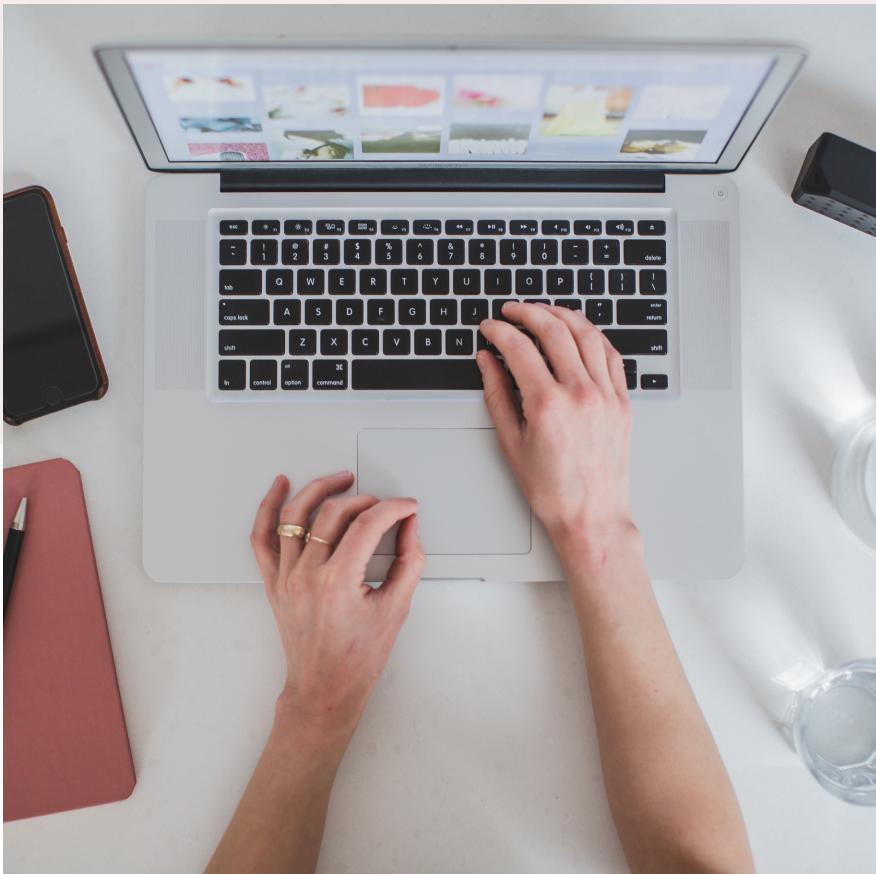


# RAPPORT DU TP MICRO-SERVICES JEE



RÉALISÉ PAR :  
OUASSINE ANAS  
ENSEIGNÉ :  
MOHAMED ELYOUSSFI

Ce document représente un rapport du TP des "Distributed High Performance Computing with Microservices architecture"

# Introduction

Le but de ce TP est de réaliser d'une application distribuée en appliquant les règles et l'architecture des micro-services en utilisant Spring Boot et Spring Cloud

Les micro-services avec lesquels l'application sera créée :

- Customer-Service
- Inventory-Service
- Billing-Service
- Gateway-Service
- Eureka Discovery-Service
- Athentification-Service

# Ennoncé

1. Créer le micro service Customer-service:

- Créer l'entité Customer
- Créer l'interface CustomerRepository basée sur Spring Data
- Déployer l'API Restful du micro-service en utilisant Spring Data Rest
- Tester le Micro service

2. Créer le micro service Inventory-service

- Créer l'entité Product
- Créer l'interface ProductRepository basée sur Spring Data
- Déployer l'API Restful du micro-service en utilisant Spring Data Rest
- Tester le Micro service

3. Créer la Gateway service en utilisant Spring Cloud Gateway

4. Créer l'annuaire Discovery Service basé sur NetFlix Eureka Server

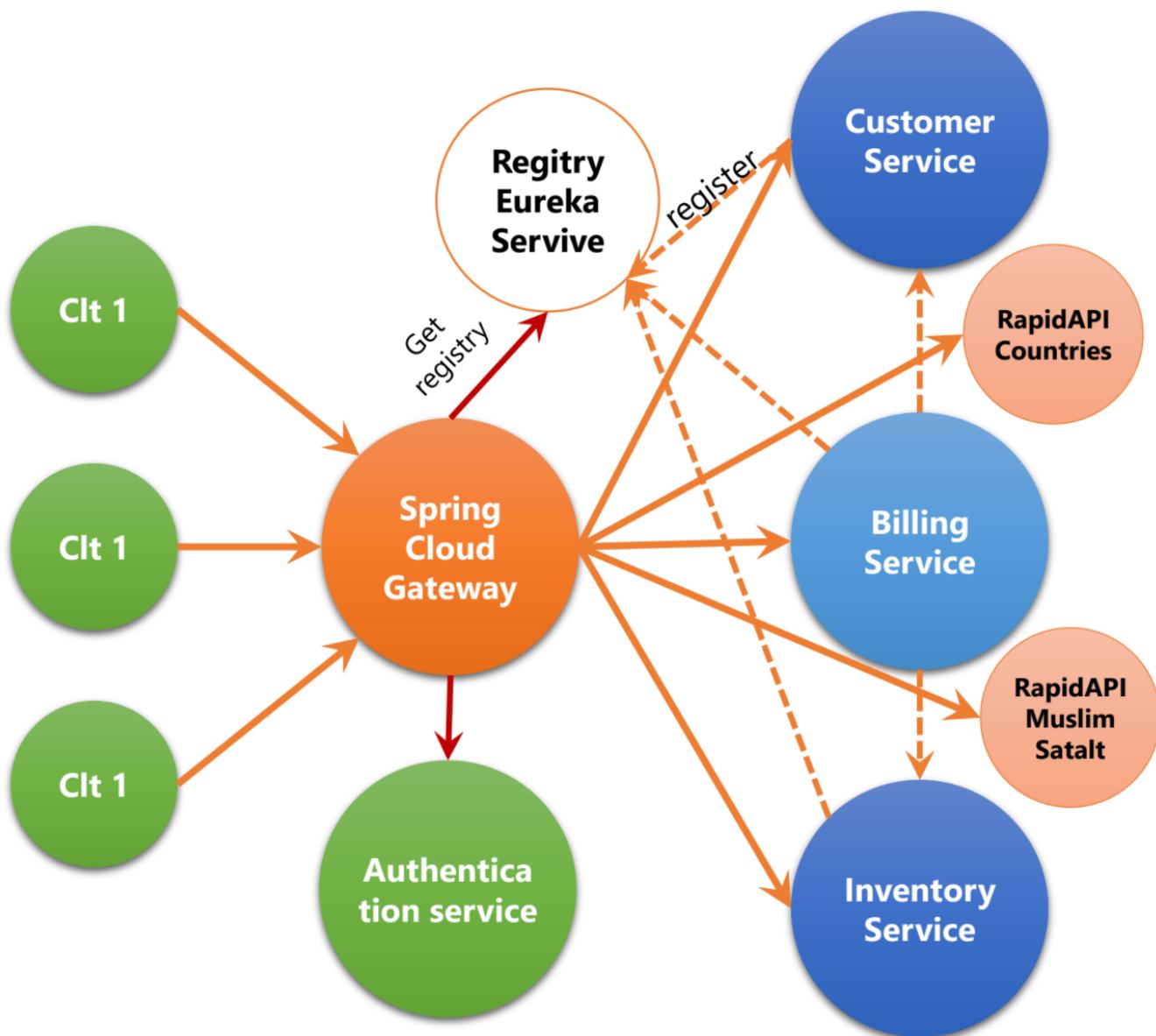
6. Créer le service Billing Service

7. Créer un service d'authentification Statless basé sur Spring Security et Json Web Token. Ce service devrait permettre de :

- Gérer les utilisateurs et les rôles de l'application
- Authentifier un utilisateur en lui délivrant un access Token et un refresh Token de type JWT
- Gérer les autorisation d'accès
- Renouveler l'access
- Token à l'aide du refresh Token

# plan de l'application

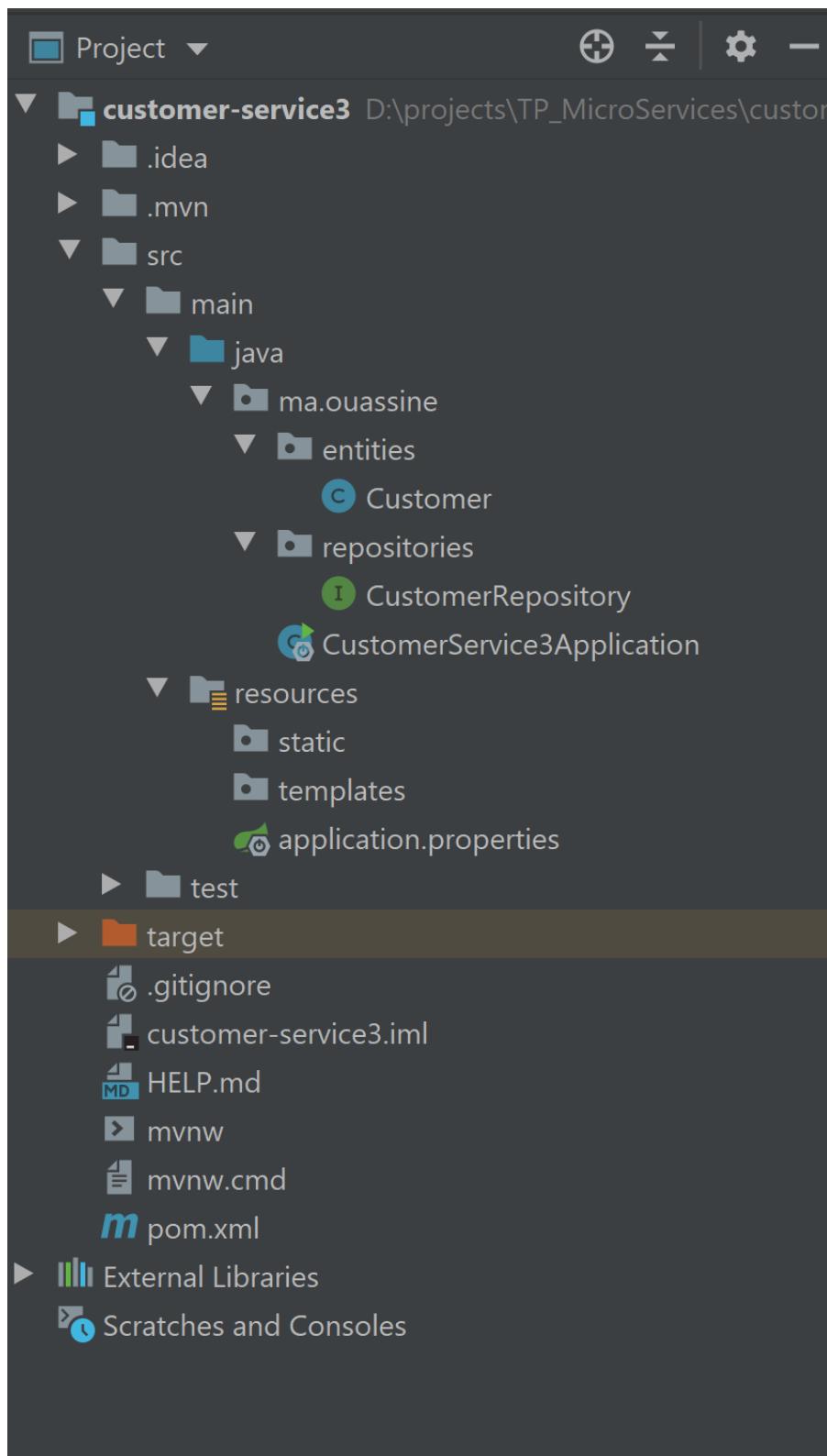
L'application consiste à créer six micro-services, le premier est pour la gestion des clients (Customer Service), le deuxième est pour la gestion de l'inventaire (Inventory Service), le troisième service est celui nommé Gateway service, et qui permet l'accès au 2 premiers services, cet accès vas se faire via le quatrième service qui est les service d'enregistrement Eureka service, le cinquième service est le service de facturation (Billing Service) et le dernier service qui est un service d'authentication qu'on va sécurisé grâce à Spring Security



# **Customer Service**

Customer service est le service responsable de la gestion des clients, c'est une application Spring où on aura besoin des dépendances suivantes :

- Spring Web
- Spring Data JPA
- H2 Database
- Rest Repositories
- Lombok
- Spring Boot Dev Tools
- Eureka Discovery Client
- Spring Boot Actuator



squelette du projet Customer service

## Package : Entities

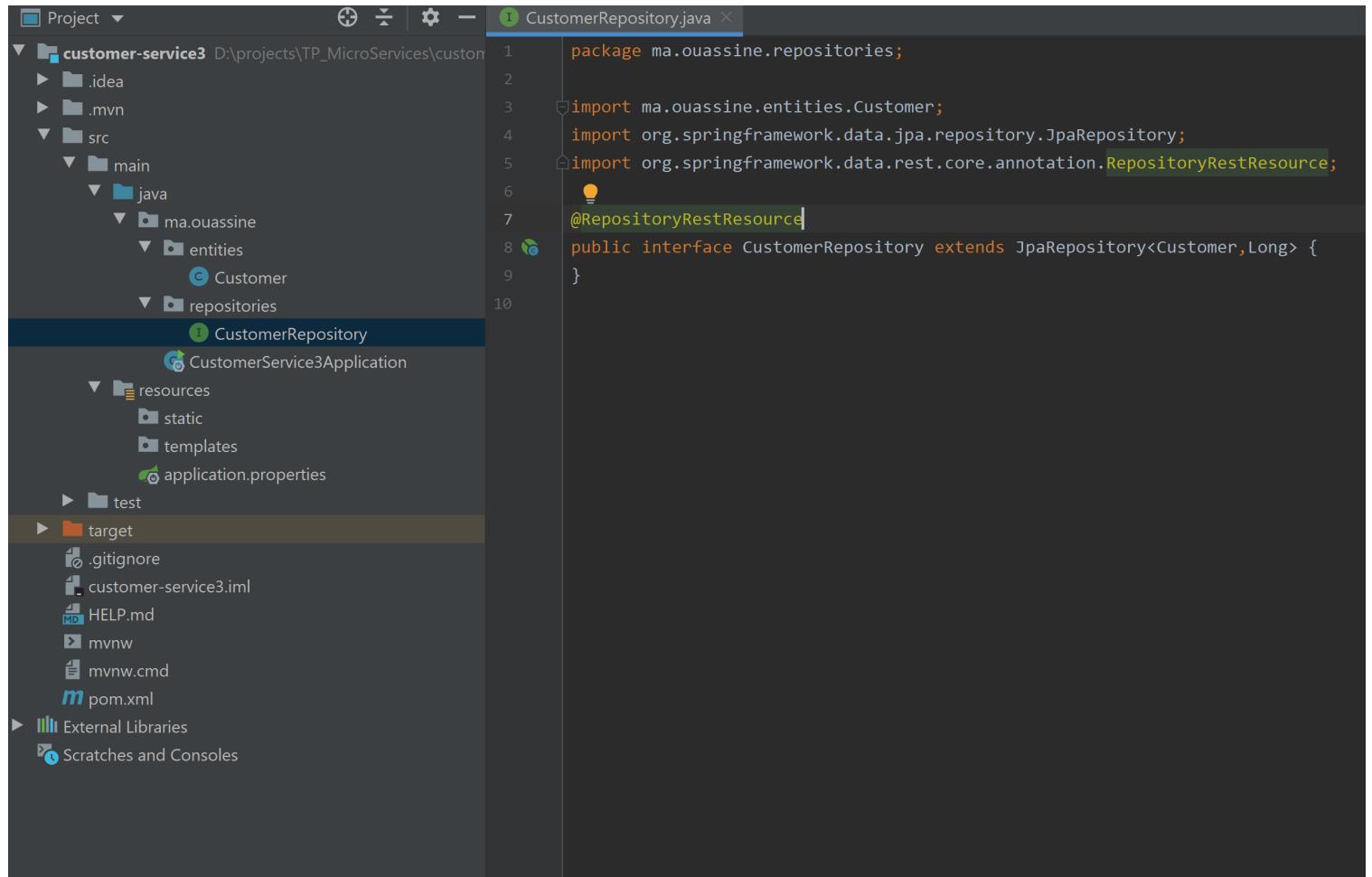
The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Tree:** The project is named "customer-service3" located at "D:\projects\TP\_MicroServices\customer-service3". It contains .idea, .mvn, src, test, target, pom.xml, and various configuration files.
- Customer.java File:** The current file is "Customer.java" located in the "ma.ouassine.entities" package under "src/main/java".
- Code Content:**

```
1 package ma.ouassine.entities;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Data;
5 import lombok.NoArgsConstructor;
6 import lombok.ToString;
7
8 import javax.persistence.Entity;
9 import javax.persistence.GeneratedValue;
10 import javax.persistence.GenerationType;
11 import javax.persistence.Id;
12
13 @Entity
14 @Data @NoArgsConstructor @AllArgsConstructor @ToString
15 public class Customer {
16     @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
17     private Long id;
18     private String name;
19     private String email;
20 }
21
```

## Class Customer

# Package : Repositories



The screenshot shows the IntelliJ IDEA interface with the following details:

- Project Bar:** Shows "customer-service3" as the active project.
- File Structure:** The left sidebar shows the project structure:
  - src:** Contains .idea, .mvn, main, resources, test, target, pom.xml, and External Libraries.
  - main.java:** Contains ma.ouassine.entities.Customer and ma.ouassine.repositories.CustomerRepository.
  - CustomerService3Application:** The main application class.
  - resources:** Contains static, templates, and application.properties.
- Code Editor:** The right pane displays the code for **CustomerRepository.java**:

```
1 package ma.ouassine.repositories;
2
3 import ma.ouassine.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 }
```

## Customer Repository

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project:** customer-service3
- File:** CustomerService3Application.java
- Code Content:**

```
import ma.ouassine.repositories.CustomerRepository;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.client.discovery.EnableDiscoveryClient;
import org.springframework.context.annotation.Bean;
import org.springframework.data.rest.core.config.RepositoryRestConfiguration;

@SpringBootApplication
public class CustomerService3Application {

    public static void main(String[] args) {
        SpringApplication.run(CustomerService3Application.class, args);
    }

    @Bean
    CommandLineRunner start(CustomerRepository customerRepository, RepositoryRestConfiguration restConfiguration) {
        restConfiguration.exposeIdsFor(Customer.class);
        return args -> {
            customerRepository.save(new Customer( id: null, name: "Ouassine", email: "oua@mail.ma"));
            customerRepository.save(new Customer( id: null, name: "lol", email: "lol@lol.lol"));
            customerRepository.save(new Customer( id: null, name: "aoa", email: "ua@oua.ma"));
            customerRepository.save(new Customer( id: null, name: "Anas", email: "oua@mail.ma"));
            for (Customer c : customerRepository.findAll()) {
                System.out.println(c.toString());
            }
        };
    }
}
```

## L'application Spring principale

The screenshot shows the IntelliJ IDEA interface with the project 'customer-service3' open. The left sidebar displays the project structure, including the src directory with main/java and main/resources, and various configuration files like pom.xml and application.properties. The right panel shows the contents of the application.properties file.

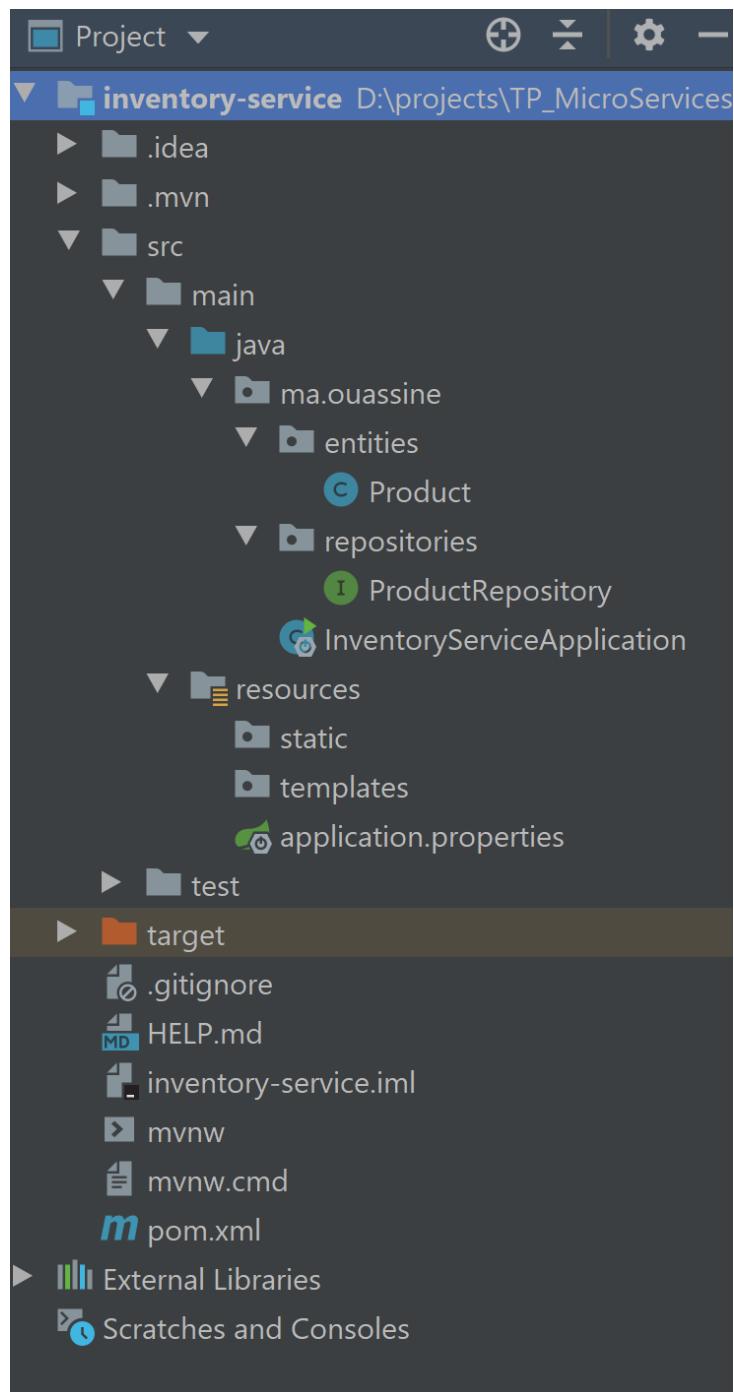
```
server.port=8081  
spring.application.name=customer-service  
spring.datasource.url=jdbc:h2:mem:customer-db  
spring.cloud.discovery.enabled=true  
#management.endpoints.web.exposure.include=*  
  
eureka.instance.prefer-ip-address=true
```

## Application.properties

# Inventory Service

Inventory service est le service responsable de la gestion des produits, les dépendances utilisées sont :

- Spring Web
- Spring Data JPA
- H2 Database
- Rest Repositories
- Lombok
- Spring Boot Dev Tools
- Eureka Discovery Client
- Spring Boot Actuator

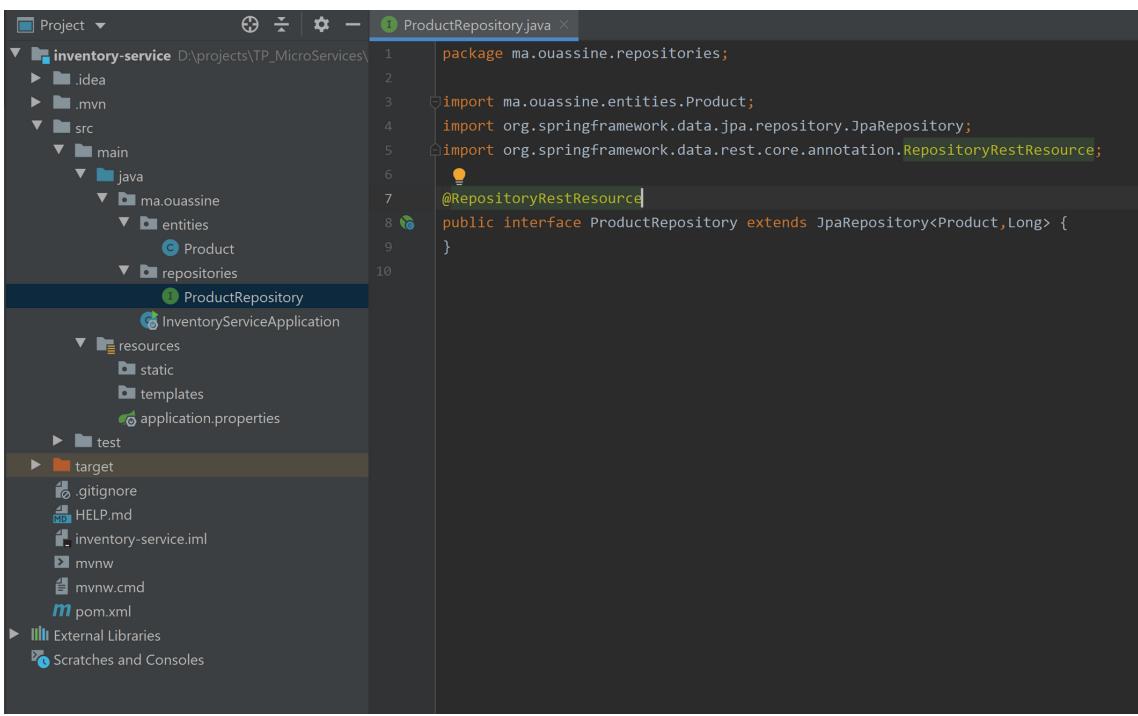


squelette du projet Inventory service

The screenshot shows the file structure of a Java project named `inventory-service` located at `D:\projects\TP_MicroServices\`. The project structure includes `.idea`, `.mvn`, `src` (containing `main` and `test`), `target`, and configuration files like `.gitignore`, `HELP.md`, `inventory-service.iml`, `mvnw`, `mvnw.cmd`, and `pom.xml`. The `src/main/java` directory contains packages `ma.ouassine` and `entities`, with a `Product` class selected. The `Product` class is annotated with `@Entity`, `@Data`, `@AllArgsConstructor`, `@NoArgsConstructor`, and `@ToString`. It has fields `id`, `name`, `price`, and `quantity`.

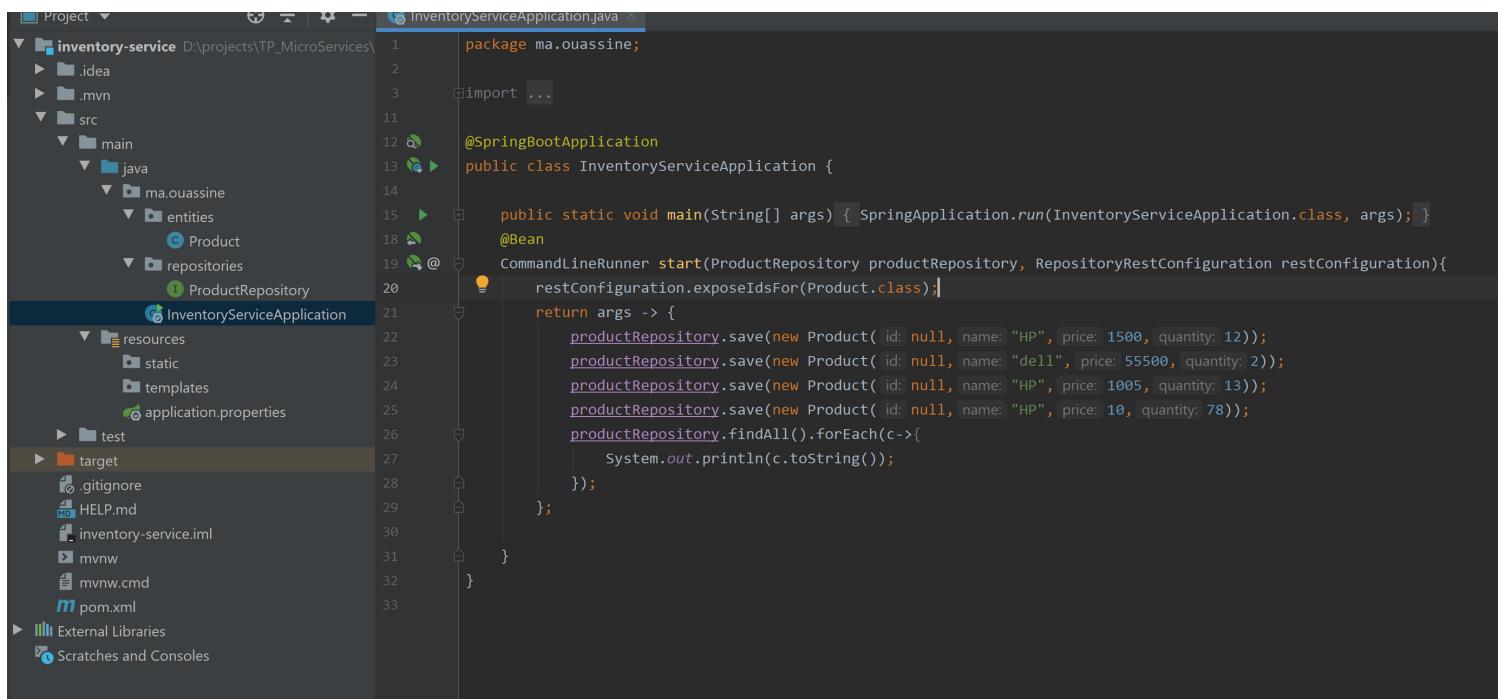
```
1 package ma.ouassine.entities;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Data;
5 import lombok.NoArgsConstructor;
6 import lombok.ToString;
7
8 import javax.persistence.Entity;
9 import javax.persistence.GeneratedValue;
10 import javax.persistence.GenerationType;
11 import javax.persistence.Id;
12
13 @Entity
14 @Data @AllArgsConstructor @NoArgsConstructor @ToString
15 public class Product {
16     @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
17     private Long id;
18     private String name;
19     private double price;
20     private double quantity;
21 }
22
```

## Class Inventory

A screenshot of an IDE showing the code for a repository interface. The project structure on the left shows a Java application named 'inventory-service' with a 'src' directory containing 'main' and 'repositories'. The 'repositories' directory contains a file named 'ProductRepository.java'. The code in the editor is as follows:

```
1 package ma.ouassine.repositories;
2
3 import ma.ouassine.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 }
```

## Inventory Repository



The screenshot shows a Java application named `InventoryServiceApplication`. The code is annotated with Spring annotations like `@SpringBootApplication`, `@Bean`, and `CommandLineRunner`. It uses a repository interface `ProductRepository` to save products and then prints them to the console. The project structure includes a `src/main/java` directory containing `ma.ouassine` and `entities` packages, and a `resources` directory with `application.properties`.

```
1 package ma.ouassine;
2
3 import ...
4
5 @SpringBootApplication
6 public class InventoryServiceApplication {
7
8     public static void main(String[] args) { SpringApplication.run(InventoryServiceApplication.class, args); }
9
10    @Bean
11    CommandLineRunner start(ProductRepository productRepository, RepositoryRestConfiguration restConfiguration){
12        restConfiguration.exposeIdsFor(Product.class);
13        return args -> {
14            productRepository.save(new Product( id: null, name: "HP", price: 1500, quantity: 12));
15            productRepository.save(new Product( id: null, name: "dell", price: 55500, quantity: 2));
16            productRepository.save(new Product( id: null, name: "HP", price: 1005, quantity: 13));
17            productRepository.save(new Product( id: null, name: "HP", price: 10, quantity: 78));
18            productRepository.findAll().forEach(c->{
19                System.out.println(c.toString());
20            });
21        };
22    }
23
24
25
26
27
28
29
30
31
32
33 }
```

## L'application Spring principale

The screenshot shows a Java-based microservice project structure in an IDE. The project is named "inventory-service" and is located at "D:\projects\TP\_MicroServices\". The "src" directory contains "main" and "resources" packages. "main" further contains "java" (with "ma.ouassine/entities/Product" and "repositories/ProductRepository"), "InventoryServiceApplication", and "resources" (with "static" and "templates"). "resources" also contains "application.properties". The "application.properties" file is open in the editor, displaying the following configuration:

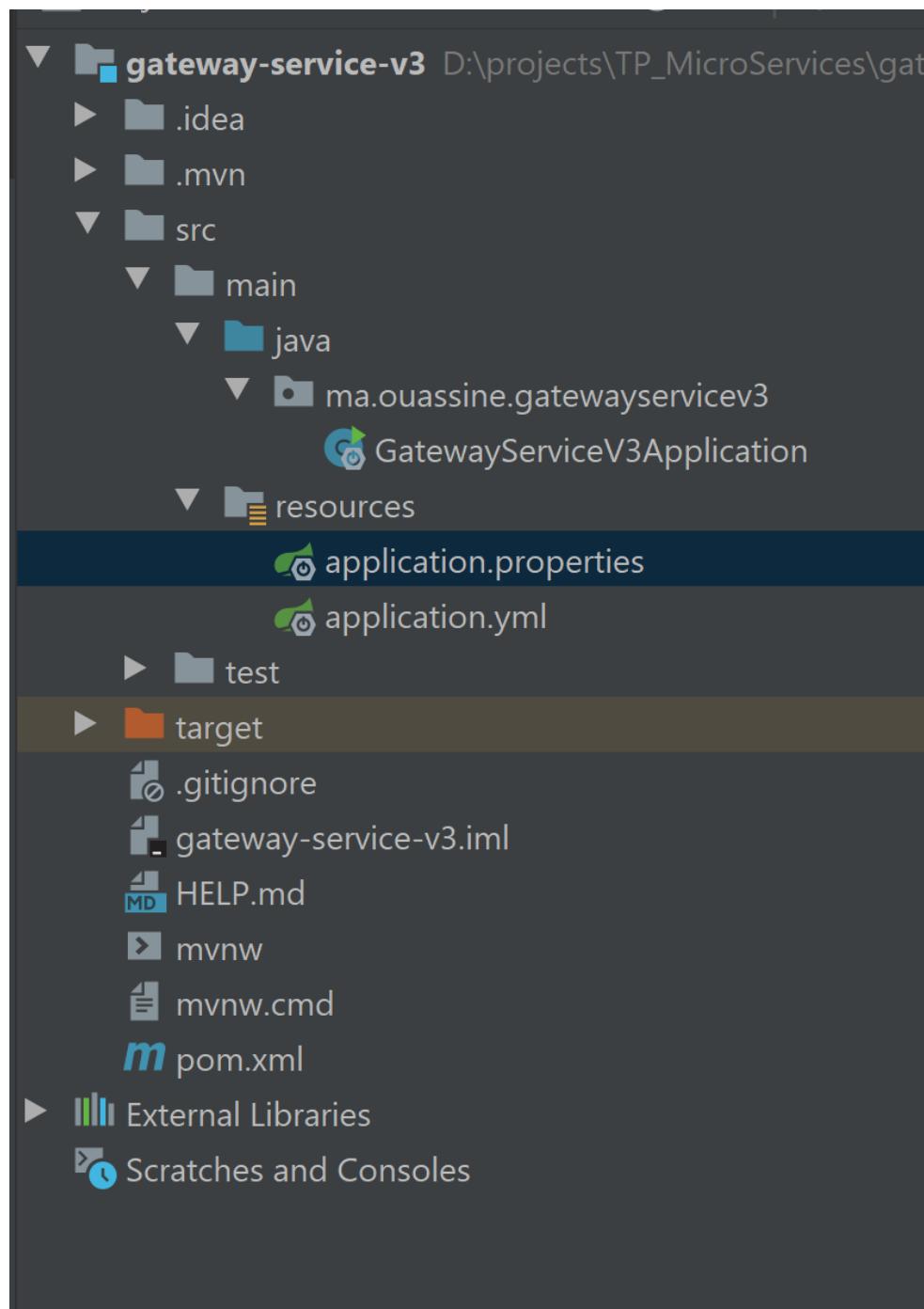
```
server.port=8082
spring.application.name=product-service
spring.datasource.url=jdbc:h2:mem:product-db
spring.cloud.discovery.enabled=true
#management.endpoints.web.exposure.include=*
eureka.instance.prefer-ip-address=true
```

## Application.Properties

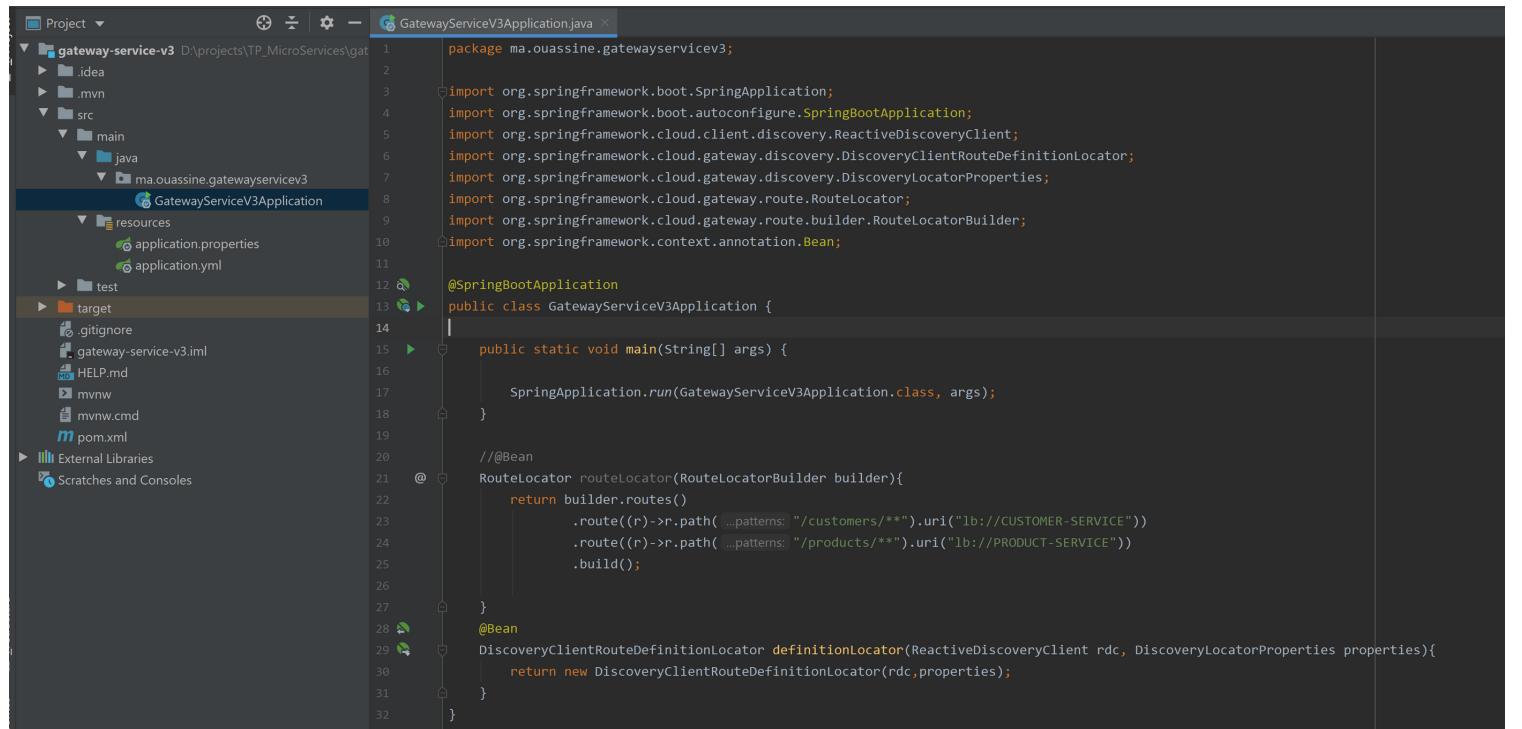
# **Gateway Service**

Les dépendances utilisées sont :

- Gateway
- Spring Boot Actuator
- Hystrix
- Eureka Discovery Client



squelette du projet Gateway service



The screenshot shows an IDE interface with a project structure on the left and a code editor on the right. The project structure for 'gateway-service-v3' includes '.idea', '.mvn', 'src' (containing 'main' and 'ma.ouassine.gatewayservicev3'), 'resources' (containing 'application.properties' and 'application.yml'), 'test', 'target', and 'External Libraries'. The code editor displays 'GatewayServiceV3Application.java' with the following content:

```
1 package ma.ouassine.gatewayservicev3;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.cloud.client.discovery.ReactiveDiscoveryClient;
6 import org.springframework.cloud.gateway.discovery.DiscoveryClientRouteDefinitionLocator;
7 import org.springframework.cloud.gateway.discovery.DiscoveryLocatorProperties;
8 import org.springframework.cloud.gateway.route.RouteLocator;
9 import org.springframework.cloud.gateway.route.builder.RouteLocatorBuilder;
10 import org.springframework.context.annotation.Bean;
11
12 @SpringBootApplication
13 public class GatewayServiceV3Application {
14
15     public static void main(String[] args) {
16
17         SpringApplication.run(GatewayServiceV3Application.class, args);
18     }
19
20     //@Bean
21     RouteLocator routeLocator(RouteLocatorBuilder builder){
22
23         return builder.routes()
24             .route((r)->r.path( ...patterns: "/customers/**").uri("lb://CUSTOMER-SERVICE"))
25             .route((r)->r.path( ...patterns: "/products/**").uri("lb://PRODUCT-SERVICE"))
26             .build();
27     }
28
29     @Bean
30     DiscoveryClientRouteDefinitionLocator definitionLocator(ReactiveDiscoveryClient rdc, DiscoveryLocatorProperties properties){
31
32         return new DiscoveryClientRouteDefinitionLocator(rdc,properties);
33     }
34 }
```

## L'application Spring principale

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure under "gateway-service-v3".
- Toolbars:** Standard IntelliJ IDEA toolbars.
- Editors:** Three tabs are visible:
  - `GatewayServiceV3Application.java`
  - `application.properties`
  - `application.yml` (selected)
- Code Content (application.yml):**

```
1  spring:
2    cloud:
3      gateway:
4        routes:
5          - id: r1
6            uri: http://localhost:8081/
7            predicates:
8              - Path= /customers/**
9          - id: r2
10            uri: http://localhost:8082/
11            predicates:
12              - Path= /products/**
```

## Application.yml

The screenshot shows the IntelliJ IDEA interface with the project 'gateway-service-v3' open. The left sidebar displays the project structure, including .idea, .mvn, src (with main and resources), test, target, and various configuration files like pom.xml and mvnw. The right panel shows the 'application.properties' file content:

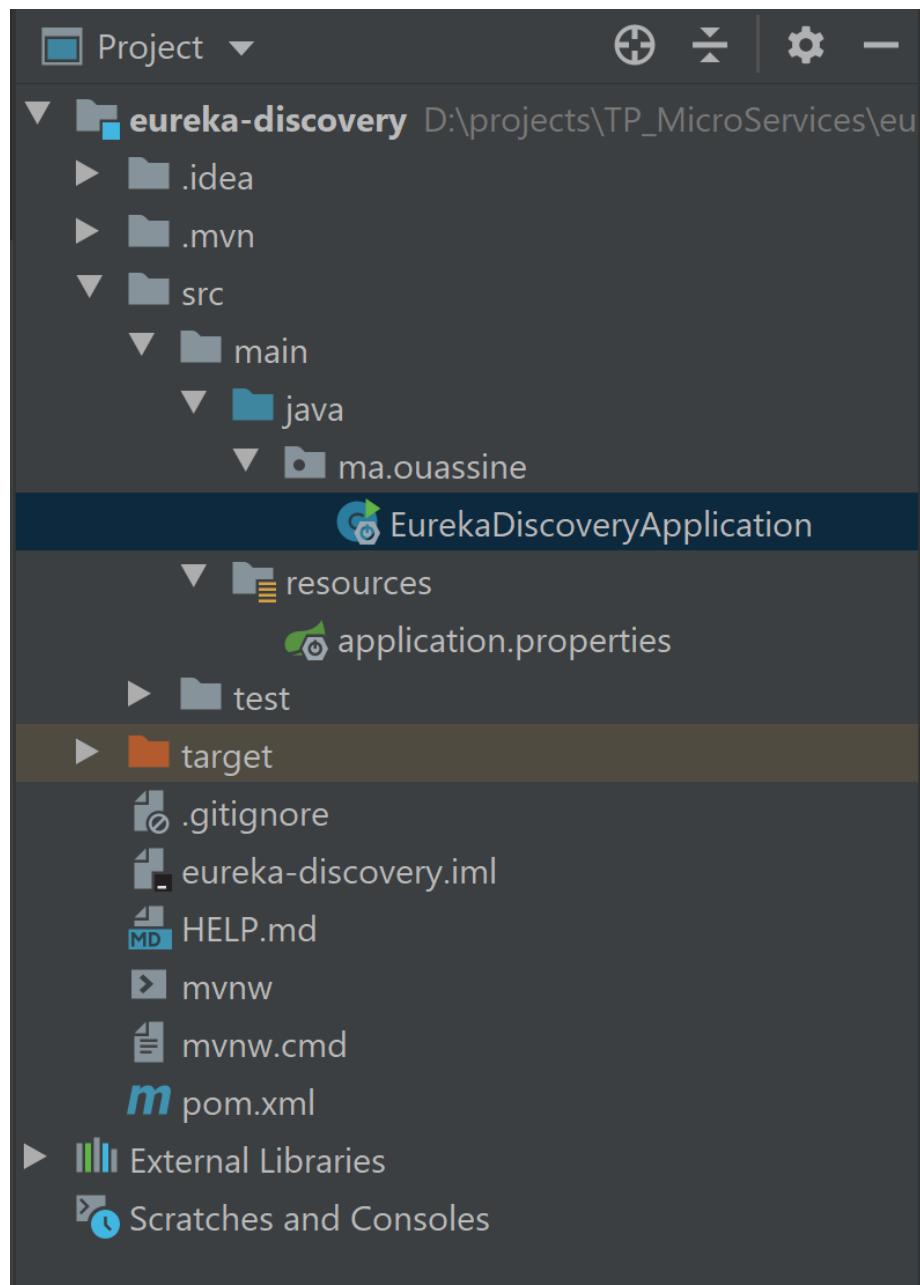
```
server.port=8888  
spring.application.name=gateway-service2  
spring.cloud.discovery.enabled=true  
spring.cloud.gateway.loadbalancer.use404=true  
spring.cloud.loadbalancer.retry.enabled=true  
eureka.instance.prefer-ip-address=true
```

## Application.properties

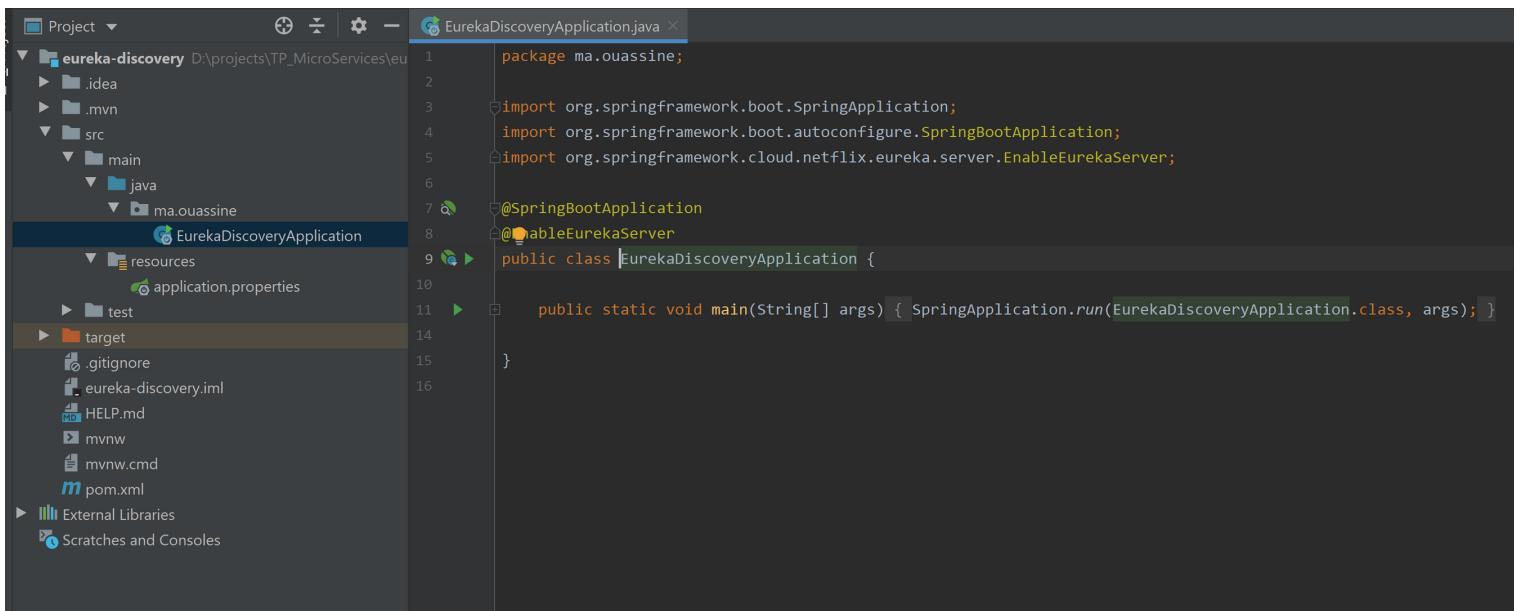
# **Eureka Service**

Le service d'enregistrement Eureka, les dépendances utilisées sont :

- Eureka Server
- Spring Cloud



squelette du projet Eureka service



The screenshot shows the IntelliJ IDEA interface with the project 'eureka-discovery' open. The left sidebar displays the project structure, including .idea, .mvn, src (with main and java subfolders), resources (containing application.properties), test, target (containing .gitignore, eureka-discovery.iml, HELP.md, mvnw, mvnw.cmd, and pom.xml), External Libraries, and Scratches and Consoles. The right pane shows the code editor with the file EurekaDiscoveryApplication.java. The code is as follows:

```
1 package ma.ouassine;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;
6
7 @SpringBootApplication
8 @EnableEurekaServer
9 public class EurekaDiscoveryApplication {
10
11     public static void main(String[] args) { SpringApplication.run(EurekaDiscoveryApplication.class, args); }
12
13 }
14
15 }
16
```

## L'application Spring principale

The screenshot shows the IntelliJ IDEA interface with the project 'eureka-discovery' open. The left sidebar displays the project structure, including .idea, .mvn, src (with main/java/ma.ouassine/EurekaDiscoveryApplication and resources/application.properties), test, target, .gitignore, eureka-discovery.iml, HELP.md, mvnw, mvnw.cmd, and pom.xml. The right panel shows the 'application.properties' file with the following content:

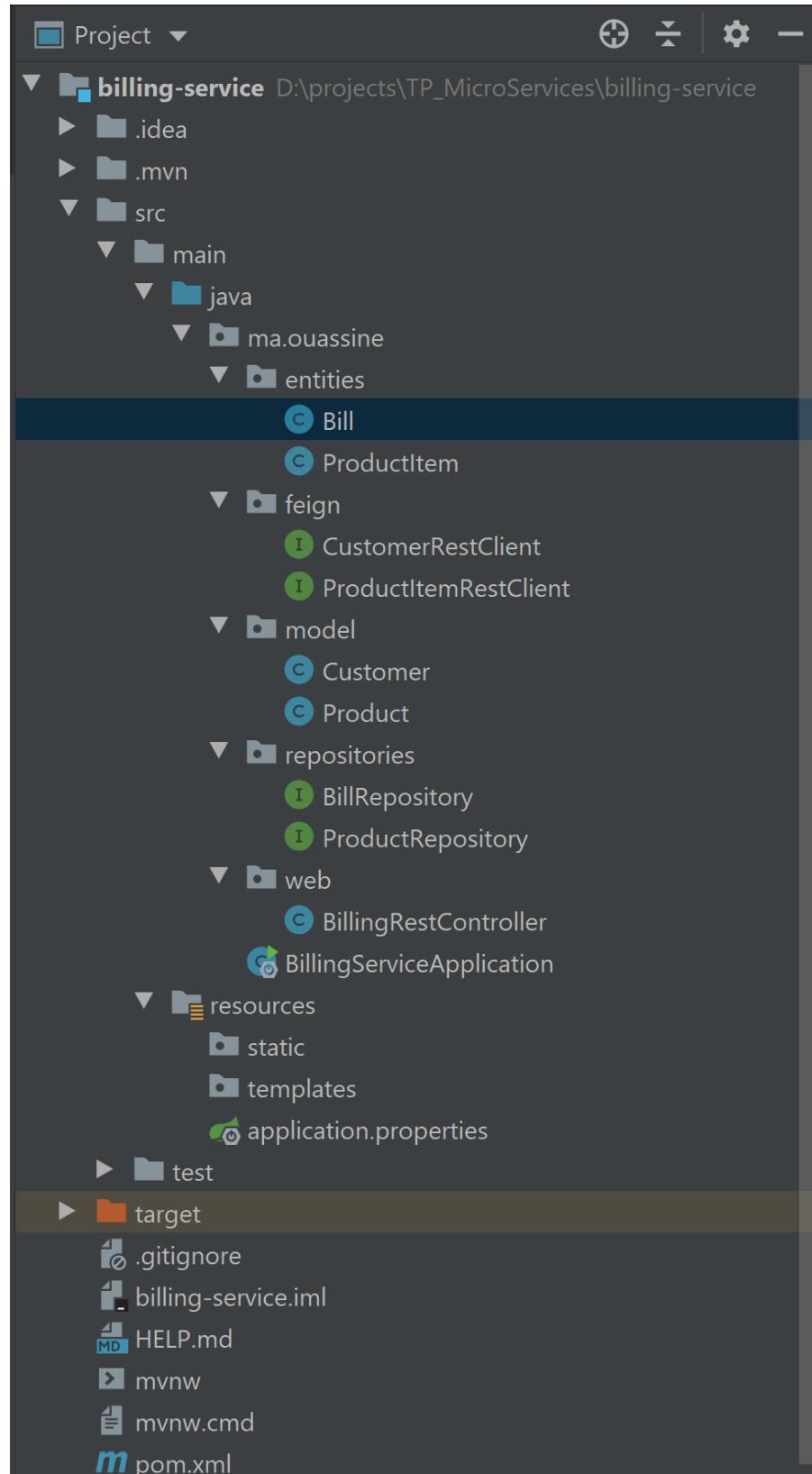
```
server.port=8761  
eureka.client.fetch-registry=false  
eureka.client.register-with-eureka=false  
eureka.instance.prefer-ip-address=true
```

## Application.properties

# **Billing Service**

Billing service est le service responsable de la gestion de facturation, les dépendances utilisées sont :

- Spring Web
- Spring Data JPA
- H2 Database
- Rest Repositories
- Lombok
- Spring Boot Dev Tools
- Eureka Discovery Client
- OpenFeign
- Spring HATEOAS



squelette du projet Billing service

## Package : entities

The screenshot shows a Java project structure in an IDE. The project is named 'billing-service' and is located at 'D:\projects\TP\_MicroServices\billing-service'. The 'src' directory contains several packages:

- ma.ouassine.entities**: Contains the **Bill** class.
- ma.ouassine.model**: Contains **Customer** and **Product**.
- ma.ouassine.repositories**: Contains **BillRepository** and **ProductRepository**.
- ma.ouassine.web**: Contains **BillingRestController** and **BillingServiceApplication**.
- feign**: Contains **CustomerRestClient** and **ProductItemRestClient**.

The **Bill.java** file is open in the editor. It defines a class **Bill** with the following code:

```
1 package ma.ouassine.entities;
2 import lombok.AllArgsConstructor;
3 import lombok.Data;
4 import lombok.NoArgsConstructor;
5 import lombok.ToString;
6 import ma.ouassine.model.Product;
7
8 import javax.persistence.*;
9 import java.util.Collection;
10 import java.util.Date;
11
12 @Entity
13 @Data @NoArgsConstructor@AllArgsConstructor@ToString
14 public class Bill {
15     @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
16     private Long id;
17     private Date billingDate;
18     @OneToMany(mappedBy = "billID")
19     private Collection<ProductItem> productItems;
20     private Long customerID;
21     @Transient
22     private Product product;
23 }
```

## Bill Class

The screenshot shows a Java project structure in an IDE. The project is named "billing-service" and is located at "D:\projects\TP\_MicroServices\billing-service". The structure includes .idea, .mvn, src (containing main, resources, test, and target), and .gitignore. The main directory contains java, resources, static, templates, and application.properties. The java directory has sub-packages ma.ouassine.entities, feign, model, repositories, and web. The ma.ouassine.entities package contains Bill and ProductItem. The feign package contains CustomerRestClient and ProductItemRestClient. The model package contains Customer and Product. The repositories package contains BillRepository and ProductRepository. The web package contains BillingRestController and BillingServiceApplication. The ProductItem.java file is open in the editor, showing its code.

```
1 package ma.ouassine.entities;
2
3 import com.fasterxml.jackson.annotation.JsonProperty;
4 import lombok.AllArgsConstructor;
5 import lombok.Data;
6 import lombok.NoArgsConstructor;
7 import lombok.ToString;
8 import ma.ouassine.model.Customer;
9 import ma.ouassine.model.Product;
10 import org.hibernate.annotations.ManyToOne;
11
12 import javax.persistence.*;
13
14 @Entity
15 @Data @NoArgsConstructor@AllArgsConstructor@ToString
16 public class ProductItem {
17     @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
18     private Long id;
19     private double quantity;
20     private double price;
21     private long productID;
22     @ManyToOne
23     @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
24     private Bill bill;
25     @Transient
26     private Product product;
27     @Transient
28     private String productName;
29 }
```

## ProductItem Class

## Package : Feign

The screenshot shows the IntelliJ IDEA interface with the following details:

- Project View:** Shows the project structure under "billing-service".
  - src:** Contains main, java, resources, and test folders.
  - java:** Contains subfolders for entities, feign, model, repositories, and web, along with CustomerRestClient.java.
  - resources:** Contains static, templates, and application.properties.
- CustomerRestClient.java:** The current file being edited, located at D:\projects\TP\_MicroServices\billing-service\src\java\feign\CustomerRestClient.java.
- Code Content:**

```
1 package ma.ouassine.feign;
2
3 import ma.ouassine.model.Customer;
4 import org.springframework.cloud.openfeign.FeignClient;
5 import org.springframework.web.bind.annotation.GetMapping;
6 import org.springframework.web.bind.annotation.PathVariable;
7
8 @FeignClient(name="CUSTOMER-SERVICE")
9 public interface CustomerRestClient {
10     @GetMapping("/customers/{id}")
11     Customer getCustomerById(@PathVariable(name = "id") Long id);
12 }
13
14 }
```

## Interface CustomerRestClient

The screenshot shows the IntelliJ IDEA interface with the project structure on the left and the code editor on the right.

**Project Structure:**

- billing-service** (Project root)
  - .idea
  - .mvn
  - src
    - main
      - java
        - ma.ouassine
          - entities
            - Bill
            - ProductItem
          - feign
            - CustomerRestClient
            - ProductItemRestClient
        - model
          - Customer
          - Product
        - repositories
          - BillRepository
          - ProductRepository
        - web
          - BillingRestController
          - BillingServiceApplication
      - resources
        - static
        - templates
        - application.properties

## Interface ProductItemRestClient

## Package : Model

The screenshot shows a Java code editor in an IDE. The left pane displays the project structure under the 'Project' tab, showing a single module named 'billing-service' located at 'D:\projects\TP\_MicroServices\billing-service'. The 'src' directory contains 'main' and 'java' packages. The 'java' package contains 'ma.ouassine' and 'model' sub-packages. Under 'ma.ouassine', there are 'entities' and 'feign' sub-packages. 'entities' contains 'Bill' and 'ProductItem' classes. 'feign' contains 'CustomerRestClient' and 'ProductItemRestClient' interfaces. The 'model' package contains 'Customer' and 'Product' classes. The right pane shows the content of 'Customer.java'. The code defines a class 'Customer' with three private fields: 'id' (Long), 'name' (String), and 'email' (String). It includes imports for 'ma.ouassine.model' and 'lombok.Data', and is annotated with '@Data'.

```
package ma.ouassine.model;  
import lombok.Data;  
@Data  
public class Customer {  
    private Long id;  
    private String name;  
    private String email;  
}
```

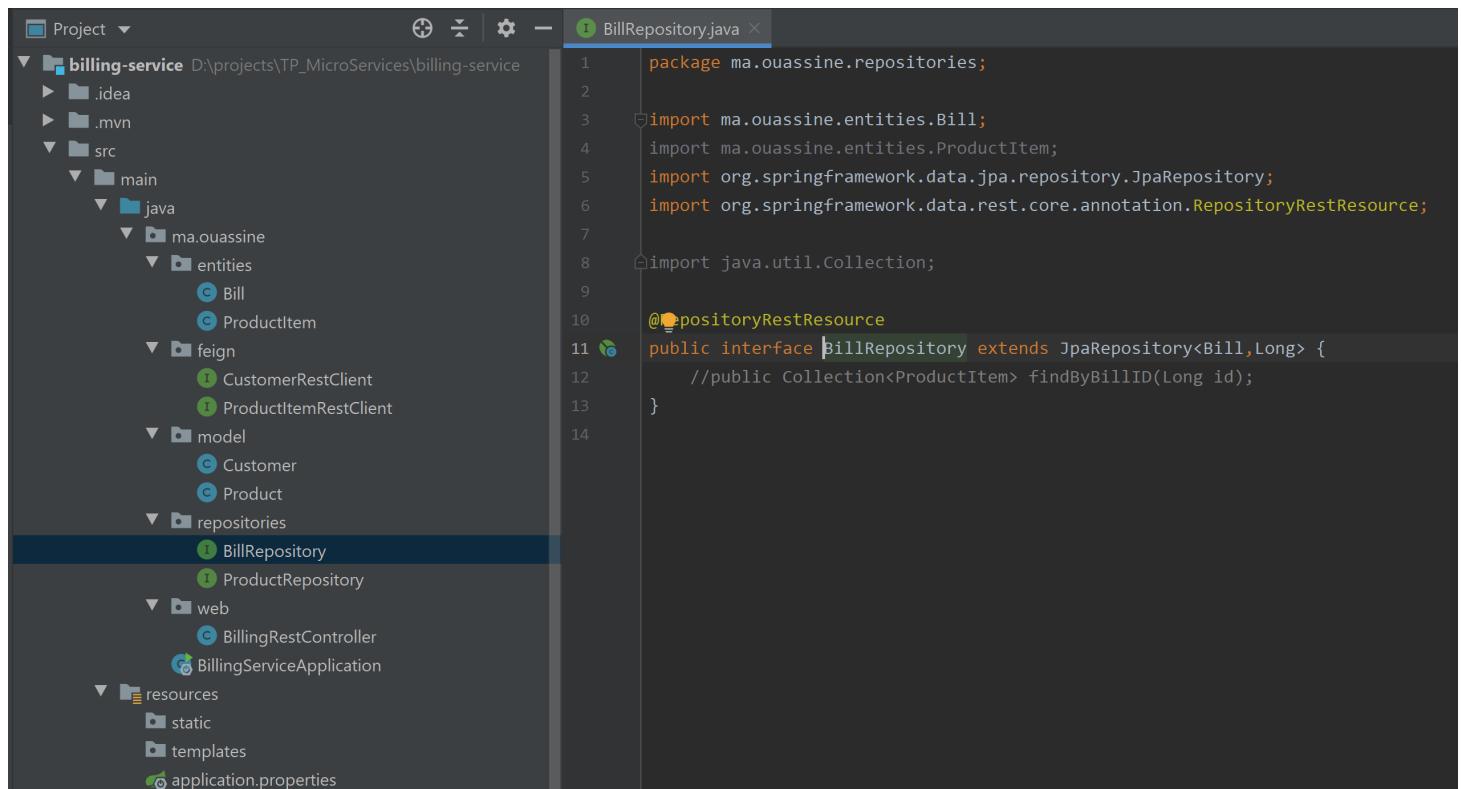
## Class Customer

The screenshot shows the IntelliJ IDEA interface. On the left, the Project tool window displays the project structure for 'billing-service' located at D:\projects\TP\_MicroServices\billing-service. The structure includes .idea, .mvn, and src folders. The src folder contains main, java, feign, and resources. The java folder has subfolders ma.ouassine.entities and ma.ouassine.feign. Inside ma.ouassine.entities are Bill and ProductItem classes. Inside ma.ouassine.feign are CustomerRestClient and ProductItemRestClient interfaces. On the right, the code editor is open to 'Product.java' with the following content:

```
1 package ma.ouassine.model;
2
3 import lombok.Data;
4
5 @Data
6 public class Product {
7     private Long id;
8     private String name;
9     private double price;
10    private double quantity;
11 }
12
```

## Class Product

## Package : Repositories



The screenshot shows a Java code editor in an IDE with the following details:

- Project Structure:** The project is named "billing-service" located at "D:\projects\TP\_MicroServices\billing-service". It contains the following directories:
  - .idea
  - .mvn
  - src
    - main
      - java
        - ma.ouassine
          - entities
            - Bill
            - ProductItem
          - feign
            - CustomerRestClient
            - ProductItemRestClient
          - model
            - Customer
            - Product
          - repositories
            - BillRepository
            - ProductRepository
        - web
          - BillingRestController
          - BillingServiceApplication
      - resources
        - static
        - templates
        - application.properties- Code Editor:** The file "BillRepository.java" is open. The code is as follows:

```
1 package ma.ouassine.repositories;
2
3 import ma.ouassine.entities.Bill;
4 import ma.ouassine.entities.ProductItem;
5 import org.springframework.data.jpa.repository.JpaRepository;
6 import org.springframework.data.rest.core.annotation.RepositoryRestResource;
7
8 import java.util.Collection;
9
10 @RepositoryRestResource
11 public interface BillRepository extends JpaRepository<Bill, Long> {
12     //public Collection<ProductItem> findByBillID(Long id);
13 }
```

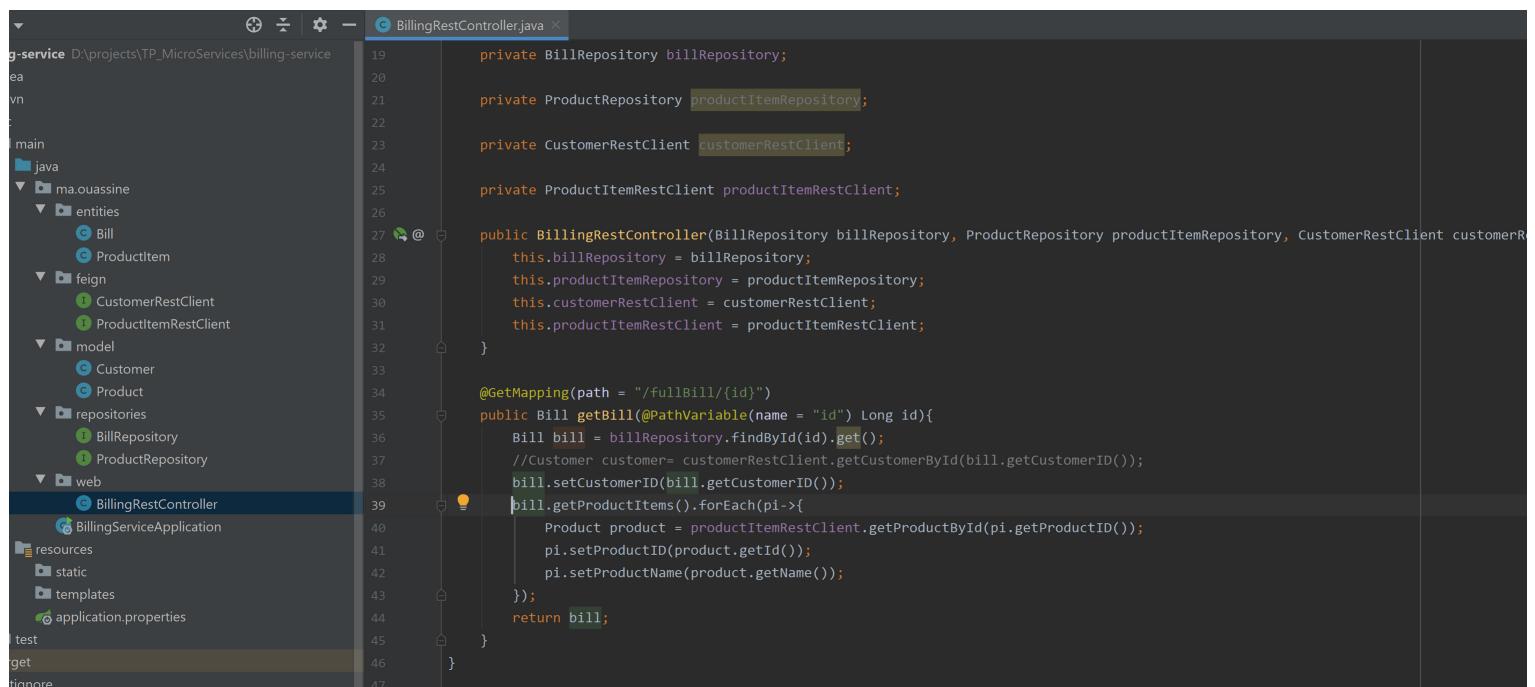
## Interface BillRepository

The screenshot shows the IntelliJ IDEA interface with the 'ProductRepository.java' file open in the editor. The left side displays the project structure under the 'billing-service' module. The 'repositories' package contains the 'ProductRepository' interface, which is highlighted in blue. The code implements the JpaRepository interface for the 'ProductItem' entity.

```
1 package ma.ouassine.repositories;
2
3 import ma.ouassine.entities.ProductItem;
4 import ma.ouassine.model.Product;
5 import org.springframework.data.jpa.repository.JpaRepository;
6 import org.springframework.data.rest.core.annotation.RepositoryRestResource;
7
8 import java.util.Collection;
9
10 @RepositoryRestResource
11 public interface ProductRepository extends JpaRepository<ProductItem, Long> {
12     public Collection<ProductItem> findByBillID(Long id);
13 }
14
```

## Interface ProductItemRepository

## Package : Web



The screenshot shows an IDE interface with the following details:

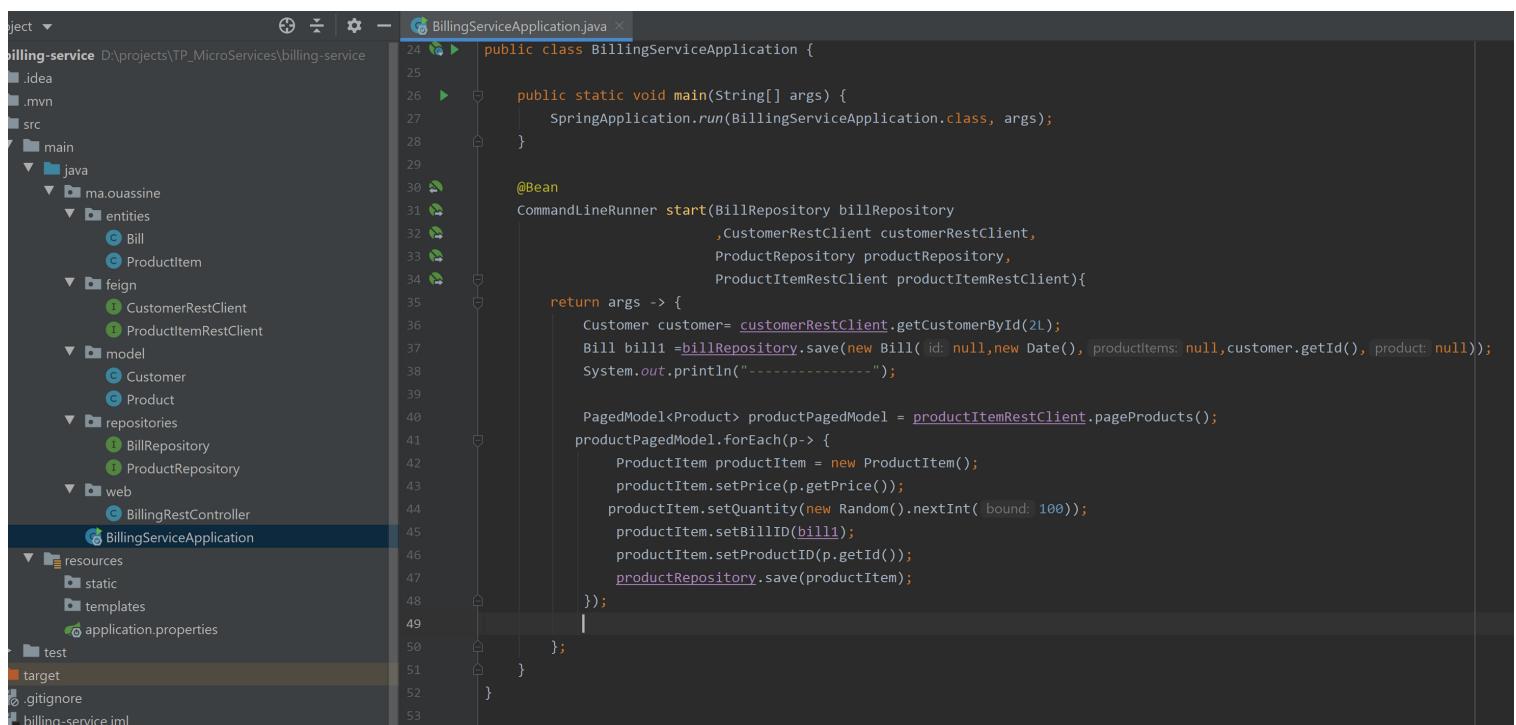
- Project Structure:** The project is named "g-service" located at "D:\projects\TP\_MicroServices\billing-service". It contains several packages:
  - main**: Contains "java" and "resources".
  - java**: Contains "ma.ouassine" (which has "entities", "feign", "model", "repositories", and "web" packages), "com", "io", and "org".
  - resources**: Contains "static" and "templates".
  - application.properties**
- Current File:** "BillingRestController.java" is open in the editor.
- Code Content:** The code defines a REST controller for bills. It includes imports for `BillRepository`, `ProductRepository`, `CustomerRestClient`, and `ProductItemRestClient`. The constructor takes these dependencies and initializes them. The `@GetMapping` annotation on line 34 maps to the URL `/fullBill/{id}`. The method retrieves a bill by ID, sets its customer ID, and then iterates over its product items to fetch products from the `ProductItemRestClient` and set their names. Finally, it returns the bill.

```
private BillRepository billRepository;
private ProductRepository productItemRepository;
private CustomerRestClient customerRestClient;
private ProductItemRestClient productItemRestClient;

public BillingRestController(BillRepository billRepository, ProductRepository productItemRepository, CustomerRestClient customerRestClient, ProductItemRestClient productItemRestClient) {
    this.billRepository = billRepository;
    this.productItemRepository = productItemRepository;
    this.customerRestClient = customerRestClient;
    this.productItemRestClient = productItemRestClient;
}

@GetMapping(path = "/fullBill/{id}")
public Bill getBill(@PathVariable(name = "id") Long id){
    Bill bill = billRepository.findById(id).get();
    //Customer customer=customerRestClient.getCustomerById(bill.getCustomerID());
    bill.setCustomerID(bill.getCustomerID());
    bill.getProductItems().forEach(pi->{
        Product product = productItemRestClient.getProductById(pi.getProductId());
        pi.setProductId(product.getId());
        pi.setProductName(product.getName());
    });
    return bill;
}
```

## Class Billing RestController



The screenshot shows an IDE interface with the following details:

- Project Tree:** On the left, the project structure is displayed under "Billing-service". It includes ".idea", ".mvn", "src", "main" (containing "java", "resources", "static", "templates", and "application.properties"), "test", "target", ".gitignore", and "billing-service.iml".
- Code Editor:** The main window shows the `BillingServiceApplication.java` file. The code is a Spring Boot application entry point. It imports `org.springframework.boot.SpringApplication`, `org.springframework.boot.autoconfigure.SpringBootApplication`, `org.springframework.context.annotation.Bean`, `org.springframework.context.annotation.Configuration`, `org.springframework.web.bind.annotation.GetMapping`, `org.springframework.web.bind.annotation.RestController`, and `java.util.List`.
- Content of BillingServiceApplication.java:**

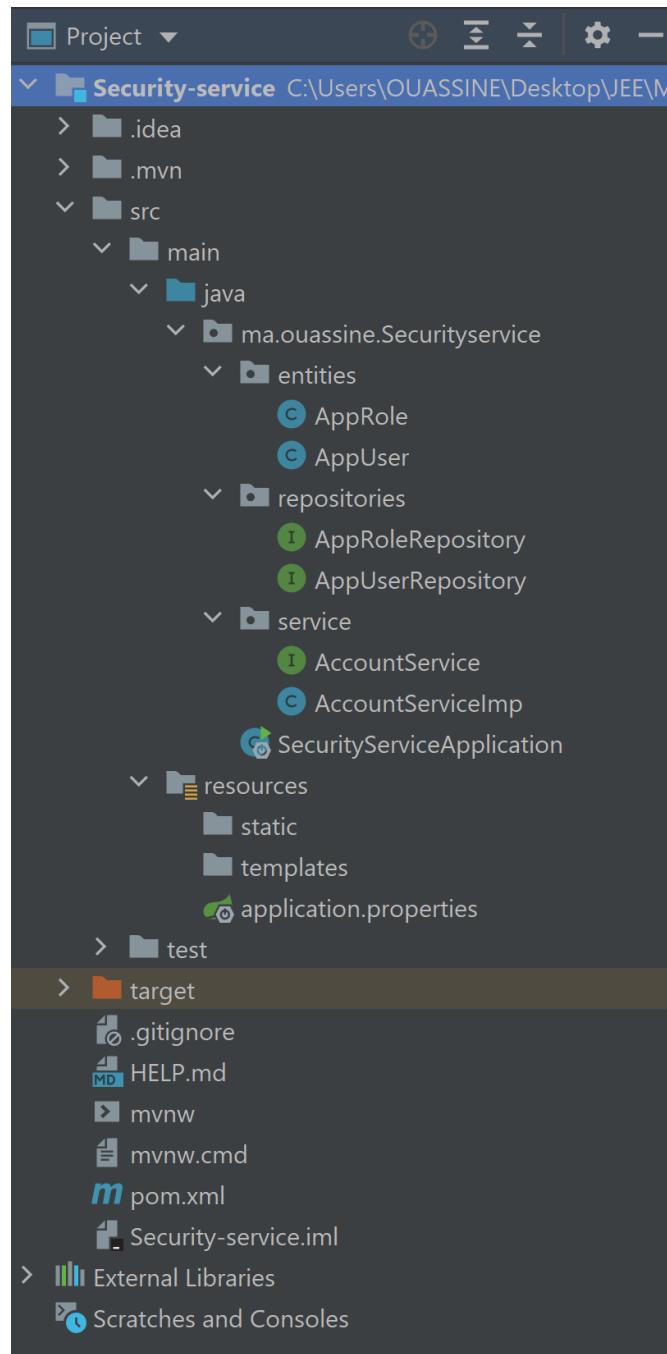
```
public class BillingServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(BillingServiceApplication.class, args);
    }

    @Bean
    CommandLineRunner start(BillRepository billRepository,
                           CustomerRestClient customerRestClient,
                           ProductRepository productRepository,
                           ProductItemRestClient productItemRestClient){
        return args -> {
            Customer customer= customerRestClient.getCustomerById(2L);
            Bill bill1 =billRepository.save(new Bill( id: null,new Date(), productItems: null, customer.getId(), product: null));
            System.out.println("-----");

            PagedModel<Product> productPagedModel = productItemRestClient.pageProducts();
            productPagedModel.forEach(p-> {
                ProductItem productItem = new ProductItem();
                productItem.setPrice(p.getPrice());
                productItem.setQuantity(new Random().nextInt( bound: 100));
                productItem.setBillID(bill1);
                productItem.setProductID(p.getId());
                productRepository.save(productItem);
            });
        };
    }
}
```

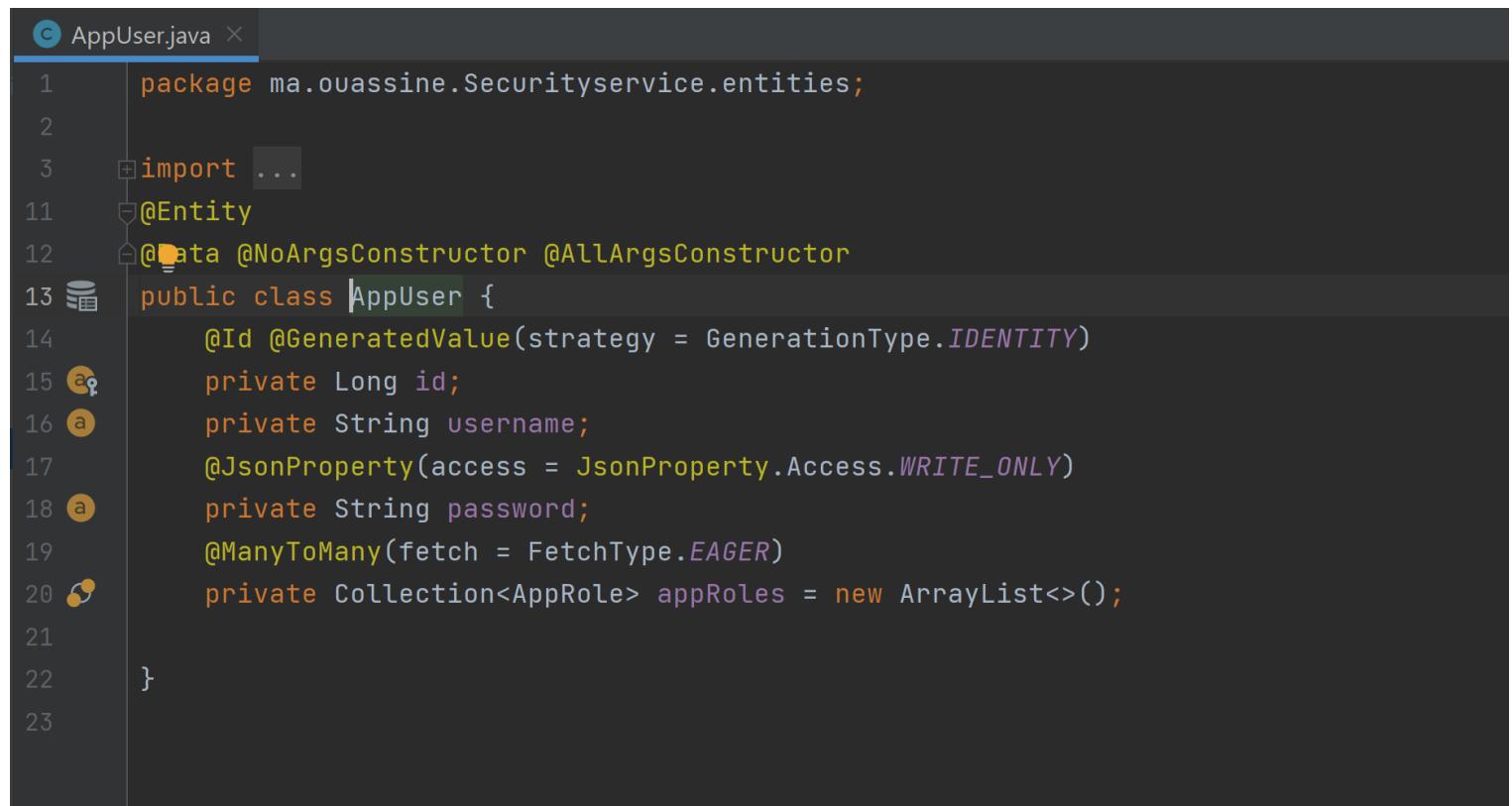
## Application Spring principale

# security Service



squelette du projet Billing service

## Package : entities



The screenshot shows a Java code editor with the file `AppUser.java` open. The code defines a class `AppUser` with annotations for persistence and security. The code is as follows:

```
1 package ma.ouassine.Securityservice.entities;
2
3 import ...
4
5 @Entity
6 @Data @NoArgsConstructor @AllArgsConstructor
7 public class AppUser {
8     @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
9     private Long id;
10    private String username;
11    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
12    private String password;
13    @ManyToMany(fetch = FetchType.EAGER)
14    private Collection<AppRole> appRoles = new ArrayList<>();
15
16 }
17
18 }
```

Class AppUser

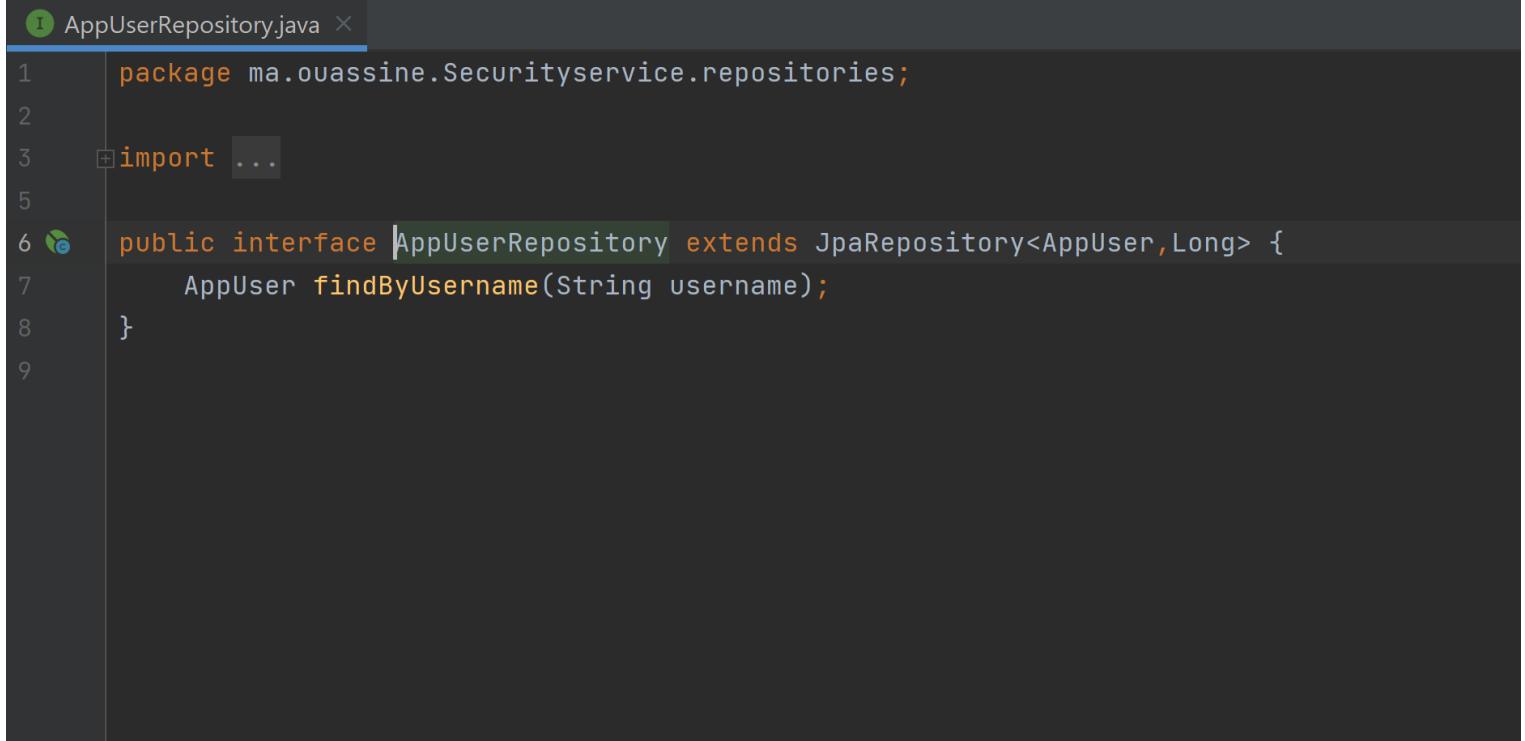
The screenshot shows a Java code editor window with the following details:

- Title Bar:** The title bar displays "AppRole.java" with a close button.
- Code Area:** The main area contains the following Java code:

```
1 package ma.ouassine.Securityservice.entities;
2
3 import ...
4
5
6
7
8
9
10
11
12 @Entity
13 @Data @AllArgsConstructor @NoArgsConstructor
14 public class AppRole {
15     @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
16     private Long id;
17     private String roleName;
18
19 }
```
- Toolbars and Status Bar:** There are several icons along the top edge, likely for file operations like save, open, and close, as well as other developer tools. The status bar at the bottom is mostly blank.

## Class AppRole

## Package : repositories



The screenshot shows a code editor window with a dark theme. The title bar says "AppUserRepository.java". The code is as follows:

```
1 package ma.ouassine.Securityservice.repositories;
2
3 import ...
4
5
6 public interface AppUserRepository extends JpaRepository<AppUser, Long> {
7     AppUser findByUsername(String username);
8 }
9
```

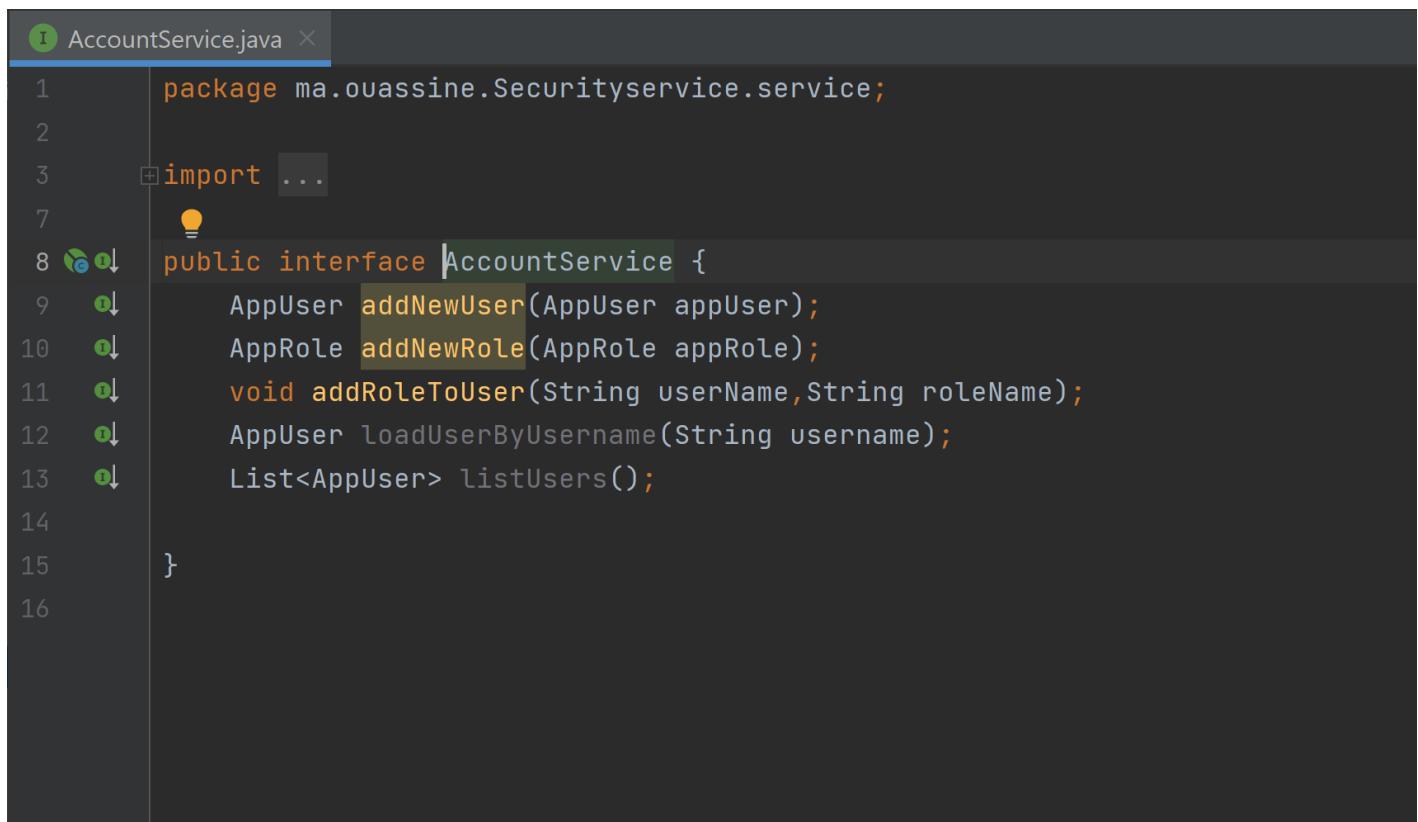
L'interface AppUserRepository

The screenshot shows a code editor window with a dark theme. The title bar says "AppRoleRepository.java". The code is as follows:

```
1 package ma.ouassine.Securityservice.repositories;
2
3 import ...
4
5
6 public interface AppRoleRepository extends JpaRepository<AppRole, Long> {
7     AppRole findByName(String roleName);
8 }
9
10
```

L'interface AppRoleRepository

## Package : service



A screenshot of a code editor showing the file `AccountService.java`. The code defines an interface for account management. The interface includes methods for adding new users, adding new roles, associating roles with users, loading users by username, and listing all users.

```
1 package ma.ouassine.Securityservice.service;
2
3 import ...
4
5 public interface AccountService {
6     AppUser addNewUser(AppUser appUser);
7     AppRole addNewRole(AppRole appRole);
8     void addRoleToUser(String userName, String roleName);
9     AppUser loadUserByUsername(String username);
10    List<AppUser> listUsers();
11}
```

L'interface Account Service

```
12  @Service
13  @Transactional
14  public class AccountServiceImp implements AccountService {
15      private AppRoleRepository appRoleRepository;
16      private AppUserRepository appUserRepository;
17
18      private PasswordEncoder passwordEncoder;
19      public AccountServiceImp(AppRoleRepository appRoleRepository, AppUserRepository appUserRepository, PasswordEncoder passwordEncoder) {
20          this.appRoleRepository = appRoleRepository;
21          this.appUserRepository = appUserRepository;
22          this.passwordEncoder = passwordEncoder;
23      }
24
25      @Override
26      public AppUser addNewUser(AppUser appUser) {
27          String pwd= appUser.getPassword();
28          appUser.setPassword(passwordEncoder.encode(pwd));
29          return appUserRepository.save(appUser);
30      }
31
32      @Override
33      public AppRole addNewRole(AppRole appRole) { return appRoleRepository.save(appRole); }
34
35      @Override
36      public void addRoleToUser(String userName, String roleName) {
37          AppUser appUser= appUserRepository.findByUsername(userName);
38          AppRole appRole = appRoleRepository.findByName(roleName);
39          appUser.getAppRoles().add(appRole);
40      }
41
42      @Override
43      public AppUser loadUserByUsername(String username) { return appUserRepository.findByUsername(username); }
44
45
46
47
48
```

## Class AccountserviceImplementation

```
SecurityServiceApplication.java ×
15 @SpringBootApplication
16 public class SecurityServiceApplication {
17
18     public static void main(String[] args) { SpringApplication.run(SecurityServiceApplication.class, args); }
19
20     PasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder(); }
21
22     @Bean
23     CommandLineRunner start(AccountService accountService){
24         return args ->{
25             accountService.addNewRole(new AppRole( id: null, roleName: "USER"));
26             accountService.addNewRole(new AppRole( id: null, roleName: "ADMIN"));
27             accountService.addNewRole(new AppRole( id: null, roleName: "CUSTOMER-MANAGER"));
28             accountService.addNewRole(new AppRole( id: null, roleName: "PRODUCT-MANAGER"));
29             accountService.addNewRole(new AppRole( id: null, roleName: "BILLS-MANAGER"));
30
31             accountService.addNewUser(new AppUser( id: null, username: "user1", password: "1234",new ArrayList<>()));
32             accountService.addNewUser(new AppUser( id: null, username: "user2", password: "abcd",new ArrayList<>()));
33             accountService.addNewUser(new AppUser( id: null, username: "user3", password: "azerty",new ArrayList<>()));
34
35             accountService.addRoleToUser( userName: "user1", roleName: "USER");
36             accountService.addRoleToUser( userName: "user2", roleName: "ADMIN");
37             accountService.addRoleToUser( userName: "user3", roleName: "PRODUCT-MANAGER");
38
39         };
40     }
41
42     }
43
44     }
45 }
46
47 }
```

L'application spring principale

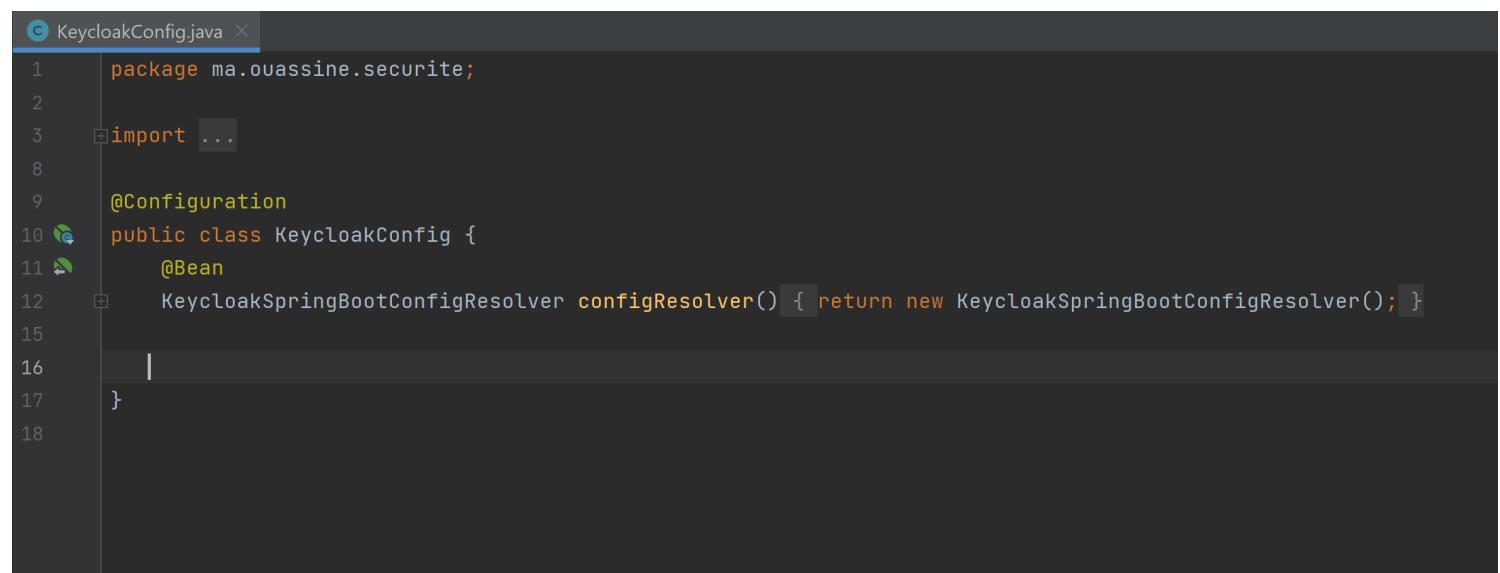
### application.properties ×

```
1 spring.datasource.url=jdbc:h2:mem:users-db  
2 spring.application.name=security-service  
3 server.port=8090  
4 |
```

appliation.properties file

Pour la sécurisation de notre application on va ajouter du code keycloak au micro services réalisés  
ça va se faire comme suit :

# **Sécurisation d'Inventory Service**



The screenshot shows a code editor window with a dark theme. The file is named 'KeycloakConfig.java'. The code is as follows:

```
1 package ma.ouassine.securite;
2
3 import ...
4
5 @Configuration
6 public class KeycloakConfig {
7     @Bean
8     KeycloakSpringBootConfigResolver configResolver() { return new KeycloakSpringBootConfigResolver(); }
9 }
10
11
12
13
14
15
16
17
18
```

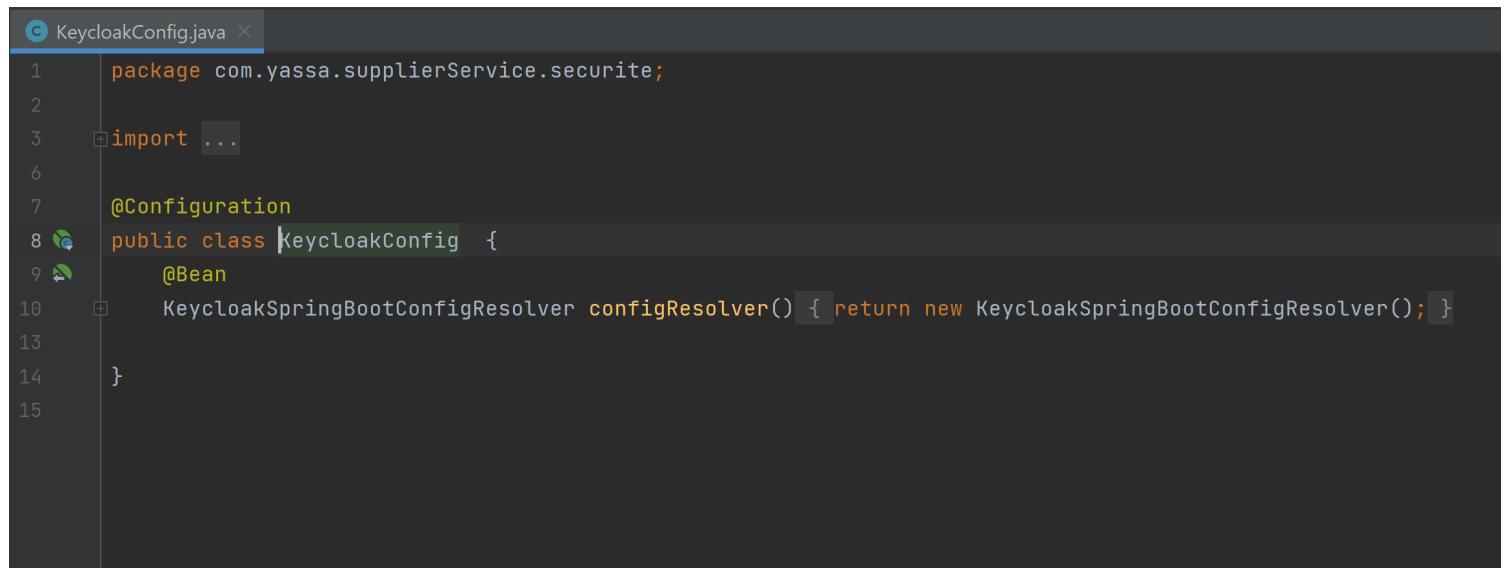
`@Configuration`

The screenshot shows a Java code editor with two tabs: 'KeycloakSpringSecuriteConfig.java' and 'KeycloakConfig.java'. The 'KeycloakSpringSecuriteConfig.java' tab is active, displaying the following code:

```
1 package ma.ouassine.securite;
2
3 import ...
4
5 @KeycloakConfiguration
6 public class KeycloakSpringSecuriteConfig extends KeycloakWebSecurityConfigurerAdapter {
7
8     @Override
9     protected SessionAuthenticationStrategy sessionAuthenticationStrategy() {
10         return new RegisterSessionAuthenticationStrategy(new SessionRegistryImpl());
11     }
12
13     @Override
14     protected void configure(AuthenticationManagerBuilder auth) throws Exception {
15         auth.authenticationProvider(keycloakAuthenticationProvider());
16     }
17
18     @Override
19     protected void configure(HttpSecurity http) throws Exception {
20         super.configure(http);
21         http.authorizeRequests().antMatchers("...antPatterns: /produits/**").authenticated();
22     }
23
24 }
25
26 }
```

@KeycloakConfiguration

# **Sécurisation du Customer Service**



A screenshot of a Java code editor showing a file named KeycloakConfig.java. The code defines a configuration class for Keycloak integration.

```
1 package com.yassa.supplierService.securite;
2
3 import ...
4
5
6
7 @Configuration
8 public class KeycloakConfig {
9     @Bean
10    KeycloakSpringBootConfigResolver configResolver() { return new KeycloakSpringBootConfigResolver(); }
11
12 }
13
14
15
```

`@Configuration`

```
C KeycloakSpringSecuriteConfig.java ×
1 package com.yassa.supplierService.securite;
2
3 import ...
10
11 @KeycloakConfiguration
12 public class KeycloakSpringSecuriteConfig extends KeycloakWebSecurityConfigurerAdapter {
13
14     @Override
15     protected SessionAuthenticationStrategy sessionAuthenticationStrategy() {
16         return new RegisterSessionAuthenticationStrategy(new SessionRegistryImpl());
17     }
18
19     @Override
20     protected void configure(AuthenticationManagerBuilder auth) throws Exception {
21         auth.authenticationProvider(keycloakAuthenticationProvider());
22     }
23
24     @Override
25     protected void configure(HttpSecurity http) throws Exception {
26         super.configure(http);
27         http.authorizeRequests().antMatchers("/suppliers/**").hasAuthority("app-manager");
28     }
29 }
```

@KeycloakConfiguration

# **Sécurisation du Billing Service**



A screenshot of a Java code editor showing a file named `KeycloakConfig.java`. The code is annotated with line numbers from 1 to 15. It contains a package declaration, imports, and a configuration class `KeycloakConfig` with a single bean method `configResolver`.

```
1 package org.sid.billingservice.securite;
2
3 import ...
4
5
6
7 @Configuration
8 public class KeycloakConfig {
9     @Bean
10    KeycloakSpringBootConfigResolver configResolver() { return new KeycloakSpringBootConfigResolver(); }
11
12
13
14 }
15
```

`@Configuration`

```
KeycloakSpringSecuriteConfig.java ×
1 package org.sid.billingservice.securite;
2
3 import ...
10
11 @KeycloakConfiguration
12 public class KeycloakSpringSecuriteConfig extends KeycloakWebSecurityConfigurerAdapter {
13
14     @Override
15     protected SessionAuthenticationStrategy sessionAuthenticationStrategy() {
16         return new RegisterSessionAuthenticationStrategy(new SessionRegistryImpl());
17     }
18
19     @Override
20     @Override
21     protected void configure(AuthenticationManagerBuilder auth) throws Exception {
22         auth.authenticationProvider(keycloakAuthenticationProvider());
23     }
24
25     @Override
26     protected void configure(HttpSecurity http) throws Exception {
27         super.configure(http);
28         http.authorizeRequests().antMatchers("/bills/**").hasAuthority("app-manager");
29     }
30 }
```

@KeycloakConfiguration