

DEPARTEMENT MATHÉMATIQUES ET INFORMATIQUE

Compte rendu

Filière :

« Ingénierie Informatique : Big Data et Cloud Computing »

II-BDCC

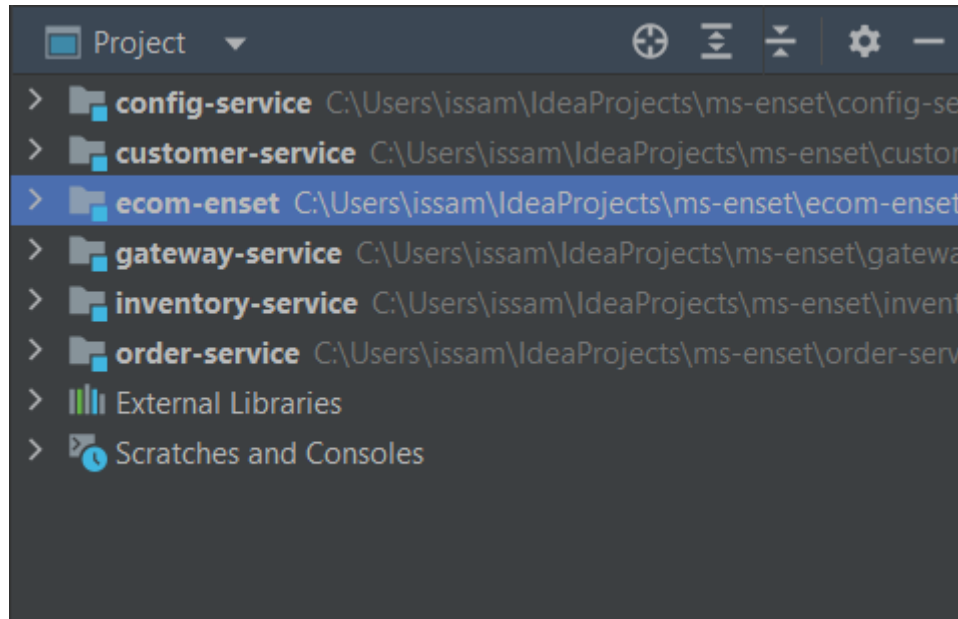
Activité pratique N°3

Réalisé par :

Issam AZEHAF

Année Universitaire : 2022-2023

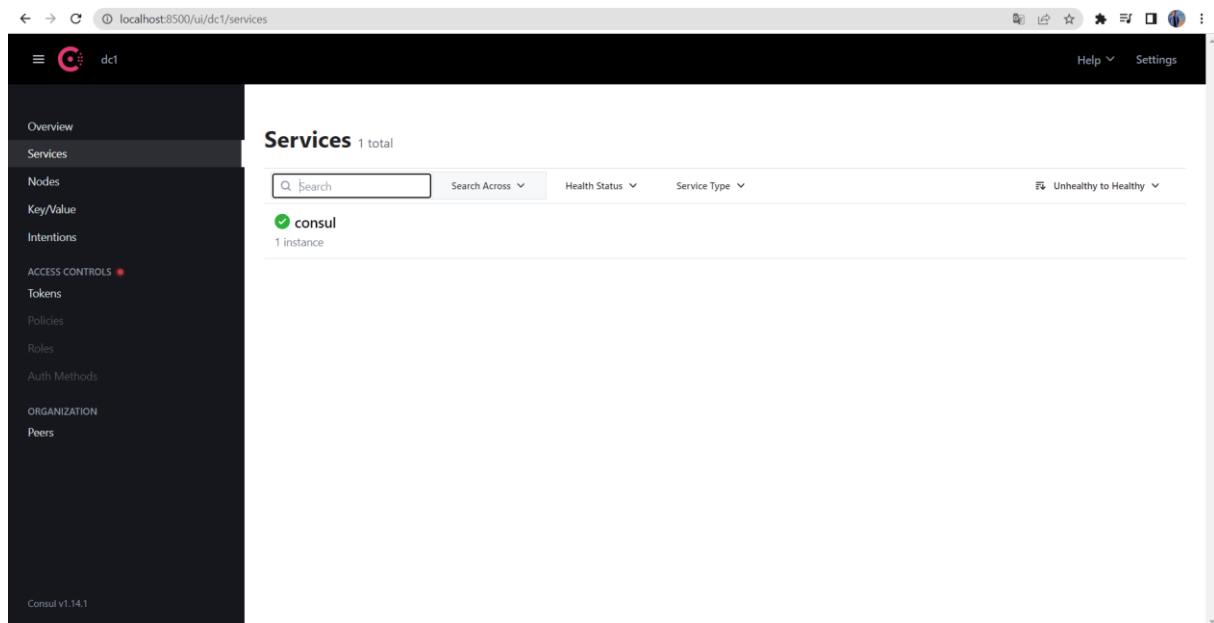
Architecture du projet :



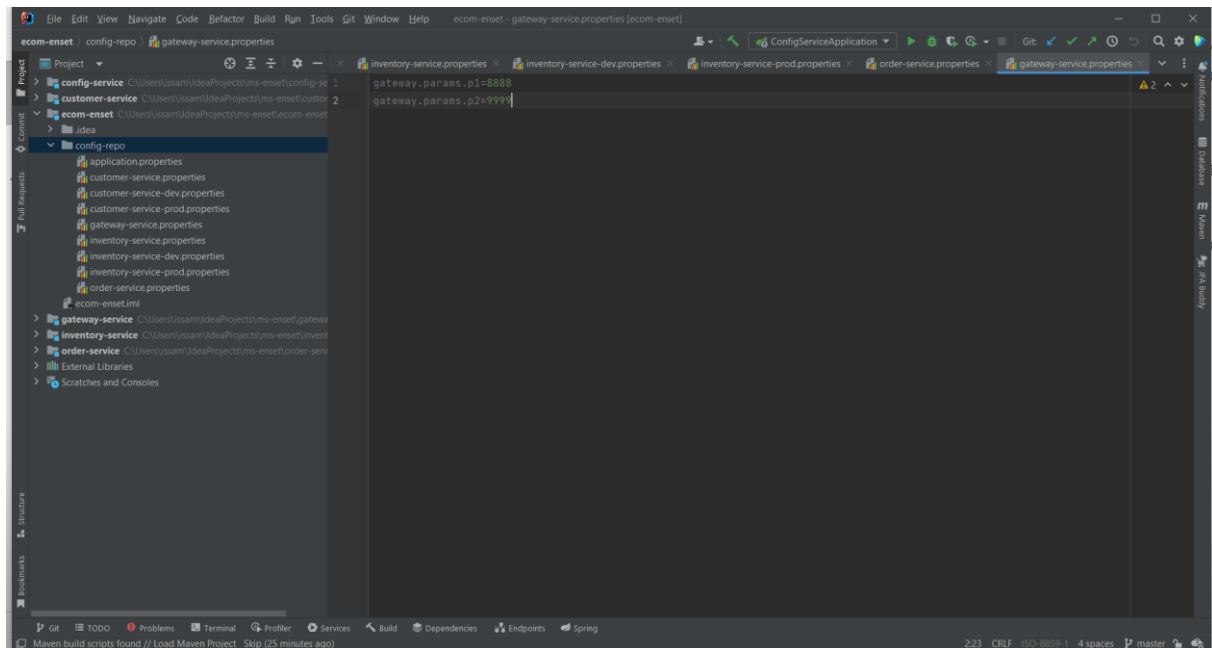
Ecom-enset contient tous les modules ou bien les microservices dans le fichier modules.xml

Platform consul :

Consul est un outil distribué, hautement disponible et compatible avec plusieurs centres de données pour la découverte, la configuration et l'orchestration de services. Consul permet un déploiement, une configuration et une maintenance rapides d'architectures orientées services à grande échelle



Après on ajoute le repository de la configuration pour que chaque microservice prend sa configuration de ce repo



Config-service :

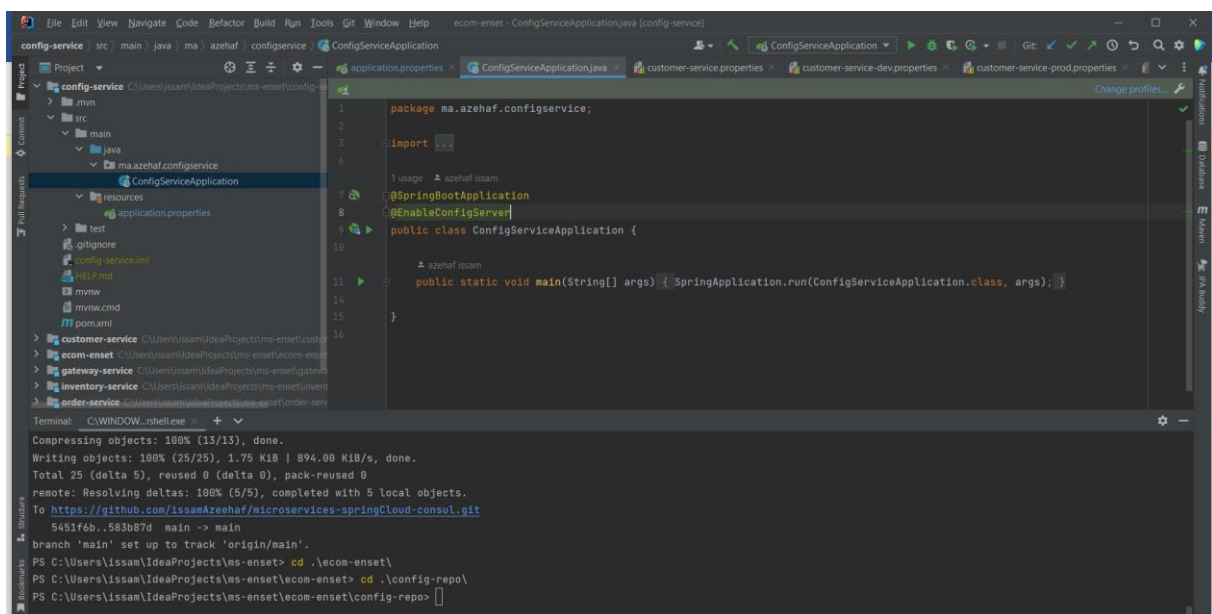


Figure 1 : configServiceApplication

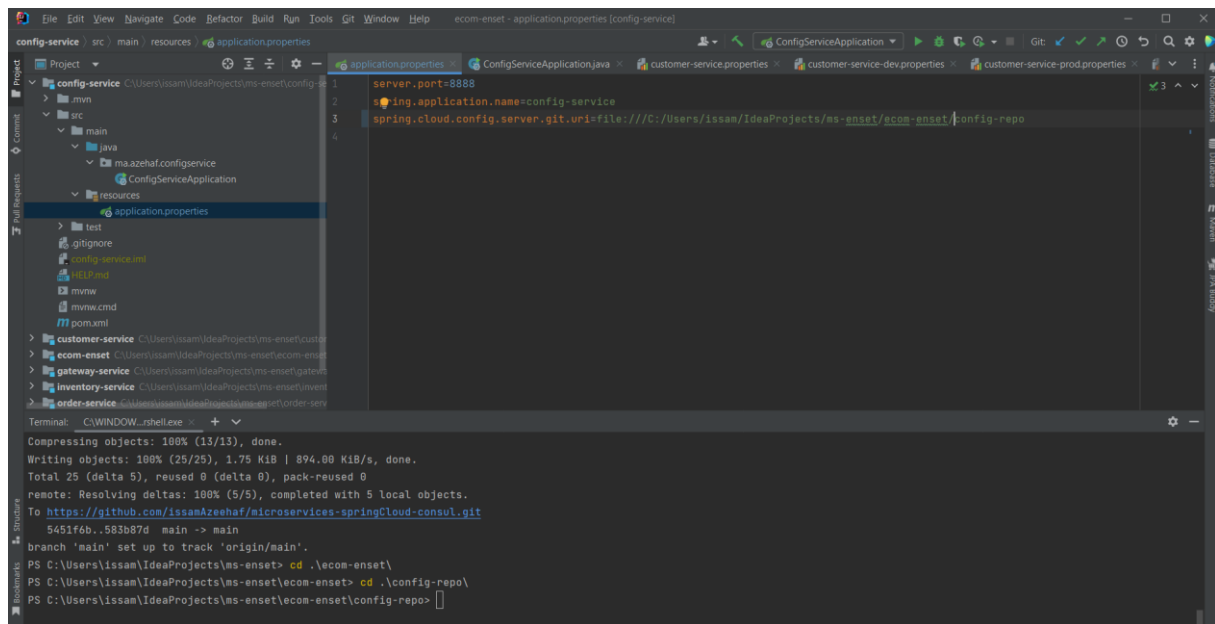
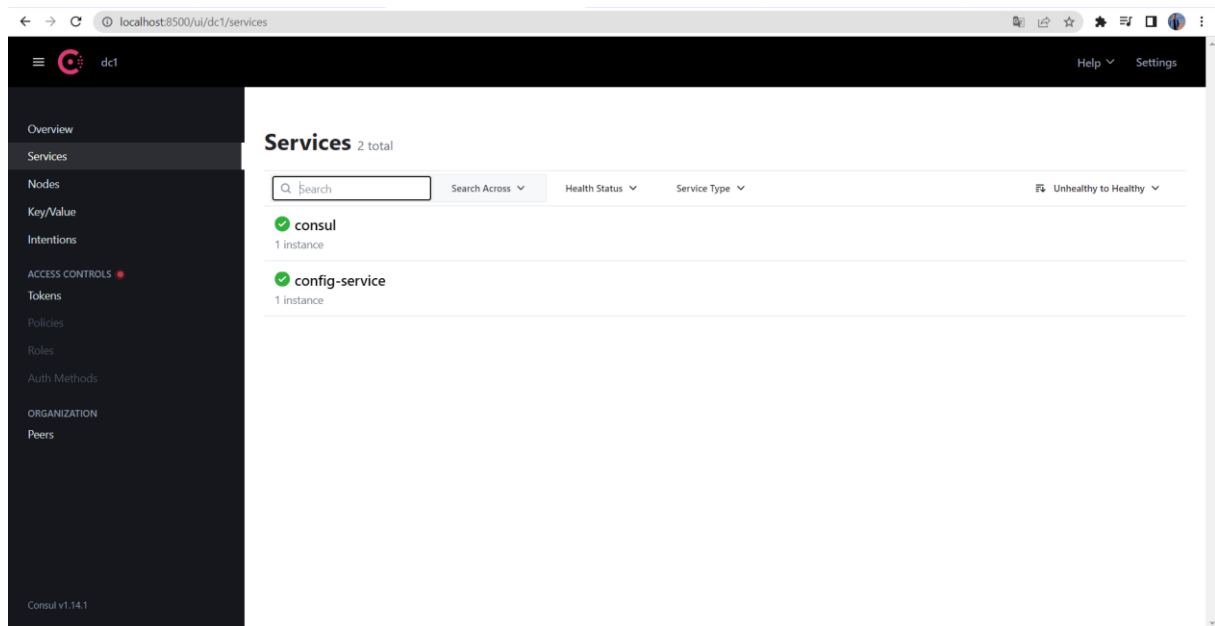


Figure 2 : Application.properties

Après on démarre notre service de configuration et on va bien qu'il est bien démarrer sur Consul



Maintenant on peut avoir toutes les configuration des micro services

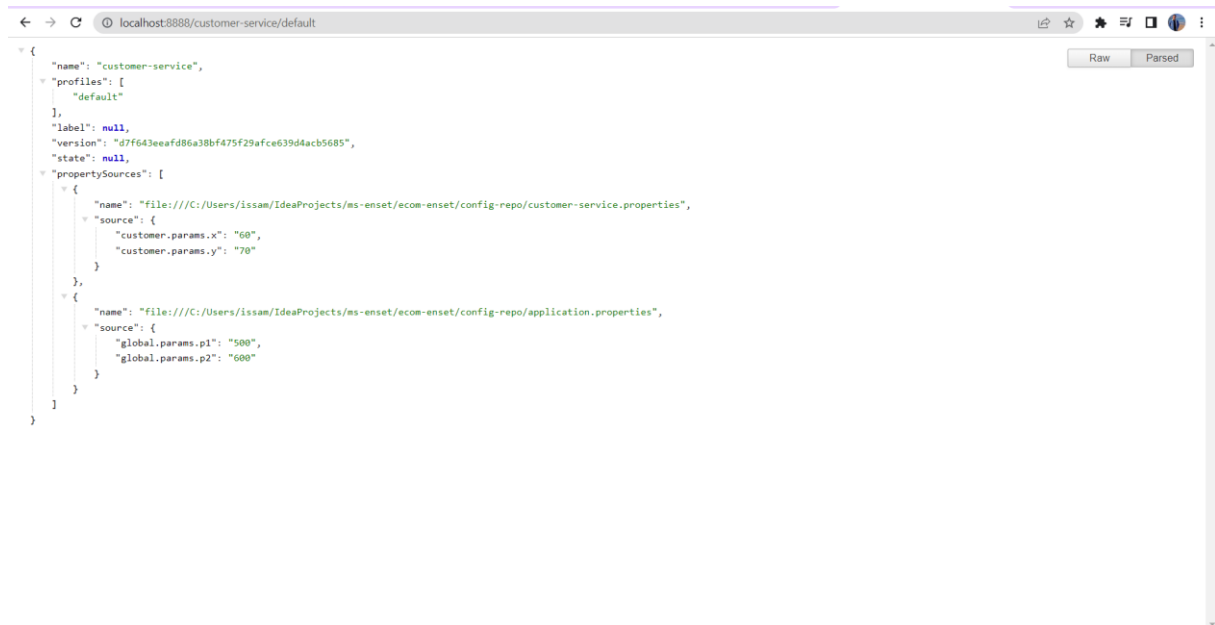


Figure 3 : customer-service configuration default

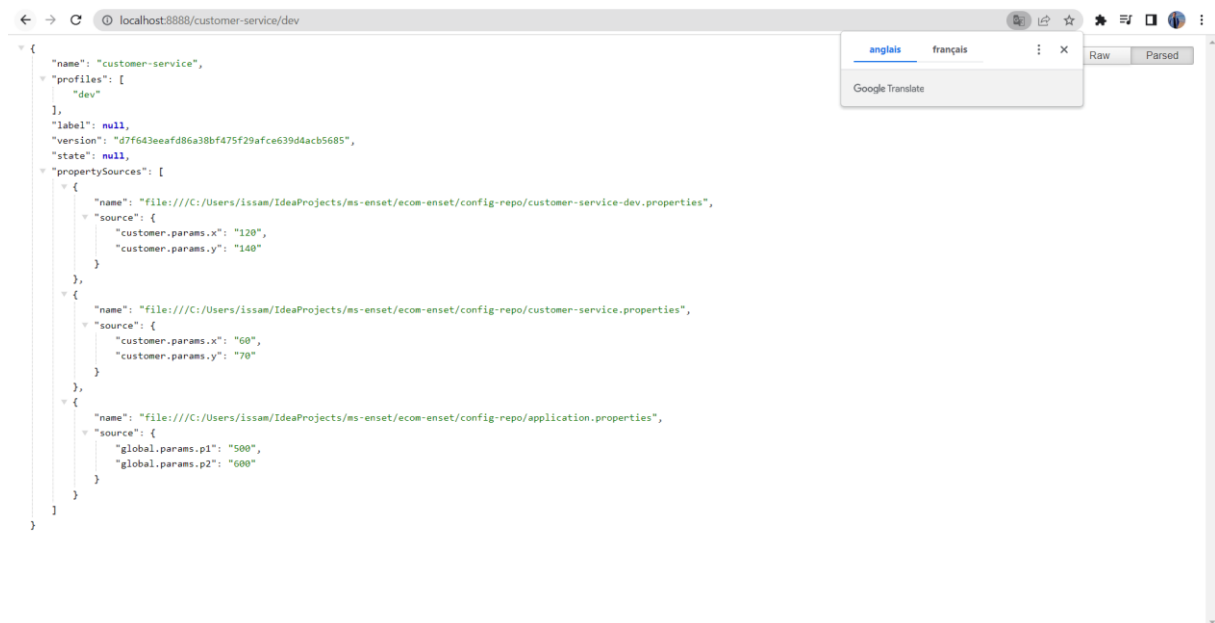


Figure 4 : customer-service configuration dev

Customer-service :

Tout d'abord on met la configuration dans le fichier customer-service.properties qui se trouve dans notre repo

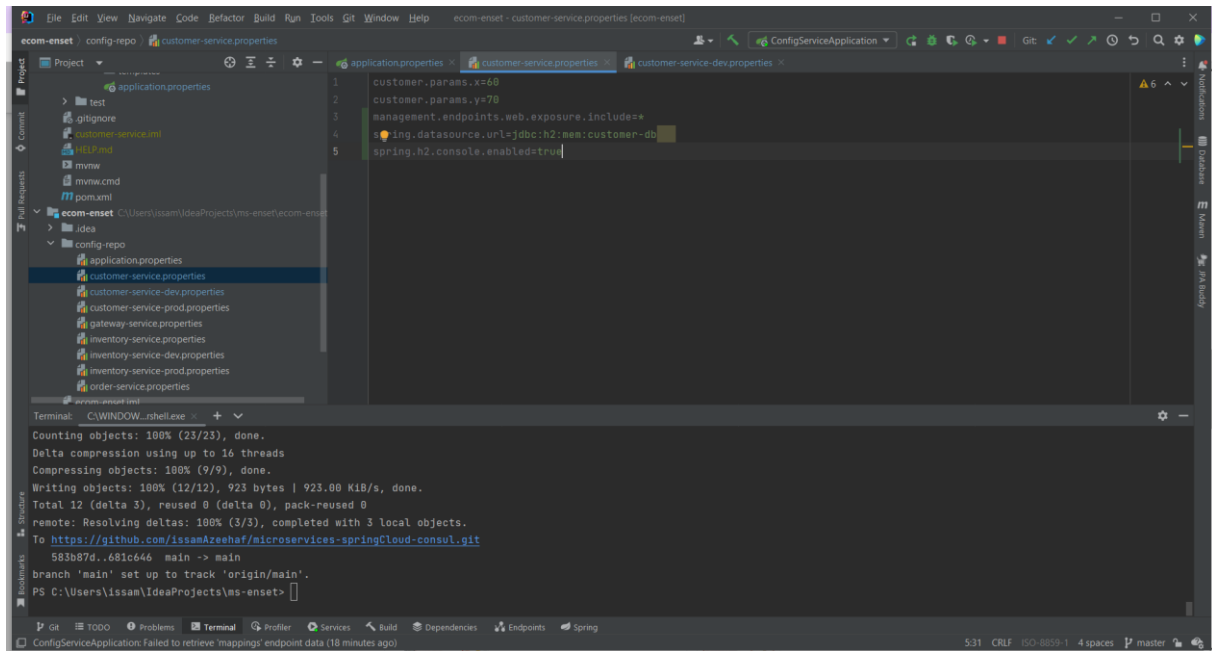


Figure 5 : customer-service configuration

Après j'ai créé ma classe Customer

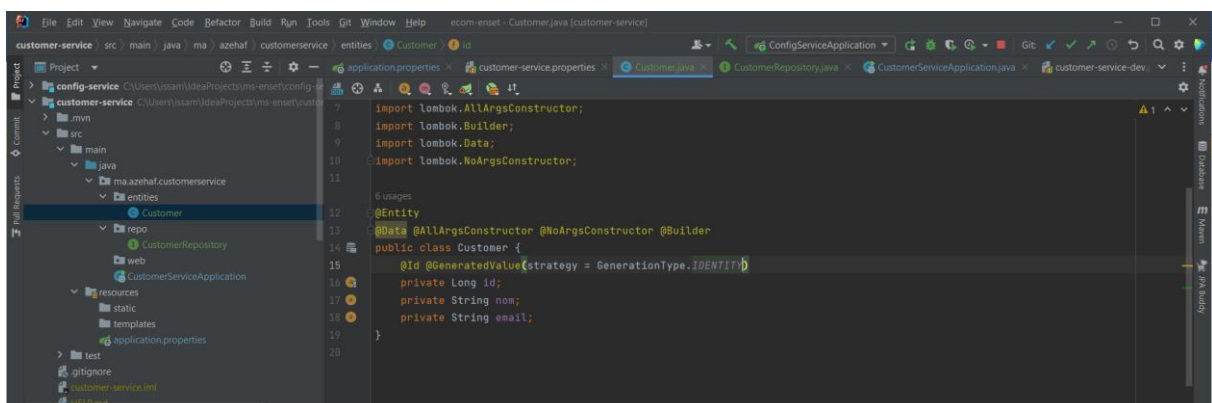


Figure 6 : classe customer

Par la suite repository

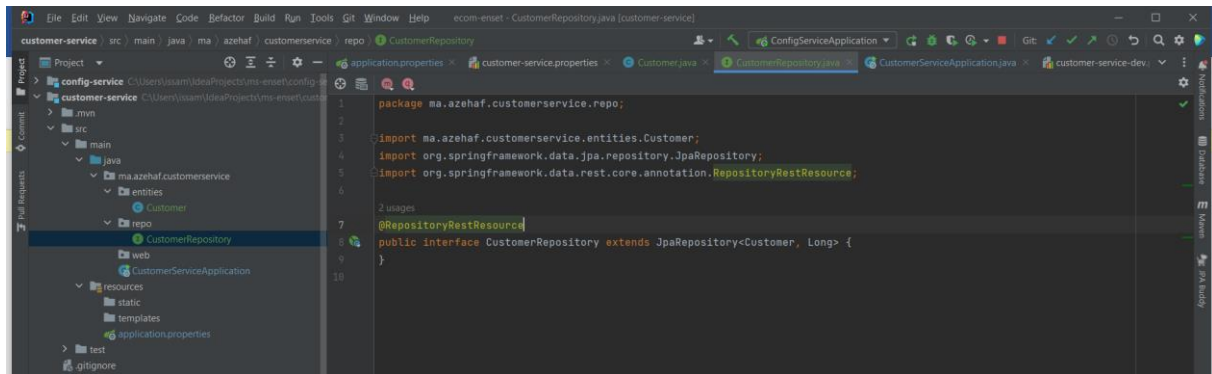
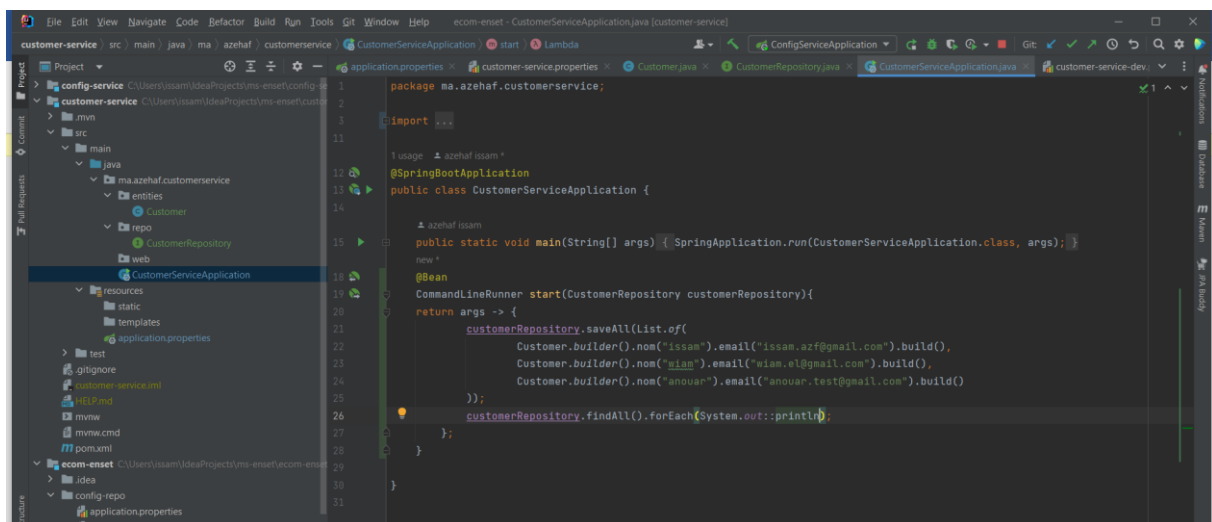
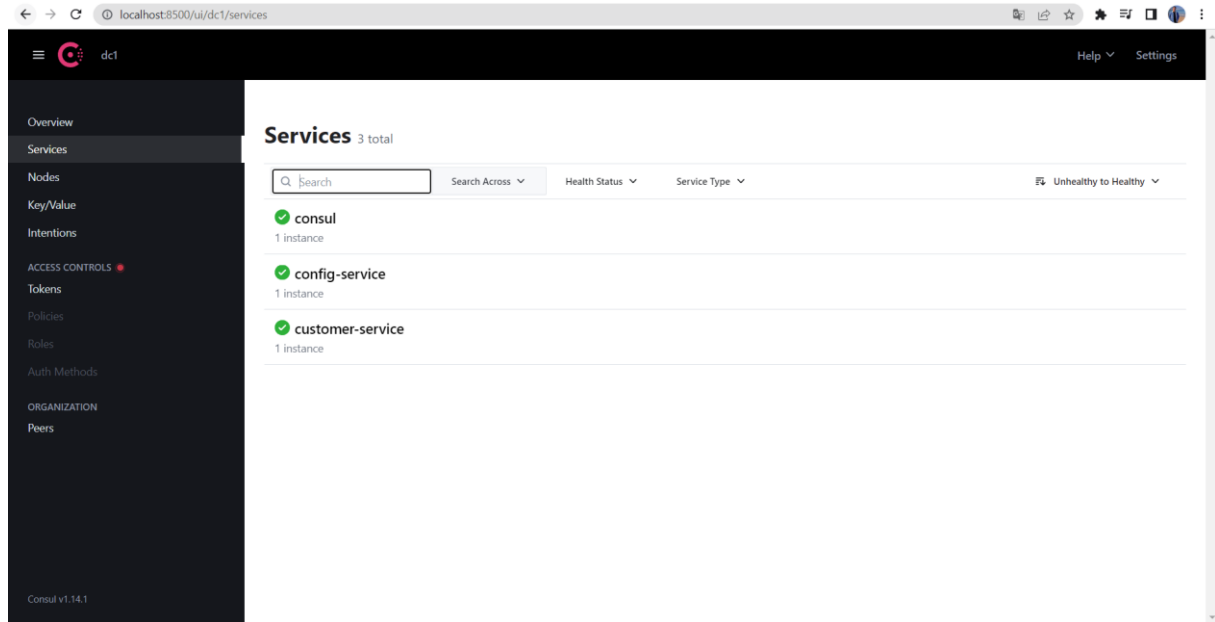


Figure 7 : customerRepository

Et dans customerServiceApplication j'ai créé quelque customer



Lorsque j'ai démarré le service on voit bien qu'il est bien enregistré dans consul



La base de données a été bien créer

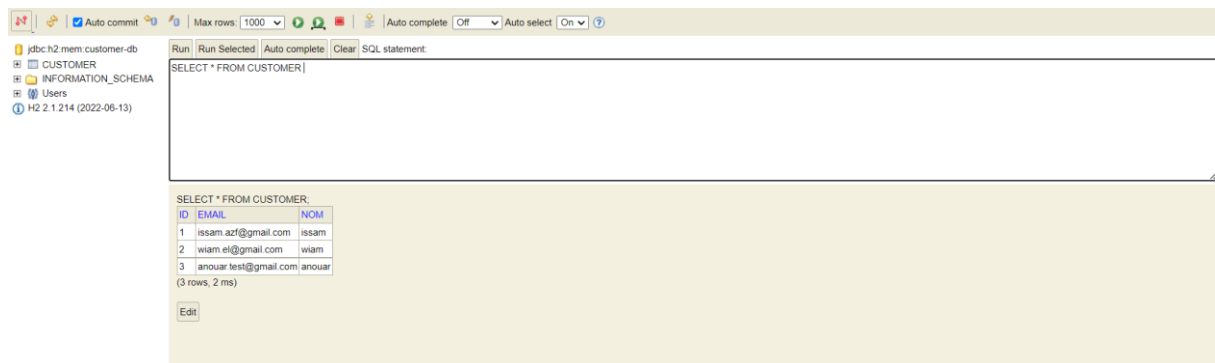
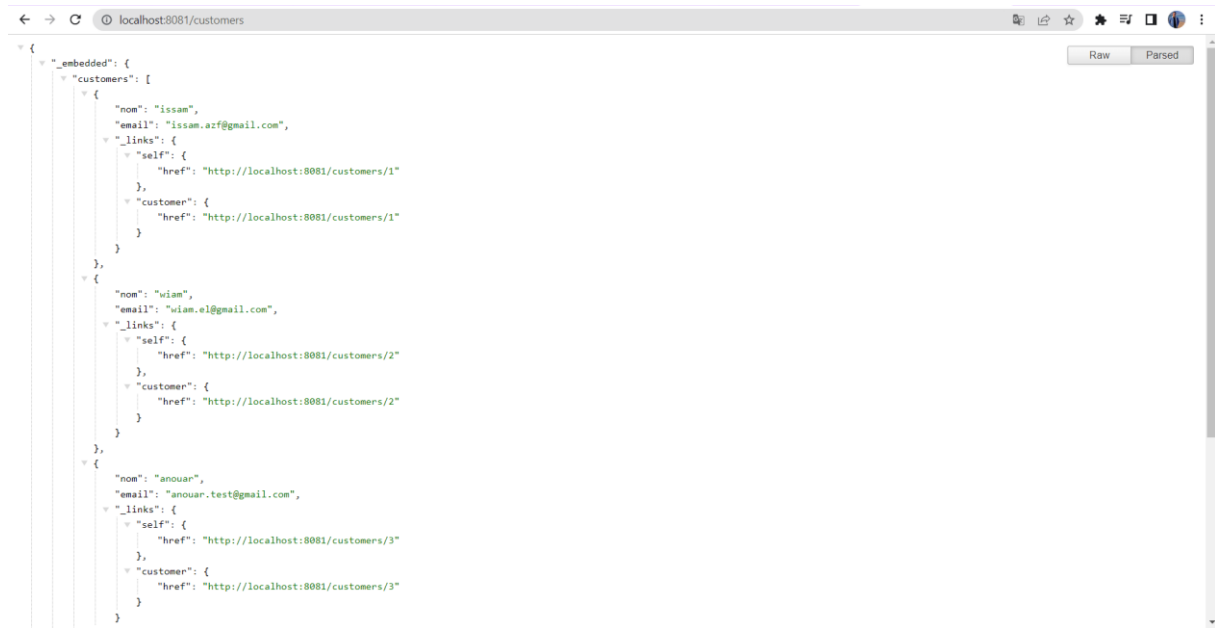


Figure 8 : base de données customer-db

A l'aide de spring data rest je vois bien la liste des customers



```
{
  "_embedded": {
    "customers": [
      {
        "nom": "issam",
        "email": "issam.azf@gmail.com",
        "_links": {
          "self": {
            "href": "http://localhost:8081/customers/1"
          },
          "customer": {
            "href": "http://localhost:8081/customers/1"
          }
        }
      },
      {
        "nom": "wiam",
        "email": "wiam.el@gmail.com",
        "_links": {
          "self": {
            "href": "http://localhost:8081/customers/2"
          },
          "customer": {
            "href": "http://localhost:8081/customers/2"
          }
        }
      },
      {
        "nom": "anouar",
        "email": "anouar-test@gmail.com",
        "_links": {
          "self": {
            "href": "http://localhost:8081/customers/3"
          },
          "customer": {
            "href": "http://localhost:8081/customers/3"
          }
        }
      }
    ]
  }
}
```

gateway-service :

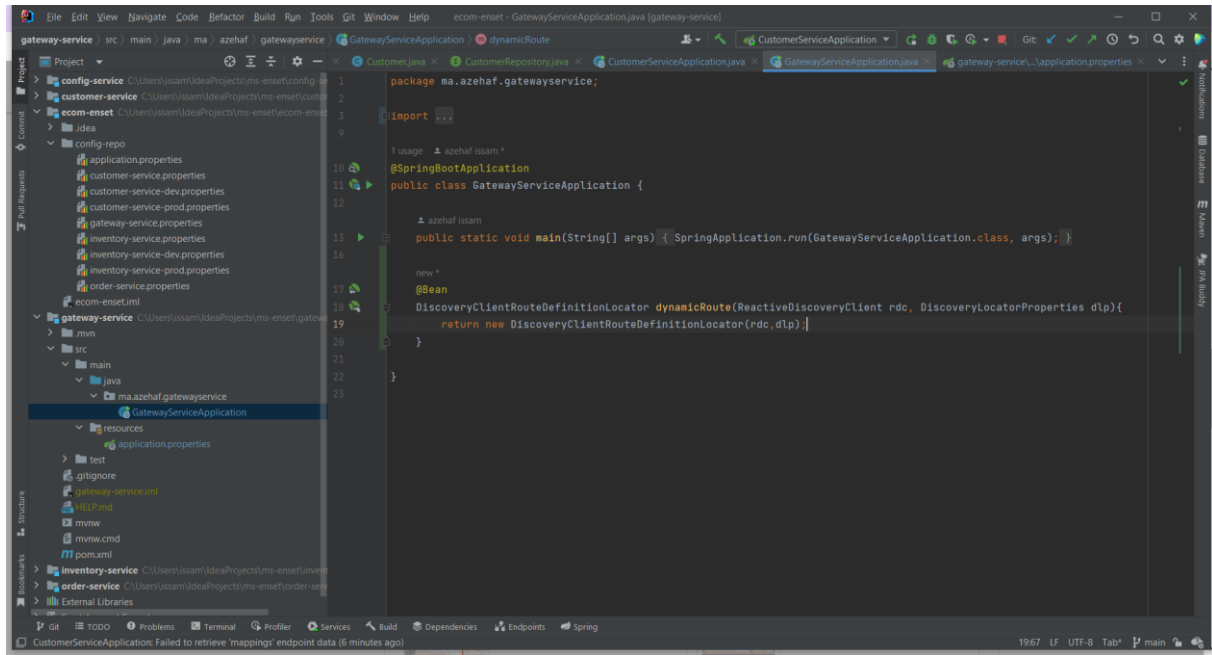


Figure 9 : dynamicRoute

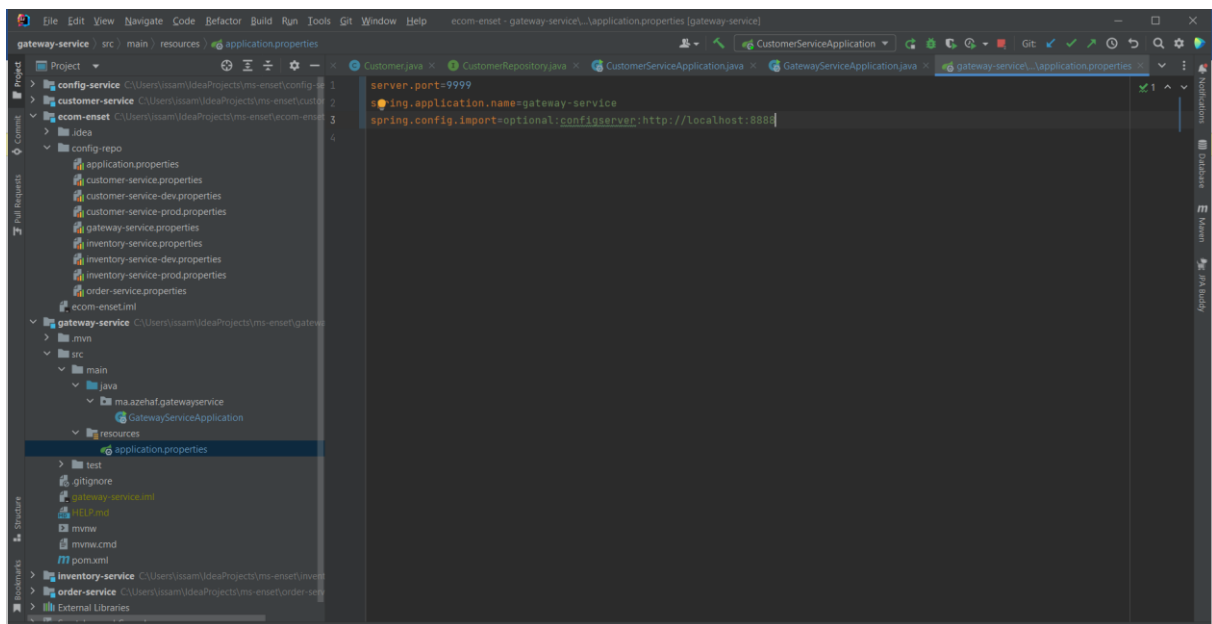
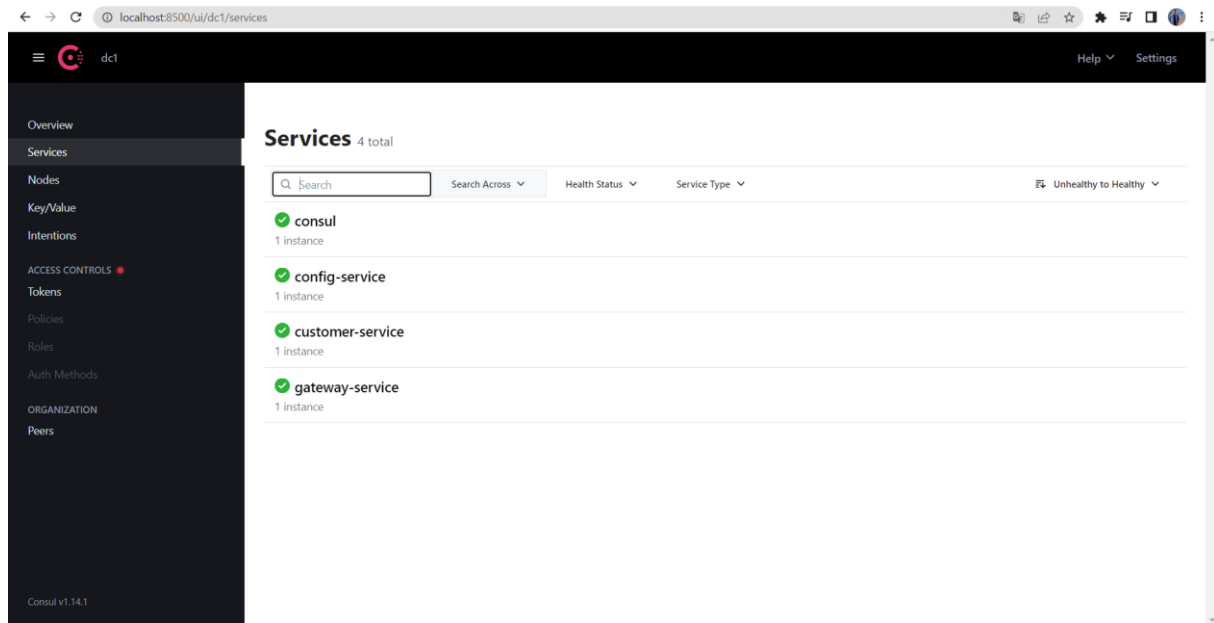
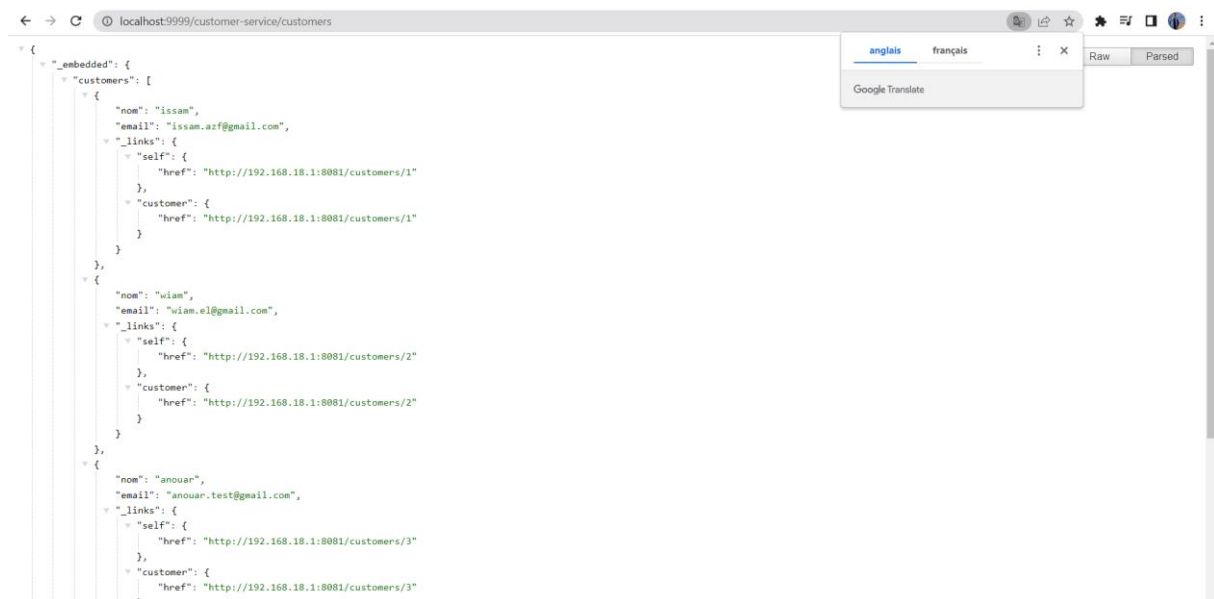


Figure 10 : application.properties

Et lorsque on demarre le service on le vois bien dans consul



Maintenant on verifie si la gateway fonctionne



Inventory-service :

tout d'abord on va créer notre entité product

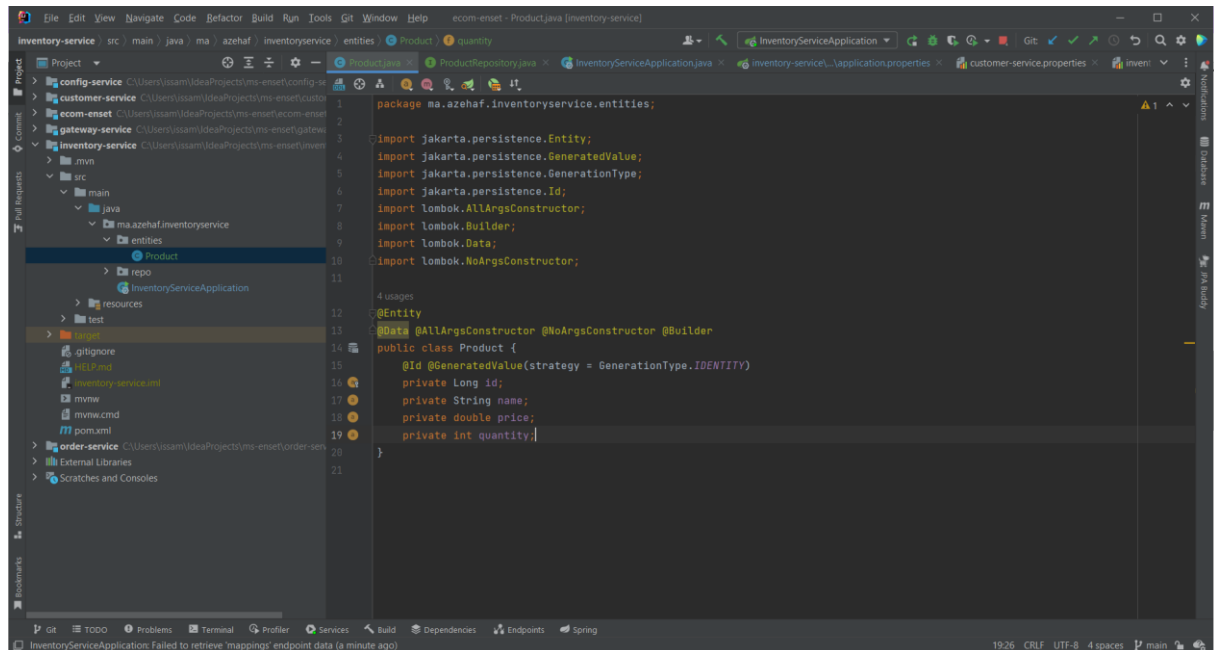
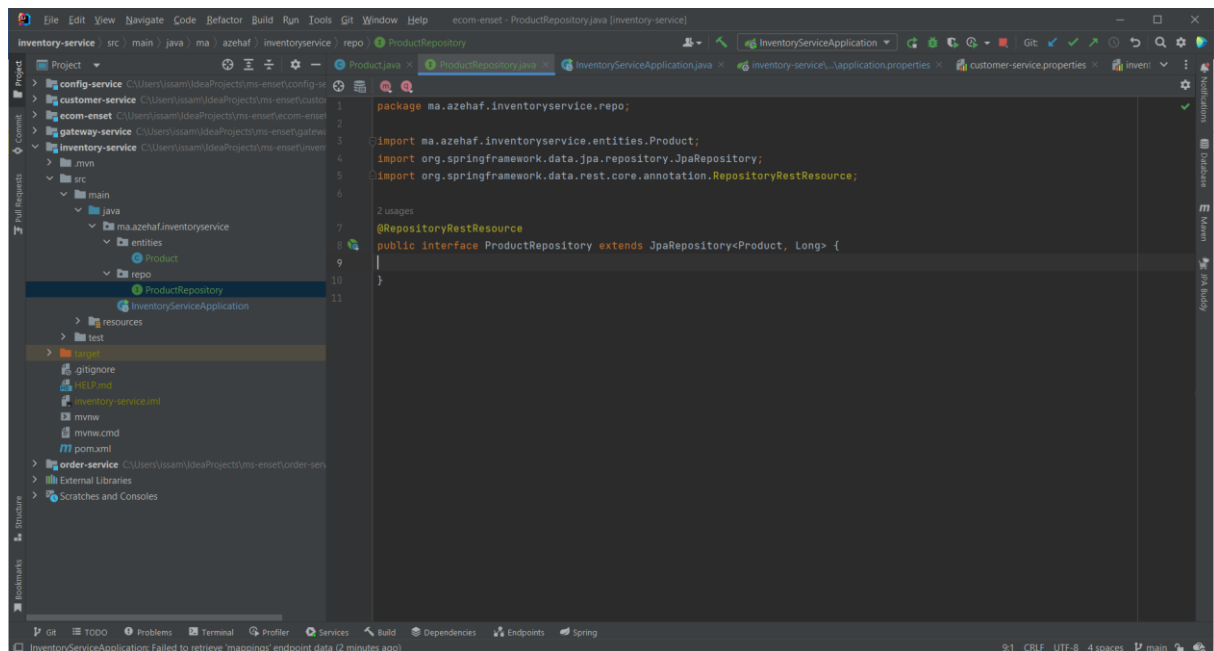


Figure 11 : classe product

Après on va créer notre repository



Par la suite on va ajouter quelque enregistrement dans la base de données

```

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.discovery.EnableDiscoveryClient;

@SpringBootApplication
@EnableDiscoveryClient
public class InventoryServiceApplication {

    @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(20))
                ));
            }
        };
    }
}

```

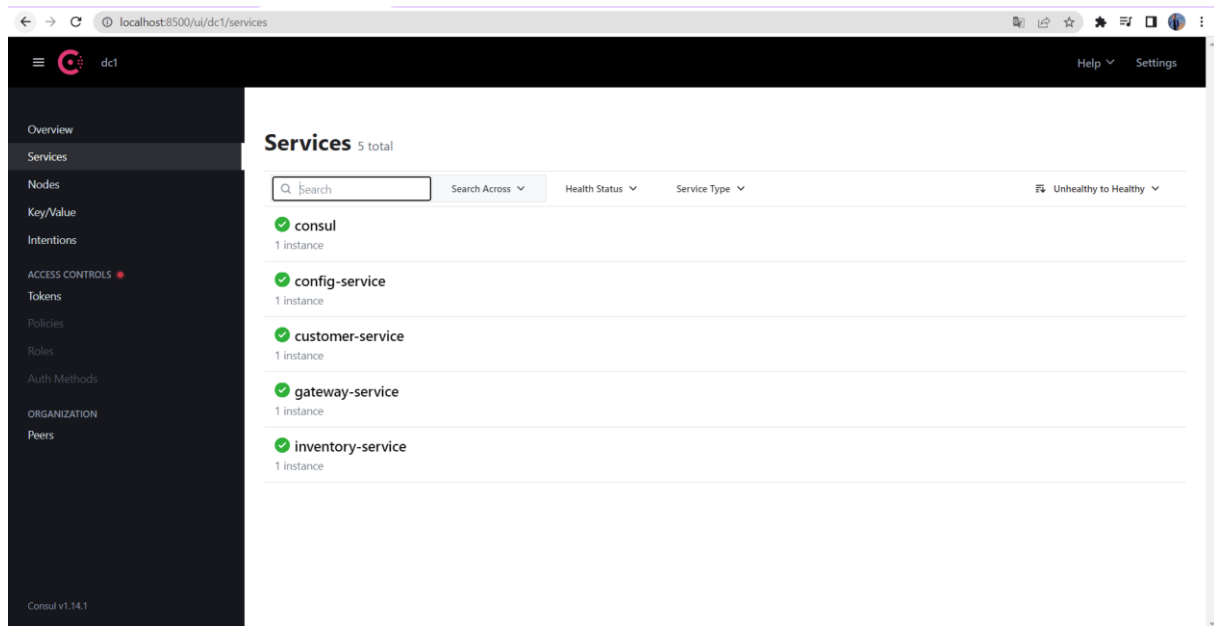
Et ensuite on va ajouter la configuration dans notre repo

```

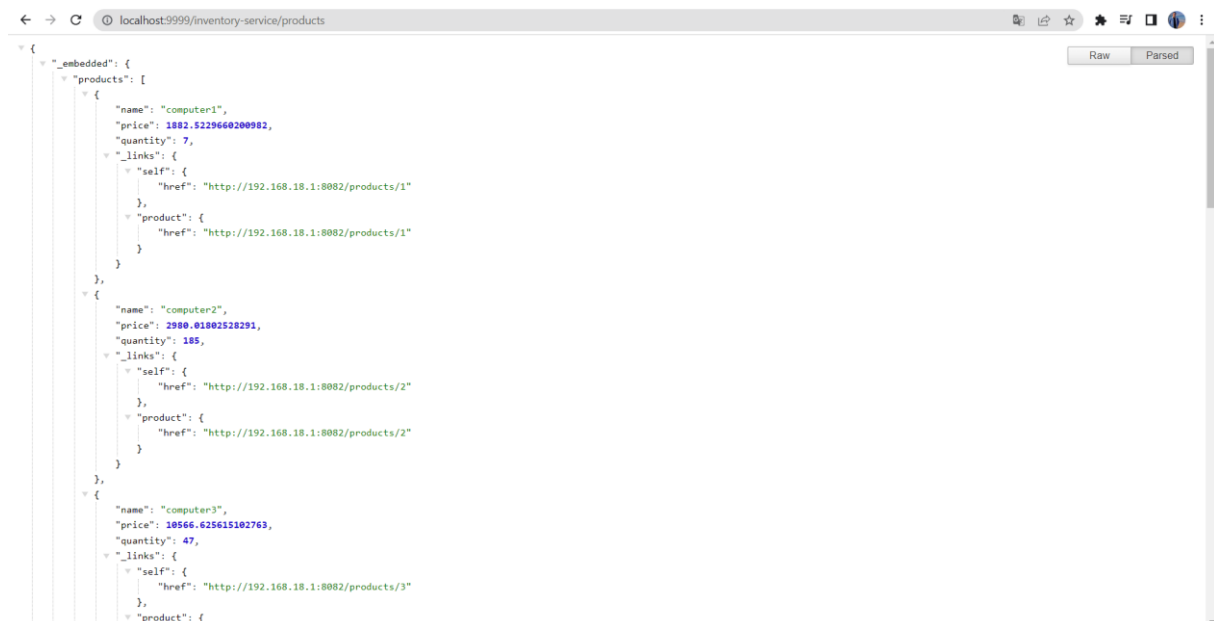
inv.params.a=100
inv.params.b=200
management.endpoints.web.exposure.include=*
spring.datasource.url=jdbc:h2:mem:products-db
spring.h2.console.enabled=true

```

Après on démarre notre service



On voit bien qu'il est bien démarrer dans consul



Et voila on voit bien les enregistrements des produits

Maintenant on va créer une projection

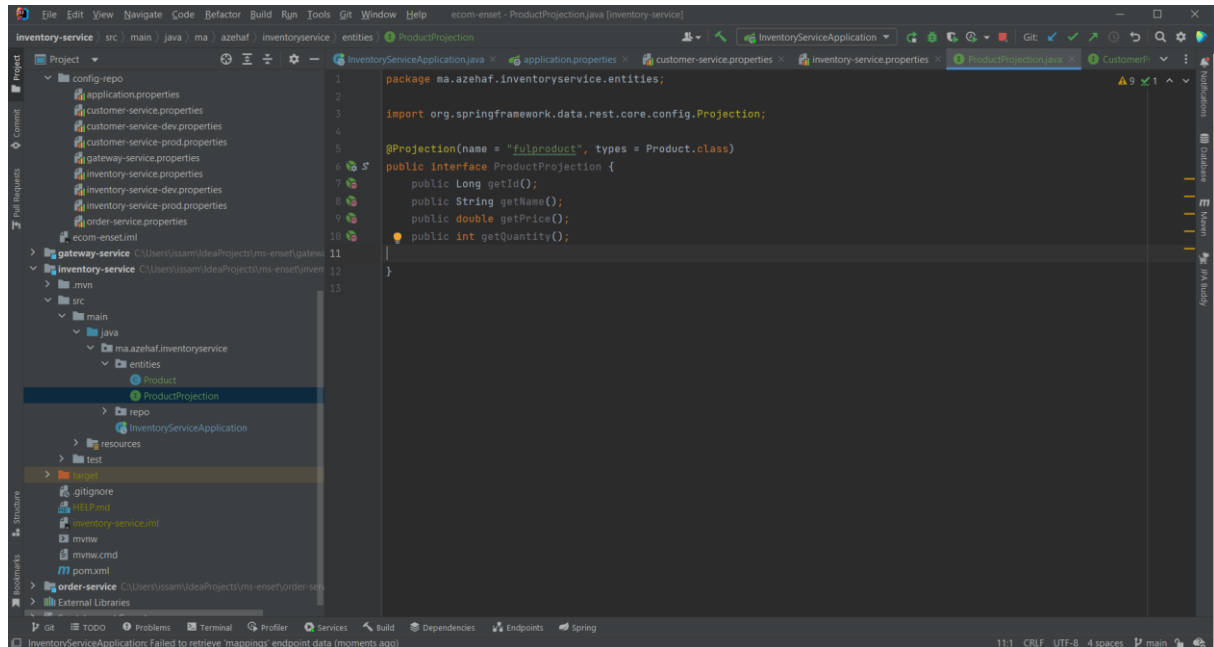
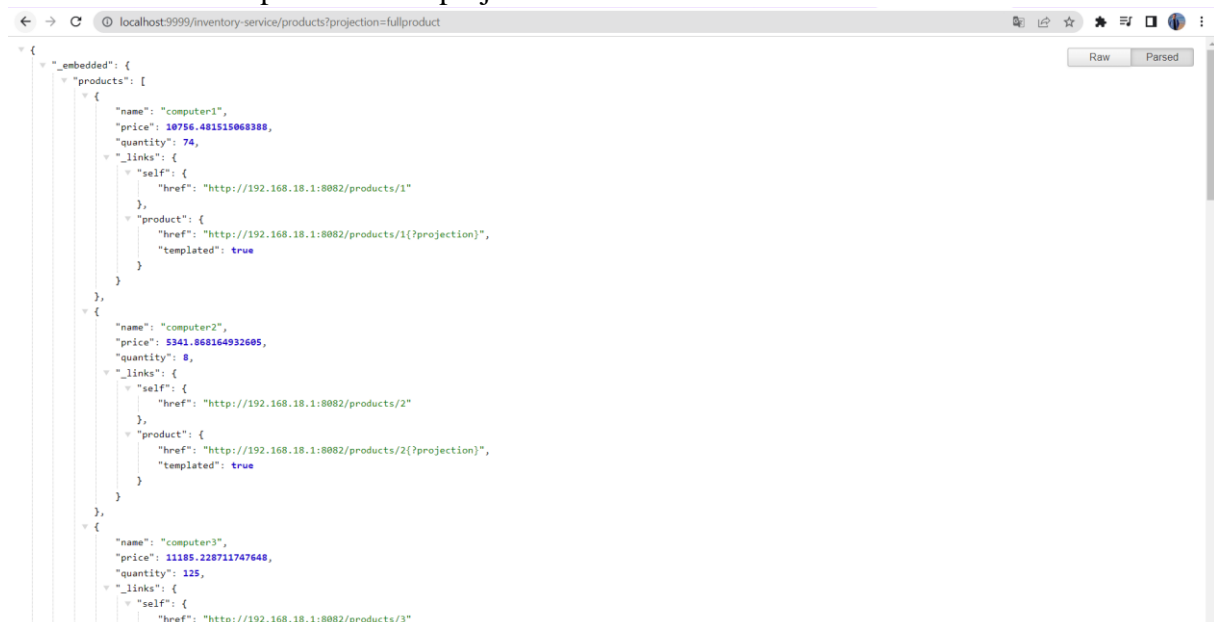


Figure 12 : productProjection

Et voici la liste des produits avec projection



Order -service :

Tout d'abord on va créer les classe qu'on va utiliser

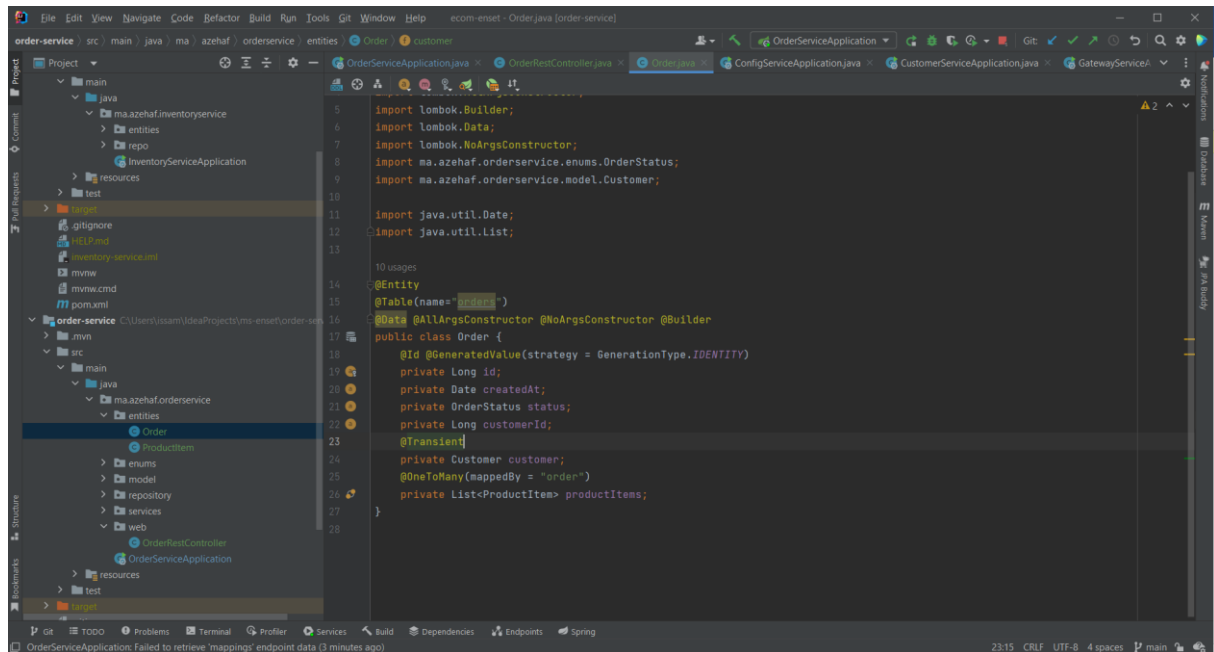


Figure 13 : classe order

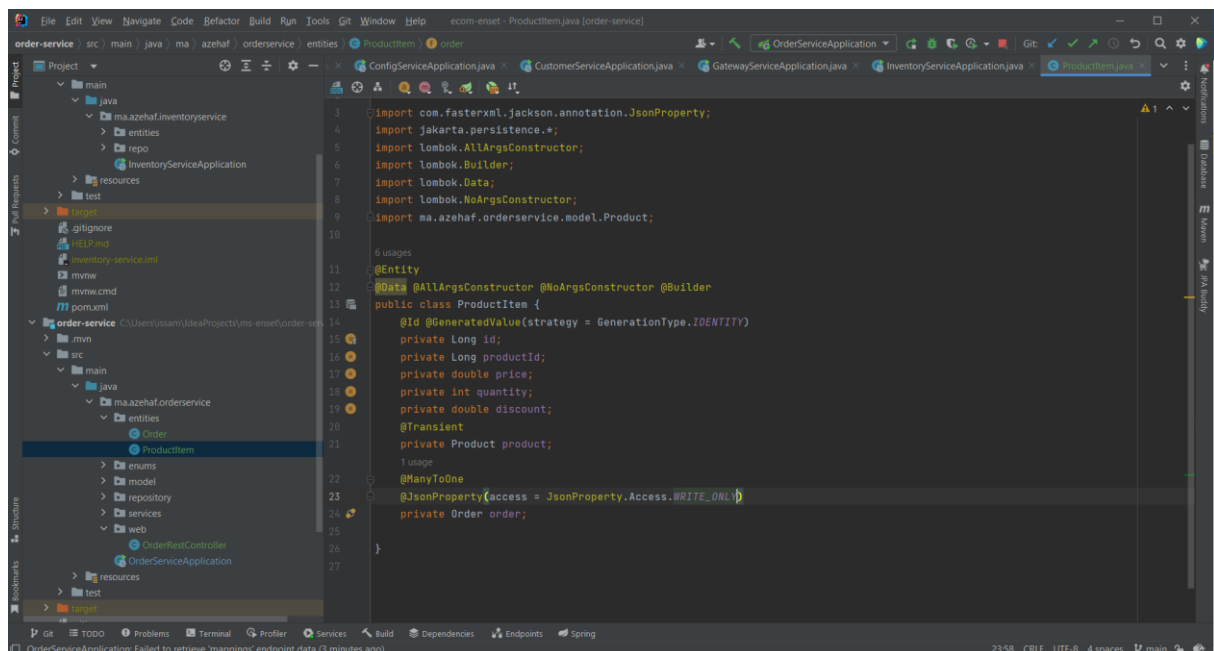
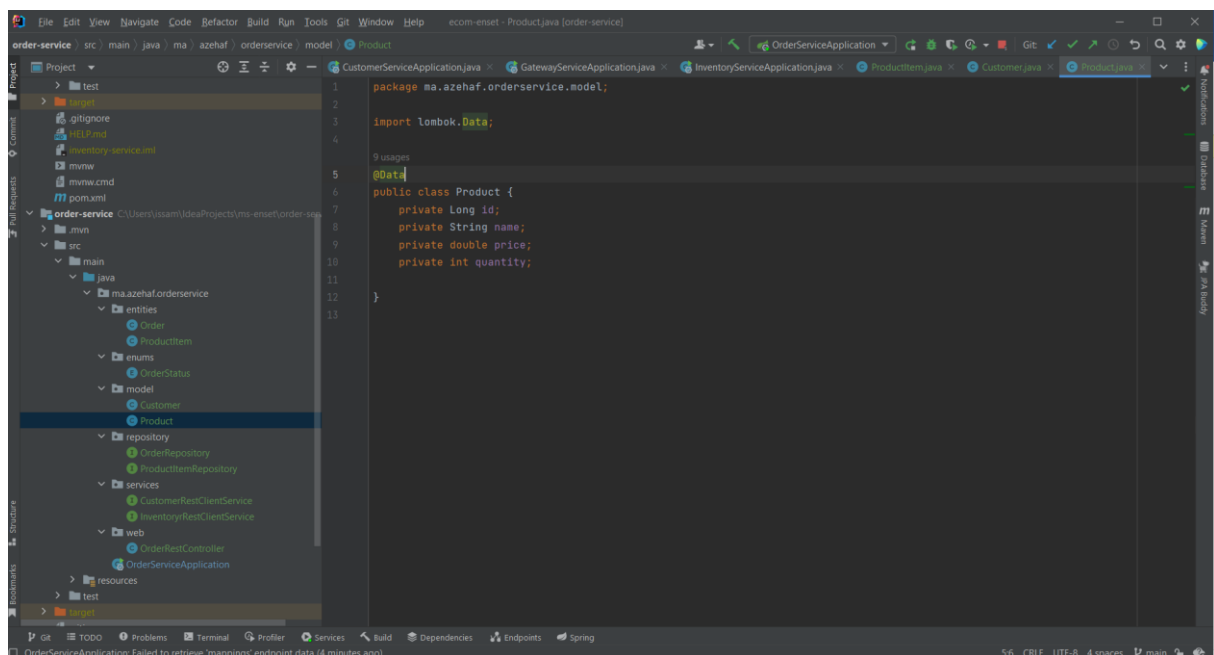
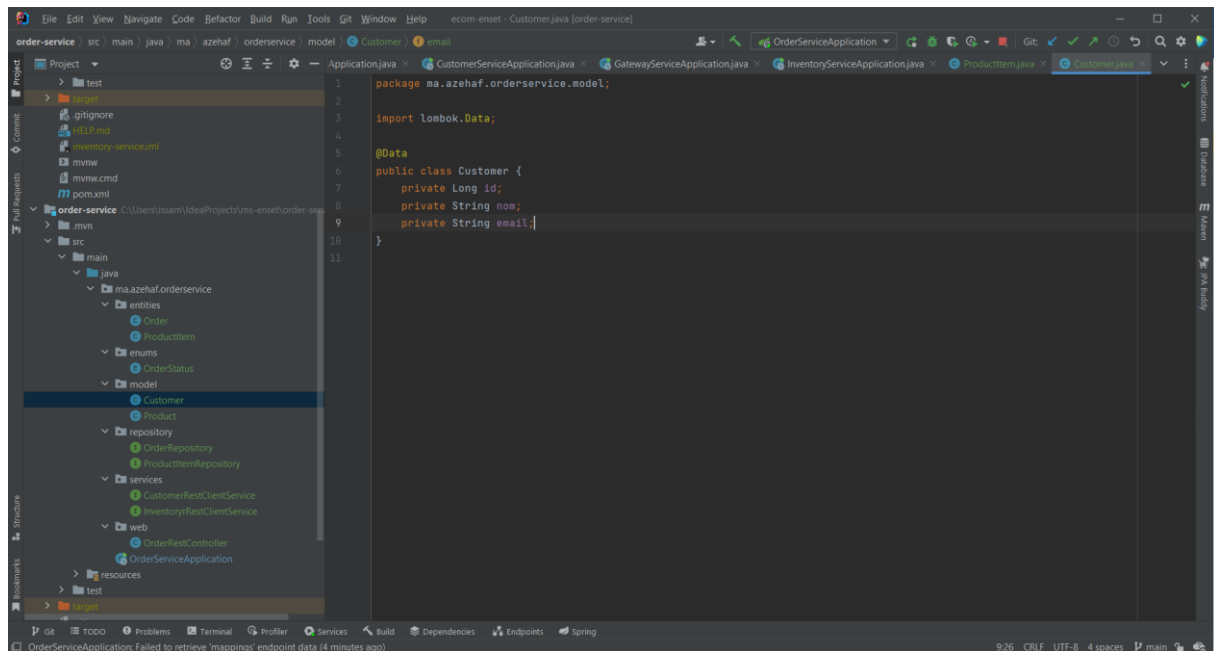
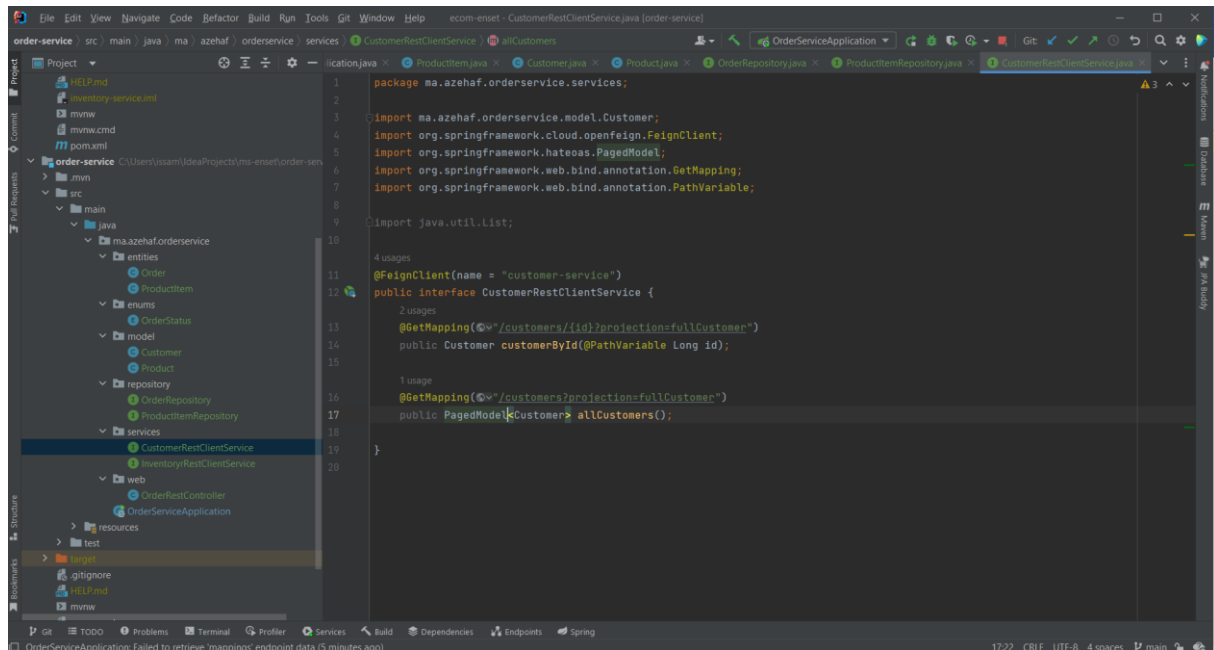


Figure 14 : classe productItem

Ensuite on va créer 2 model qui contient deux classe customer et product avec que le setter et getter



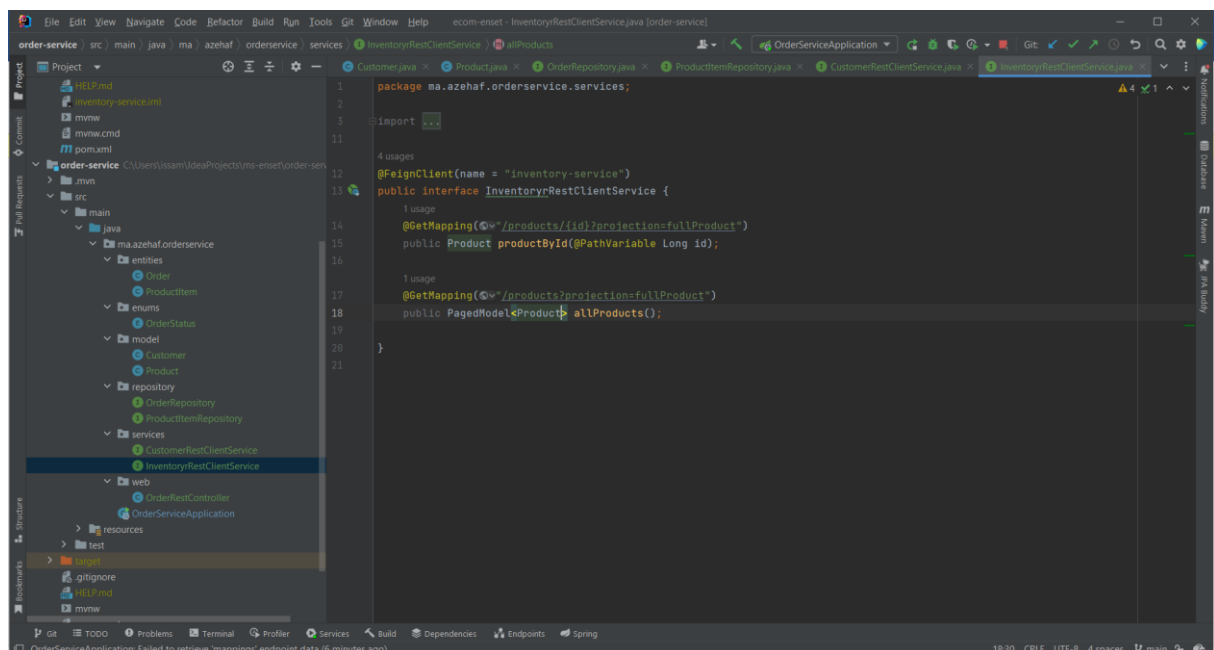
Par la suite nos repository et nos services



```

1 package ma.azehaf.orderservice.services;
2
3 import ma.azehaf.orderservice.model.Customer;
4 import org.springframework.cloud.openfeign.FeignClient;
5 import org.springframework.hateoas.PagedModel;
6 import org.springframework.web.bind.annotation.GetMapping;
7 import org.springframework.web.bind.annotation.PathVariable;
8
9 import java.util.List;
10
11 @FeignClient(name = "customer-service")
12 public interface CustomerRestClientService {
13     @GetMapping("/customers/{id}?projection=fullCustomer")
14     public Customer customerById(@PathVariable Long id);
15
16     @GetMapping("/customers?projection=fullCustomer")
17     public PagedModel<Customer> allCustomers();
18 }

```



```

1 package ma.azehaf.orderservice.services;
2
3 import org.springframework.cloud.openfeign.FeignClient;
4 import org.springframework.hateoas.PagedModel;
5 import org.springframework.web.bind.annotation.GetMapping;
6 import org.springframework.web.bind.annotation.PathVariable;
7
8 import java.util.List;
9
10 @FeignClient(name = "inventory-service")
11 public interface InventoryRestClientService {
12     @GetMapping("/products/{id}?projection=fullProduct")
13     public Product productById(@PathVariable Long id);
14
15     @GetMapping("/products?projection=fullProduct")
16     public PagedModel<Product> allProducts();
17 }

```

Et ensuite un package web qui contient notre controller

```

18 import ma.azehaf.orderservice.services.InventoryRestClientService;
19 import org.springframework.web.bind.annotation.GetMapping;
20 import org.springframework.web.bind.annotation.PathVariable;
21 import org.springframework.web.bind.annotation.RestController;
22
23 @RestController
24 @AllArgsConstructor
25 public class OrderRestController {
26     1 usage
27     private OrderRepository orderRepository;
28     private ProductItemRepository productItemRepository;
29     1 usage
30     private CustomerRestClientService customerRestClientService;
31     1 usage
32     private InventoryRestClientService inventoryRestClientService;
33
34     @GetMapping("/{id}")
35     public Order getOrder(@PathVariable Long id){
36         Order order = orderRepository.findById(id).get();
37         Customer customer = customerRestClientService.customerById(order.getCustomerId());
38         order.setCustomer(customer);
39         order.getProductItems().forEach(pi -> {
40             Product product = inventoryRestClientService.productById(pi.getProductId());
41             pi.setProduct(product);
42         });
43         return order;
44     }
45 }

```

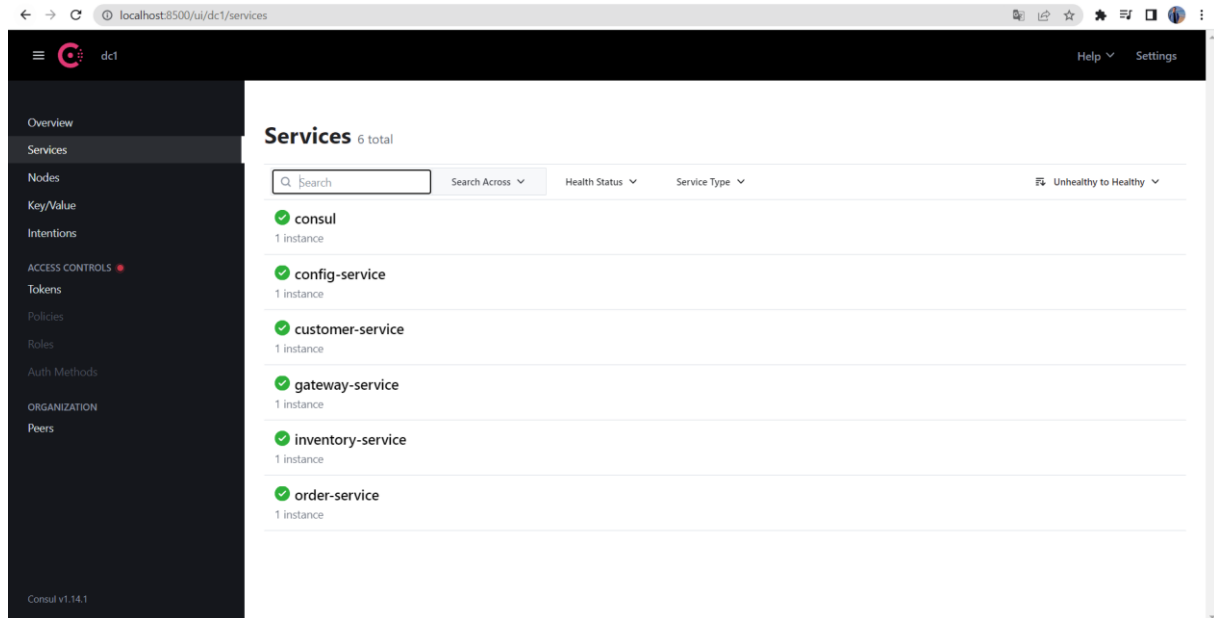
Et enfin on va remplir nos order avec des enregistrements

```

32 @Bean
33 CommandLineRunner start(OrderRepository orderRepository, ProductItemRepository productItemRepository,
34     CustomerRestClientService customerRestClientService,
35     InventoryRestClientService inventoryRestClientService){
36
37     return args -> {
38         List<Customer> customers = customerRestClientService.allCustomers().getContent().stream().toList();
39         List<Product> products = inventoryRestClientService.allProducts().getContent().stream().toList();
40         Long customerId = 1L;
41         Random random = new Random();
42         Customer customer = customerRestClientService.customerById(customerId);
43         for (int i=0; i<20; i++){
44             Order order = Order.builder()
45                 .customerId(customers.get(random.nextInt(customers.size())).getId())
46                 .status(Math.random()>0.5? OrderStatus.PENDING:OrderStatus.CREATED)
47                 .createdAt(new Date())
48                 .build();
49             Order savedOrder = orderRepository.save(order);
50             for (int j = 0; j<products.size(); j++){
51                 if (Math.random()>0.70){
52                     ProductItem productItem = ProductItem.builder()
53                         .order(savedOrder)
54                         .productId(products.get(j).getId())
55                         .price(products.get(j).getPrice())
56                         .quantity(1+random.nextInt( bound: 10))
57                         .discount(Math.random())
58                         .build();
59                     productItemRepository.save(productItem);
60                 }
61             }
62         }
63     };
64 }

```

Et maintenant on démarre notre order-service



On le voit sur la platform consul

Et voilà lorsque on consulte un order

