

Rapport sur Création d'une application basée sur une architecture micro-service qui permet de gérer les factures contenant des produits et appartenant à un client.

ACHBAD Fatima

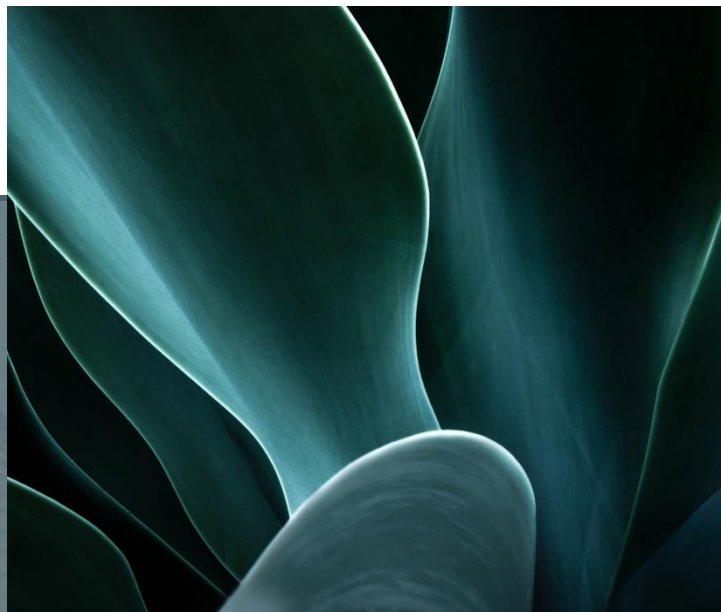
8/11/2023

—

Mohamed YOUSSEFI

Introduction

Les microservices sont une approche architecturale en plein essor dans le domaine du développement logiciel. Elle consiste à découper une application en de petits services indépendants, offrant de nombreux avantages tels que l'évolutivité, la facilité de maintenance, l'indépendance technologique et l'isolation des pannes. Cette architecture permet aux équipes de développement de travailler de manière plus agile, de réduire les risques de régressions et d'optimiser l'utilisation des ressources.



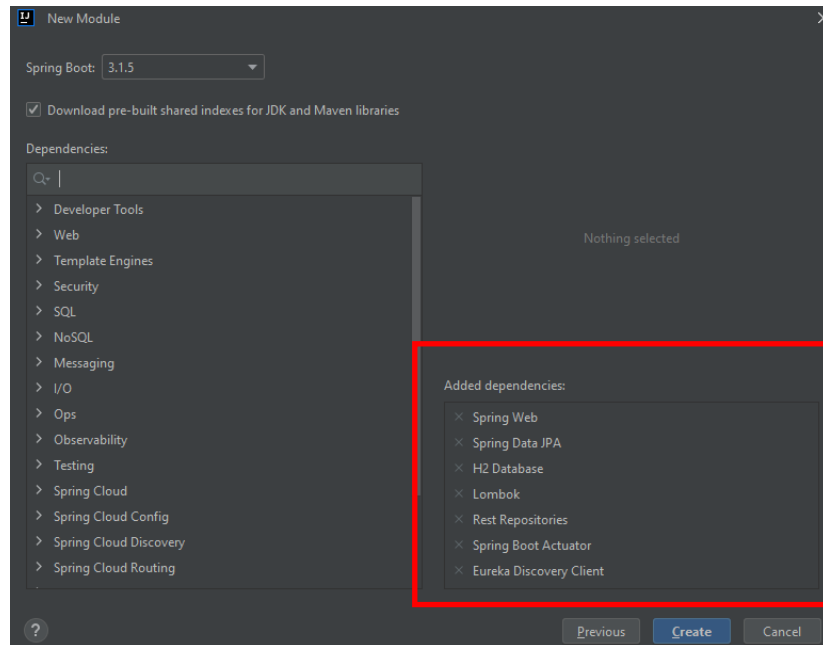


Objectifs du rapport

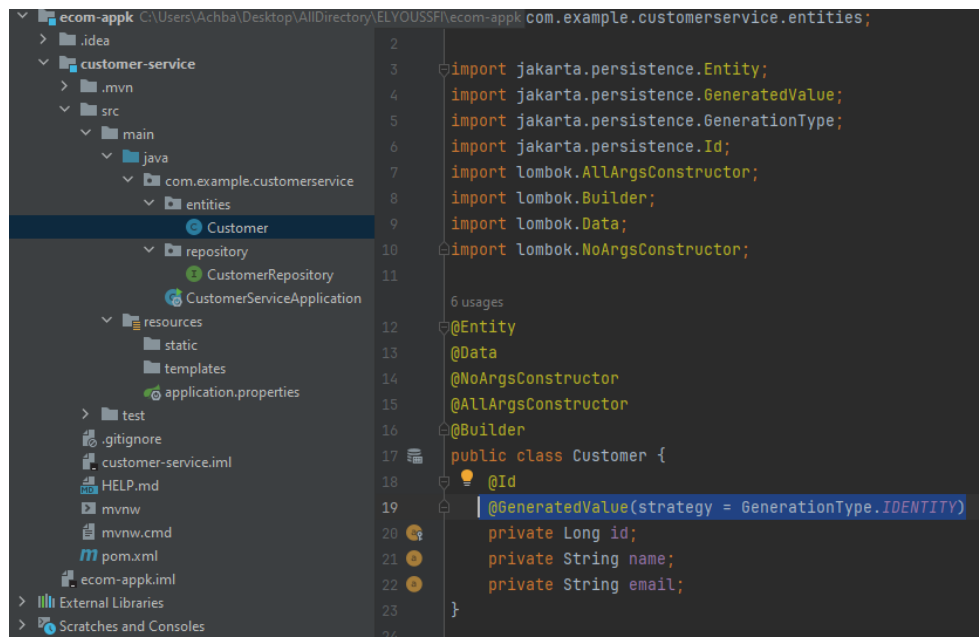
-
1. Créer le micro-service customer-service qui permet de gérer les client
 2. Créer le micro-service inventory-service qui permet de gérer les produits
 3. Créer la Gateway Spring cloud Gateway
 4. Configuration statique du système de routage
 5. Créer l'annuaire Eureka Discovery Service
 6. Faire une configuration dynamique des routes de la gateway
 7. Créer le service de facturation Billing-Service en utilisant Open Feign
 8. Créer un client Web Angular (Clients, Produits, Factures)
-

I. Créer le micro-service customer-service :

L'ensemble des module dans cette application seront créer comme un projet spring avec les dépendances suivante :



Création d'entité :



•Création de l'interface CustomerRepository et l'utilisation de @RepositoryRestResource pour permettre à Spring Boot de créer automatiquement un point de terminaison RESTful pour ce référentiel. Cela signifie que vous pouvez accéder aux opérations

sur les entités gérées par ce référentiel via des requêtes HTTP, telles que GET, POST, PUT et DELETE.

- Dans notre classe ,on vas créer des élément .

```
CustomerRepository.java
1 package com.example.customerservice.repository;
2
3 import com.example.customerservice.entities.Customer;
4 import org.springframework.data.jpa.repository.JpaRepository;
5 import org.springframework.data.rest.core.annotation.RepositoryRestResource;
6
7 @RepositoryRestResource
8 public interface CustomerRepository extends JpaRepository<Customer, Long> {
9
10 }

CustomerServiceApplication.java
1 package com.example.customerservice;
2
3 import ...
4
5 @SpringBootApplication
6 public class CustomerServiceApplication {
7
8     public static void main(String[] args) { SpringApplication.run(CustomerServiceApplication.class, args); }
9
10     @Bean
11     public CommandLineRunner commandLineRunner(CustomerRepository customerRepository){
12         return args -> {
13             customerRepository.saveAll(
14                 List.of(
15                     Customer.builder().name("Hassan").email("hassan@gmail.com").build(),
16                     Customer.builder().name("Hanane").email("hanane@gmail.com").build(),
17                     Customer.builder().name("Imane").email("imane@gmail.com").build()
18                 )
19             );
20             customerRepository.findAll().forEach(c->{
21                 System.out.println(c);
22             });
23         };
24     }
25 }
```

Configuration de la base de donn  

```
CustomerRepository.java application.properties
1 spring.application.name=customer-service
2 server.port=9081
3 spring.datasource.url=jdbc:h2:mem:customer-db
4 spring.h2.console.enabled=true
5 spring.cloud.discovery.enabled=false
6
```

Affichage :

localhost:9081/h2-console/login.do?jsessionId=4a59ae006c78b5418864b364016fb624

stagecentredouri Apprendre java Apprendre Laravel outlook pfe

Auto commit Max rows: 1000 Auto complete Off Auto select On

jdbc:h2:mem:customer-db

CUSTOMER

- ID
- EMAIL
- NAME
- Indexes

INFORMATION_SCHEMA

Users

H2 2.1.214 (2022-06-13)

Run Run Selected Auto complete Clear SQL statement:

SELECT * FROM CUSTOMER|

SELECT * FROM CUSTOMER;

ID	EMAIL	NAME
1	hassan@gmail.com	Hassan
2	hanane@gmail.com	Hanane
3	imane@gmail.com	Imane

(3 rows, 1 ms)

Edit

tester notre micro-service on effectue les opération suivante

localhost:9081/customers/1

stagecentredouri Apprendre java Apprendre Laravel

```
{
  "name": "Hassan",
  "email": "hassan@gmail.com",
  "_links": {
    "self": {
      "href": "http://localhost:9081/customers/1"
    },
    "customer": {
      "href": "http://localhost:9081/customers/1"
    }
  }
}
```

localhost:9081/customers

stagecentredouri Apprendre java Apprendre Laravel outlook

```
{
  "_embedded": {
    "customers": [ {
      "name": "Hassan",
      "email": "hassan@gmail.com",
      "_links": {
        "self": {
          "href": "http://localhost:9081/customers/1"
        },
        "customer": {
          "href": "http://localhost:9081/customers/1"
        }
      }
    }, {
      "name": "Hanane",
      "email": "hanane@gmail.com",
      "_links": {
        "self": {
          "href": "http://localhost:9081/customers/2"
        },
        "customer": {
          "href": "http://localhost:9081/customers/2"
        }
      }
    }, {
      "name": "Imane",
      "email": "imane@gmail.com",
      "_links": {
        "self": {
          "href": "http://localhost:9081/customers/3"
        },
        "customer": {
          "href": "http://localhost:9081/customers/3"
        }
      }
    }
  ]
},
  "_links": {
    "self": {
      "href": "http://localhost:9081/customers?page=0&size=20"
    },
    "profile": {
      "href": "http://localhost:9081/profile/customers"
    }
  },
  "page": {
    "size": 20,
    "totalElements": 3,
    "totalPages": 1,
    "number": 0
  }
}
```

On remarque donc que l'id n'est pas affiché .
Pour résoudre le problème ,on modifie la configuration et on expose l'id .

```
package com.example.customerservice;

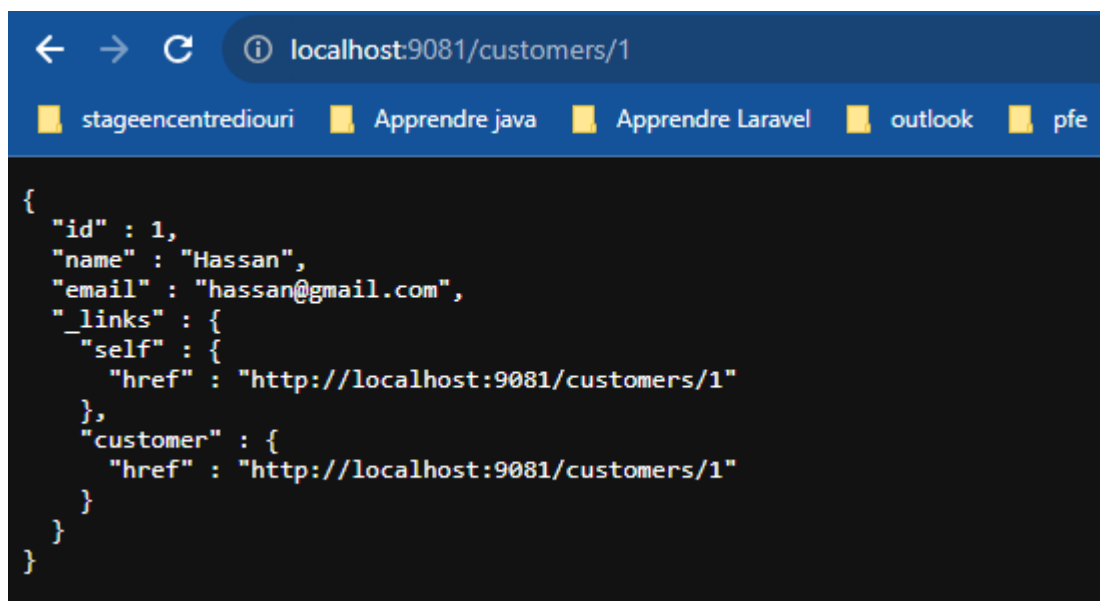
import ...

@SpringBootApplication
public class CustomerServiceApplication {

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

    @Bean
    public CommandLineRunner commandLineRunner(CustomerRepository customerRepository, RepositoryRestConfiguration restConfiguration) {
        return args -> {
            restConfiguration.exposeIdsFor(Customer.class);

            customerRepository.saveAll(
                List.of(
                    Customer.builder().name("Hassan").email("hassan@gmail.com").build(),
                    Customer.builder().name("Hanane").email("hanane@gmail.com").build(),
                    Customer.builder().name("Imane").email("imane@gmail.com").build()
                )
            );
            customerRepository.findAll().forEach(c -> {
                System.out.println(c);
            });
        };
    }
}
```

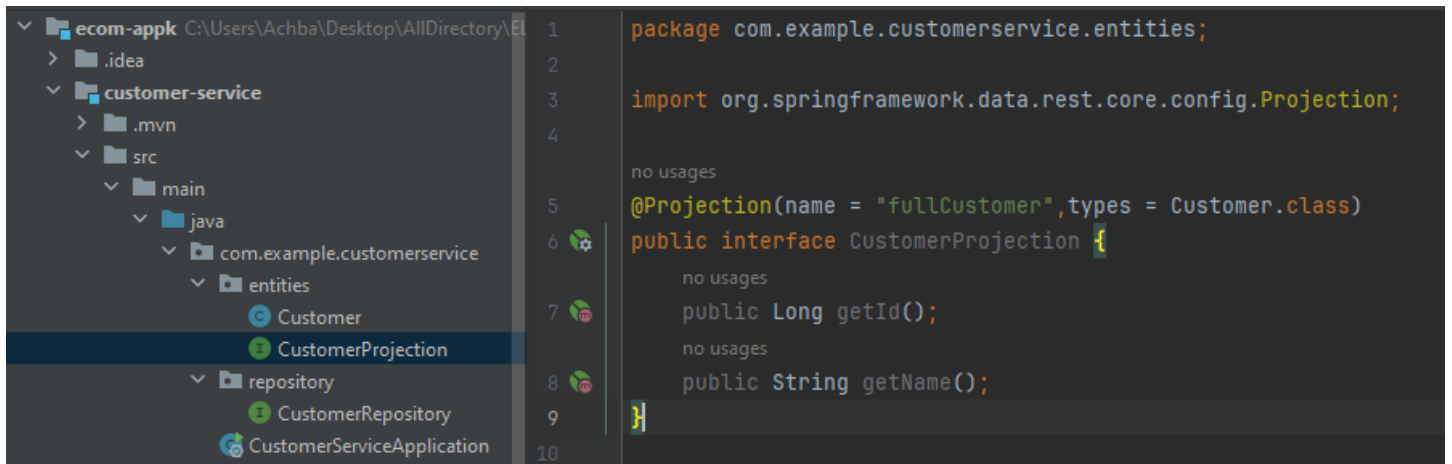


← → ↺ ⓘ localhost:9081/customers/1

stagecentredïouri Apprendre java Apprendre Laravel outlook pfe

```
{
  "id" : 1,
  "name" : "Hassan",
  "email" : "hassan@gmail.com",
  "_links" : {
    "self" : {
      "href" : "http://localhost:9081/customers/1"
    },
    "customer" : {
      "href" : "http://localhost:9081/customers/1"
    }
  }
}
```

On peut toujours Contrôler nos affichage en utilisant @Projection



```
package com.example.customerservice.entities;

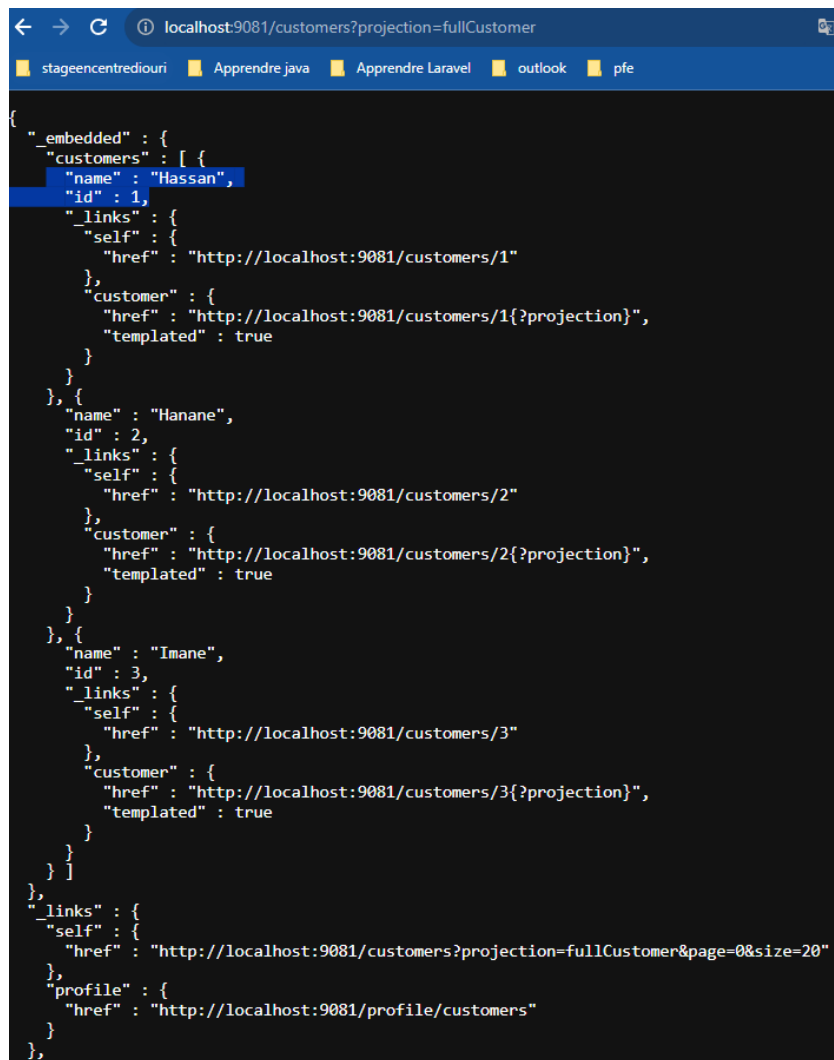
import org.springframework.data.rest.core.config.Projection;

no usages
@Projection(name = "fullCustomer", types = Customer.class)
public interface CustomerProjection {

    no usages
    public Long getId();

    no usages
    public String getName();
}
```

J'ai donc que l'affichage de l'id et son nom .



```
{
  "_embedded" : {
    "customers" : [ {
      "name" : "Hassan",
      "id" : 1,
      "_links" : {
        "self" : {
          "href" : "http://localhost:9081/customers/1"
        },
        "customer" : {
          "href" : "http://localhost:9081/customers/1{?projection}",
          "templated" : true
        }
      }
    }, {
      "name" : "Hanane",
      "id" : 2,
      "_links" : {
        "self" : {
          "href" : "http://localhost:9081/customers/2"
        },
        "customer" : {
          "href" : "http://localhost:9081/customers/2{?projection}",
          "templated" : true
        }
      }
    }, {
      "name" : "Imane",
      "id" : 3,
      "_links" : {
        "self" : {
          "href" : "http://localhost:9081/customers/3"
        },
        "customer" : {
          "href" : "http://localhost:9081/customers/3{?projection}",
          "templated" : true
        }
      }
    }
  ]
}, {
  "_links" : {
    "self" : {
      "href" : "http://localhost:9081/customers?projection=fullCustomer&page=0&size=20"
    },
    "profile" : {
      "href" : "http://localhost:9081/profile/customers"
    }
  }
}
```


II. Créer le micro-service inventory-service :

Par la même manière ,on crée ce service.

```
ProductRepository.java x
1 package com.example.inventoryservice.repository;
2
3 import com.example.inventoryservice.entities.Product;
4 import org.springframework.data.jpa.repository.JpaRepository;
5 import org.springframework.data.rest.core.annotation.RepositoryRestResource;
6
7 @RepositoryRestResource
8 public interface ProductRepository extends JpaRepository<Product,Long> {
9 }

InventoryServiceApplication.java x
12
13 @SpringBootApplication
14 public class InventoryServiceApplication {
15     public static void main(String[] args) { SpringApplication.run(InventoryServiceApplication.class, args); }
16
17     @Bean
18     CommandLineRunner start(ProductRepository productRepository, RepositoryRestConfiguration restConfiguration){
19         return args -> {
20             restConfiguration.exposeIdsFor(Product.class);
21             productRepository.saveAll(
22                 List.of(
23                     Product.builder().name("Computer").quantity(12).price(1200).build(),
24                     Product.builder().name("Printer").quantity(32).price(120).build(),
25                     Product.builder().name("Smartphone").quantity(31).price(900).build()
26                 )
27             )
28         }
29     }
30 }
```

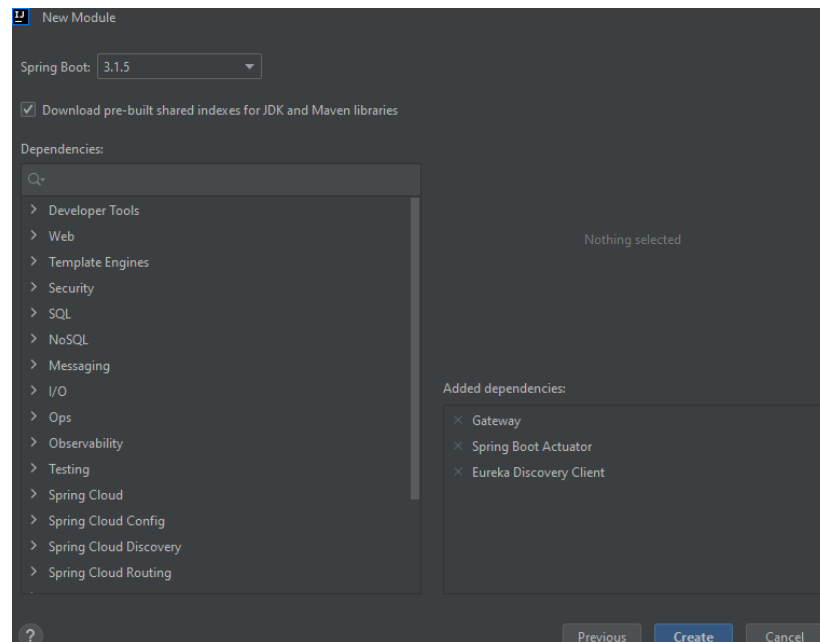
```
localhost:9082/products
stagecentredioni Apprendre java Apprendre Laravel outl

{
  "embedded": {
    "products": [ {
      "id": 1,
      "name": "Computer",
      "price": 1200.0,
      "quantity": 12,
      "links": {
        "self": {
          "href": "http://localhost:9082/products/1"
        }
      },
      "product": {
        "href": "http://localhost:9082/products/1"
      }
    }, {
      "id": 2,
      "name": "Printer",
      "price": 120.0,
      "quantity": 32,
      "links": {
        "self": {
          "href": "http://localhost:9082/products/2"
        }
      },
      "product": {
        "href": "http://localhost:9082/products/2"
      }
    }, {
      "id": 3,
      "name": "Smartphone",
      "price": 900.0,
      "quantity": 31,
      "links": {
        "self": {
          "href": "http://localhost:9082/products/3"
        }
      },
      "product": {
        "href": "http://localhost:9082/products/3"
      }
    }
  ]
}, {
  "links": {
    "self": {
      "href": "http://localhost:9082/products?page=0&size=20"
    }
  },
  "profile": {
    "href": "http://localhost:9082/profile/products"
  }
}, {
  "page": {
    "size": 20,
    "totalElements": 3,
    "totalPages": 1,
    "number": 0
  }
}
}
```

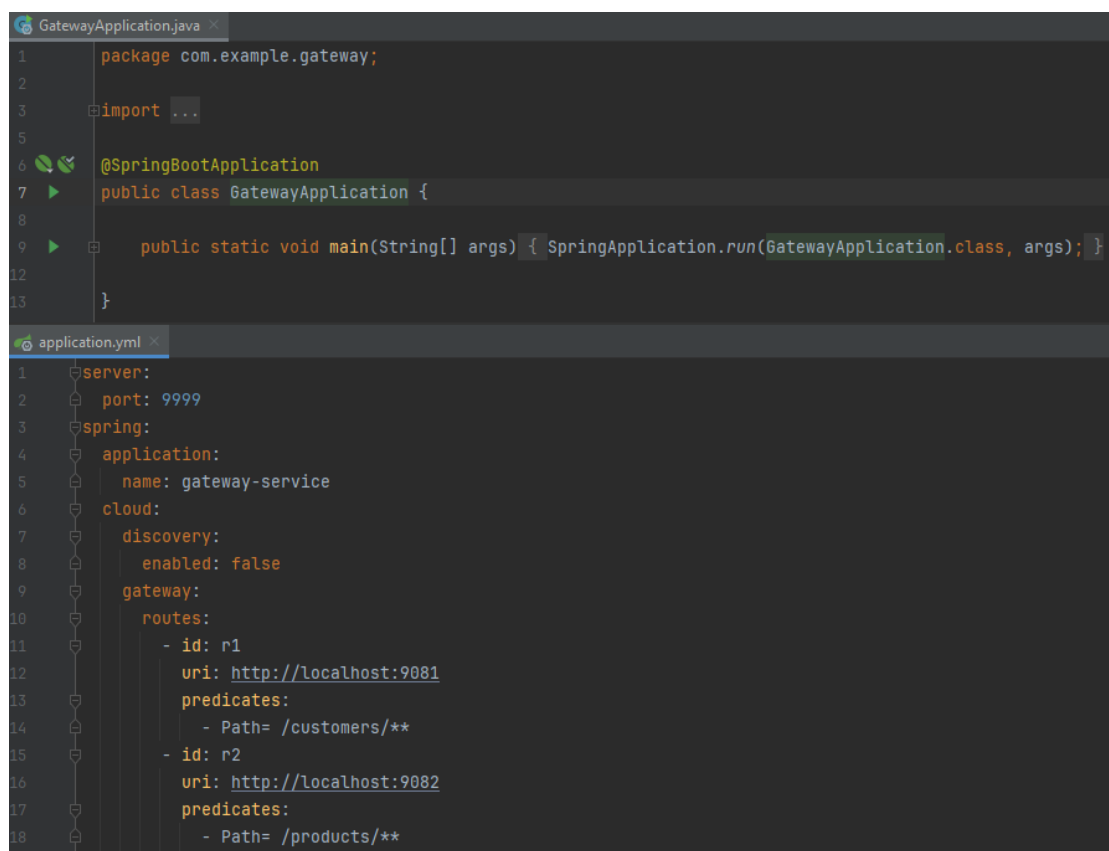
```
application.properties x
1 spring.application.name=inventory-service
2 server.port=9082
3 spring.datasource.url=jdbc:h2:mem:products-db
4 spring.cloud.discovery.enabled=false
5
```

III. Créer la Gateway :

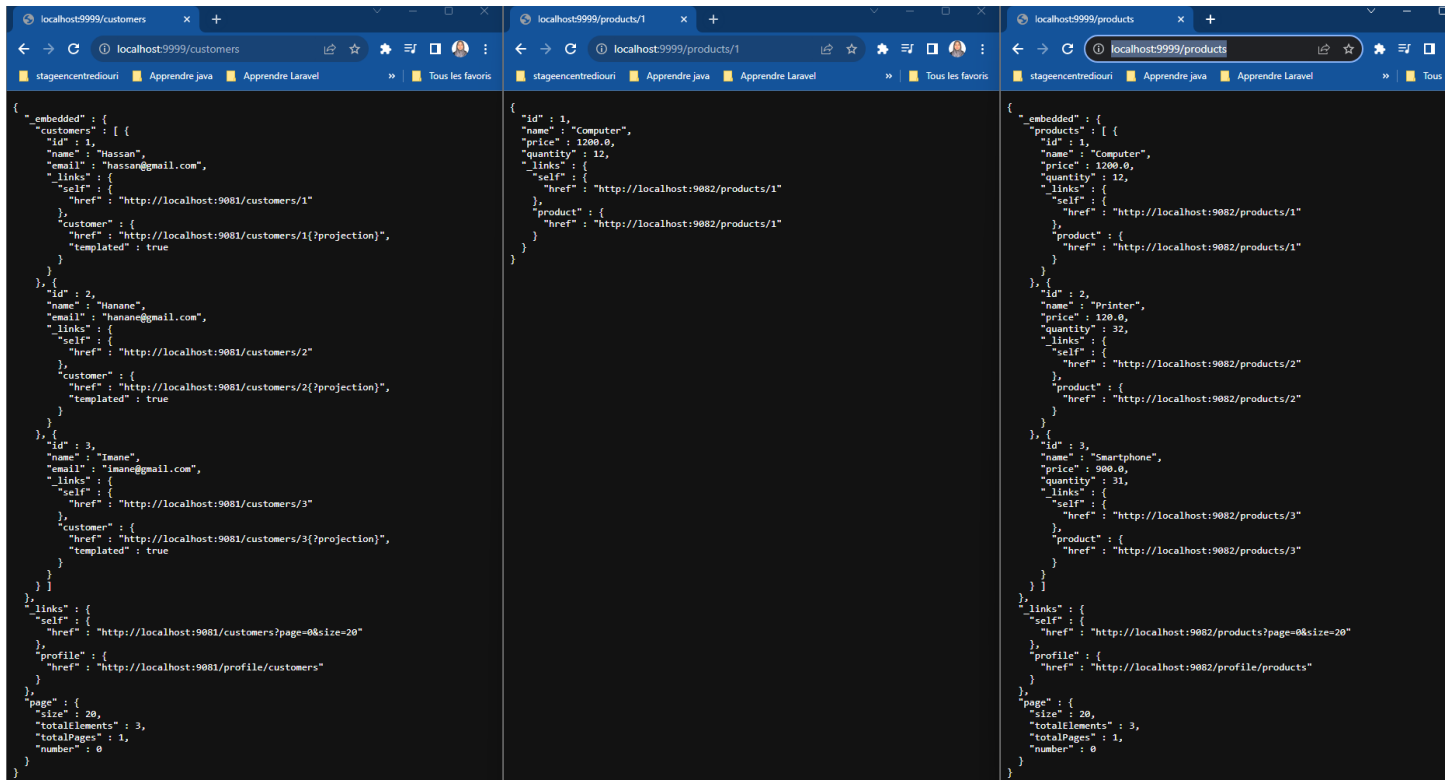
Pour créer la gateway, on crée un projet spring avec ces dépendances .



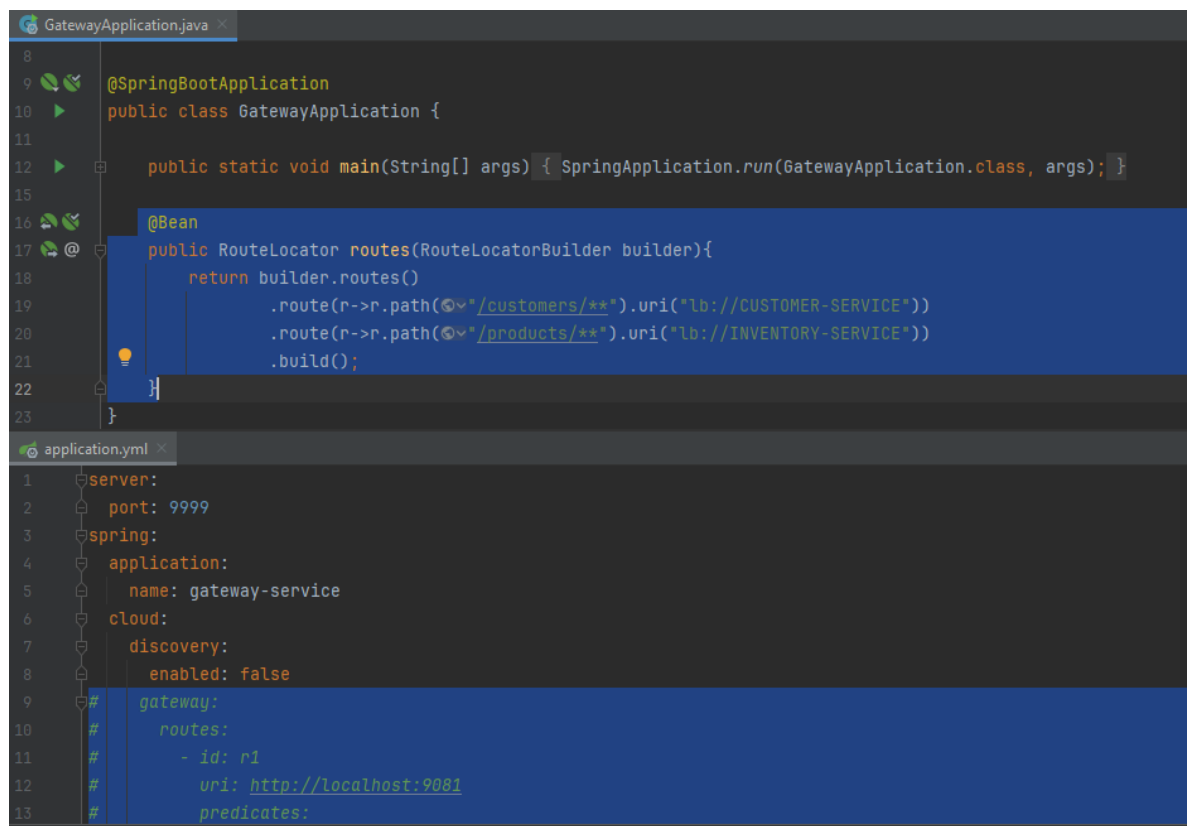
Pour gérer la configuration statique de la Gateway , on utilise un fichier yml comme définie dans l'image .



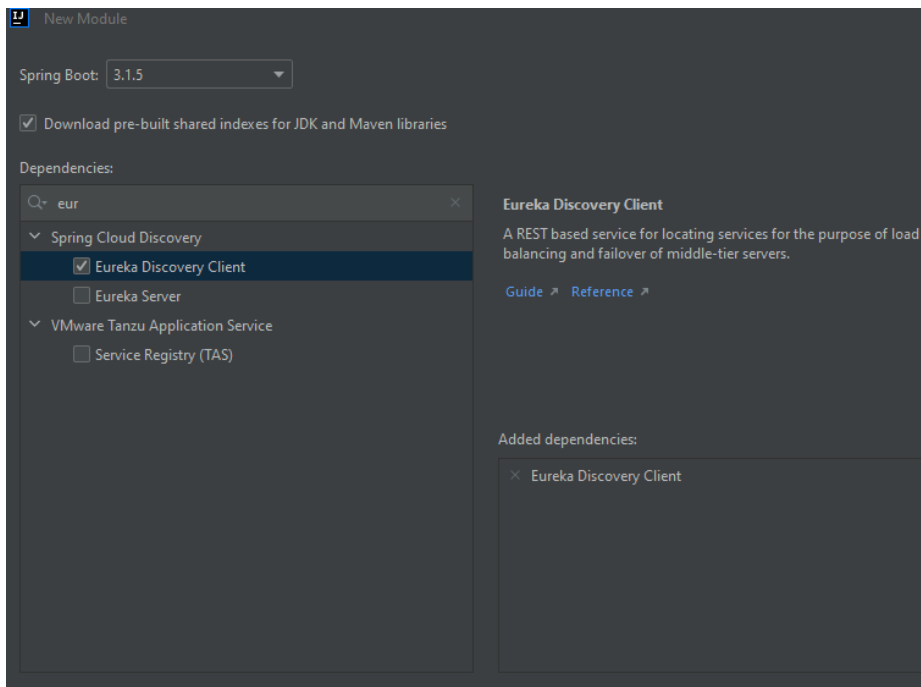
Je peut donc consulter les deux webservices en utilisant la Gateway .



Il existe une autre méthode statique , qui est l'utilisation d'une classe java .



IV. Créer le Discovery-service :



On ajoute @ENAb

