

## Département Mathématiques et Informatique

### Cycle Ingénieur

#### « Ingénierie Informatique – Big Data et Cloud Computing »

### COMPTE-RENDU:

#### Activité pratique N° 3 - Architectures Micro services avec (Spring Cloud Config, Consul Discovery, Consul Config, Vault)



Réalisé par: Asmaa ELASRI

Encadré par: Pr. Mohamed YOUSSEFI

Année Universitaire : 2022-2023

# Sommaire

Département Mathématiques et Informatique.....	1
« Ingénierie Informatique – Big Data et Cloud Computing » .....	1
<b>Activité pratique N° 3 - Architectures Micro services avec (Spring Cloud Config, Consul Discovery, Consul Config,Vault).....</b>	<b>1</b>
<b>Travail à faire .....</b>	<b>1</b>
<b>PARTIE 1 : Configuration des microservices .....</b>	<b>2</b>
1. Démarrer le service Consul Discovery.....	2
2. Activer le service de configuration des microservices: .....	2
3. Création d'un dossier contient les fichiers de configuration des ms : .....	2
4. Configuration partagée par les microservices : .....	3
5. Exemple de configuration (Customer-service) : .....	3
6. Configuration de service de configuration crée : .....	3
7. Démarrer le service de configuration : .....	3
<b>PARTIE 2 : Création de Gateway .....</b>	<b>3</b>
1. Configuration automatique de Gateway : .....	4
2. Configuration automatique de Gateway : .....	4
<b>PARTIE 3 : Création de customer-service .....</b>	<b>4</b>
1. Structure du projet: .....	4
2. Configuration de service : .....	4
3. Entité de Customer: .....	5
4. Repository de customer-service: .....	5
5. Récupérer la configuration de customer-service : .....	5
6. Tester le customer-service : .....	6
<b>PARTIE 4 : Création de inventory-service.....</b>	<b>6</b>
1. Configuration de service : .....	6
2. Entité de Product : .....	7
3. Projection de inventory -service: .....	7
4. Tester le inventory -service : .....	7
<b>PARTIE 5 : Création de order-service .....</b>	<b>9</b>
1. Structure du projet : .....	9
2. Configuration de service : .....	9
3. Entité de Order: .....	10
4. Entité ProductItem: .....	10
5. Les modules utilisés dans order-service : .....	11

6.	Communiquer le order-service avec customer-service avec OpenFeign : .....	11
7.	Récupérer le customer et la liste des produits d'ordre : .....	12
8.	Tester le order-service : .....	12
<b>PARTIE 6 : Création de billing-service .....</b>		<b>13</b>
1.	Dépendances .....	13
2.	Structure.....	14
3.	Configuration de billing-service : .....	14
4.	Partager le secret avec Vault : .....	14
<b>PARTIE 7 : Frontend avec Angular.....</b>		<b>16</b>
1.	Liste des produits :.....	16
2.	Liste des customer : .....	17
3.	Liste des order de customer 1 :.....	17
4.	Détails de order 5 : .....	17

# Travail à faire

**Création d'une application de e-commerce basée sur les micro services:**

**Consul Discovery**

**Spring Cloud Config**

**Spring Cloud Gateway**

**Customer-service**

**Inventory Service**

**Order Service**

**Consul Config (Billing Service)**

**Vault (Billing Service)**

**Frontend Web avec Angular**

# PARTIE 1 : Configuration des microservices

## 1. Démarrer le service Consul Discovery.

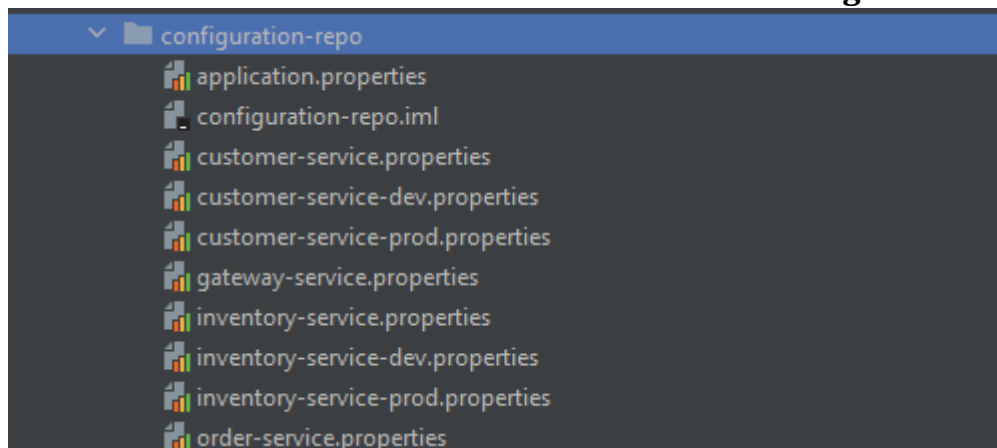
```
C:\ProgramData\consul>consul agent -server -bootstrap-expect=1 -data-dir=consul-data -ui -bind=192.168.11.109
==> Starting Consul agent...
    Version: '1.14.3'
    Build Date: '2022-12-13 17:13:55 +0000 UTC'
    Node ID: 'ec12efa6-63c8-d6b5-ea32-e216c9699494'
    Node name: 'DESKTOP-BOJP3KP'
    Datacenter: 'dc1' (Segment: '<all>')
    Server: true (Bootstrap: true)
    Client Addr: [127.0.0.1] (HTTP: 8500, HTTPS: -1, gRPC: -1, gRPC-TLS: 8503, DNS: 8600)
    Cluster Addr: 192.168.11.109 (LAN: 8301, WAN: 8302)
    Gossip Encryption: false
    Auto-Encrypt-TLS: false
    HTTPS TLS: Verify Incoming: false, Verify Outgoing: false, Min Version: TLSv1_2
    gRPC TLS: Verify Incoming: false, Min Version: TLSv1_2
    Internal RPC TLS: Verify Incoming: false, Verify Outgoing: false (Verify Hostname: false), Min Version: TLSv1_2
==> Log data will now stream in as it occurs:
2022-12-24T14:10:36.500+0100 [WARN] agent: BootstrapExpect is set to 1; this is the same as Bootstrap mode.
2022-12-24T14:10:36.500+0100 [WARN] agent: bootstrap = true: do not enable unless necessary
```



## 2. Activer le service de configuration des microservices:

```
8 @SpringBootApplication
9 @EnableConfigServer
10 @EnableDiscoveryClient
11 public class ConfigServiceApplication {
12
13     no usages
14     public static void main(String[] args) { SpringApplication.run(ConfigServiceApplication.class, args); }
15
16 }
17 }
```

## 3. Création d'un dossier contient les fichiers de configuration des ms :



#### 4. Configuration partagée par les microservices :

```
application.properties
1 global.params.p1 = 15
2 global.params.p2 = 22
3 management.endpoints.web.exposure.include=*
4 |
```

#### 5. Exemple de configuration (Customer-service) :

```
customer-service.properties
1 customer.params.x = 999
2 customer.params.y = 888
3 spring.datasource.url=jdbc:h2:mem:customer-db
4 spring.h2.console.enabled=true
5 |
```

```
customer-service-dev.properties
1 customer.params.x = 89
2 customer.params.y = 63
```

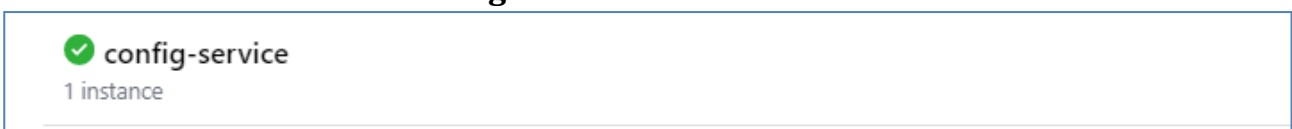
```
customer-service-prod.properties
1 customer.params.x = 9
2 customer.params.y = 6
3 |
```

#### 6. Configuration de service de configuration crée :

On va stocker la configuration de l'ensemble de microservices dans (repository git en local), Quand le service de configuration démarre va chercher dans le repository où se trouve la configuration de l'ensemble de microservices.

```
application.properties
1 server.port=8888
2 spring.application.name=config-service
3 spring.cloud.config.server.git.uri=file:///E:/Desktop/Important/ENSET/Activite_3/ecom-enet/configuration-repo
4 |
```

#### 7. Démarrer le service de configuration :



## PARTIE 2 : Création de Gateway

## 1. Configuration automatique de Gateway :

```
GatewayServiceApplication.java
10 @SpringBootApplication
11 public class GatewayServiceApplication {
12
13     no usages
14     public static void main(String[] args) { SpringApplication.run(GatewayServiceApplication.class, args); }
15
16     //Configuration dynamique
17     no usages
18     @Bean
19     DiscoveryClientRouteDefinitionLocator dynamicRoutes(ReactiveDiscoveryClient rdc,
20                                                         DiscoveryLocatorProperties dlp){
21         return new DiscoveryClientRouteDefinitionLocator(rdc, dlp);
22     };
23 }
```

## 2. Configuration automatique de Gateway :

```
1 server.port=9999
2 spring.application.name=gateway-service
3 spring.config.import=optional:configserver:http://localhost:8888
4
```

## PARTIE 3 : Création de customer-service

### 1. Structure du projet:

```
customer-service
├── .mvn
├── src
│   ├── main
│   │   ├── java
│   │   │   ├── org.sid.customerservice
│   │   │   │   ├── entities
│   │   │   │   │   ├── Customer
│   │   │   │   │   ├── CustomerProjection
│   │   │   │   │   └── CustomerRepository
│   │   │   │   ├── repo
│   │   │   │   │   └── CustomerRepository
│   │   │   │   ├── web
│   │   │   │   │   ├── CustomerConfigTestController
│   │   │   │   │   └── CustomerServiceApplication
│   │   │   └── resources
│   │   │       └── application.properties
│   └── test
```

### 2. Configuration de service :

Le service customer-service va chercher sa configuration dans le service de configuration config-service qui a le port 8888

```
application.properties x
1 server.port=8081
2 spring.application.name=customer-service
3 spring.config.import=optional:configserver:http://localhost:8888
4 |
```

### 3. Entité de Customer:

```
13 @Entity
14 @Data @NoArgsConstructor @AllArgsConstructor @Builder
15 public class Customer {
16     no usages
17     @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
18     private Long id;
19     no usages
20     private String name;
21     no usages
22     private String email;
23 }
```

### 4. Repository de customer-service:

```
8 @RepositoryRestResource
9 public interface CustomerRepository extends JpaRepository<Customer, Long> {
10 }
11 |
```

### 5. Récupérer la configuration de customer-service :

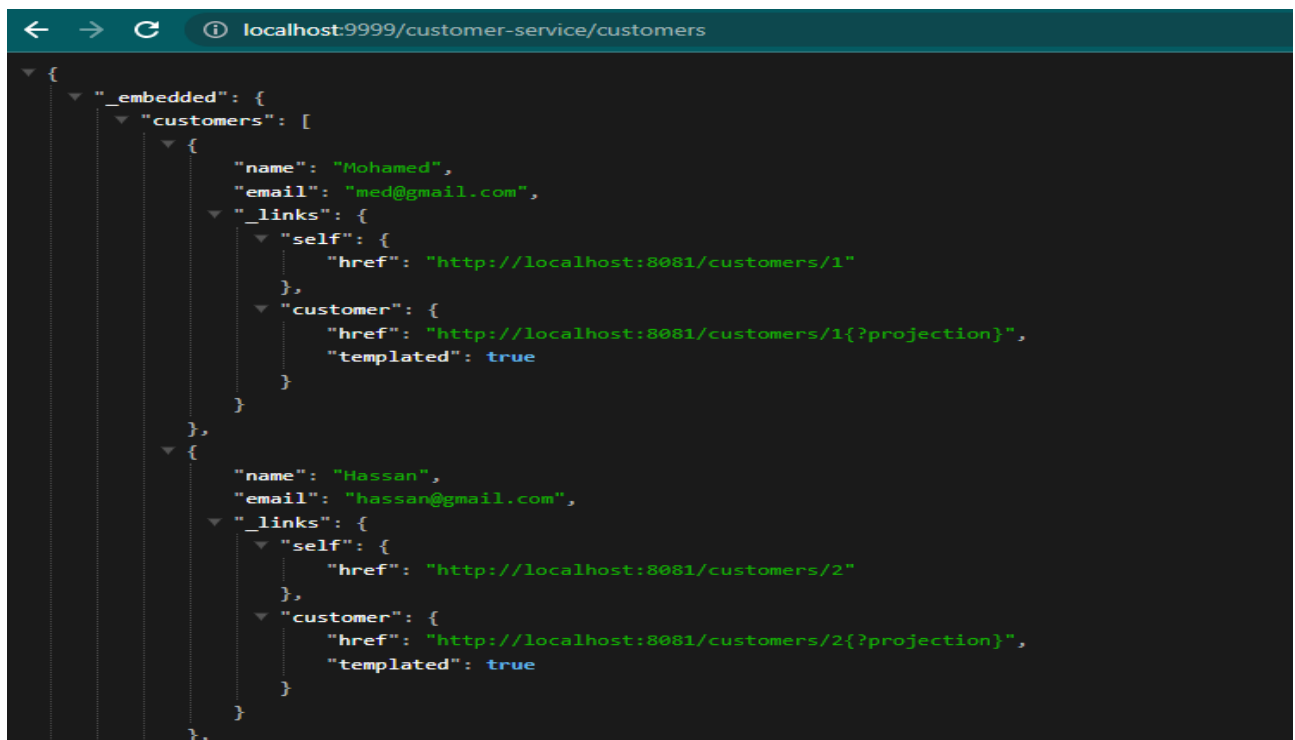
```
10 @RestController
11 @RefreshScope
12 public class CustomerConfigTestController {
13
14     @Value("${global.params.p1}")
15     private String p1;
16
17     @Value("${global.params.p2}")
18     private String p2;
19
20     @Value("${customer.params.x}")
21     private String x;
22
23     @Value("${customer.params.y}")
24     private String y;
25 }
```

```
localhost:9999/customer-service/params
{
  "y": "888",
  "p2": "22",
  "p1": "15",
  "x": "999"
}
```



## 6. Tester le customer-service :

```
12 @SpringBootApplication
13 ▶ public class CustomerServiceApplication {
14
15 ▶     no usages
16     public static void main(String[] args) {
17         SpringApplication.run(CustomerServiceApplication.class, args);
18     }
19     no usages
20     @Bean
21     CommandLineRunner start(CustomerRepository customerRepository){
22         return args -> {
23             customerRepository.saveAll(List.of(
24                 Customer.builder().name("Mohamed").email("med@gmail.com").build(),
25                 Customer.builder().name("Hassan").email("hassan@gmail.com").build(),
26                 Customer.builder().name("Imane").email("imane@gmail.com").build()
27             ));
28             customerRepository.findAll().forEach(System.out::println);
29         };
30     }
31 }
```



```
{
  "_embedded": {
    "customers": [
      {
        "name": "Mohamed",
        "email": "med@gmail.com",
        "_links": {
          "self": {
            "href": "http://localhost:8081/customers/1"
          },
          "customer": {
            "href": "http://localhost:8081/customers/1?projection",
            "templated": true
          }
        }
      },
      {
        "name": "Hassan",
        "email": "hassan@gmail.com",
        "_links": {
          "self": {
            "href": "http://localhost:8081/customers/2"
          },
          "customer": {
            "href": "http://localhost:8081/customers/2?projection",
            "templated": true
          }
        }
      }
    ]
  }
}
```

## PARTIE 4 : Création de inventory-service

### 1. Configuration de service :

Le service inventory-service va chercher sa configuration dans le service de configuration config-service qui a le port 8888

```
1 server.port=8082
2 spring.application.name=inventory-service
3 spring.config.import=optional:configserver:http://localhost:8888
4
```

## 2. Entité de Product :

```
@Entity
@Data @NoArgsConstructor @AllArgsConstructor @Builder
public class Product {
    no usages
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    no usages
    private String name;
    no usages
    private double price;
    no usages
    private int quantity;
}
```

## 3. Projection de inventory -service:

```
5 @Projection(name = "fullProduct", types = Product.class)
6 public interface ProductProjection {
7     public Long getId();
8     public String getName();
9     public double getPrice();
10    public int getQuantity();
11 }
```

## 4. Tester le inventory -service :

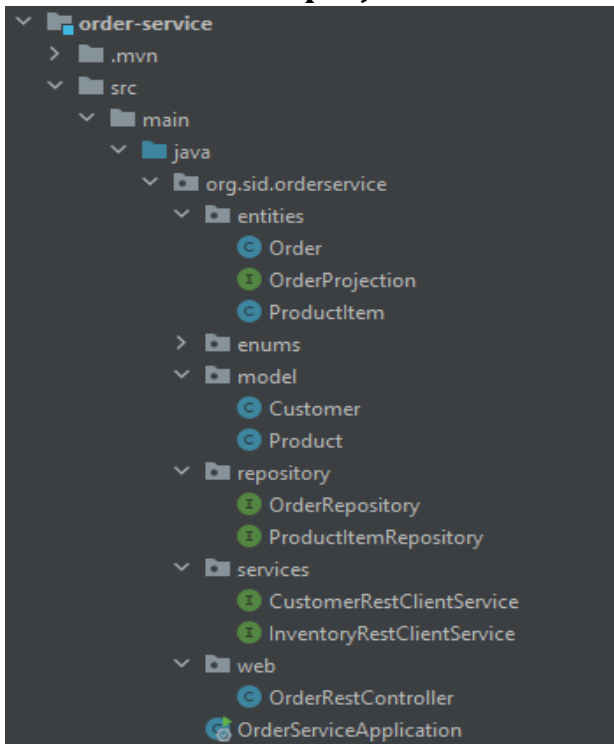
```
22 @Bean
23 CommandLineRunner start(ProductRepository productRepository){
24     return args -> {
25         Random random = new Random();
26         for (int i=1 ; i<10 ; i++){
27             productRepository.saveAll(List.of(
28                 Product.builder()
29                     .name("Computer"+ i)
30                     .price(1200+Math.random()*10000)
31                     .quantity(1+random.nextInt( bound: 200))
32                     .build()
33             ));
34         }
35     };
36 }
37 }
```

```
localhost:9999/inventory-service/products?projection=fullProduct/

{
  "_embedded": {
    "products": [
      {
        "name": "Computer1",
        "price": 11163.430444692056,
        "quantity": 181,
        "_links": {
          "self": {
            "href": "http://localhost:8082/products/1"
          },
          "product": {
            "href": "http://localhost:8082/products/1{?projection}",
            "templated": true
          }
        }
      },
      {
        "name": "Computer2",
        "price": 7846.667382509763,
        "quantity": 188,
        "_links": {
          "self": {
            "href": "http://localhost:8082/products/2"
          },
          "product": {
            "href": "http://localhost:8082/products/2{?projection}",
            "templated": true
          }
        }
      }
    ]
  }
}
```

## PARTIE 5 : Création de order-service

### 1. Structure du projet :



### 2. Configuration de service :

Le service order-service va chercher sa configuration dans le service de configuration config-service qui a le port 8888

```
1  server.port=8083
2  spring.application.name=order-service
3  spring.config.import=optional:configserver:http://localhost:8888
4  logging.level.org.sid.orderservice.services.CustomerRestClientService=debug
5  logging.level.org.sid.orderservice.services.InventoryRestClientService=debug
6  feign.client.config.default.loggerLevel=full
7
```

Sa configuration dans le répertoire de configuration :

```
1  order.params.c = 9
2  order.params.d = 252
3  spring.datasource.url=jdbc:h2:mem:order-db
4  spring.h2.console.enabled=true
```

### 3. Entité de Order:

```
14 @Entity
15 @Table(name = "orders")
16 @Data @AllArgsConstructor @NoArgsConstructor @Builder
17 public class Order {
18     no usages
19     @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
20     private Long id;
21     no usages
22     private Date createdAt;
23     no usages
24     private OrderStatus status;
25     no usages
26     private Long customerId;
27     no usages
28     @Transient
29     private Customer customer;
30     1 usage
31     @OneToMany(mappedBy = "order")
32     private List<ProductItem> productItems;
```

### 4. Entité ProductItem:

```
12 @Entity
13 @Data @AllArgsConstructor @NoArgsConstructor @Builder
14 public class ProductItem {
15     no usages
16     @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
17     private Long id;
18     no usages
19     private Long productId;
20     no usages
21     @Transient
22     private Product product;
23     1 usage
24     private double price;
25     1 usage
26     private int quantity;
27     1 usage
28     private double discount;
29     no usages
30     @ManyToOne
31     @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
32     private Order order;
```

## 5. Les modules utilisés dans order-service :

```
Customer.java x
1 package org.sid.orderservice.model;
2
3 import lombok.Data;
4
5 @Data
6 public class Customer {
7     no usages
8     private Long id;
9     no usages
10    private String name;
11    no usages
12    private String email;
13 }
14
```

```
Customer.java x Product.java x
1 package org.sid.orderservice.model;
2
3 import lombok.Data;
4
5 @Data
6 public class Product {
7     no usages
8     private Long id;
9     no usages
10    private String name;
11    no usages
12    private double price;
13    no usages
14    private int quantity;
15 }
16
```

## 6. Communiquer le order-service avec customer-service avec OpenFeign :

```
1 package org.sid.orderservice.services;
2
3 import ...
4
5 @FeignClient(name = "customer-service")
6 public interface CustomerRestClientService {
7
8     @GetMapping("/customers/{id}?projection=fullCustomer")
9     public Customer customerById(@PathVariable Long id);
10
11     //List Customers
12
13     @GetMapping("/customers?projection=fullCustomer")
14     public PagedModel<Customer> allCustomers();
15 }
16
```

## 7. Récupérer le customer et la liste des produits d'ordre :

```
14 @RestController
15 public class OrderRestController {
16
17     2 usages
18     private OrderRepository orderRepository;
19     1 usage
20     private ProductItemRepository productItemRepository;
21     2 usages
22     private CustomerRestClientService customerRestClientService;
23     2 usages
24     private InventoryRestClientService inventoryRestClientService;
25     no usages
26     @GetMapping("/{id}")
27     public Order getOrder(@PathVariable Long id){
28         Order order = orderRepository.findById(id).get();
29         Customer customer= customerRestClientService.customerById(order.getCustomerId());
30         order.setCustomer(customer);
31         order.getProductItems().forEach(pi -> {
32             Product product= inventoryRestClientService.productById(pi.getProductId());
33             pi.setProduct(product);
34         });
35         return order;
36     }
37 }
```

## 8. Tester le order-service :

```
public class OrderServiceApplication {
    no usages
    public static void main(String[] args){ SpringApplication.run(OrderServiceApplication.class, args); }
    //Des interactions Rest avec le client
    no usages
    @Bean
    CommandLineRunner start(
        OrderRepository orderRepository,
        ProductItemRepository productItemRepository,
        CustomerRestClientService customerRestClientService,
        InventoryRestClientService inventoryRestClientService){
        return args -> {
            List<Customer> customers= customerRestClientService.allCustomers().getContent().stream().toList();
            List<Product> products= inventoryRestClientService.allProducts().getContent().stream().toList();
            Long customerId = 1L; //Créer un order pour un client précis
            Random random = new Random();
            Customer customer= customerRestClientService.customerById(customerId);
            //Créer un order et chercher le client
            for (int i = 0; i < 20; i++) {
                Order order= Order.builder()
                    .customerId(customers.get(random.nextInt(customers.size())).getId())
                    .status(Math.random()>0.5? OrderStatus.PENDING: OrderStatus.CREATED)
                    .createdAt(new Date())
                    .build();
            }
        };
    }
}
```

```
localhost:9999/order-service/fullOrder/1

{
  "id": 1,
  "createdAt": "2022-12-24T18:34:08.309+00:00",
  "status": "CREATED",
  "customerId": 3,
  "customer": {
    "id": 3,
    "name": "Imane",
    "email": "imane@gmail.com"
  },
  "productItems": [
    {
      "id": 1,
      "productId": 1,
      "product": {
        "id": 1,
        "name": "Computer1",
        "price": 11163.430444692056,
        "quantity": 181
      },
      "price": 11163.430444692056,
      "quantity": 3,
      "discount": 0.48761997704537996,
      "amount": 17159.75624251086
    },
    {
      "id": 2,
      "productId": 3,
      "product": {
        "id": 3,
        "name": "Computer3",

```

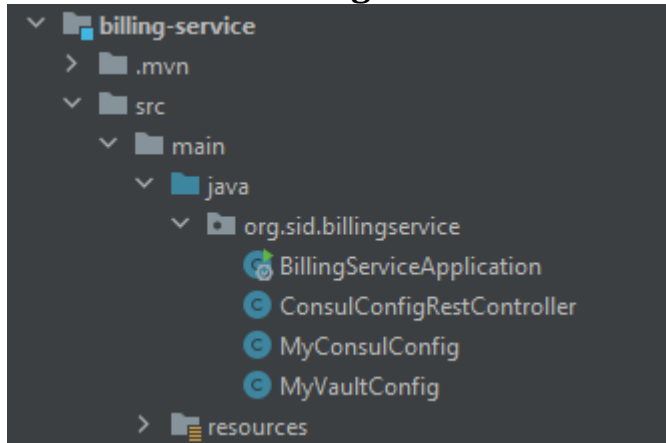
## PARTIE 6 : Création de billing-service

### 1. Dépendances :

```
20 <dependencies>
21 <dependency>
22 <groupId>org.springframework.boot</groupId>
23 <artifactId>spring-boot-starter-actuator</artifactId>
24 </dependency>
25 <dependency>
26 <groupId>org.springframework.boot</groupId>
27 <artifactId>spring-boot-starter-web</artifactId>
28 </dependency>
29 <dependency>
30 <groupId>org.springframework.cloud</groupId>
31 <artifactId>spring-cloud-starter-consul-config</artifactId>
32 </dependency>
33 <dependency>
34 <groupId>org.springframework.cloud</groupId>
35 <artifactId>spring-cloud-starter-consul-discovery</artifactId>
36 </dependency>
37 <dependency>
38 <groupId>org.springframework.cloud</groupId>
39 <artifactId>spring-cloud-starter-vault-config</artifactId>
40 </dependency>
41 <dependency>
42 <groupId>org.projectlombok</groupId>
```



## 2. Structure de billing-service :



## 3. Configuration de billing-service :

```
1  spring.application.name=billing-service
2  server.port=8084
3  spring.cloud.vault.token=hvs.YyT0fgt5SU9ewWwp3oIIm49J
4  spring.cloud.vault.scheme=http
5  spring.cloud.vault.kv.enabled=true
6  spring.config.import=optional:consul:, vault://
7  management.endpoints.web.exposure.include=*
```

## 4. Partager le secret avec Vault :

```
C:\ProgramData\vault>vault server -dev
==> Vault server configuration:

Api Address: http://127.0.0.1:8200
```

```
2022-12-24T20:33:04.939+0100 [INFO] secrets.kv.kv_48ecf246: done collecting keys: num
_keys=1
2022-12-24T20:33:04.939+0100 [INFO] secrets.kv.kv_48ecf246: upgrading keys finished
WARNING! dev mode is enabled! In this mode, Vault runs entirely in-memory
and starts unsealed with a single unseal key. The root token is already
authenticated to the CLI, so you can immediately begin using Vault.

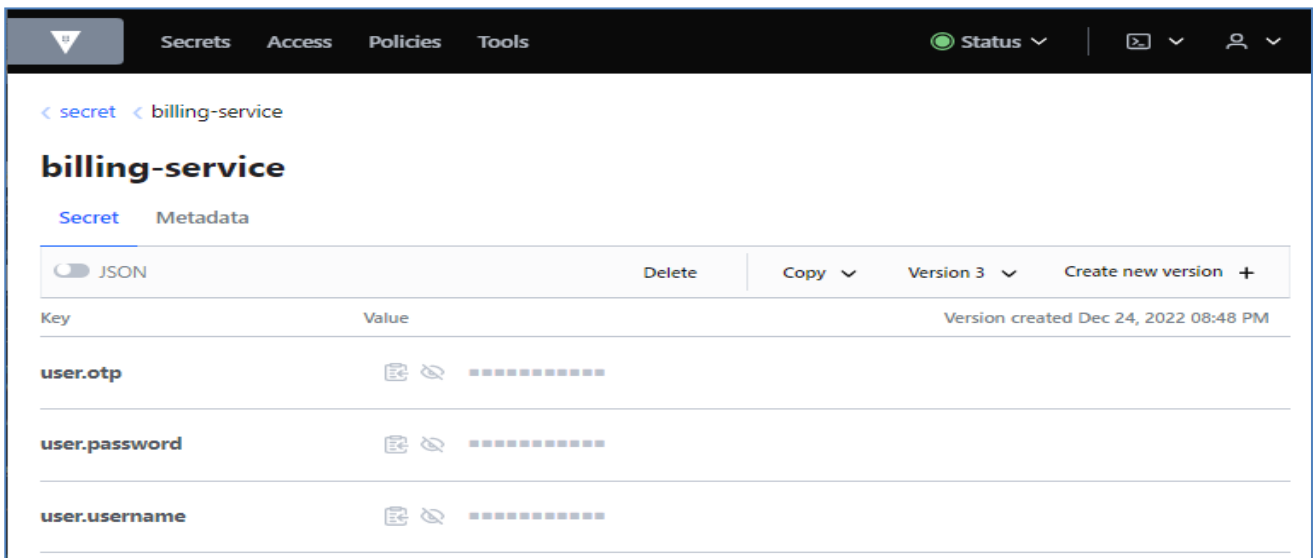
You may need to set the following environment variables:

PowerShell:
    $env:VAULT_ADDR="http://127.0.0.1:8200"
cmd.exe:
    set VAULT_ADDR=http://127.0.0.1:8200

The unseal key and root token are displayed below in case you want to
seal/unseal the Vault or re-authenticate.

Unseal Key: 9T+CYvgSRlrNDA+ybhgryZh8BJvK26nwZh05nZ4VmZA=
Root Token: hvs.YyT0fgt5SU9ewWwp3oIIm49J

Development mode should NOT be used in production installations!
```



Les paramètres de configuration dans Consul Config

```

6      @Component
7      @ConfigurationProperties(prefix = "user")
8      @Data
9      public class MyVaultConfig {
10         no usages
11         private String username;
12         no usages
13         private String password;
14         no usages
15         private String otp;
16     }

```

Contrôleur de test des secrets :

```

10     @RestController
11     // @RefreshScope //refresher la configuration a chaque fois elle change
12     public class ConsulConfigRestController {
13
14         1 usage
15         @Autowired
16         private MyConsulConfig myConsulConfig;
17
18         1 usage
19         @Autowired
20         private MyVaultConfig myVaultConfig;
21         no usages
22         @GetMapping("/myConfig")
23         public Map<String, Object> myConfig(){
24             return Map.of( k1: "consulConfig", myConsulConfig, k2: "vaultConfig", myVaultConfig);
25         }
26     }

```

```

localhost:8084/myConfig
{
  "consulConfig": {
    "accessTokenTimeout": 54000,
    "refreshTokenTimeout": 540000
  },
  "vaultConfig": {
    "username": "mohammed",
    "password": "123456",
    "otp": "998877"
  }
}

```

Dans cette situation le microservice billing-service le seul qui a le secret, il veut le partager avec les autres microservices mais il ne fait pas confiance, c'est pour cela, il va le donner à Vault, c'est comme ça il est sûr que seules les microservices qui ont droit d'accès à Vault qui peuvent le voir comme ça on partage les secrets.

```

12  @SpringBootApplication
13  public class BillingServiceApplication {
14
15      1 usage
16      @Autowired
17      private VaultTemplate vaultTemplate;
18
19      no usages
20      public static void main(String[] args) { SpringApplication.run(BillingServiceApplication.class, args); }
21
22      no usages
23      @Bean
24      CommandLineRunner commandLineRunner(){
25          return args -> {
26              vaultTemplate.opsForVersionedKeyValue( path: "secret")
27                  .put( s: "keypair", Map.of( k1: "privkey", v1: "54321", k2: "pubkey", v2: "8999"));
28          };
29      }

```

[secret](#) < [keypair](#)

## keypair

[Secret](#)
[Metadata](#)

☐ JSON
 Delete
 Copy
 Version 2
 Create new version +

Key	Value	Version created Dec 24, 2022 08:57 PM
privkey	.....	
pubkey	.....	

## PARTIE 7 : Frontend avec Angular

### 1. Liste des produits :

Products	Customers	Dropdown		
ID	Name	Price	Quantity	
1	Computer1	11163.430444692056	181	
2	Computer2	7846.667382509763	188	
3	Computer3	10817.315581346962	4	
4	Computer4	8804.903382213186	68	
5	Computer5	9115.636912110962	170	
6	Computer6	1610.905540511295	136	
7	Computer7	4910.835645693165	136	
8	Computer8	1319.87294487101	127	
9	Computer9	4246.89683177993	160	

## 2. Liste des customer :

Products Customers Dropdown ▾

ID	Name	Email	
1	Mohamed	med@gmail.com	Orders
2	Hassan	hassan@gmail.com	Orders
3	Imane	imane@gmail.com	Orders

## 3. Liste des order de customer 1 :

Products Customers Dropdown ▼

ID	Date	Status	CustomerId	
5	24/12/2022	PENDING	1	Orders Details
8	24/12/2022	PENDING	1	Orders Details
15	24/12/2022	PENDING	1	Orders Details

## 4. Détails de order 5 :

Products Customers Dropdown

Order ID: 5

Order Id : 5

Date : 24/12/2022

Status : PENDING

Customer Id : 1

Customer Name : Mohamed

Customer Email : med@gmail.com

Product Items

Product Id	Product Name	Quantity	Price	Discount	Amount
2	Computer2	3	7,846.667	0.359	15,093.80
4	Computer4	1	8,804.903	0.595	3,566.671
9	Computer9	3	4,246.897	0.516	6,160.82
Total					24,821.29