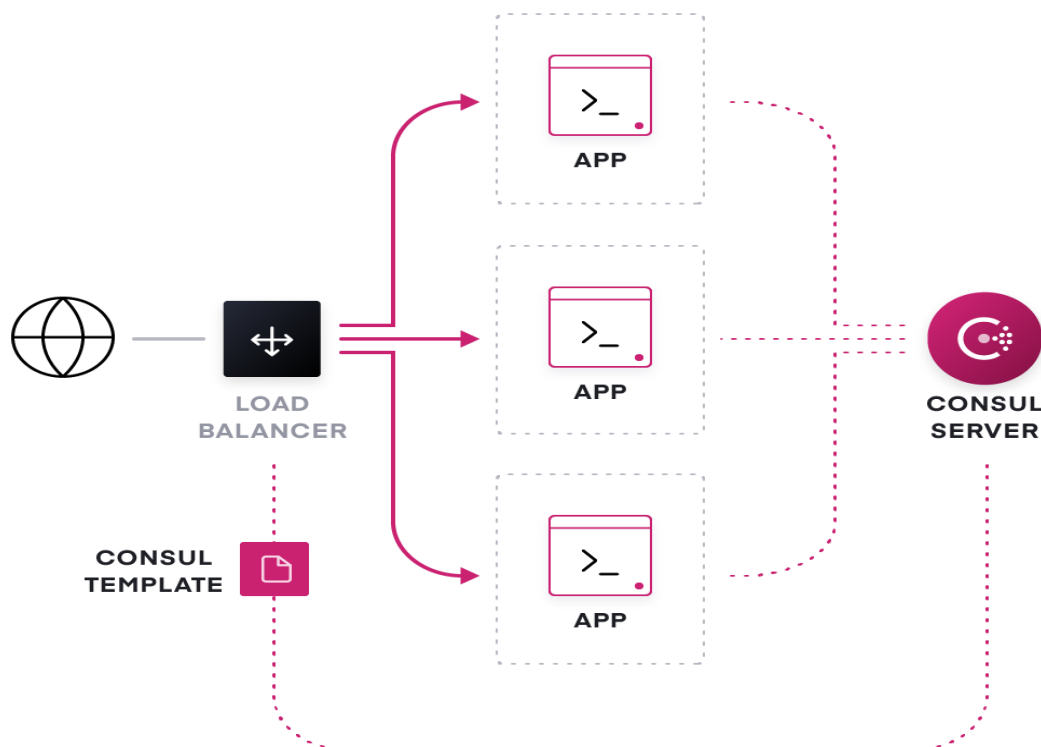


## COMPTE-RENDU : ACTIVITE PRATIQUE N° 3 - ARCHITECTURES MICRO SERVICES AVEC (SPRING CLOUD CONFIG, CONSUL DISCOVERY, CONSUL CONFIG, VAULT)

Filière : « Ingénierie Informatique : Big Data et Cloud  
Computing » II-BDCC



Réalisé par :

Khadija BENJILALI

Encadré par :

Pr. Mohamed YOUSSEFI

**Année Universitaire : 2022-2023**

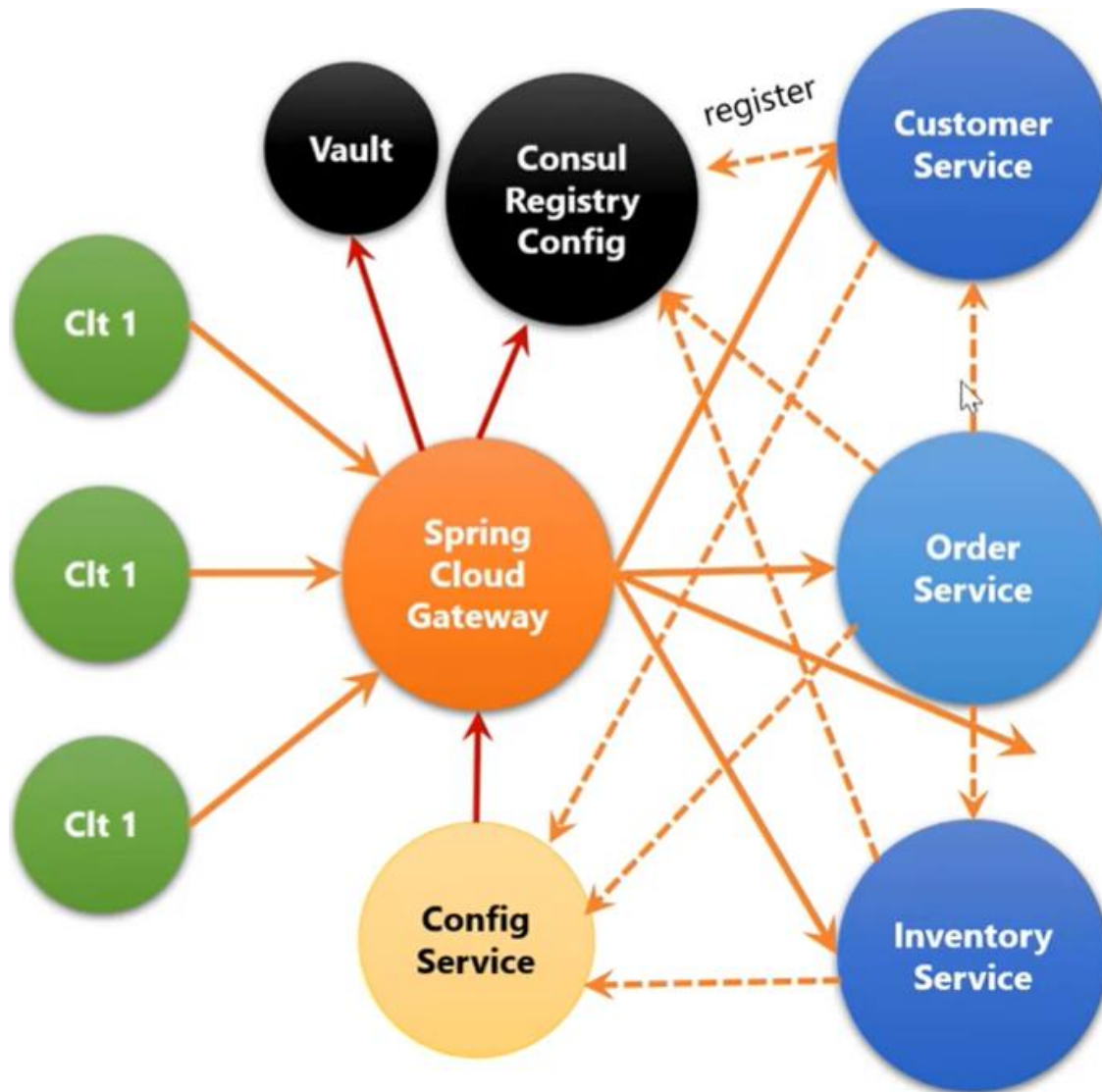
# Sommaire

Travail à faire.....	1
<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 :.....	3
4. Configuration partagée par les microservices : .....	3
5. Exemple de configuration (Customer-service) : .....	3
6. Configuration de service de configuration crée : .....	4
7. Démarrer le service de configuration : .....	4
<b>PARTIE 2 : Création de Gateway .....</b>	<b>4</b>
1. Configuration automatique de Gateway : .....	4
2. Propriétés de configuration de Gateway : .....	4
<b>PARTIE 3 : Création de customer-service .....</b>	<b>5</b>
1. Structure du projet : .....	5
2. Configuration de service : .....	5
3. Entité de Customer:.....	5
4. Repository de customer-service: .....	5
5. Récupérer la configuration de customer-service : .....	6
6. Tester le customer-service : .....	6
<b>PARTIE 4 : Création de inventory-service .....</b>	<b>7</b>
1. Configuration de service : .....	7
2. Entité de Product : .....	8
3. Projection de inventory -service:.....	8
4. Tester le inventory -service : .....	8
<b>PARTIE 5 : Création de order-service .....</b>	<b>10</b>
1. Structure du projet:.....	10
2. Configuration de service : .....	10
3. Entité de Order: .....	11
4. Entité ProductItem: .....	11
5. Les modules utilisés dans order-service : .....	11
6. Communiquer le order-service avec customer-service avec OpenFeign : .....	12
7. Récupérer le customer et la liste des produits d' order : .....	12
8. Tester le order-service : .....	13
<b>PARTIE 6 : Création de billing-service .....</b>	<b>14</b>

1. Dépendances :	14
2. Structure de billing-service :	14
3. Configuration de billing-service :	15
4. Partager le secret avec Vault :	15
<b>PARTIE 7 : Frontend avec Angular</b>	<b>17</b>
1. Liste des produits :	17
2. Liste des customer :	18
3. Liste des order de customer 1 :	19
4. Détails de order 3 :	19

## Travail à faire

## Créer une application de e-commerce basée sur les micro services :

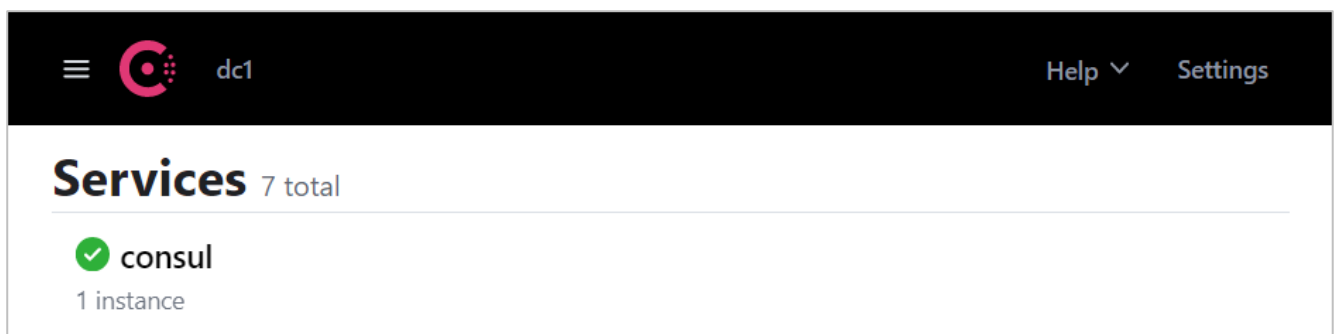


1. Consul Discovery
2. Spring Cloud Config
3. Spring Cloud Gateway
4. Customer-service
5. Inventory Service
6. Order Service
7. Consul Config (Billing Service)
8. Vault (Billing Service)
9. Frontend Web avec Angular

## 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=192.168.0.106
==> Starting Consul agent...
    Version: '1.13.3'
    Build Date: '2022-10-19 19:49:59 +0000 UTC'
    Node ID: '84c1732c-e84f-c316-8f31-2f72ea116473'
    Node name: 'DESKTOP-N5BFQBK'
    Datacenter: 'dc1' (Segment: '<all>')
    Server: true (Bootstrap: true)
    Client Addr: [127.0.0.1] (HTTP: 8500, HTTPS: -1, gRPC: -1, DNS: 8600)
    Cluster Addr: 192.168.0.106 (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:
```



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

```
8  @SpringBootApplication
9  @EnableConfigServer
10 @EnableDiscoveryClient
11 public class ConfigServiceApplication {
12
13     public static void main(String[] args) {
14         SpringApplication.run(ConfigServiceApplication.class, args);
15     }
}
```

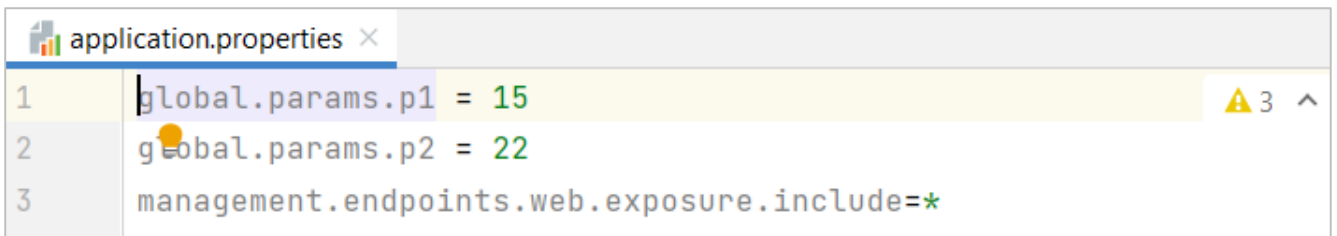
### 3. Création d'un dossier contient les fichiers de configuration des ms :



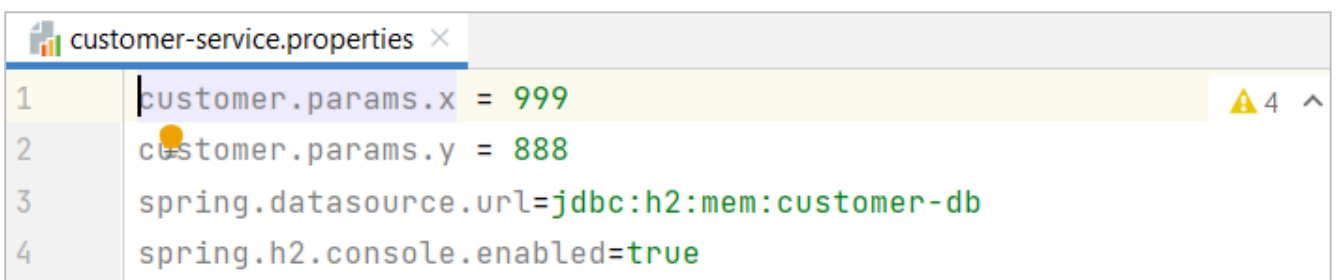
Ajouter et faire un commit chaque fois on change dans les propriétés de configuration de chaque microservices.

```
PS C:\Users\dell\Desktop\tp3-application-de-ecommerce\ecom-enset\configuration-repo> git add .
PS C:\Users\dell\Desktop\tp3-application-de-ecommerce\ecom-enset\configuration-repo> git commit -m "V2"
On branch master
nothing to commit, working tree clean
PS C:\Users\dell\Desktop\tp3-application-de-ecommerce\ecom-enset\configuration-repo> █
```

### 4. Configuration partagée par les microservices :



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



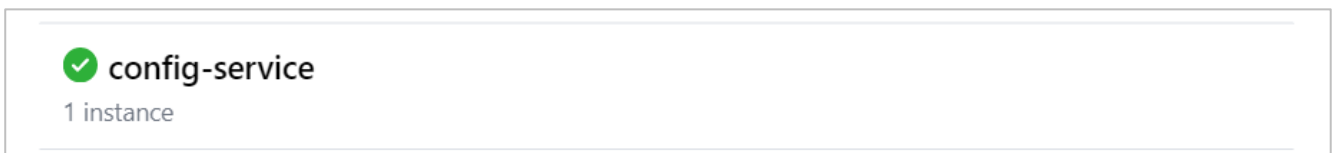
```
customer-service-prod.properties x
1 customer.params.x = 9
2 customer.params.y = 6
```

## 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 x
1 server.port=8888
2 spring.application.name=config-service
3 spring.cloud.config.server.git.uri=file:///C:/Users/dell/Desktop/tp3-application-de-ecommerce/ecom-enst/configuration-repo
4
```

## 7. Démarrer le service de configuration :



# PARTIE 2 : Création de Gateway

## 1. Configuration automatique de Gateway :

```
@SpringBootApplication
public class GatewayServiceApplication {

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

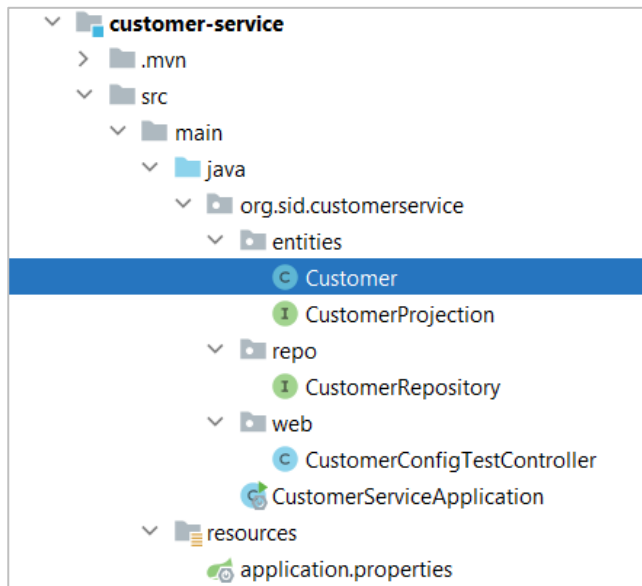
    //Configuration dynamique
    @Bean
    DiscoveryClientRouteDefinitionLocator dynamicRoutes(ReactiveDiscoveryClient rdc,
        DiscoveryLocatorProperties dlp){
        return new DiscoveryClientRouteDefinitionLocator(rdc, dlp);
    }
}
```

## 2. Propriétés de configuration de Gateway :

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

## PARTIE 3 : Création de customer-service

### 1. Structure du projet :



### 2. Configuration de service :

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

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

### 3. Entité de Customer:

```
@Entity
@Data @NoArgsConstructor @AllArgsConstructor @Builder
public class Customer {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String email;
}
```

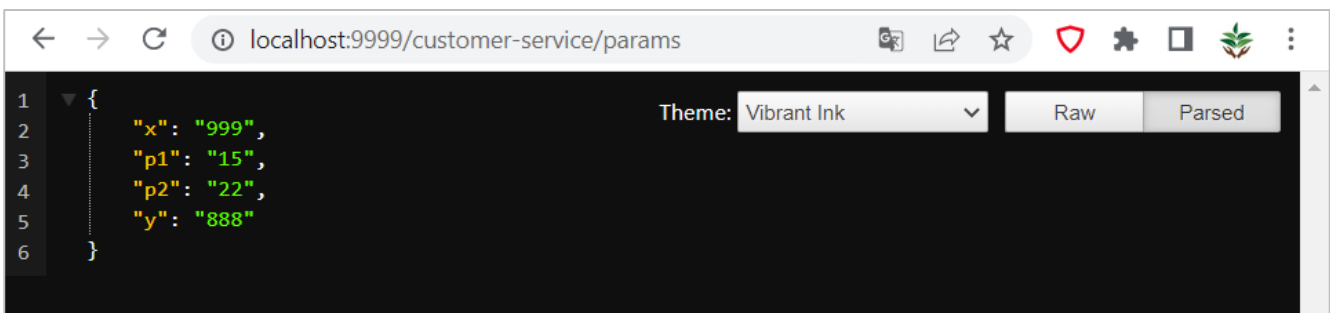
### 4. Repository de customer-service:

```
// Spring Data REST pour explorer le web service RESTFUL qui permet de gérer les customers
//pour : http://localhost:8081/customers
@RepositoryRestResource
public interface CustomerRepository extends JpaRepository<Customer, Long> {
}
```



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

```
10 @RestController
11 @RefreshScope
12 // cette annotation de Acuator indique au controleur q'elle va surveiller ses paramètres p1,p2...
13 public class CustomerConfigTestController {
14     @Value("${global.params.p1}")
15     private String p1;
16     @Value("${global.params.p2}")
17     private String p2;
18     @Value("${customer.params.x}")
19     private String x;
20     @Value("${customer.params.y}")
21     private String y;
22
23     @GetMapping("/params")
24     public Map<String,String> params() { return Map.of( k1: "p1", p1, k2: "p2", p2, k3: "x", x, k4: "y", y); }
27 }
```

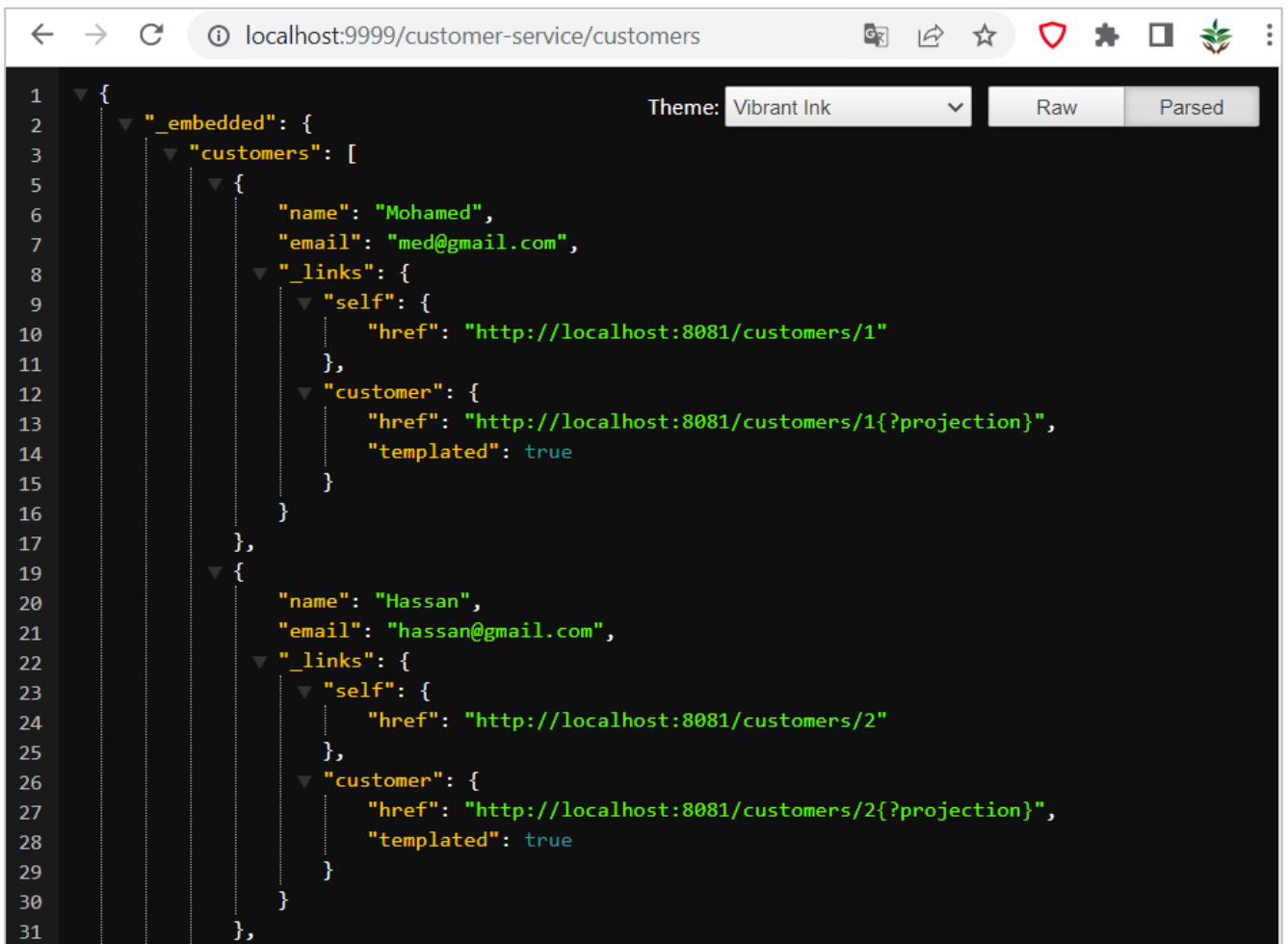


localhost:9999/customer-service/params

```
1 {
2   "x": "999",
3   "p1": "15",
4   "p2": "22",
5   "y": "888"
6 }
```

## 6. Tester le customer-service :

```
@SpringBootApplication
public class CustomerServiceApplication {
    public static void main(String[] args) { SpringApplication.run(CustomerServiceApplication.class, args); }
    @Bean
    CommandLineRunner start(CustomerRepository customerRepository){
        return args -> {
            customerRepository.saveAll(List.of(
                Customer.builder().name("Mohamed").email("med@gmail.com").build(),
                Customer.builder().name("Hassan").email("hassan@gmail.com").build(),
                Customer.builder().name("Imane").email("imane@gmail.com").build()
            ));
            customerRepository.findAll().forEach(System.out::println);
        };
    }
}
```



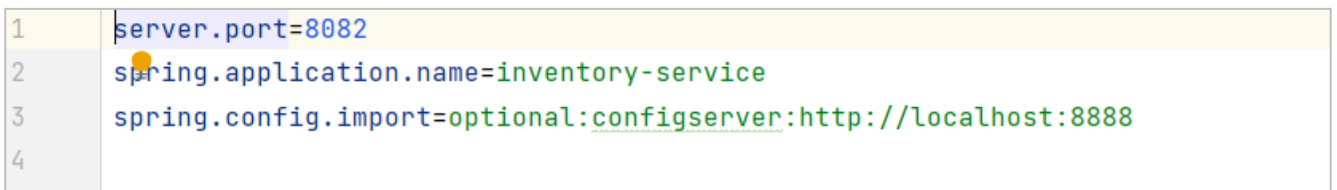
The screenshot shows a web browser at the URL `localhost:9999/customer-service/customers`. The browser's developer tools are open, displaying a JSON response in the 'Parsed' tab. The JSON structure is as follows:

```
1 {
2   "_embedded": {
3     "customers": [
4       {
5         "name": "Mohamed",
6         "email": "med@gmail.com",
7         "_links": {
8           "self": {
9             "href": "http://localhost:8081/customers/1"
10          },
11          "customer": {
12            "href": "http://localhost:8081/customers/1{?projection}",
13            "templated": true
14          }
15        }
16      },
17      {
18        "name": "Hassan",
19        "email": "hassan@gmail.com",
20        "_links": {
21          "self": {
22            "href": "http://localhost:8081/customers/2"
23          },
24          "customer": {
25            "href": "http://localhost:8081/customers/2{?projection}",
26            "templated": true
27          }
28        }
29      }
30    ]
31  }
}
```

## 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



The screenshot shows a code editor with the following configuration properties:

```
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 {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private double price;
    private int quantity;
}
```

## 3. Projection de inventory -service:

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

## 4. Tester le inventory -service :

```
@Bean
CommandLineRunner start(ProductRepository productRepository){
    return args -> {
        Random random = new Random();
        for (int i=1 ; i<10 ; i++){
            productRepository.saveAll(List.of(
                Product.builder()
                    .name("Computer"+ i)
                    .price(1200+Math.random()*10000)
                    .quantity(1+random.nextInt( bound: 200))
                    .build()
            ));
        }
    };
}
```

```

1  {
2    "_embedded": {
3      "products": [
4        {
5          "name": "Computer1",
6          "id": 1,
7          "quantity": 72,
8          "price": 2191.3330359550087,
9          "_links": {
10             "self": {
11               "href": "http://localhost:8082/products/1"
12             },
13             "product": {
14               "href": "http://localhost:8082/products/1{?projection}",
15               "templated": true
16             }
17           }
18         },
19         {
20           "name": "Computer2",
21           "id": 2,
22           "quantity": 43,
23           "price": 7761.139261905172,
24           "_links": {
25             "self": {
26               "href": "http://localhost:8082/products/2"
27             },
28             "product": {
29               "href": "http://localhost:8082/products/2{?projection}",
30               "templated": true
31             }
32           }
33         }
34       ]
35     }
36   }

```

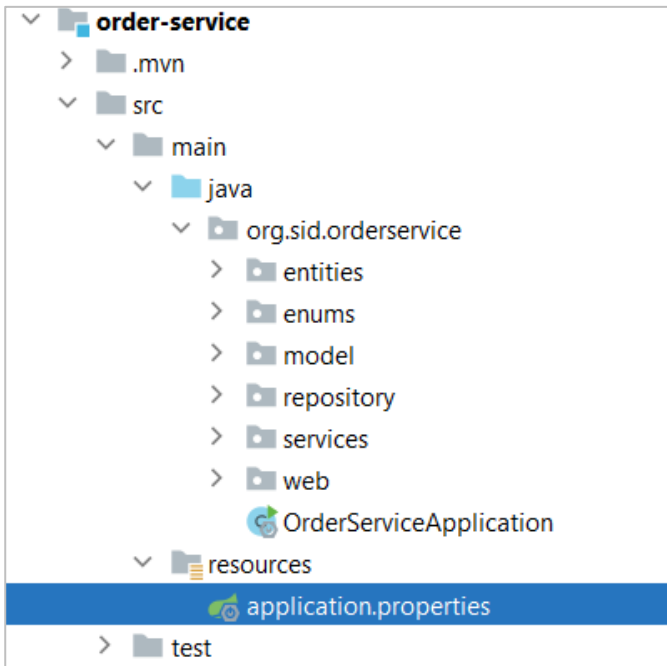
SELECT \* FROM PRODUCT;

ID	NAME	PRICE	QUANTITY
1	Computer1	2191.3330359550087	72
2	Computer2	7761.139261905172	43
3	Computer3	8404.50600085095	194
4	Computer4	1986.0542876000586	35
5	Computer5	10091.26646207548	175
6	Computer6	6075.983084696809	174
7	Computer7	2661.0998047002113	86
8	Computer8	9966.521482983058	172
9	Computer9	1595.9589564115884	1

(9 rows, 12 ms)

## 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 # pour la journalisation de OpenFeign(Afficher les messages de debug)
5 logging.level.org.sid.orderservice.services.CustomerRestClientService=debug
6 logging.level.org.sid.orderservice.services.InventoryRestClientService=debug
7 feign.client.config.default.loggerLevel=full
```

Sa configuration dans le répertoire de configuration :

```
order-service.properties x
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:

```
@Entity
@Table(name = "orders")
@Data @AllArgsConstructor @NoArgsConstructor @Builder
public class Order {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private Date createdAt;
    private OrderStatus status;
    private Long customerId;
    @Transient
    private Customer customer;
    @OneToMany(mappedBy = "order")
    private List<ProductItem> productItems;
```

### 4. Entité ProductItem:

```
@Entity
@Data @AllArgsConstructor @NoArgsConstructor @Builder
public class ProductItem {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private Long productId;
    @Transient
    private Product product;
    private double price;
    private int quantity;
    private double discount;
    @ManyToOne
    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    private Order order;
```

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

```
Customer.java ×
1 package org.sid.orderservice.model;
2 import lombok.Data;
3 @Data
4 public class Customer {
5     private Long id;
6     private String name;
7     private String email;
8 }
```

```

c Product.java x
1 package org.sid.orderservice.model;
2 import lombok.Data;
3 @Data
4 public class Product {
5     private Long id;
6     private String name;
7     private double price;
8     private int quantity;
9 }

```

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

```

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

```

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

```

@RestController @AllArgsConstructor
public class OrderRestController {
    |
    private OrderRepository orderRepository;
    private ProductItemRepository productItemRepository;
    private CustomerRestClientService customerRestClientService;
    private InventoryRestClientService inventoryRestClientService;

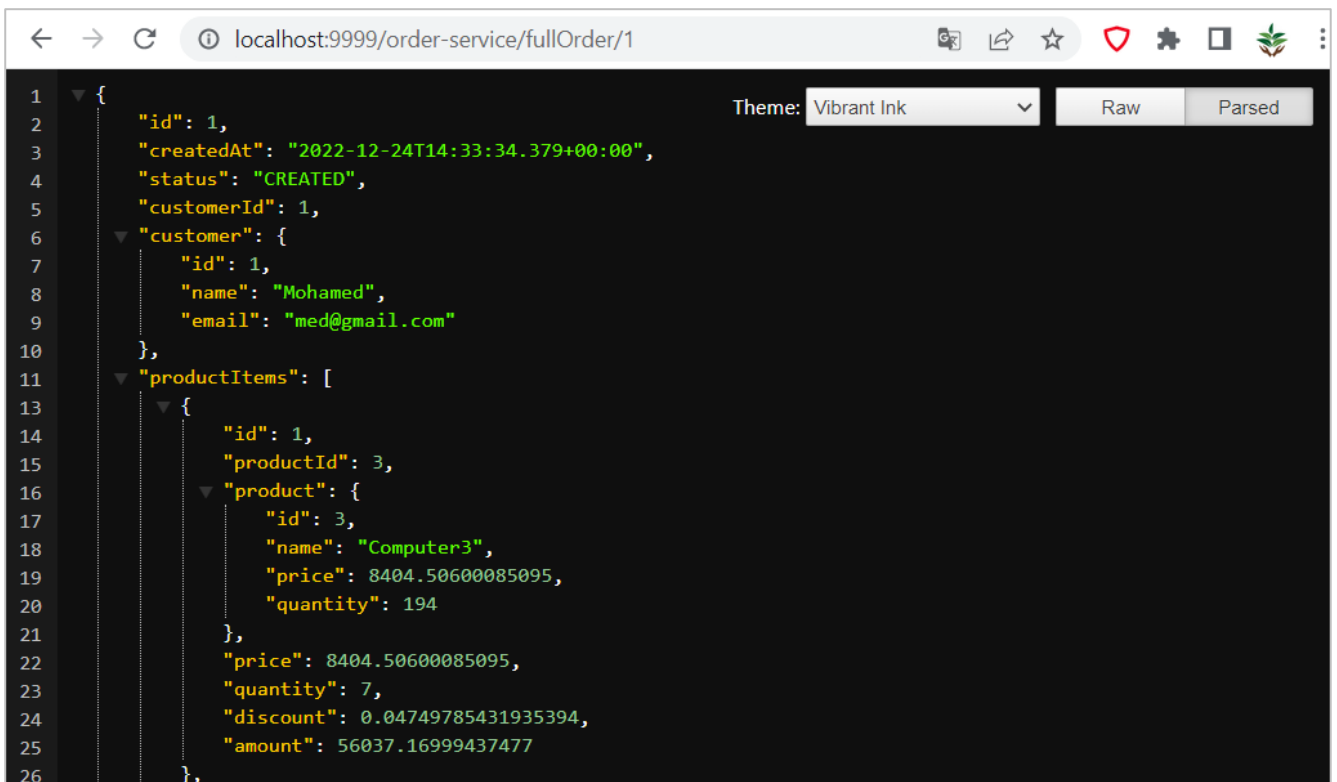
    @GetMapping("/fullOrder/{id}")
    public Order getOrder(@PathVariable Long id){
        Order order = orderRepository.findById(id).get();
        Customer customer= customerRestClientService.customerById(order.getCustomerId());
        order.setCustomer(customer);
        order.getProductItems().forEach(pi -> {
            Product product= inventoryRestClientService.productById(pi.getProductid());
            pi.setProduct(product);
        });
        return order;
    }
}

```

## 8. Tester le order-service :

```
@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; //Cr  er un order pour un client precis
        Random random = new Random();
        Customer customer= customerRestClientService.customerById(customerId);
        //Cr  er un order at attribuer    lui les produits
        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();
            Order savedOrder= orderRepository.save(order);
            for (int j = 0; j < products.size() ; j++)
            {if (Math.random()>0.70){
                ProductItem productItem= ProductItem.builder()
                    .order(savedOrder)
                    .productId(products.get(j).getId())
                    .price(products.get(j).getPrice())
                    .quantity(1+random.nextInt( bound: 10))
                    .discount(Math.random())
                    .build();
                productItemRepository.save(productItem);}
            }
        }
    }
}
```



The screenshot shows a web browser window with the address bar displaying `localhost:9999/order-service/fullOrder/1`. The browser's developer tools are open, showing a REST client interface. The response is a JSON object representing an order. The JSON structure is as follows:

```
{
  "id": 1,
  "createdAt": "2022-12-24T14:33:34.379+00:00",
  "status": "CREATED",
  "customerId": 1,
  "customer": {
    "id": 1,
    "name": "Mohamed",
    "email": "med@gmail.com"
  },
  "productItems": [
    {
      "id": 1,
      "productId": 3,
      "product": {
        "id": 3,
        "name": "Computer3",
        "price": 8404.50600085095,
        "quantity": 194
      },
      "price": 8404.50600085095,
      "quantity": 7,
      "discount": 0.04749785431935394,
      "amount": 56037.16999437477
    }
  ]
}
```

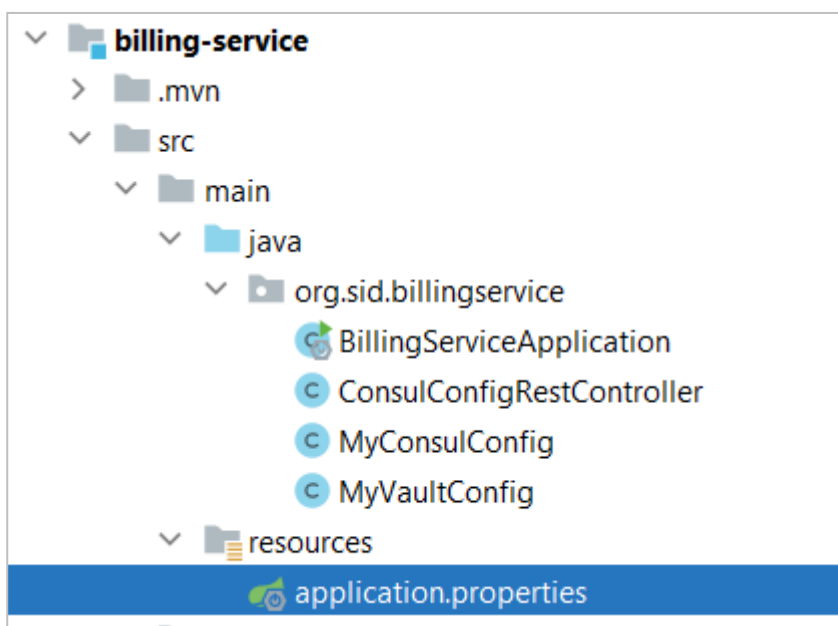


## PARTIE 6 : Création de billing-service

### 1. Dépendances :

```
<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-consul-config</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-consul-discovery</artifactId>
</dependency>
<dependency>
  <groupId>org.springframework.cloud</groupId>
  <artifactId>spring-cloud-starter-vault-config</artifactId>
</dependency>
<dependency>
  <groupId>org.projectlombok</groupId>
  <artifactId>lombok</artifactId>
  <optional>true</optional>
</dependency>
```

### 2. Structure de billing-service :



### 3. Configuration de billing-service :

```
1 server.port=8084
2 spring.application.name=billing-service
3 spring.cloud.vault.token=hvs.wyJMD3aU71DfTGwsE4LVT6K6
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 :

```
2022-12-24T16:46:20.209+0100 [INFO] secrets.kv.kv_d6f4889e: collecting keys to upgrade
2022-12-24T16:46:20.209+0100 [INFO] secrets.kv.kv_d6f4889e: done collecting keys: num_keys=1
2022-12-24T16:46:20.209+0100 [INFO] secrets.kv.kv_d6f4889e: 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: aJ0H6CQ6egEFHCTp0aJSmXQAivbv8pGGG4meSkufNWo=
Root Token: hvs.wyJMD3aU71DfTGwsE4LVT6K6

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

[secret](#) < [billing-service](#)

## billing-service

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



☐ JSON



Delete



Copy ▾

Version 4 ▾

Create new version +

**user.otp**  
  998877

**user.password**  
  ■■■■■■■■

**user.username**  
  ■■■■■■■■

## Les paramètres de configuration dans Consul Config

```
@Component
@ConfigurationProperties(prefix = "token")
@Data
public class MyConsulConfig {
    private long accessTokenTimeout;
    private long refreshTokenTimeout;
}
```

## Les paramètres de configuration dans Vault

```
@Component
@ConfigurationProperties(prefix = "user")
@Data
public class MyVaultConfig {
    private String username;
    private String password;
    private String otp;
}
```

## Contrôleur de test des secrets :

```
@RestController
@RefreshScope //refresher la configuration a chaque fois ça change pas besoin d'utiliser actuator
public class ConsulConfigRestController {

    @Autowired
    private MyConsulConfig myConsulConfig;

    @Autowired
    private MyVaultConfig myVaultConfig;

    //@Value("${token.accessTokenTimeout}")
    //private long accessTokenTimeout;
    //@Value("${token.refreshTokenTimeout}")
    //private long refreshTokenTimeout;

    @GetMapping("/myConfig") // Object soit MyConsulConfig ou MyVaultConfig
    public Map<String, Object> myConfig(){
        return Map.of( k1: "consulConfig", myConsulConfig, k2: "vaultConfig", myVaultConfig);
    }
}
```

```

1  {
2    "consulConfig": {
3      "accessTokenTimeout": 11111,
4      "refreshTokenTimeout": 540000
5    },
6    "vaultConfig": {
7      "username": "khadija",
8      "password": "1234567",
9      "otp": "998877"
10   }
11 }

```

Theme: Vibrant Ink   Raw   Parsed

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.

```

@Bean
CommandLineRunner commandLineRunner(){
    return args -> {
        // Créer des secret => reponse de type Metadata
        vaultTemplate.opsForVersionedKeyValue( path: "secret")
            .put( s: "keypair", Map.of( k1: "privkey", v1: "54321", k2: "pubkey", v2: "8999"));
    };
}

```



< secret < keypair

## keypair

Secret   Metadata



☒ JSON   Delete   Copy ▾   Version 1 ▾   Create new version +

**privkey**

  ■■■■■■■■

---

**pubkey**

  ■■■■■■■■

## PARTIE 7 : Frontend avec Angular

### 1. Liste des produits :

<a href="#">←</a> <a href="#">→</a> <a href="#">↻</a> <span>localhost:4200/products</span> <span>🔍</span> <span>🔖</span> <span>🛡️</span> <span>⚙️</span> <span>🖱️</span> <span>🌱</span> <span>⋮</span>			
<a href="#">Products</a> <a href="#">Customers</a> <a href="#">Dropdown ▼</a>			
ID	Name	Price	Quantity
1	Computer1	2191.3330359550087	72
2	Computer2	7761.139261905172	43
3	Computer3	8404.50600085095	194
4	Computer4	1986.0542876000586	35
5	Computer5	10091.26646207548	175
6	Computer6	6075.983084696809	174
7	Computer7	2661.0998047002113	86
8	Computer8	9966.521482983058	172
9	Computer9	1595.9589564115884	1

## 2. Liste des customer :

<a href="#">←</a> <a href="#">→</a> <a href="#">↻</a> <span>localhost:4200/customers</span> <span>🔍</span> <span>🔖</span> <span>🛡️</span> <span>⚙️</span> <span>🖱️</span> <span>🌱</span> <span>⋮</span>			
<a href="#">Products</a> <a href="#">Customers</a> <a href="#">Dropdown ▼</a>			
ID	Name	Email	
1	Mohamed	med@gmail.com	<a href="#">Orders</a>
2	Hassan	hassan@gmail.com	<a href="#">Orders</a>
3	Imane	imane@gmail.com	<a href="#">Orders</a>

### 3. Liste des order de customer 1 :

Products Customers Dropdown ▼

ID	Date	Status	CustomerId	
3	24/12/2022	PENDING	1	Orders Details
4	24/12/2022	PENDING	1	Orders Details
6	24/12/2022	CREATED	1	Orders Details
10	24/12/2022	CREATED	1	Orders Details

### 4. Détails de order 3 :

←	→	↺	🔍 localhost:4200/order-details/3	🔗	🌟	🛡️	⚙️	📱	🌿	⋮
Products	Customers	Dropdown ▼								
Order ID: 3										
Order Id : 3										
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					
3	Computer3	3	8,404.506	0.456	13,705.598					
5	Computer5	7	10,091.266	0.20	56,499.57					
6	Computer6	7	6,075.983	0.635	15,511.429					
Total					85,716.597					