Prof. Bernardo Copstein

Prof. Júlio Machado

Roteiro 1: Introdução aos micros serviços

Criando um par de micros serviços independentes

Nosso primeiro exemplo será composto por dois micros serviços:

- Um que fornece valores de câmbio (*currency-exange*);

- Um que converte valores em uma moeda para outra (*currency-conversion*).

Ambos expõem "endpoints" e atendem a requisições HTTP e retornam JSON como resposta. Como cliente iremos usar o "Insomnia" ou "Postman" ou outro software qualquer capaz de gerar requisições HTTP. A figura 1 apresenta a arquitetura básica do sistema:

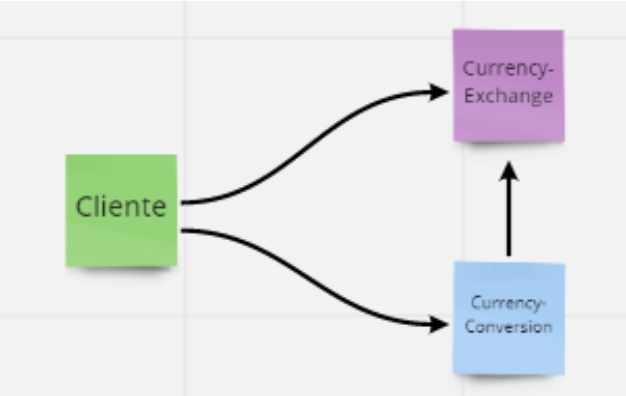

Figura 1 – arquitetura básica do sistema de câmbio

Note que os clientes podem mandar requisições diretamente para o micro serviço de câmbio (*currency-exchange*) para saber a cotação de uma moeda ou para o micro serviço de conversão (*currency-conversion*) para saber quanto vale uma certa quantia (por exemplo, quanto valem 300 euros em reais). Para resolver esta questão o micro serviço de conversão irá questionar o micro serviço de câmbio.

Passo 1: criando o micro serviço de câmbio

A tecnologia de implementação será o Java através do *framework* Spring. Adicionalmente o sistema de *build* será realizado através do Maven e, portanto, o primeiro passo será a configuração das dependências.

Para facilitar o processo, será utilizado o sistema "Spring Initializr" via web no endereço https://start.spring.io/. A figura 2 traz as configurações necessárias, já a figura 3 apresenta o arquivo de dependências correspondente. Clique no botão "GENERATE" e faça o download do projeto em um arquivo "zip" para um diretório local.

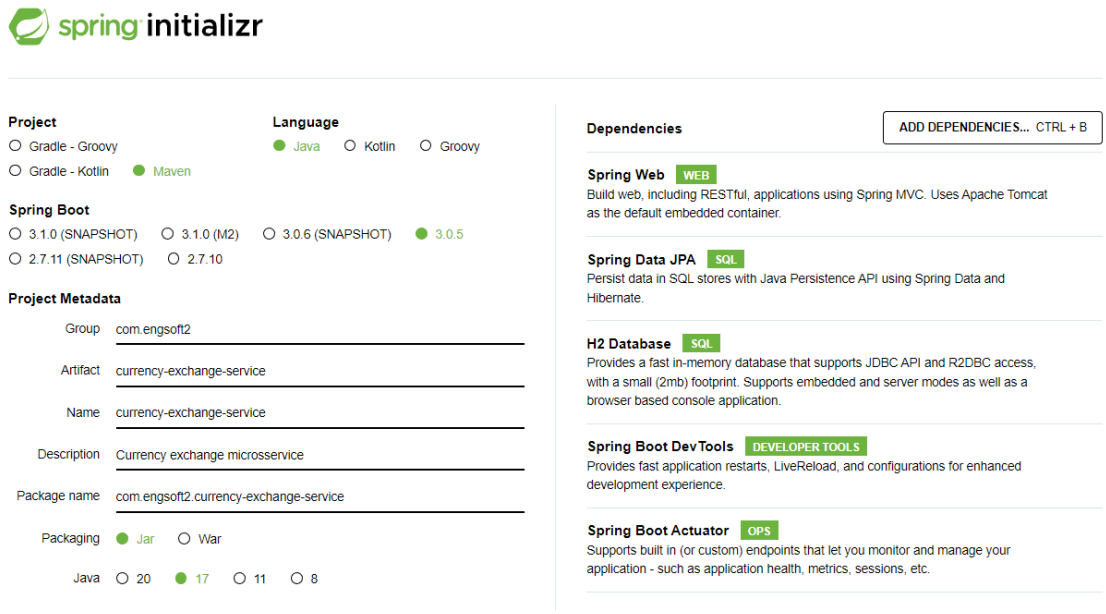

Figura 2 – configuração do Spring Initializr

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <parent>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-parent</artifactId>
                <version>3.0.5</version>
                <relativePath/> <!-- lookup parent from repository -->
        </parent>
        <groupId>com.engsoft2</groupId>
        <artifactId>currency-exchange-service</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <name>currency-exchange-service</name>
        <description>Currency exchange microsservice</description>
        <properties>
                <java.version>17</java.version>
        </properties>
        <dependencies>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-actuator</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-data-jpa</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-web</artifactId>
                </dependency>

                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-devtools</artifactId>
                        <scope>runtime</scope>
                        <optional>true</optional>
                </dependency>
                <dependency>
                        <groupId>com.h2database</groupId>
                        <artifactId>h2</artifactId>
                        <scope>runtime</scope>
                </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-test</artifactId>
```

```
                        <scope>test</scope>
                </dependency>
        </dependencies>

        <build>
                <plugins>
                        <plugin>
                                <groupId>org.springframework.boot</groupId>
                                <artifactId>spring-boot-maven-plugin</artifactId>
                        </plugin>
                </plugins>
        </build>

</project>
```

Figura 3: arquivo POM do micro serviço de câmbio.

A figura 4 apresenta o código das classes desse micro serviço. Note que se trata de um "backend de serviços REST" padrão nos moldes do Spring Boot, embora, por simplicidade não siga uma arquitetura "Clean" (toda a lógica está no próprio "controller"). Alguns detalhes a serem observados:

- A forma como o construtor é gerado automaticamente a partir das anotações "@Autowired" junto dos atributos;
- A forma como a requisição HTTP é estruturada. A requisição HTTP para esse "endpoint" é: http://localhost:8000/currency-exchange/from/USD/to/BRL
- A indicação de quais atributos da entidade serão persistidos. Na verdade, essa não é uma boa prática de programação. A classe "CurrencyExchange" está sendo usada como "Entity" e "DTO" ao mesmo tempo. Por isso os atributos "conversionMultiple" e "environment" não estão sendo persistidos. No caso optou-se por esta abordagem para simplificar o exemplo.
- Note que o repositório é gerado automaticamente usando a JPA. Seu acionamento é feito diretamente no "controller". Não há separação de camadas.

```
import java.math.BigDecimal;
import jakarta.persistence.Column;
import jakarta.persistence.Entity;
import jakarta.persistence.Id;

@Entity
public class CurrencyExchange {
    @Id
    private Long id;
    @Column(name = "currency_from")
    private String from;
    @Column(name = "currency_to")
    private String to;
    private BigDecimal conversionMultiple;
    private String environment;

    public CurrencyExchange() {
    }

    public CurrencyExchange(Long id, String from, String to, BigDecimal
conversionMultiple) {
        this.id = id;
        this.from = from;
        this.to = to;
        this.conversionMultiple = conversionMultiple;
    }

    public Long getId() {
        return id;
    }
```

```
    public void setId(Long id) {
        this.id = id;
    }

    public String getFrom() {
        return from;
    }

    public void setFrom(String from) {
        this.from = from;
    }

    public String getTo() {
        return to;
    }

    public void setTo(String to) {
        this.to = to;
    }

    public BigDecimal getConversionMultiple() {
        return conversionMultiple;
    }

    public void setConversionMultiple(BigDecimal conversionMultiple) {
        this.conversionMultiple = conversionMultiple;
    }

    public String getEnvironment() {
        return environment;
    }

    public void setEnvironment(String environment) {
        this.environment = environment;
    }
}
```

```
import org.springframework.data.repository.CrudRepository;

public interface CurrencyExchangeRepository extends
CrudRepository<CurrencyExchange,Long> {
    CurrencyExchange findByFromAndTo(String from, String to);
}
```

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.core.env.Environment;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class CurrencyExchangeController {
    private Logger logger =
LoggerFactory.getLogger(CurrencyExchangeController.class);
    @Autowired
    private CurrencyExchangeRepository repository;
    @Autowired
    private Environment environment;

    @GetMapping("/currency-exchange/from/{from}/to/{to}")
    public CurrencyExchange retrieveExchangeValue(@PathVariable String from,
@PathVariable String to) {
        logger.info("retrieveExchangeValue called with {} to {}", from, to);
        CurrencyExchange currencyExchange = repository.findByFromAndTo(from,
to);
```

```
        if(currencyExchange ==null) {
            throw new RuntimeException("Unable to Find data for " + from + "
to " + to);
        }
        String port = environment.getProperty("local.server.port");
        currencyExchange.setEnvironment(port);
        return currencyExchange;
    }
}
```

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class CurrencyExchangeServiceApplication {

      public static void main(String[] args) {
            SpringApplication.run(CurrencyExchangeServiceApplication.class,
args);
      }
}
```

Figura 4 – código do serviço de câmbio

Além do código propriamente dito, necessitamos configurar algumas propriedades no arquivo "application.properties" no diretório "resources" como pode ser visto na figura 5. As duas primeiras propriedades definem o nome do micro serviço e a porta que será monitorada. As outras três são relativas ao uso do JPA.

```
server.port=8000
spring.application.name=currency-exchange

spring.jpa.show-sql=true
spring.jpa.defer-datasource-initialization=true
spring.datasource.url=jdbc:h2:mem:testdb
spring.h2.console.enabled=true
```

Figura 5: arquivo de propriedades

A inicialização do banco de dados é feita com um "script" armazenado no arquivo "data.sql" no diretório "resources" (ver figura 6).

```
insert into currency_exchange
(id,currency_from,currency_to,conversion_multiple,environment)
values(10001,'USD','INR',65,'');
insert into currency_exchange
(id,currency_from,currency_to,conversion_multiple,environment)
values(10002,'EUR','INR',75,'');
insert into currency_exchange
(id,currency_from,currency_to,conversion_multiple,environment)
values(10003,'AUD','INR',25,'');
insert into currency_exchange
(id,currency_from,currency_to,conversion_multiple,environment)
values(10004,'USD','BRL',5,'');
```

Figura 6: script de inicialização do banco de dados

Para "subir" este micro serviço use o comando: "mvn spring-boot:run". Teste se ele está funcionando com a requisição apresentada anteriormente (http://localhost:8000/currency-exchange/from/USD/to/BRL).

Passo 2: criando o micro serviço de conversão

A tecnologia de implementação será o Java através do *framework* Spring. Adicionalmente o sistema de *build* será realizado através do Maven e, portanto, o primeiro passo será a configuração das dependências.

Para facilitar o processo, será utilizado o sistema "Spring Initializr" via web no endereço https://start.spring.io/. A figura 7 traz as configurações necessárias, já a figura 8 apresenta o arquivo de dependências correspondente. Clique no botão "GENERATE" e faça o download do projeto em um arquivo "zip" para um diretório local.

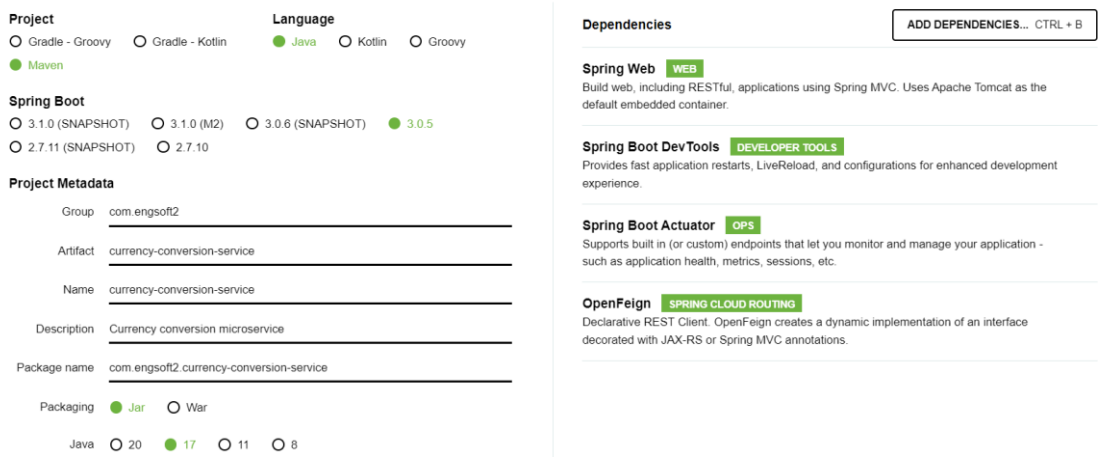

Figura 7 – configuração do Spring Initializr

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <parent>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-parent</artifactId>
                <version>3.0.5</version>
                <relativePath/> <!-- lookup parent from repository -->
        </parent>
        <groupId>com.engsoft2</groupId>
        <artifactId>currency-conversion-service</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <name>currency-conversion-service</name>
        <description>Currency conversion microservice</description>
        <properties>
                <java.version>17</java.version>
                <spring-cloud.version>2022.0.2</spring-cloud.version>
        </properties>
        <dependencies>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-actuator</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-web</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.springframework.cloud</groupId>
                        <artifactId>spring-cloud-starter-openfeign</artifactId>
                </dependency>

                <dependency>
```

```
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-devtools</artifactId>
                        <scope>runtime</scope>
                        <optional>true</optional>
                </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-test</artifactId>
                        <scope>test</scope>
                </dependency>
        </dependencies>
        <dependencyManagement>
                <dependencies>
                        <dependency>
                                <groupId>org.springframework.cloud</groupId>
                                <artifactId>spring-cloud-dependencies</artifactId>
                                <version>${spring-cloud.version}</version>
                                <type>pom</type>
                                <scope>import</scope>
                        </dependency>
                </dependencies>
        </dependencyManagement>

        <build>
                <plugins>
                        <plugin>
                                <groupId>org.springframework.boot</groupId>
                                <artifactId>spring-boot-maven-plugin</artifactId>
                        </plugin>
                </plugins>
        </build>

</project>
```

Figura 8: arquivo POM do micro serviço de câmbio.

A figura 9 apresenta o código do micro serviço de conversão. Ele está com dois “endpoints” que fazem exatamente a mesma coisa. O objetivo é facilitar o entendimento dos recursos do “Spring-boot”.

O primeiro endpoint “/currency-conversion” faz a conversão dos valores. Ele questiona o micro serviço de cambio sobre o valor corrente da moeda para então poder fazer a conversão. Seu funcionamento é convencional. Ele recebe os parâmetros através de “path variables”. Para se comunicar de forma síncrona com o micro serviço de câmbio, usa uma instância de “RestTemplate”. Note que por ser uma comunicação síncrona a execução é suspensa até que o micro serviço demandado encaminhe a resposta. O método “getForEntity” da classe “RestTemplate” é usado para retornar uma instância de “ResponseEntity”. A classe “ResponseEntity” armazena uma série de informações sobre a comunicação em si. O método “getBody” dessa classe retorna o “corpo” da mensagem HTTP que se traduz na classe resposta. O envio da solicitação através do método “getForEntity” exige a string com o “endpoint” destino, a classe que será retornada e um dicionário com os parâmetros que serão informados na solicitação. É uma construção trabalhosa, mas funciona.

O segundo endpoint “/currency-conversion” faz exatamente a mesma coisa – solicita o câmbio para o micro serviço de cambio e retorna a conversão – só que de uma forma mais simples. Antes de tudo define-se a interface “CurrencyExchangeProxy” anotada com “@FeignClient”. A API Feign é um gerador de clientes para o protocolo HTTP. Uma classe concreta que implementa essa interface é gerada pelo SpringBoot e age como um “proxy”, ou seja, faz o papel do micro serviço de câmbio, mas na verdade comunica-se com ele. Esta classe é injetada no “controller” para poder ser acessada pelo “endpoint”. O uso de “clientes proxy” na comunicação entre micro serviços simplifica bastante esta comunicação principalmente quando for o caso de se usar balanceamento de carga como será visto mais adiante. Para que este recurso funcione, porém

é necessário a inclusão da dependência da API correspondente no arquivo POM.xml. Sugere-se o uso desta segunda abordagem.

```
import java.math.BigDecimal;

public class CurrencyConversion {
    private Long id;
       private String from;
       private String to;
       private BigDecimal quantity;
       private BigDecimal conversionMultiple;
       private BigDecimal totalCalculatedAmount;
       private String environment;

    public CurrencyConversion() {
    }

    public CurrencyConversion(Long id, String from, String to,
                              BigDecimal quantity,
                              BigDecimal conversionMultiple,
                              BigDecimal totalCalculatedAmount,
                              String environment) {
        this.id = id;
        this.from = from;
        this.to = to;
        this.quantity = quantity;
        this.conversionMultiple = conversionMultiple;
        this.totalCalculatedAmount = totalCalculatedAmount;
        this.environment = environment;
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getFrom() {
        return from;
    }

    public void setFrom(String from) {
        this.from = from;
    }

    public String getTo() {
        return to;
    }

    public void setTo(String to) {
        this.to = to;
    }

    public BigDecimal getQuantity() {
        return quantity;
    }

    public void setQuantity(BigDecimal quantity) {
        this.quantity = quantity;
    }

    public BigDecimal getConversionMultiple() {
        return conversionMultiple;
    }
```

```
    public void setConversionMultiple(BigDecimal conversionMultiple) {
        this.conversionMultiple = conversionMultiple;
    }

    public BigDecimal getTotalCalculatedAmount() {
        return totalCalculatedAmount;
    }

    public void setTotalCalculatedAmount(BigDecimal totalCalculatedAmount) {
        this.totalCalculatedAmount = totalCalculatedAmount;
    }

    public String getEnvironment() {
        return environment;
    }

    public void setEnvironment(String environment) {
        this.environment = environment;
    }
}
```

```
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;

@FeignClient(name = "currency-exchange", url = "localhost:8000")
public interface CurrencyExchangeProxy {
    @GetMapping("/currency-exchange/from/{from}/to/{to}")
    public CurrencyConversion retrieveExchangeValue(@PathVariable String from,
                                                    @PathVariable String to);
}
```

```
import java.math.BigDecimal;
import java.util.HashMap;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.client.RestTemplate;

@RestController
public class CurrencyConversionController {
    @Autowired
    private CurrencyExchangeProxy proxy;

@GetMapping("/currency-conversion/from/{from}/to/{to}/quantity/{quantity}")
public CurrencyConversion calculateCurrencyConversion(
                                                @PathVariable String from,
                                                @PathVariable String to,
                                                @PathVariable BigDecimal quantity){
        HashMap<String, String> uriVariables = new HashMap<>();
        uriVariables.put("from",from);
        uriVariables.put("to",to);
        ResponseEntity<CurrencyConversion> responseEntity =
                      new RestTemplate().getForEntity(
                        "http://localhost:8000/currency-exchange/from/{from}/to/{to}",
                        CurrencyConversion.class,
                        uriVariables);
        CurrencyConversion currencyConversion = responseEntity.getBody();
        return new CurrencyConversion(
                      currencyConversion.getId(),
                      from, to, quantity,
                      currencyConversion.getConversionMultiple(),
                      quantity.multiply(currencyConversion.getConversionMultiple()),
                      currencyConversion.getEnvironment()+ " " + "rest template"
        );
    }
```

```
    @GetMapping("/currency-conversion-feign/from/{from}/to/{to}/quantity/{quantity}")
    public CurrencyConversion calculateCurrencyConversionFeign(
                                        @PathVariable String from,
                                        @PathVariable String to,
                                        @PathVariable BigDecimal quantity) {
        CurrencyConversion currencyConversion = proxy.retrieveExchangeValue(from, to);
        return new CurrencyConversion(
                    currencyConversion.getId(),
                    from, to, quantity,
                    currencyConversion.getConversionMultiple(),
                    quantity.multiply(currencyConversion.getConversionMultiple()),
                    currencyConversion.getEnvironment() + " " + "feign"
        );
    }
}
```

```
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.openfeign.EnableFeignClients;

@SpringBootApplication
@EnableFeignClients
public class CurrencyConversionServiceApplication {

       public static void main(String[] args) {
              SpringApplication.run(
                  CurrencyConversionServiceApplication.class,
                  Args);
       }

}
```

Conteúdo do arquivo de propriedades:

```
spring.application.name=currency-conversion
server.port=8100
```

Figura 9 – código do conversor de moeda

Para executar o micro serviço de conversão, abra uma segunda janela de console e use o mesmo comando “mvn spring-boot:run”. Teste o uso dos dois “endpoints” com as seguintes requisições HTTP:

http://localhost:8100/currency-conversion/from/USD/to/INR/quantity/10

http://localhost:8100/currency-conversion-feign/from/USD/to/INR/quantity/10

Conclusões

Este roteiro mostrou como executar os micros serviços de câmbio e de conversão de moedas e como eles podem interagir de forma síncrona. Além do processo todo ser manual, é necessário saber em que porta cada serviço foi alocado deixando a aplicação muito engessada. Em um próximo roteiro vamos acrescentar um servidor de nomes e um balanceador de carga, de maneira a facilitar a interação com a aplicação.

Exercício

Crie um micro serviço adicional chamado coletor de informações (information-collector). Toda vez que o serviço de câmbio é consultado sobre uma cotação ele informa o micro serviço de coleta de informações sobre a consulta feita. O micro serviço de coleta de informações armazena então esta informação (moeda, quantidade, valor, data e hora). Futuramente ele irá disponibilizar consultas sobre que moedas são mais consultadas, quais as quantidades mais

frequentes, em que horários etc. Por simplicidade você pode manter estas informações em memória. Este micro serviço deve dispor de um endpoint que se acionado retorna todas as informações armazenadas.