurekaServer
9 public class EurekaServerApplication {
10
11     public static void main(String[] args) {
12         SpringApplication.run(EurekaServerApplication.class, args);
13     }
14
15 }
16
```

Vamos criar mais uma requisição no Postman, um get e direcionar para: <http://localhost:8761/eureka/apps>. Quando você faz essa requisição, ele vai nos listar quais são as aplicações que já se registraram no Eureka.



Não vai retornar nada ainda por que você tem que adicionar a configuração eureka nos outros YML e também adicionar a dependência 'Eureka Discovery Client' em todos os serviços.:

```
Y application.yml  J EurekaServerApplication.java  Y application.yml ✕
1 server:
2   port: 8081
3
4 eureka:
5   client:
6     register-with-eureka: true
7     fetch-registry: true
8     service-url:
9       defaultZone: http://localhost:8761/eureka
10
```

```
Y application.yml  J EurekaServerApplication.java  Y application.yml  Y *application.yml ✕
1 spring:
2   application:
3     name: 'Floricultura'
4
5 eureka:
6   client:
7     register-with-eureka: true
8     fetch-registry: true
9     service-url:
10    defaultZone: http://localhost:8761/eureka
```

Temos a loja registrada e temos o fornecedor registrado. A floricultura, quando ela for fazer uma requisição para o fornecedor, ela vai fazer uma requisição através do nome da aplicação.