EST – IPCB

LICENCIATURA EM ENGENHARIA INFORMÁTICA

Aplicações Distribuídas

A minha primeira Aplicação Spring Boot baseada na arquitetura de Microservicos

3º Ano / 1º Semestre – 2022/2023

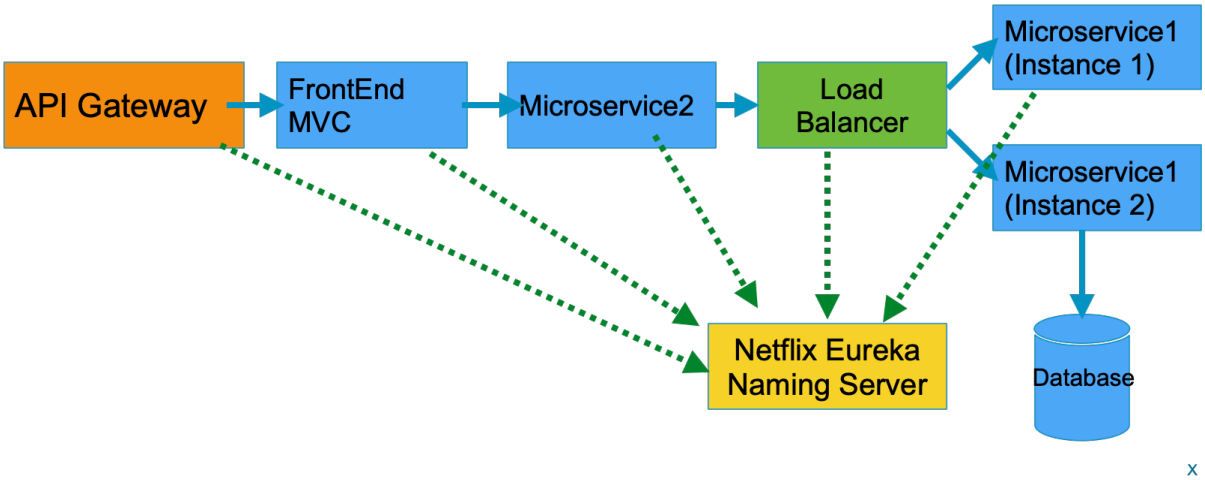
Versão 21 novembro de 2022 (updated a 25 novembro 2022)

Atividade Prática n.º8 - A minha primeira Aplicação Spring Boot com Spring Cloud baseada na arquitetura de Microservicos

Ao longo desta atividade, pretende-se que ao realizar os passos seguintes experimente a implementação de microserviços e a utilização de algumas funcionalidades base do Spring Cloud de suporte ao desenvolvimento deste tipos de aplicações.

Para melhor compreensão dos conteúdos deste guião sugere-se a consulta do bloco de slides: AD-2023-Modulo-7.2_Microservicos_com_Spring_Boot.pdf.

Durante a realização desta atividade serão implementados pequenos projetos cada correspondente a um microserviço. No final a aplicação terá uma arquitetura semelhante a:



x

PARTE 1: CRIAÇÃO e IMPLEMENTAÇÃO DOS PROJECTOS DOS MICROSERVIÇOS (EUREKA, MICROSERVICE1, e MICROSERVICE2)

Passo 1: Criar 3 Projetos maven Spring Boot com os seguintes nomes e dependências:

Nota: Por agora crie apenas os projetos do Microservico2 e do Servidor Spring Eureka (naming-server). Os restantes apenas quando for necessário.

- Microservico-FrontEnd
- Spring Web
- Spring Boot Actuator
- Eureka Discovery Client
- Spring Cloud OpenFeign
- Thymeleaf

- Microservico2-faturacao
- Spring Web
- Spring Boot Actuator
- Eureka Discovery Client
- Spring Data JPA
- H2 Database
- Lombok
- Spring Cloud OpenFeign

- Microservico1-informacao-potencias
- Spring Web
- Spring Actuator
- Eureka Discovery Client
- Spring Data JPA
- H2 Database
- Lombok

- Naming-server
- Eureka Server
- Spring Actuator

- API-Cloud-Gateway
- Spring Cloud Routing / Gateway
- Eureka Discovery Client
- Spring Actuator

Passo 2: Abra o Projeto **naming-server** e realize as seguintes tarefas:

- Anote a classe principal da aplicação com **`@EnableEurekaServer`**
- Defina as seguintes **application.properties** para o servidor de nomes (naming server).

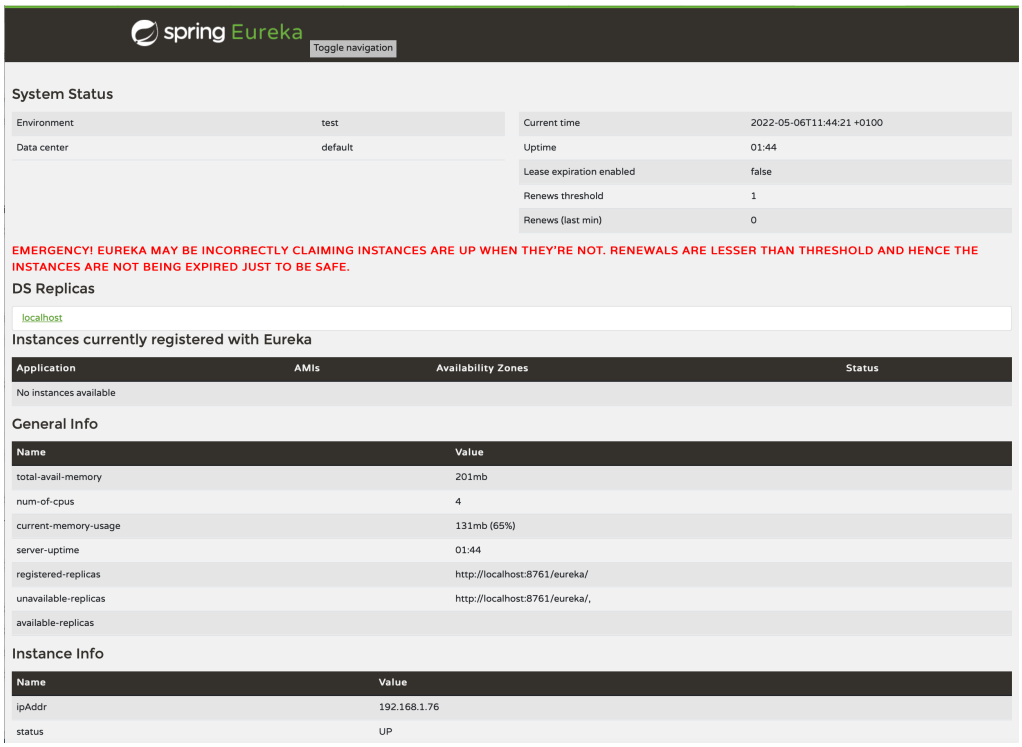
```
## Nome da aplicação Spring
spring.application.name=naming-server

## Porta por Omissão Servidor Eureka
server.port=8761

## O Servidor Eureka não se deve registar
eureka.client.register-with-eureka=false
eureka.client.fetch-registry=false

logging.level.com.netflix.eureka=OFF
logging.level.com.netflix.discovery=OFF
```

- Execute o projeto.
- Abra o browser e introduza o endereço **http://localhost:8761**
- Deve obter a página Spring Eureka:



Passo 3: Abra o projeto do **microservico1** e realize as seguintes tarefas:

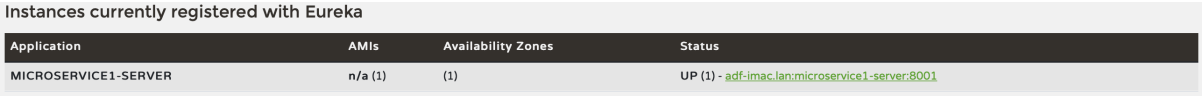
- Defina as seguintes **application.properties** para o microservice1.

```
spring.application.name=microservice1-server
server.port=8001
eureka.client.serviceUrl.defaultZone=http://localhost:8761/eureka
```

- Execute e teste o projeto.
- Nota: adicione a seguinte dependência, em caso de problema durante o *build* do microserviço.

```
<dependency>
    <groupId>com.sun.jersey.contribs</groupId>
    <artifactId>jersey-apache-client4</artifactId>
    <version>1.19.4</version>
</dependency>
```

- Refresque a página web do Spring Eureka para verificar se o microservice1 se registou.



Passo 4: Exponha o link actuator **/info** preenchendo e ativando a variável de ambiente info no ficheiro **application.properties**.

```
## Configuring info endpoint for Atuator
info.app.name=Microservice 1
info.app.description=This is my first spring boot microservice 1
info.app.version=1.0.0

## Expose all actuator endpoints
management.endpoints.web.exposure.include=*

## Expose info Environment Variable
management.info.env.enabled = true

info.java-vendor = ${java.specification.vendor}
```

- Para testar, re-execute o **microservico1**, volte à página web do Spring Eureka e clique no nome do microserviço para visualizar a variável **/info**

http://<microservico>:<porta>/info

```
{"app":{"name":"Microservice 1","description":"This is my first spring boot microservice 1","version":"1.0.0"}}
```

- Consulte os links disponíveis invocando o endpoint **/actuator**

http://<microservico>:<porta>/actuator

Nota: Deverão ser apresentados 13 links.

- Para observar o estado do microserviço, teste o link /actuator/health

```
http://<microservico>:<porta>/actuator/health
```

```
{"status":"UP"}
```

- Ative a visualização detalhada sobre a saúde (health) do microserviço fazendo a seguinte configuração:

```
management.endpoint.health.show-details=always
```

- Teste novamente invocando o link /actuator/health

Imagem Ilustrativa:


```
{"status":"UP","components":{"discoveryComposite":{"status":"UP","component
s":{"discoveryClient":{"status":"UP","details":{"services":["microservice1-
server","microservice2-server"]}},"eureka":{"description":"Remote status
from Eureka
server","status":"UP","details":{"applications":{"MICROSERVICE1-
SERVER":1,"MICROSERVICE2-
SERVER":1}}}},"diskSpace":{"status":"UP","details":{"total":1027680514048,
"free":588721520640,"threshold":10485760,"exists":true}},"ping":{"status":"
UP"},"refreshScope":{"status":"UP"}}}
```


Passo 5: Repita os passos 3 e 4 para o microservice2. A porta servidor a considerar para este microserviço é: `server.port=8201`

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
MICROSERVICE1-SERVER	n/a (1)	(1)	UP (1) - afs-macbook-air.lan:microservice1-server:8001
MICROSERVICE2-SERVER	n/a (1)	(1)	UP (1) - localhost:microservice2-server:8201

Passo 6: Nos microservices 1 e 2 crie uma classe ControladorRest ou anote a Main Classe com @RestController e inclua os dois métodos seguintes para consultar os nomes de todos os microserviços registados no Spring Eureka, e obter os detalhes das instâncias de um microserviço (pode existir mais do que uma, embora neste ponto não seja o caso):

```
@AutoWired
DiscoveryClient discoveryClient;

@GetMapping(value = "/instancias-servico/{nomeservico}")
public List<ServiceInstance>
getAllInstancesServicoByNomeMicroservico
(@PathVariable String nomeservico) {
      return this.discoveryClient.getInstances(nomeservico);
```

```
}

@GetMapping(value = "/servicos")
public List<String> getAllServicos() {
	return this.discoveryClient.getServices();
}
```

Passo 7: Crie um método a **hello** anotado com **@GetMapping("hello")** para pingar o microserviço. Este método retorna a string "Olá do Microserviço x". Para obter o nome leia do ficheiro **application.properties** a *property* **spring.application.name**

Passo 8: Integre o módulo Swagger-ui, adicionando a seguinte dependência à lista de dependências do projeto (nesta ou versão superior):

```
<dependency>
	<groupId>org.springdoc</groupId>
	<artifactId>springdoc-openapi-ui</artifactId>
	<version>1.6.6</version>
</dependency>
```

A página do Swagger UI ficará disponível no url http://server:port/context-path/swagger-ui/index.html e a documentação OpenAPI no formato json no url: http://server:port/context-path/v3/api-docs

Experimente a ferramenta de testes disponibilizada.
No nosso caso deve utilizar o URI:
http://localhost:8001/swagger-ui/index.html
http://localhost:8001/v3/api-docs

PARTE 2: IMPLEMENTACÃO DA LÓGICA DE NEGÓCIO DOS MICROSERVIÇOS

Passo 9: Considere as seguintes funções para cada microserviço. Utilize o **Spring Cloud OpenFeign** para interligar os microserviços (para mais informação consultar a página oficial do Spring Cloud OpenFeign em https://spring.io/projects/spring-cloud-openfeign)

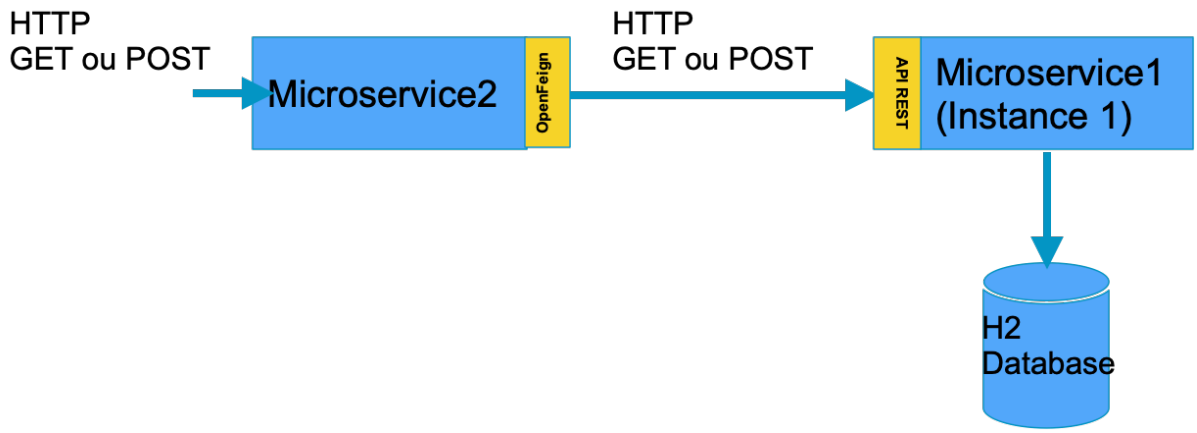
2.1. Implementação do Microservico2

- O Microservico2 calcula a fatura mensal em euros para o consumo energético mensal de um aparelho eletrodoméstico.
- A implementação é inspirada em:
  https://www.rapidtables.com/calc/electric/electricity-calculator.html
- O Microservico2 disponibiliza simultaneamente um método **GET** e um método **POST** para receber os dados necessários aos cálculos da fatura (ver URI propostos em baixo).

- Utilize códigos http adequados tal conforme uma das fichas sobre Controladores REST.
- Numa primeira implementação implemente o método **GET** na versão "monolítica", incluindo uma pequena base de dados H2.
- Este serviço monolítico guarda nesta as potências dos eletrodomésticos em Watts numa Entidade Potencia ou Aparelho: **<id (PK), <aparelho>, <potencia>.** Utilize as anotações do projeto Lombok. As definições da ligação podem ser:

```
## BAse de dados H2
spring.datasource.url=jdbc:h2:mem:potenciasbd
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=admin
spring.datasource.password=admin
spring.datasource.initialization-mode=always
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.h2.console.enabled=true
```

- Quando arrancar cria um registo para um aquecedor de 2000 Watts e um registo para uma ventoinha de 800 Watts.
- Só depois de implementar o microservico1 é que implemente a versão final do método "microserviço" e chame a operação sobre o **Microservico1**.
- Nesta altura precisa de:
  - Criar uma interface **ProxyMicroservico1** anotada com **@FeignClient(value="service-name-to-connect",url="url-microservice:port").**
  - Decorar a classe Main da aplicação ou Controlador com **@EnableFeignClients**
  - Injectar (@Autowired) uma instância no controlador REST que usa o proxy.
  - Para apanhar as exceções **FeignException** recomenda-se que use um bloco **try-catch** que invoca os métodos do microservice1.



URIs válidos a considerar consoante a versão monolítica ou microserviço:
GET /monolitico/faturas/{aparelho}/{HUD}/{KC} – versão monolítica
GET /faturas/{aparelho}/{HUD}/{KC} – versão microserviços

POST /faturas
Nota: No caso da requisição POST os parâmetros (aparelho, HUD, KC) do método devem ser anotados com **@RequestParam** ou então são recebidos num parâmetro do tipo **DadosFatura** que pode criar com os respetivos atributos e anote o parâmetro com **@RequestBody**.

Ambos os métodos Retornam o valor da Fatura mensal calculado:
Fatura= (PAW/1000) x HUD x 30 x KC/100
Sucesso: Http 200 Ok ou 201 Created

Erro (caso não encontre o aparelho)
Http 404 Not Found
Outros erros: 400 Bad Request

Onde:
- *Potência do Aparelho em Watts (PAW)*
- *Horas de uso por Dia (HUD)*
- *Custo 1 kW/h(KC) em cts de Euro*

- Execute o Projeto e Teste os URIs do microserviço usando o Swagger-UI (casos de sucesso e de erro).

2.2 Implementação do Microservico1

- **Microservico1** fornece ao **microservico2** a informação sobre o consumo nominal típico em Watts de um eletrodoméstico através da sua API pública.
- **Microservico1** esconde (hide), conforme se recomenda, a sua Base Dados H2 dos restantes microserviços.
- Este microserviço guarda numa Base de Dados H2 as potências dos eletrodomésticos em Watts numa Entidade Potencia ou Aparelho: **<id (PK), <aparelho>, <potencia>**
- Quando arranca cria um registo para um **aquecedor a óleo** de 2000 Watts e de uma **ventoinha** de 800 Watts.
- **(Quando realizar a Parte 3)** Para que várias instâncias do microserviço guardem os dados na mesma BD, defina que a Base de Dados H2 é guardada num ficheiro externo localizado no diretório de trabalho do microservice1 (onde é arrancado o Jar):

```
spring.datasource.url=jdbc:h2:file:./potenciasdb
```

URIs disponibilizados pelo microservico1

GET /potencias/{nomeaparelho}
Consulta a potência do aparelho pelo nome
Retorna:
Sucesso: 200 OK + Double com o valor da Potência do Aparelho em Watts (PAW).
Erro: 404 NOT Found no caso de não encontrar o aparelho na BD.
Outros erros: 400 Bad Request

POST /potencias
Cria uma instância da potência de um aparelho
Retorna:
Sucesso: 201 CREATED + Json da instância criada.
Erro: 400 Bad Request

Inclua ainda um método que devolve todos os aparelhos na BD. Para o retorno, defina códigos de estado Http adequados.

- Execute o Projeto e Teste os URIs do microserviço usando o Swagger-UI (casos de sucesso e de erro)

2.3 Microserviço FrontEnd MVC

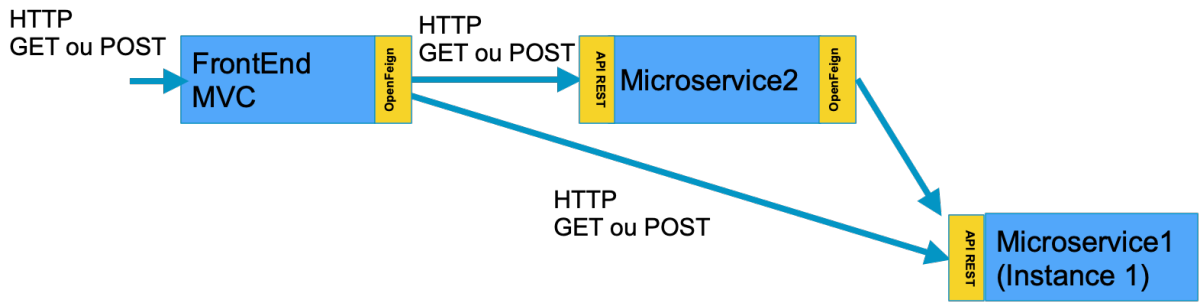
- De seguida, implemente um microserviço FrontEnd MVC cuja página **index.html** permite selecionar 3 links:
- **Um Link1 para um formulário Web Thymeleaf** onde pode preencher os seguintes campos (por agora não precisa de ser perfeito, use apenas os elementos *input*):
    - Nome do aparelho elétrico
    - Número Horas ligado por dia (H/dia)
    - Custo kW/h em cêntimos de euro
    - Após submissão será retornada uma página com uma tabela com o nome do eletrodoméstico, os dados submetidos e o correspondente valor da fatura mensal em Euros.

- Um Link2 para um formulário Web Th que permite:
    - Adicionar registos de aparelhos à BD do microservico1
    - Após submissão deverá ser ecoado o registo criado.

- Um Link3 que permite:
    - Consultar os registos de todos os aparelhos registados

Conforme nos pontos 2.1 e 2.2, implemente na API Publica dos microservice1 e microservice2, os métodos REST que dão suporte as estas operações.

Implemente as conexões com os microserviços 1 e 2 usando o Spring Cloud OpenFeign.

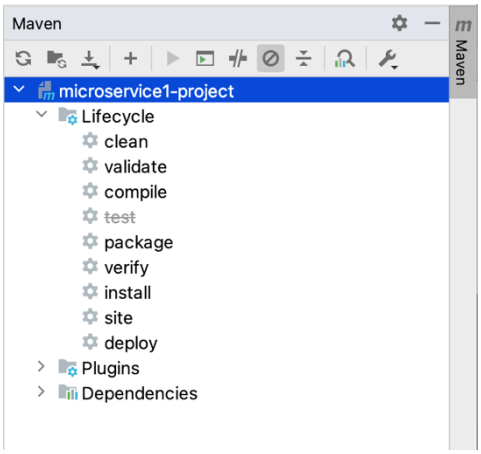


- Execute o Projeto e Teste a utilização das Vistas Web desenvolvidas e do controlador MVC do Microserviço FrontEnd (casos de sucesso e de erro)

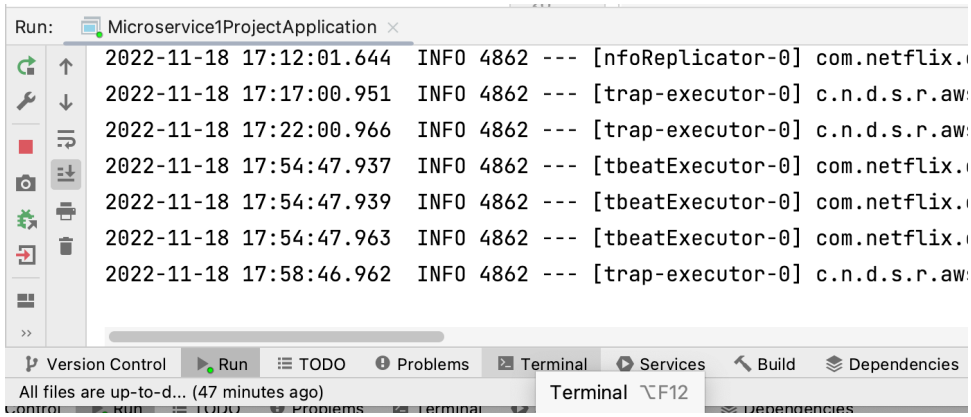
PARTE 3: BALANCEAMENTO DE CARGA

Passo 10: Replique o microservice1. Para isso:

- Comece por preparar o microserviço para que possamos aquando do seu lançamento usando o comando **java** passar um porto personalizado e distinto do Default (8001)
    - Por exemplo, defina **`server.port=${porta:8001}`**
    - Alternativamente pode utilizar uma porta aleatória definindo **`server.port=0`**, mas neste caso convém criar um **instance-id** único para serem visíveis as várias instâncias no Eureka (ver bloco de slides).
- Crie o **jar** do projeto executando o comando **maven** que cria o package disponível no menu **maven** lateral. O Jar aparecerá no diretório **/target** do projeto



- Abra uma janela terminal e execute o comando java para executar a nova instância do microserviço e passe a porta 8002. Depois repita para uma 3.ª instancia e passe a porta 8003 … conforme o número de instâncias.




```
java -Dporta=8002 -jar target/microservice1-project-0.0.1-SNAPSHOT.jar
```

Nota: Atenção que o nome do Jar pode ser diferente.

- Verifique que as instâncias do microserviço1 se registam no serviço Eureka.

Passo 11: Ative o balanceamento de carga Cliente-side. Basta que remova o URL do cliente Feign na anotação **`@FeignClient`**, mas mantenha o nome do microserviço, ou seja:

```
@FeignClient("MICROSERVICE1-SERVER")
```

Teste a implementação.
Invoque várias vezes pedidos do tipo:
http://localhost:8201/faturas/ventoinha/5/20

Não notará qualquer alteração ao funcionamento da App, contudo serão usadas as instâncias normalmente de acordo com uma política *Round-Robin*.

Seguir para a parte 4 ...

PARTE 4: ADIÇÃO DE UM SPRING CLOUD API GATEWAY

Passo 11: Crie um novo Projeto **API-Gateway** com as seguintes dependências

- Spring Cloud Routing / Gateway
- Eureka Discovery Cliente
- Spring Boot Actuator

Configure as seguintes **application.properties** (nome, porta e url do Eureka)

```
spring.application.name=api-gateway
server.port=8755
eureka.client.serviceUrl.defaultZone=http://localhost:8761/eur
eka
```

Execute o projeto e verifique o registo do Gateway no serviço Eureka da aplicação.

3.1 Encaminhamento Automático ou Rotas Automáticas

Passo 12: Conceba os URIs para os microservico1 e microservico2 através do API Gateway. Todos os pedidos aos microserviços vindos do exterior têm que passar pelo API Gateway.

- Crie um ficheiro de texto **urls** e anote o nome dos Microservicços vistos pelo Eureka

1. MICROSERVICE1-SERVER
2. MICROSERVICE2-SERVER

- Escreva no ficheiro a PATH para ambos os microserviços:
  http://localhost:8201/monolitico/faturas/ventoinha/5/20
  http://localhost:8201/faturas/ventoinha/5/20
  http://localhost:8001/potencias/ventoinha

- Substitua os endereços dos microserviços localhost:8x0y pelo endereço e porta do API Gateway + nome do microserviço:

URIs propostos API Gateway:
http://localhost:8755/MICROSERVICE1-SERVER/potencias/ventoinha
http://localhost:8755/MICROSERVICE2-SERVER/faturas/ventoinha/5/20

Teste os URIs num browser -> È retornado 404 Request not found

Para resolver o erro adicionar propriedade seguinte que o API Gateway crie automaticamente rotas com base na informação disponível no Eureka.

```
spring.cloud.gateway.discovery.locator.enabled=true
```

Passo 13: Melhoria do URI

- Adicione propriedade seguinte:

```
spring.cloud.gateway.discovery.locator.lowerCaseServiceId=true
```

- Testes os URI:

```
http://localhost:8755/microservice1-server/potencias/ventoinha
http://localhost:8755/microservice2-server/faturas/ventoinha/5/20
```

3.2. Encaminhamento Personalizado no Spring Cloud API Gateway

Passo 14: Desative o encaminhamento automático colocando:

```
spring.cloud.gateway.discovery.locator.enabled=false
```

Passo 15: Configure o Encaminhamento no ficheiro application.properties.

- 15.1 Usando encaminhamento dinâmico no caso do microserviço1
- 15.2 Usando encaminhamento dinâmico no caso do microserviço2

```
spring.cloud.gateway.discovery.locator.enabled=false
spring.cloud.gateway.routes[0].id=microservice1
spring.cloud.gateway.routes[0].uri=lb://microservice1-server
spring.cloud.gateway.routes[0].predicates[0]=Path=/potencias/**
spring.cloud.gateway.routes[0].predicates[1]=Method=GET

spring.cloud.gateway.routes[1].id=microservice2
spring.cloud.gateway.routes[1].uri=http://localhost:8201
##spring.cloud.gateway.routes[1].uri=lb://microservice2-server
spring.cloud.gateway.routes[1].predicates[0]=Path=/faturas/**
spring.cloud.gateway.routes[1].predicates[1]=Method=POST
```

- Teste os acessos através do API Gateway.

```
http://localhost:8755/potencias/ventoinha
http://localhost:8755/faturas/ventoinha/5/20
```

OPS ... Not Found. Acrescente também ao predicado **Method** do microservico2 as requisições HTTP GET:

```
spring.cloud.gateway.routes[1].predicates[1]=Method=POST,GET
```

- Crie uma terceira rota estática para atingir o serviço FrontEnd.
- Teste todos os acessos através do API Gateway.

http://localhost:8755/potencias/ventoinha
http://localhost:8755/faturas/ventoinha/5/20
http://localhost:8755/

Passo 16: Experimente em alternativa configurar o Encaminhamento numa classe @Configuration instanciando um @Bean do tipo RouteLocator.

- 16.1 Usando encaminhamento estático no caso do microserviço1
- 16.2 Usando encaminhamento dinâmico no caso do microserviço2

```
@Configuration
public class Configuracao {
        @Bean
        public RouteLocator configureRoute(RouteLocatorBuilder builder)
{
            return builder.routes()
                    .route("rota_microservice1", r-
>r.path("/potencias/**").uri("lb://microservice1-server")) //dynamic
routing
                    .route("rota_microservice2", r-
>r.path("/faturas/**").uri("http://localhost:8201")) //static routing
                    .build();
                    }
 }
```

FIM