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 :

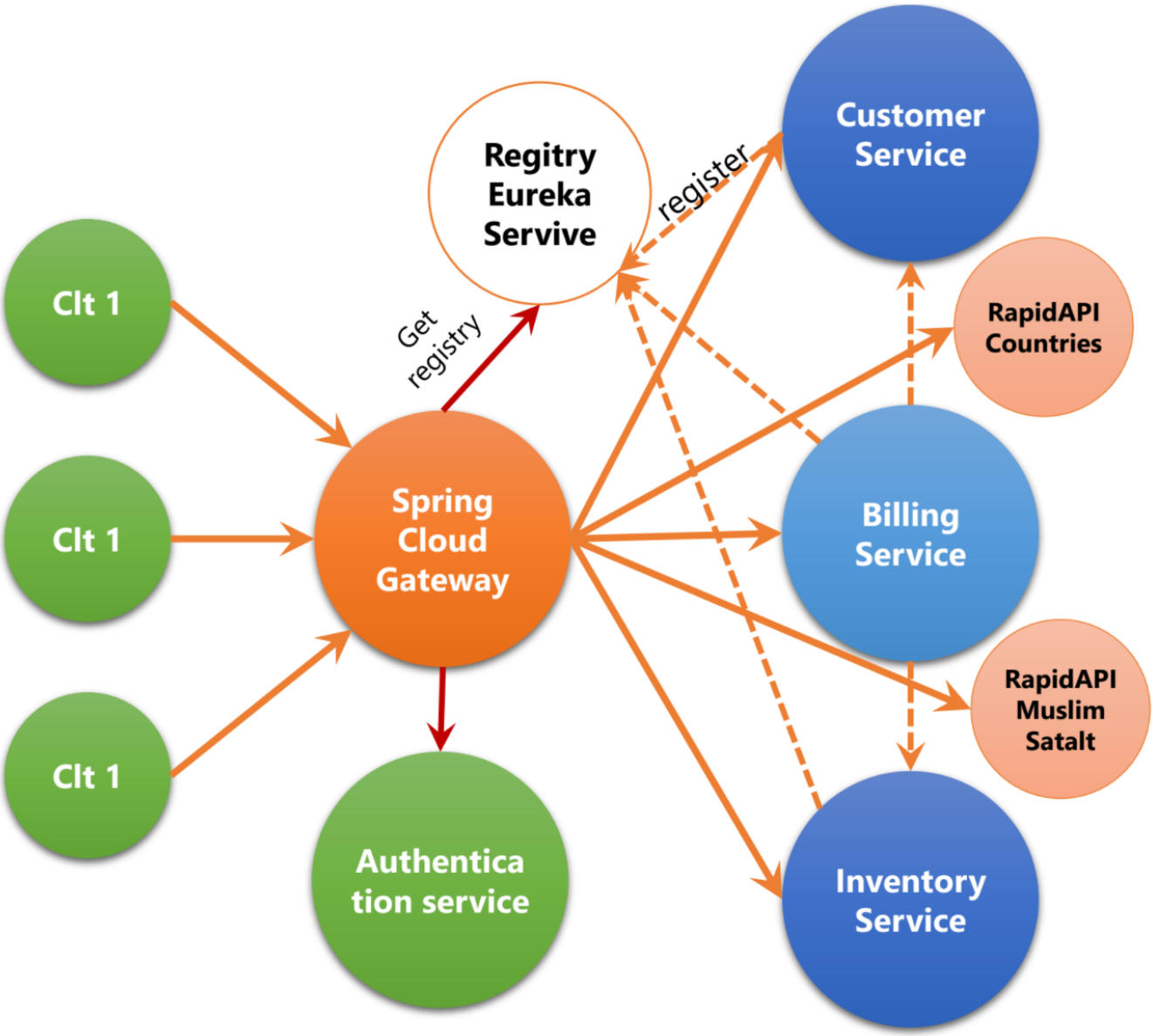
- Customer-Service
- Inventory-Service
- Billing-Service
- Gateway-Service
- Eureka Discovery-Service
- Athentification-Service

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

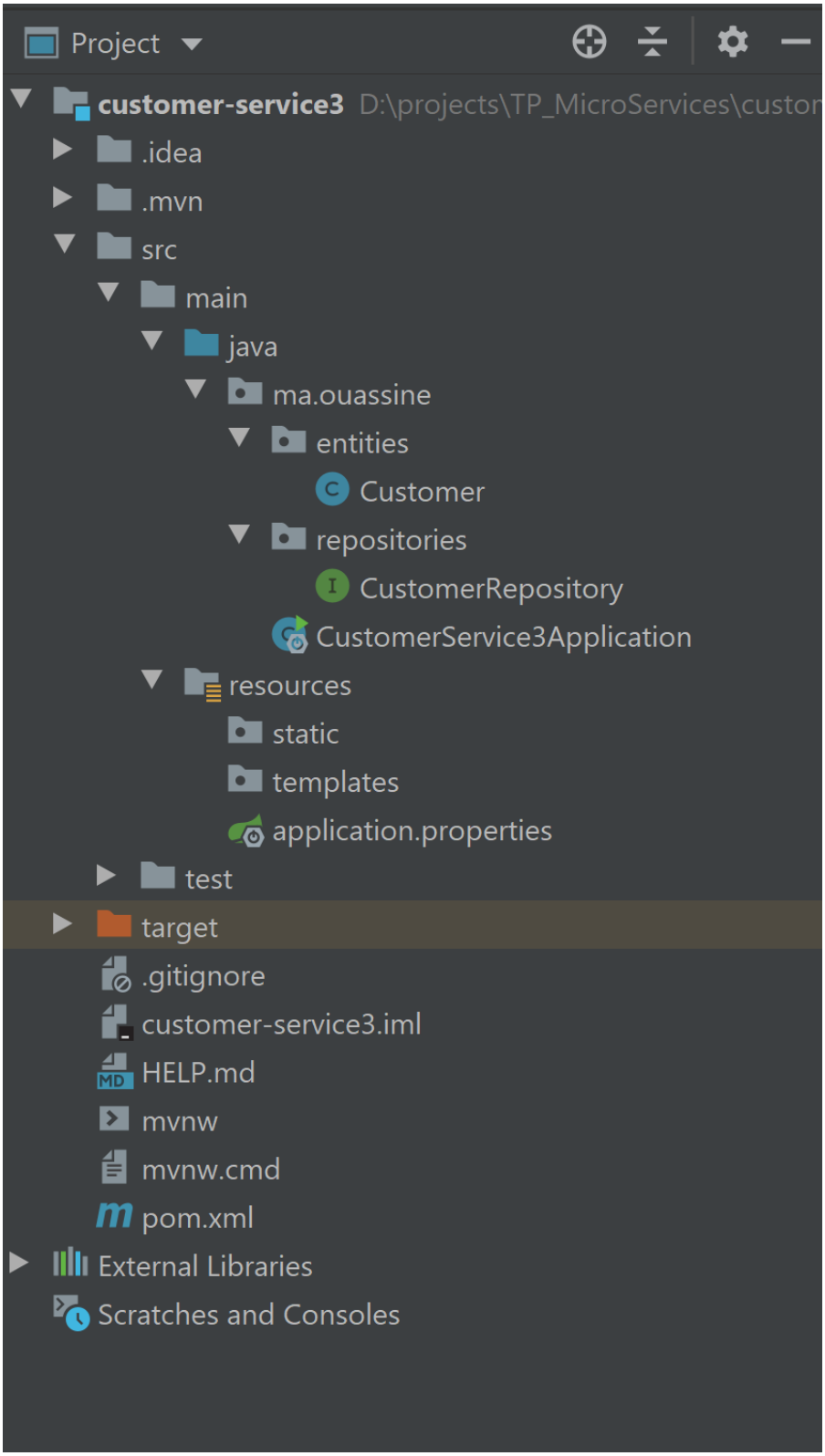
L'application consiste à créer six micro-services, le premier est pour la gestion des clients (Customer Service), le deuxième est pour la gestion de l'inventaire (Inventory Service), le troisième service est celui nommé Gateway service, et qui permet l'accès au 2 premiers services, cet accès vas se faire 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

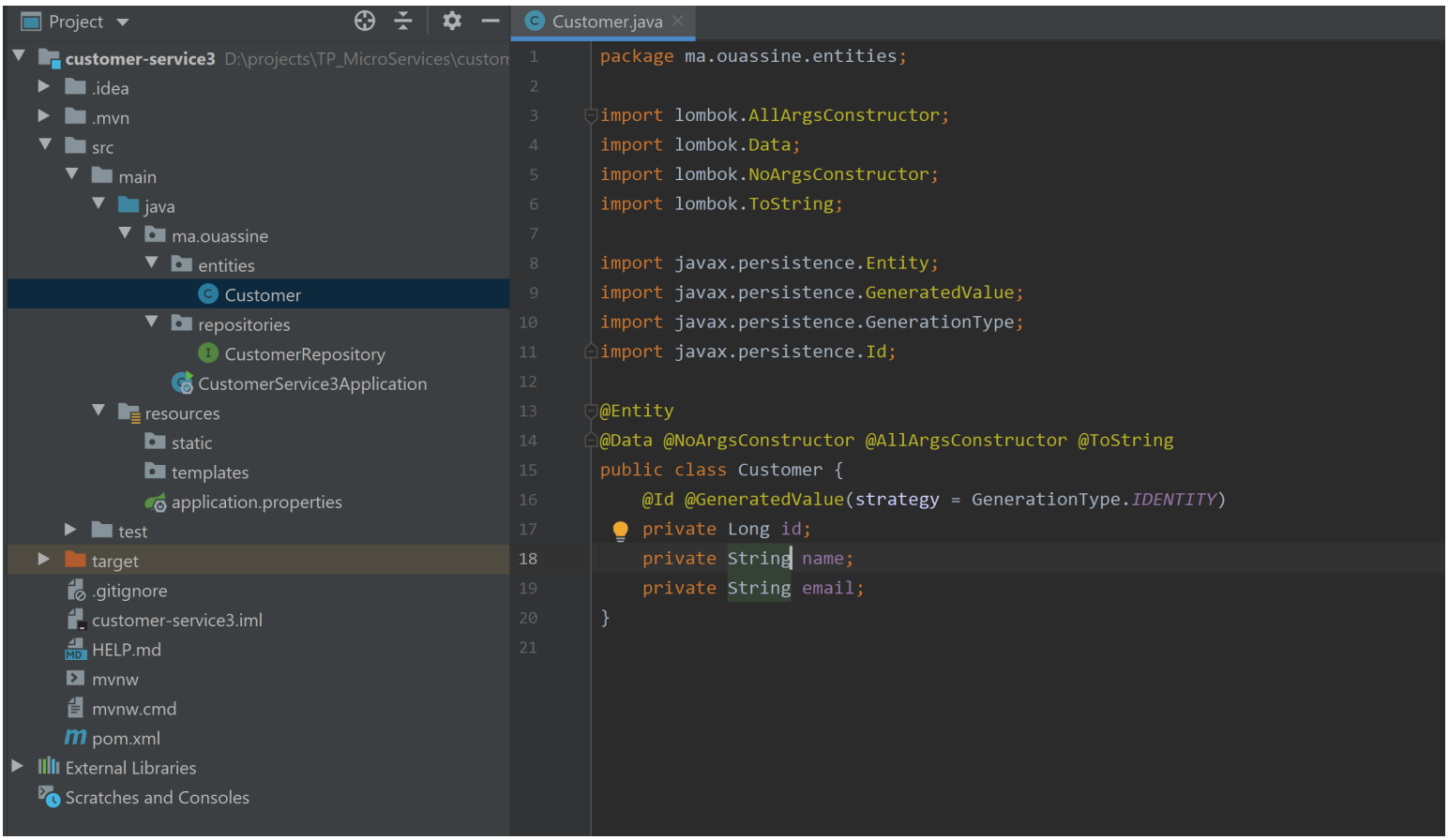
Customer service est le service responsable de la gestion des clients, c'est une application Srping où on aura besoin des dépendances suivantes :

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



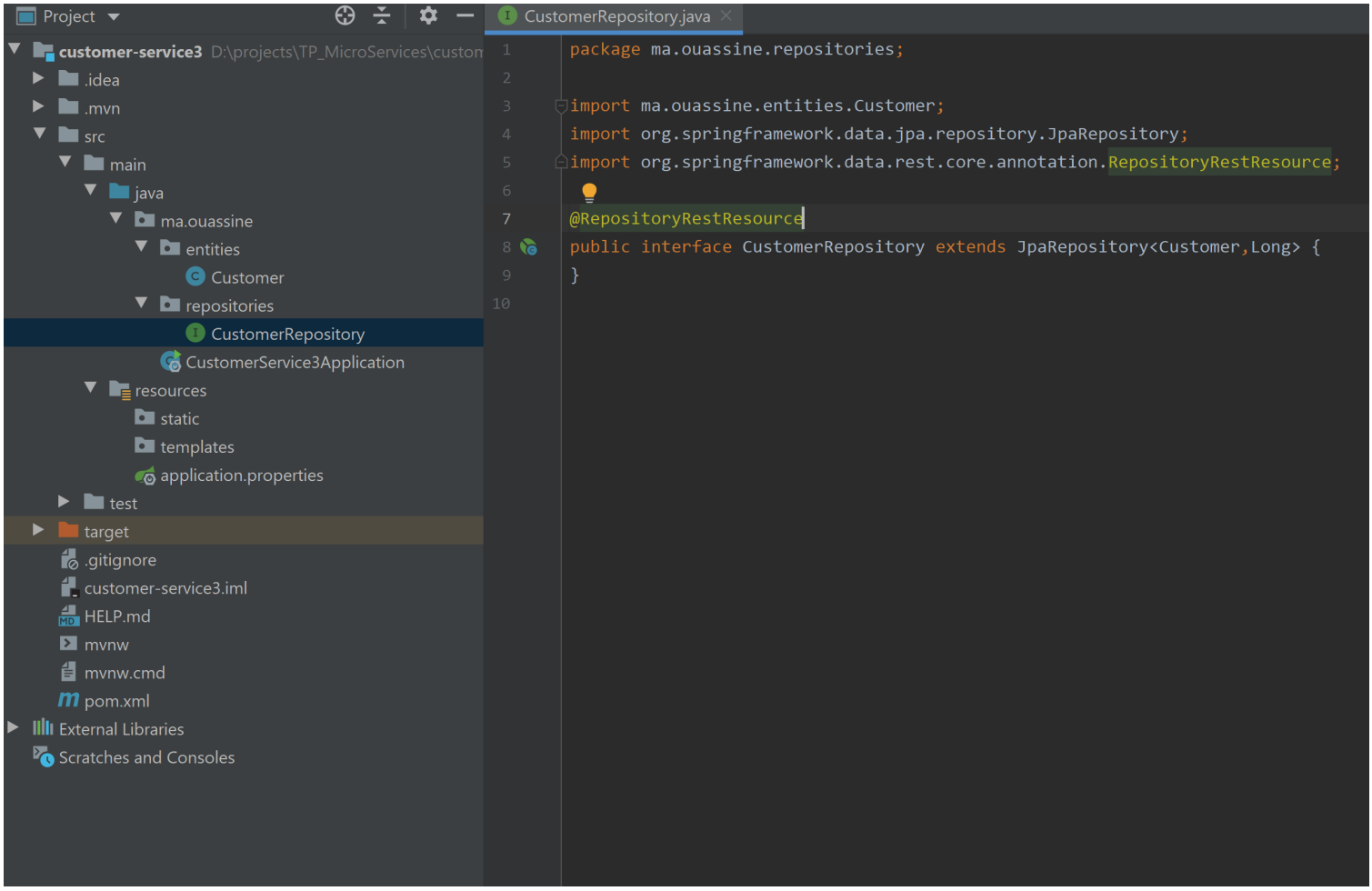
squelette du projet Customer service

Package : Entities

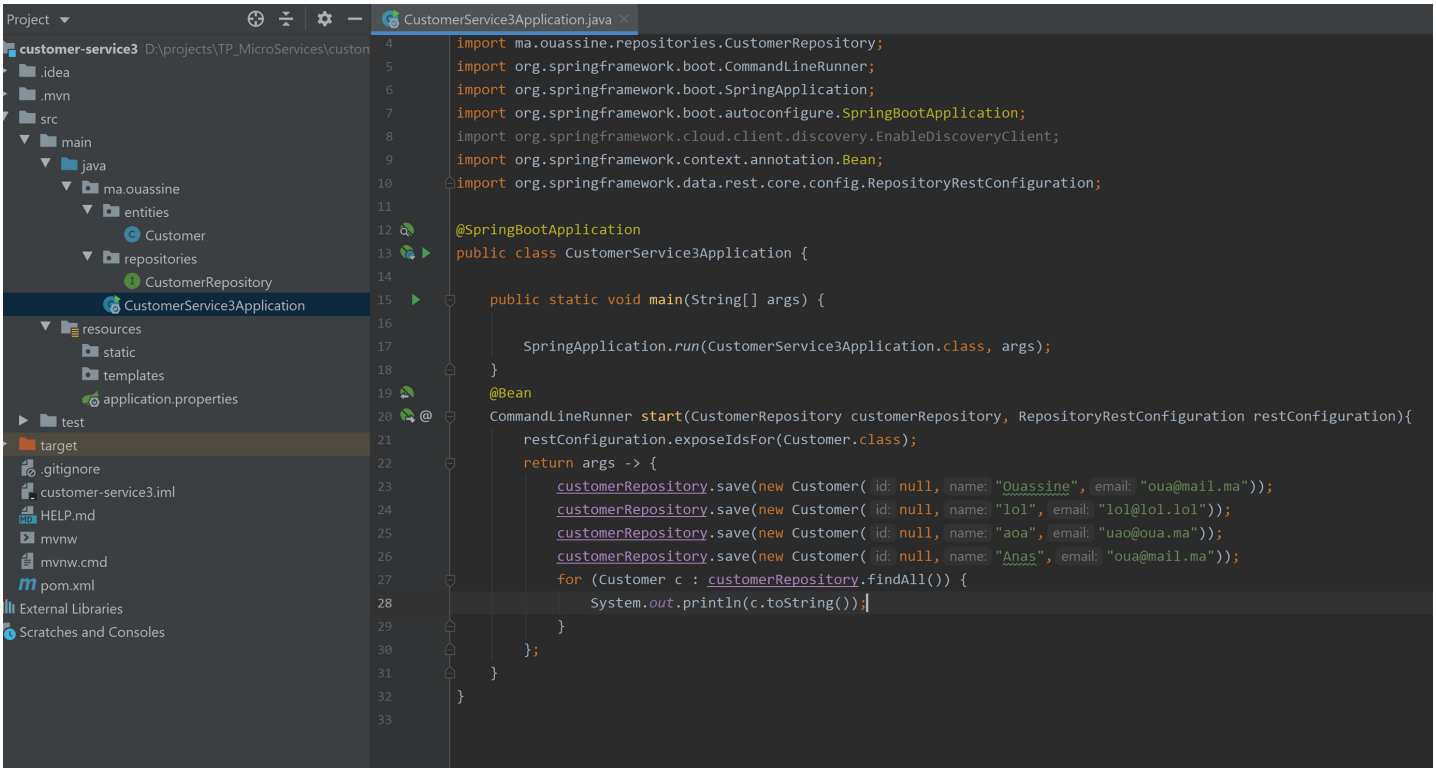


Class Customer

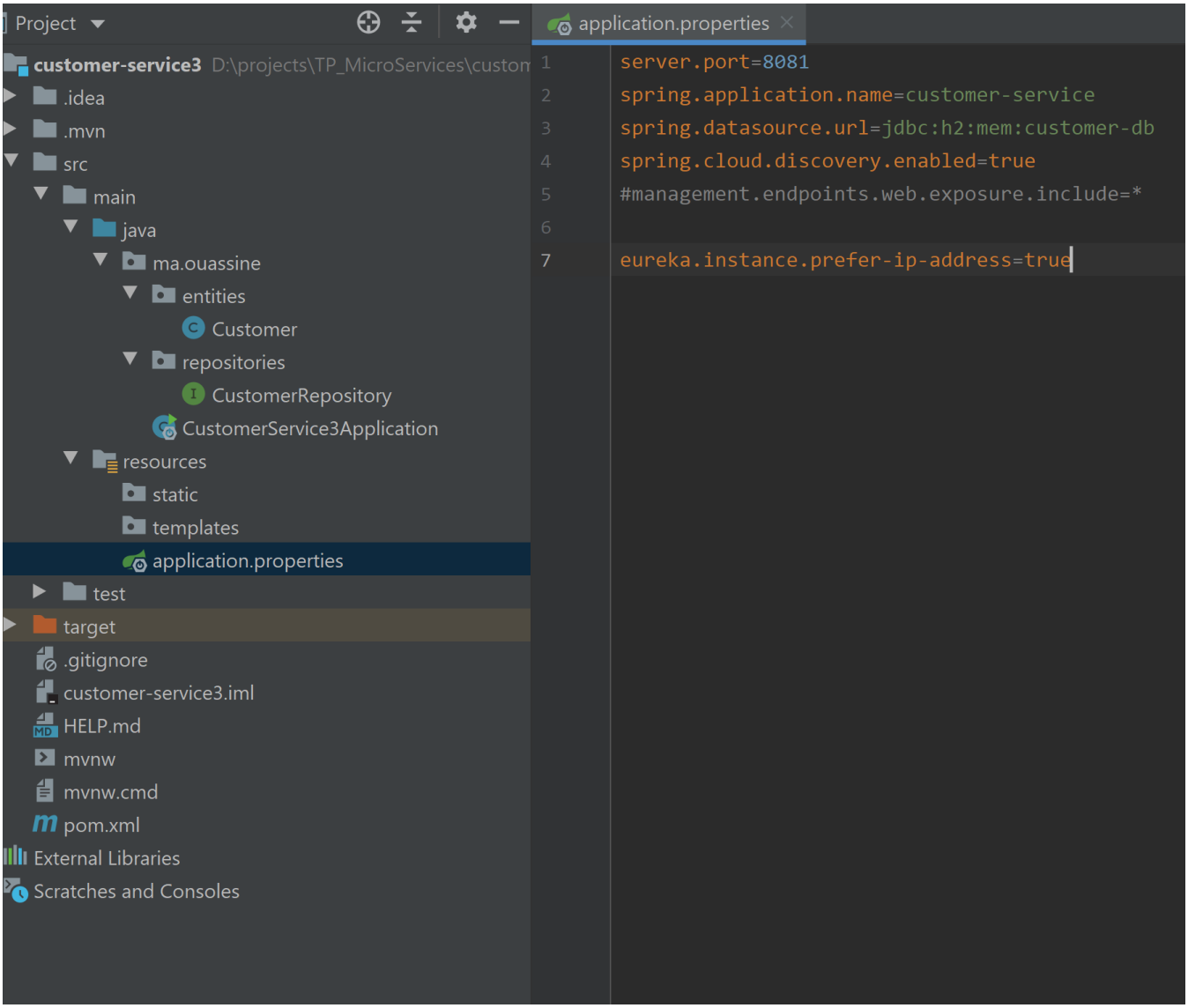
Package : Repositories



Customer Repository



L'application Spring principale

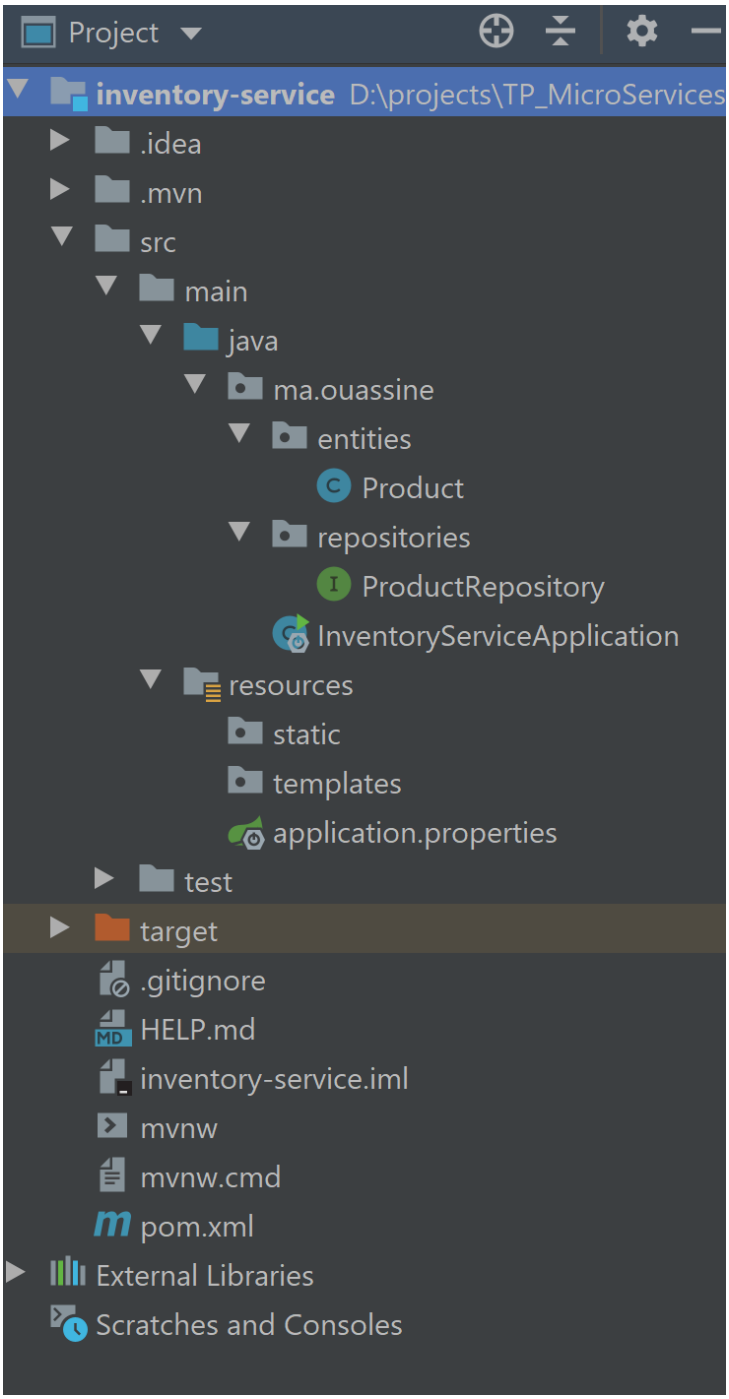


Application.properties

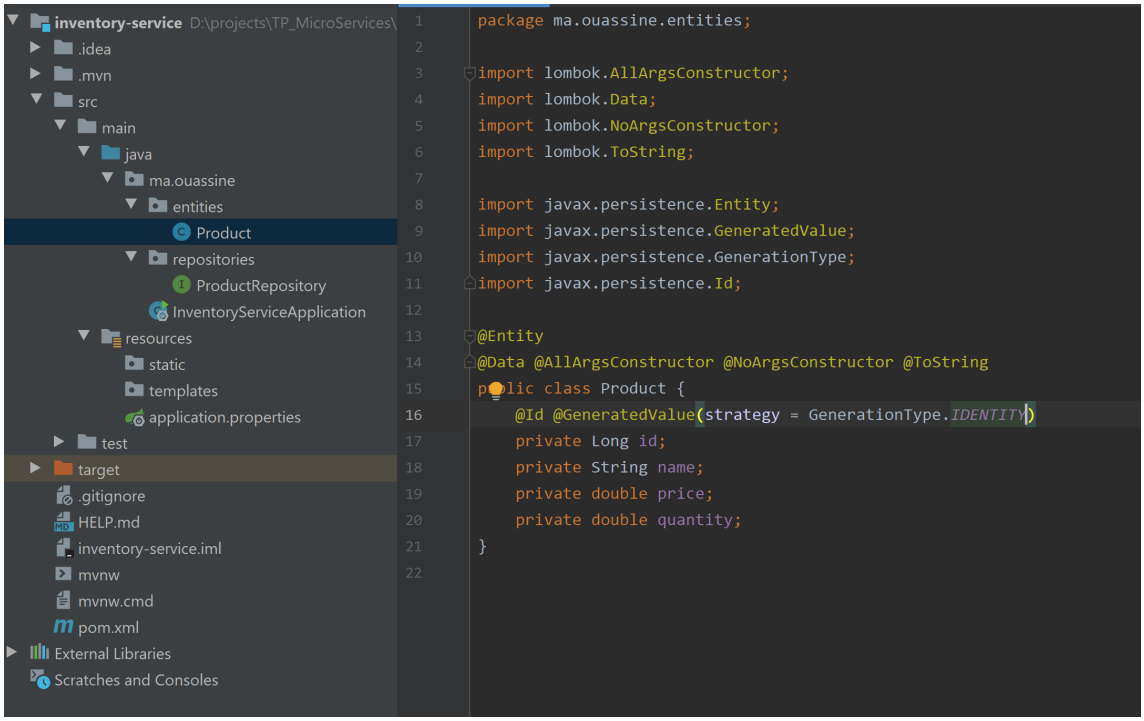
Inventory Service

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

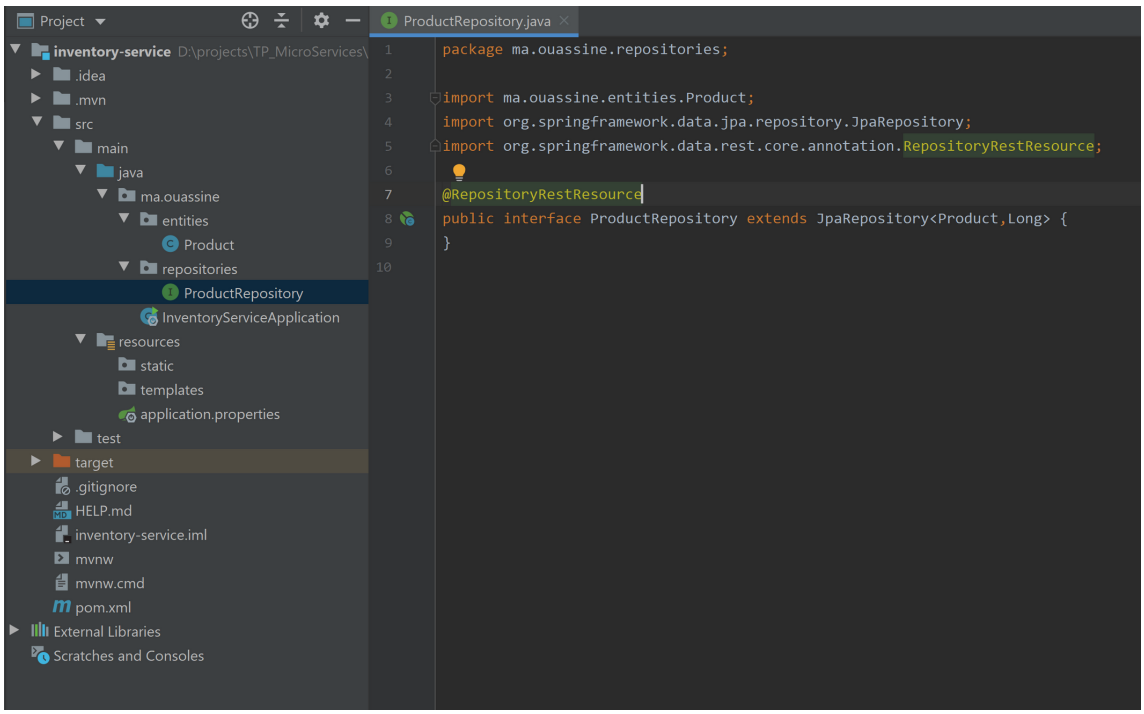
- Spring Web
- Spring Data JPA
- H2 Database
- Rest Repositories
- Lombok
- Spring Boot Dev Tools
- Eureka Discovery Client
- Spring Boot Actuator



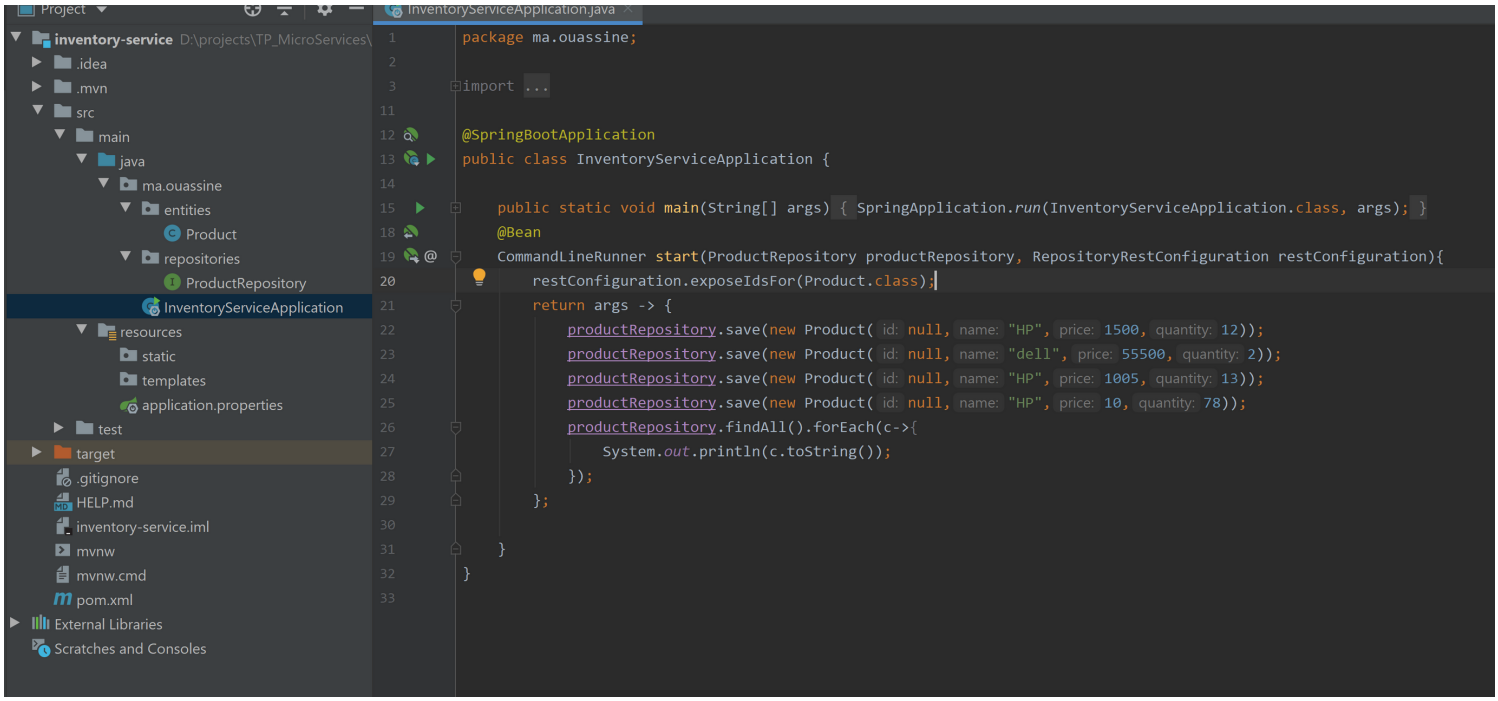
squelette du projet Inventory service



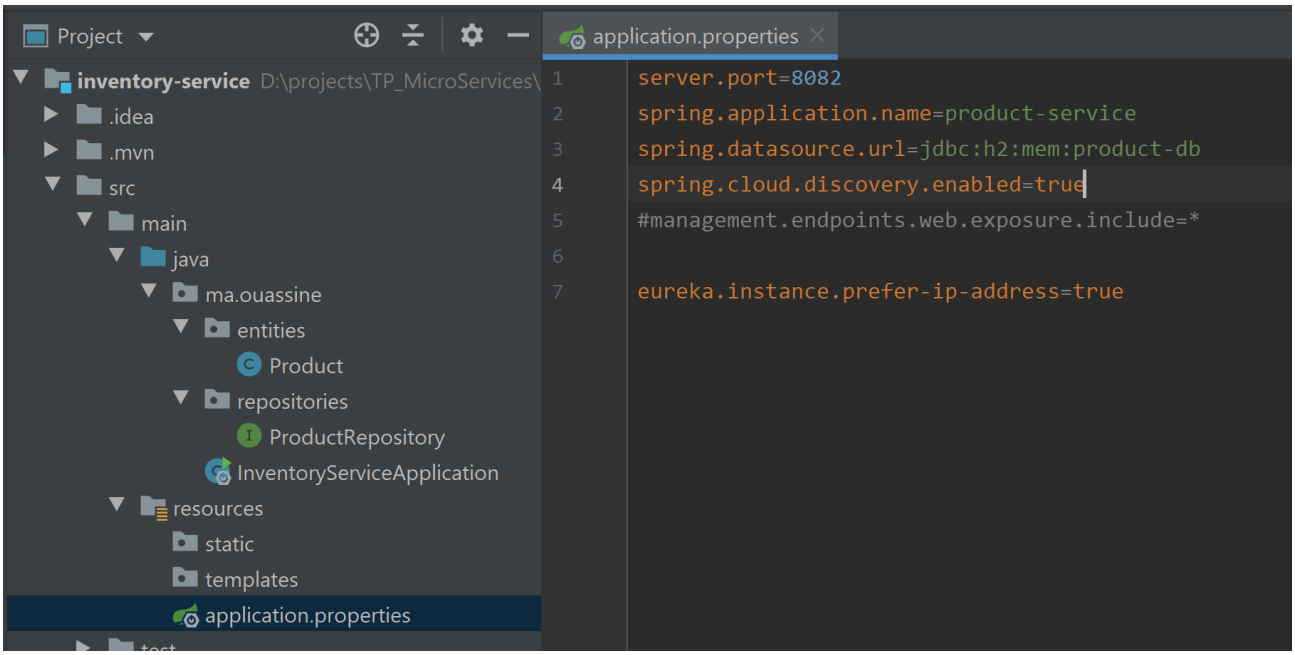
Class Inventory



Inventory Repository



L'application Spring principale

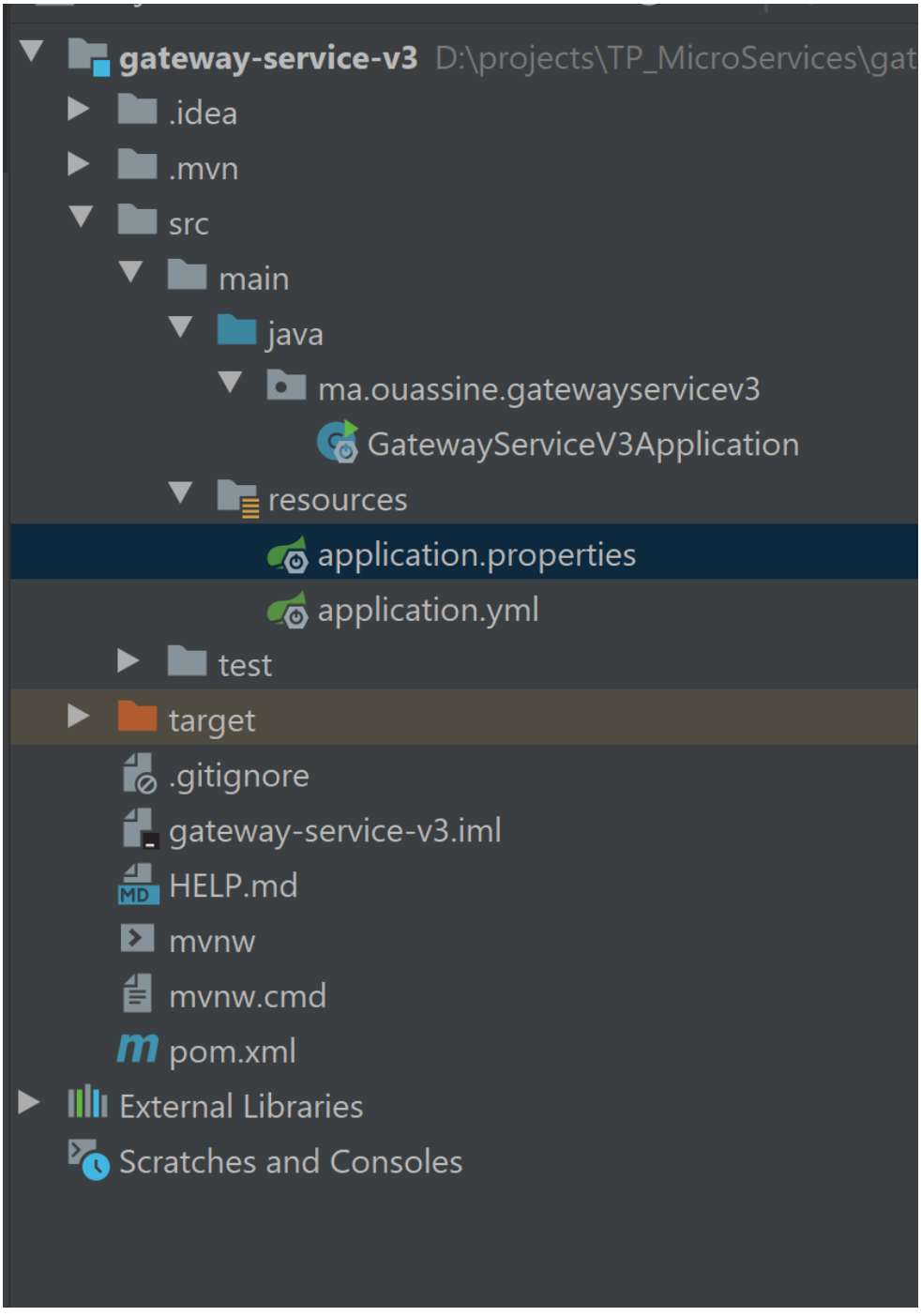


Application.Properties

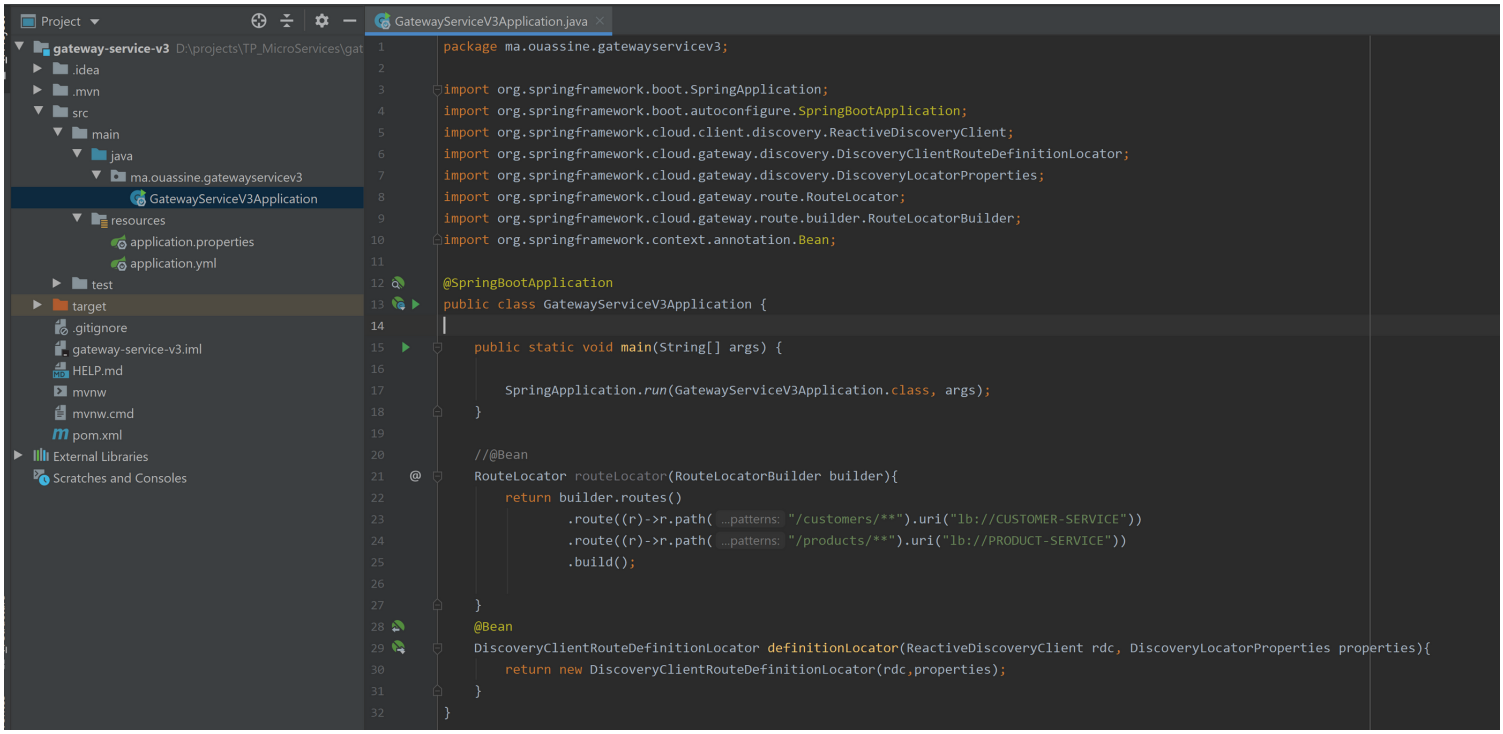
Gateway Service

Les dépendances utilisées sont :

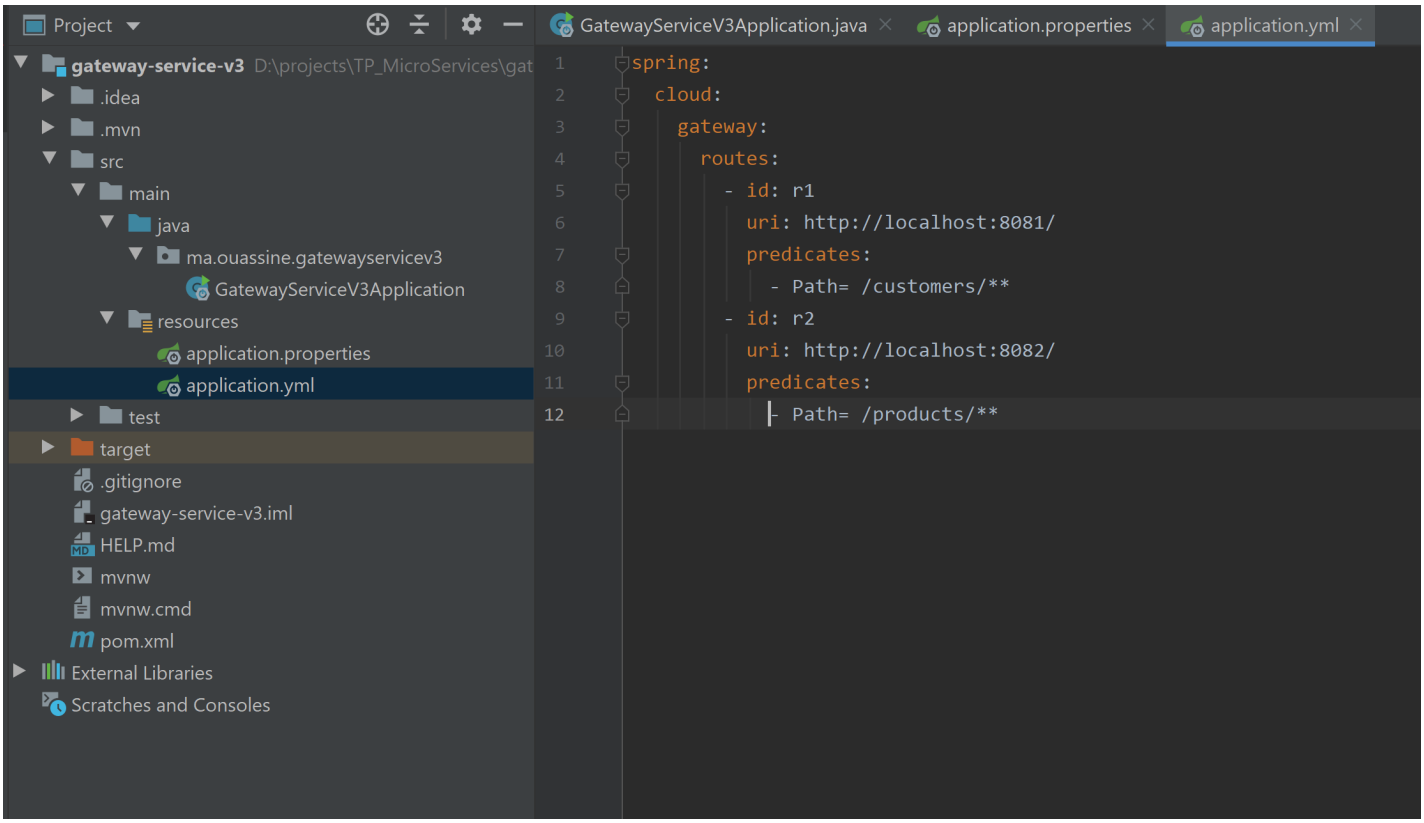
- Gateway
- Spring Boot Actuator
- Hystrix
- Eureka Discovery Client



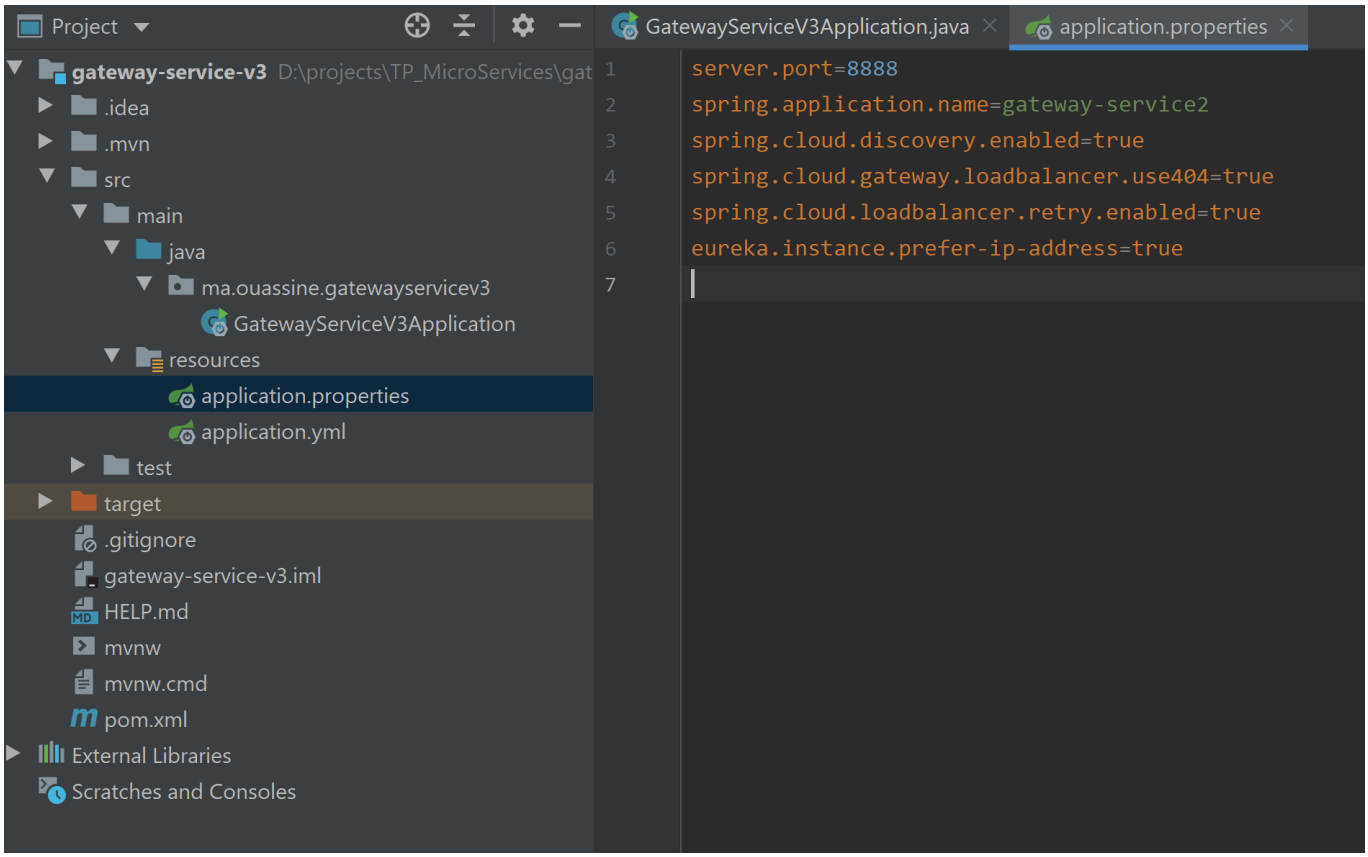
squelette du projet Gateway service



L'application Spring principale



Application.yml

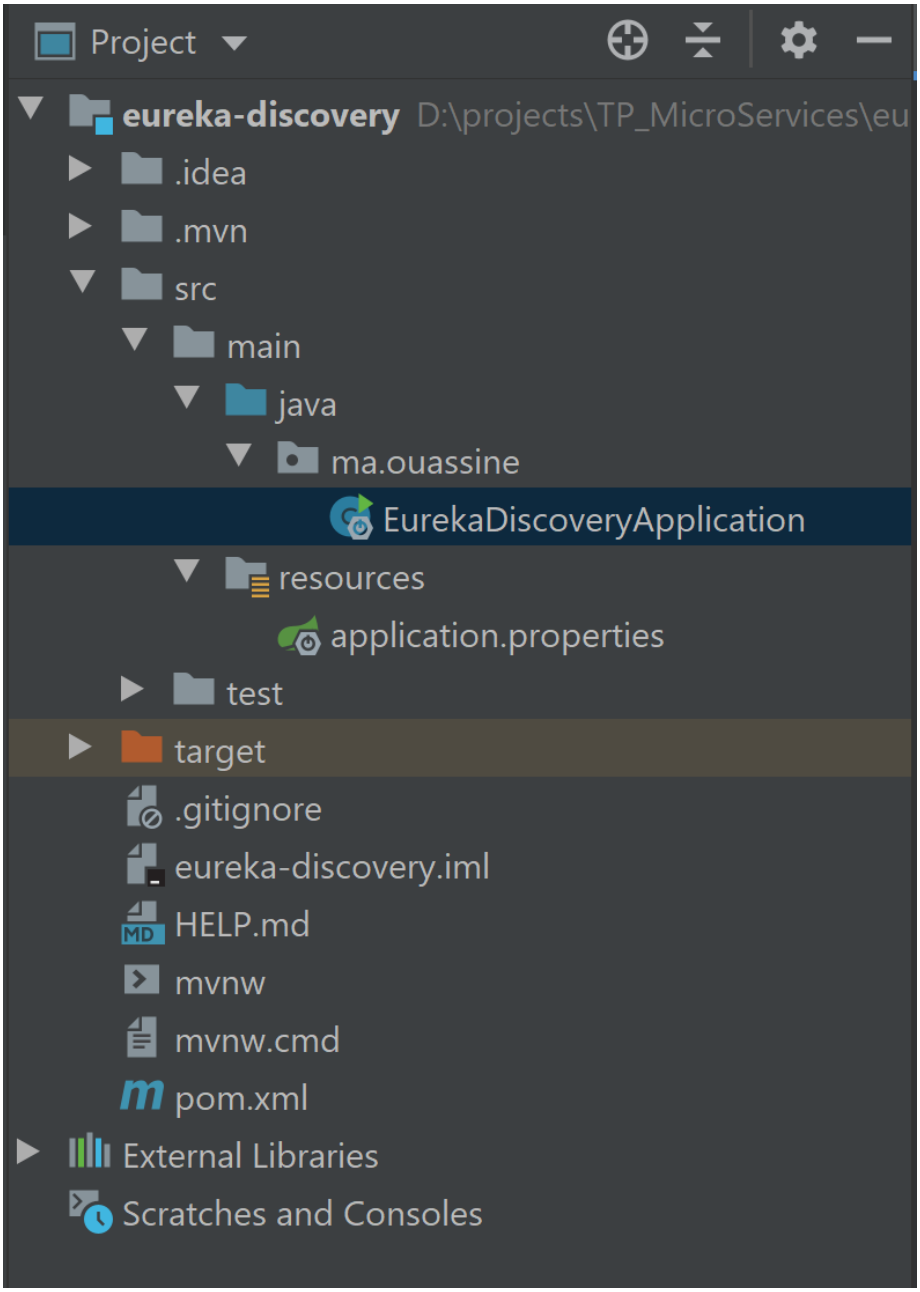


Application.properties

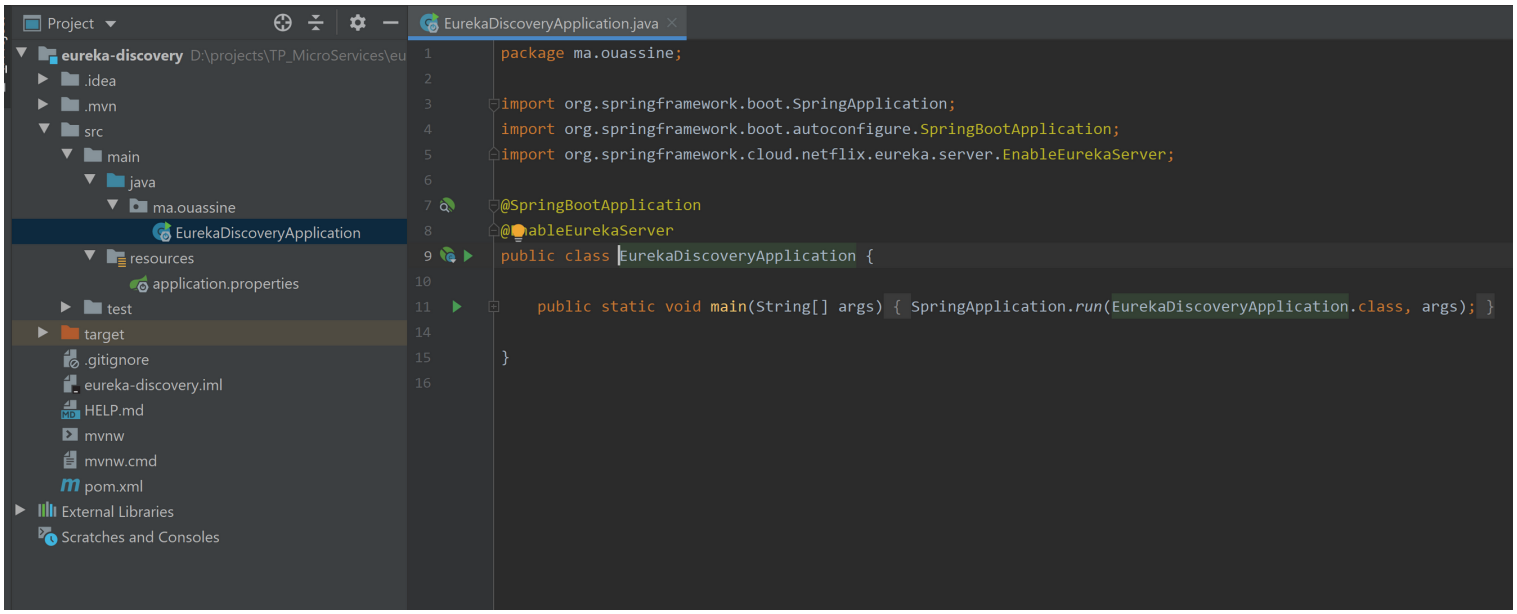
Eureka Service

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

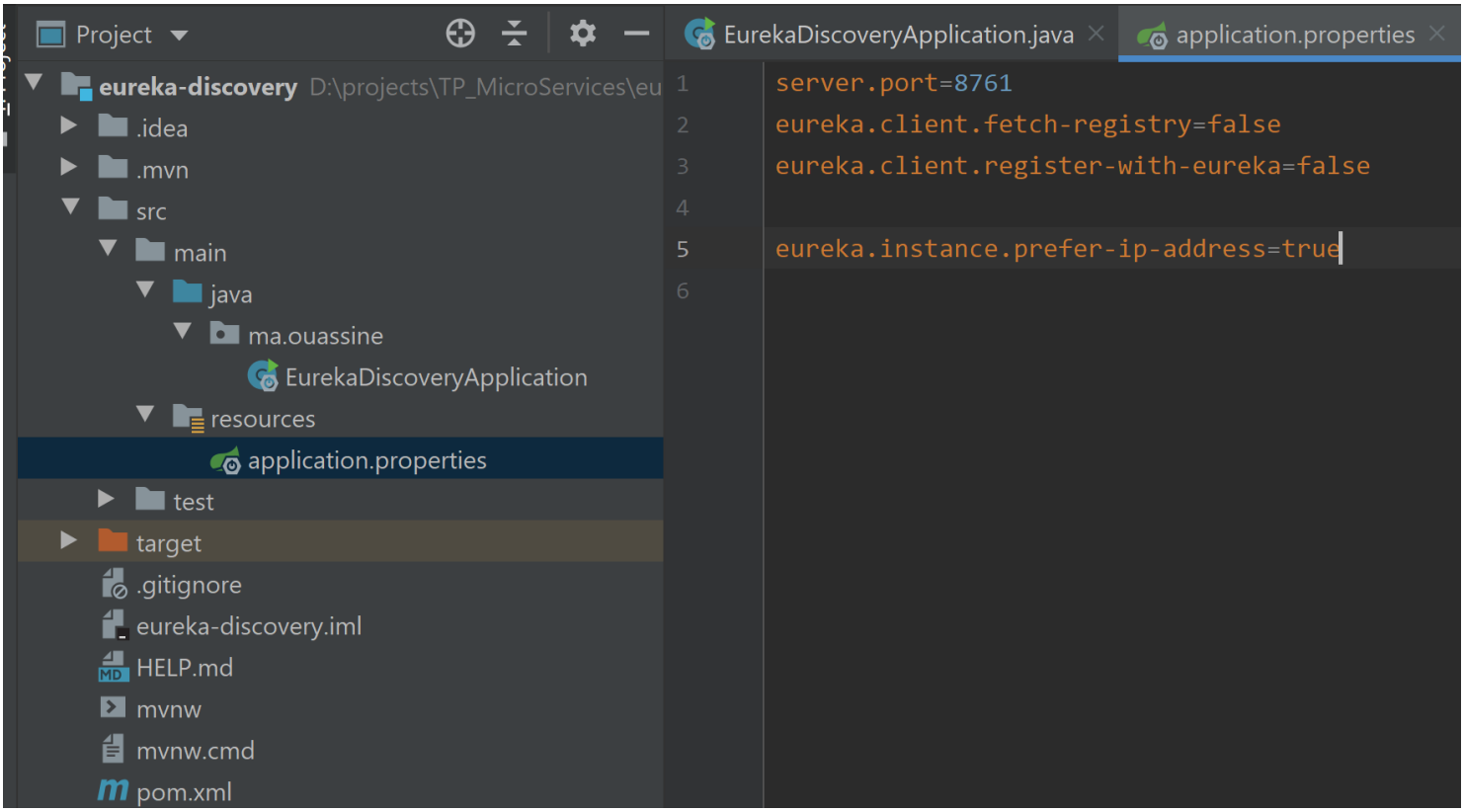
- Eureka Server
- Spring Cloud



squelette du projet Eureka service



L'application Spring principale

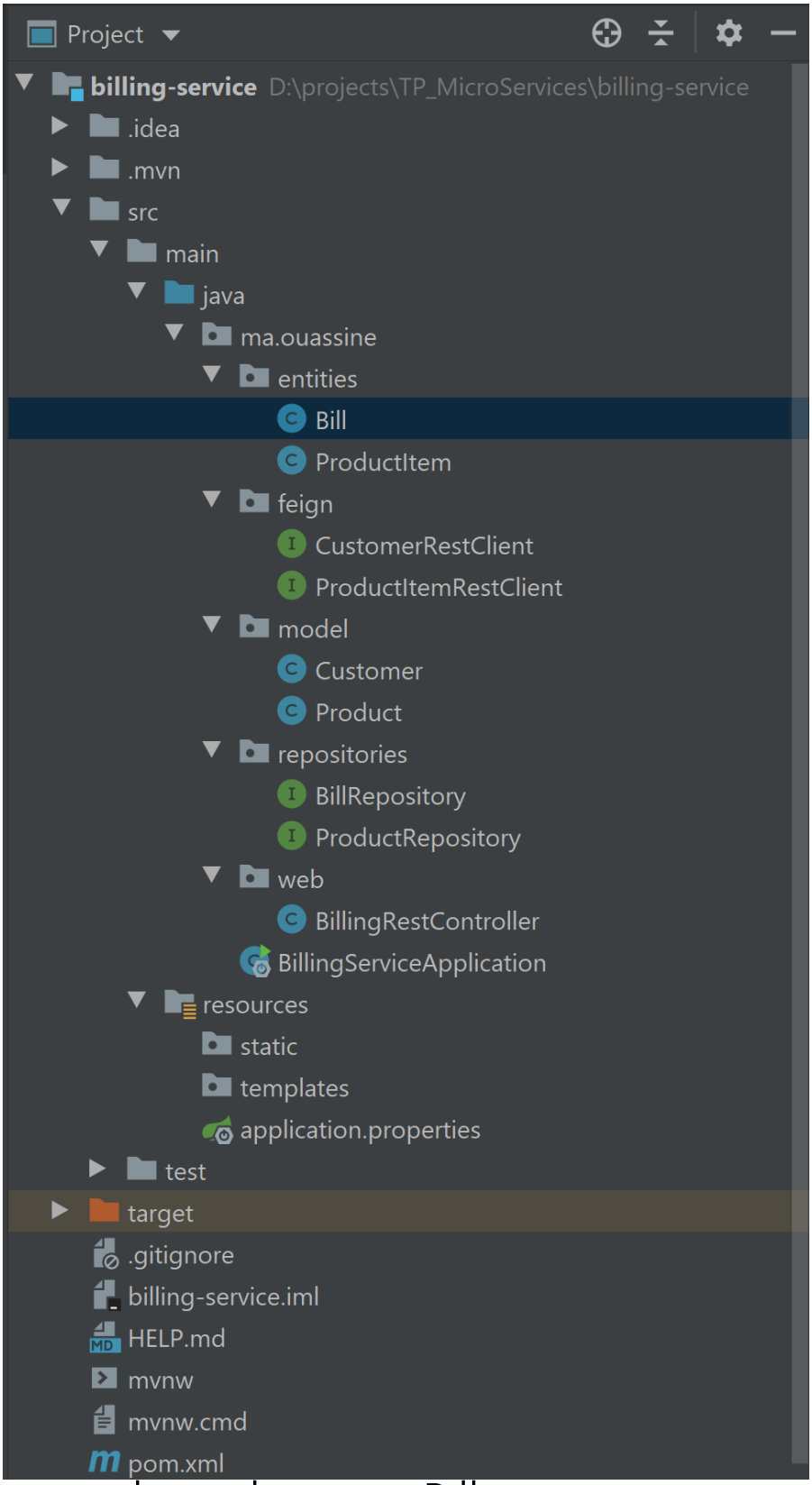


Application.properties

Billing Service

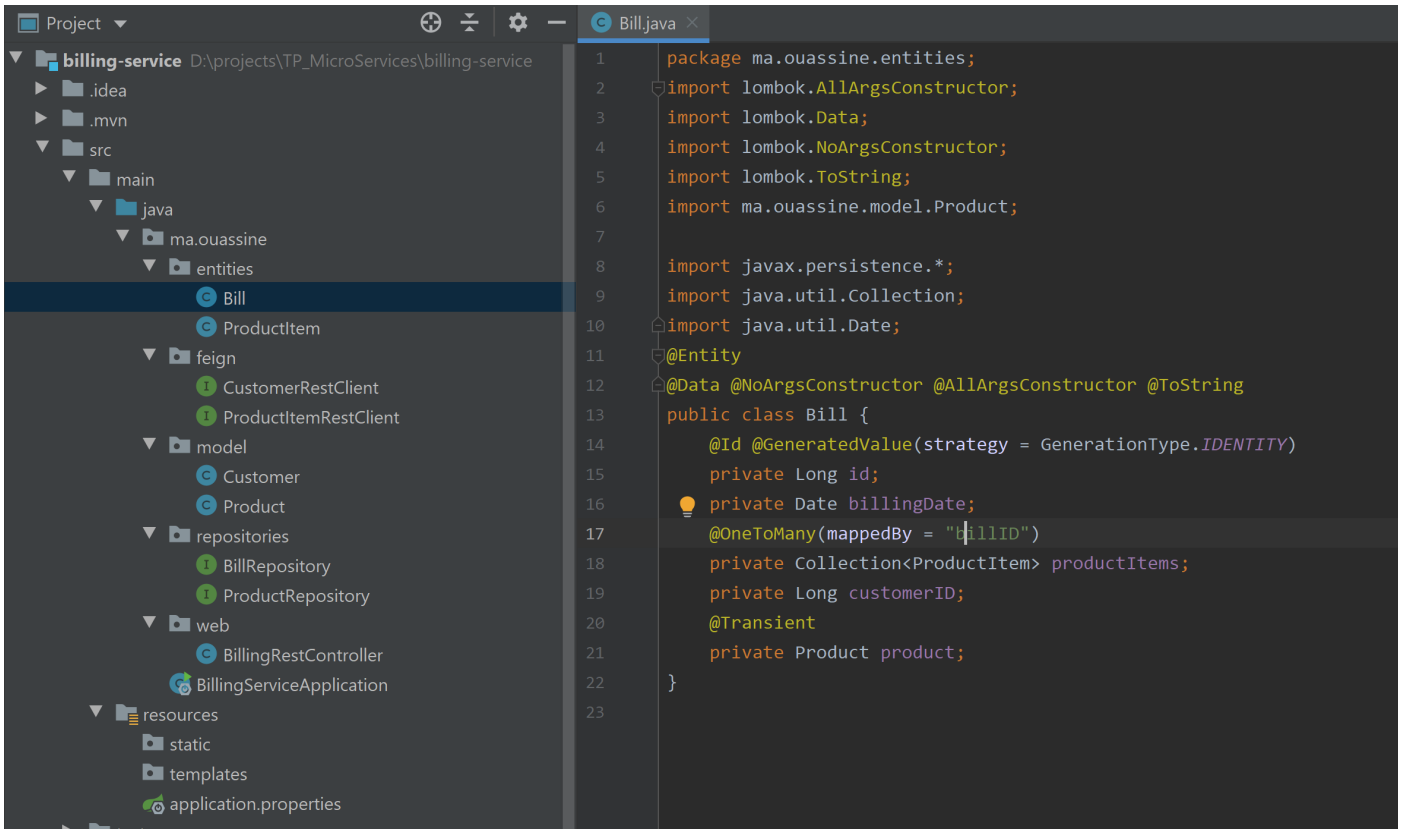
Billing service est le service responsable de la gestion de facturation, les dépendances utilisées sont :

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

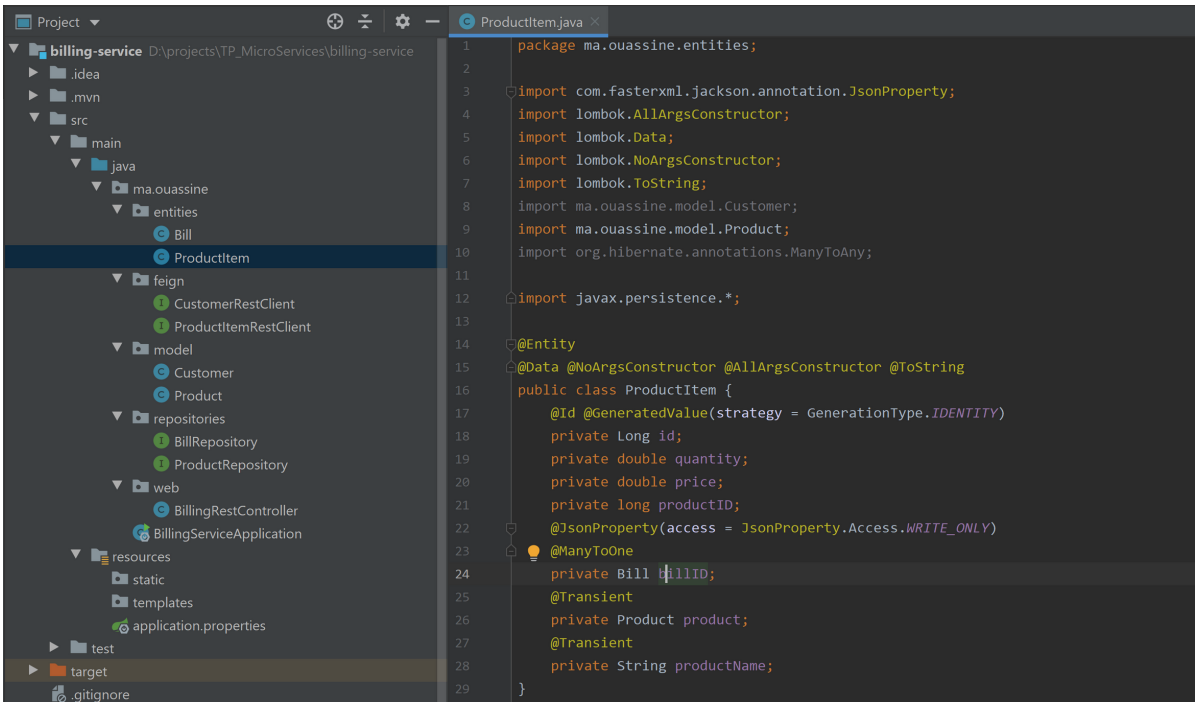


squelette du projet Billing service

Package : entities

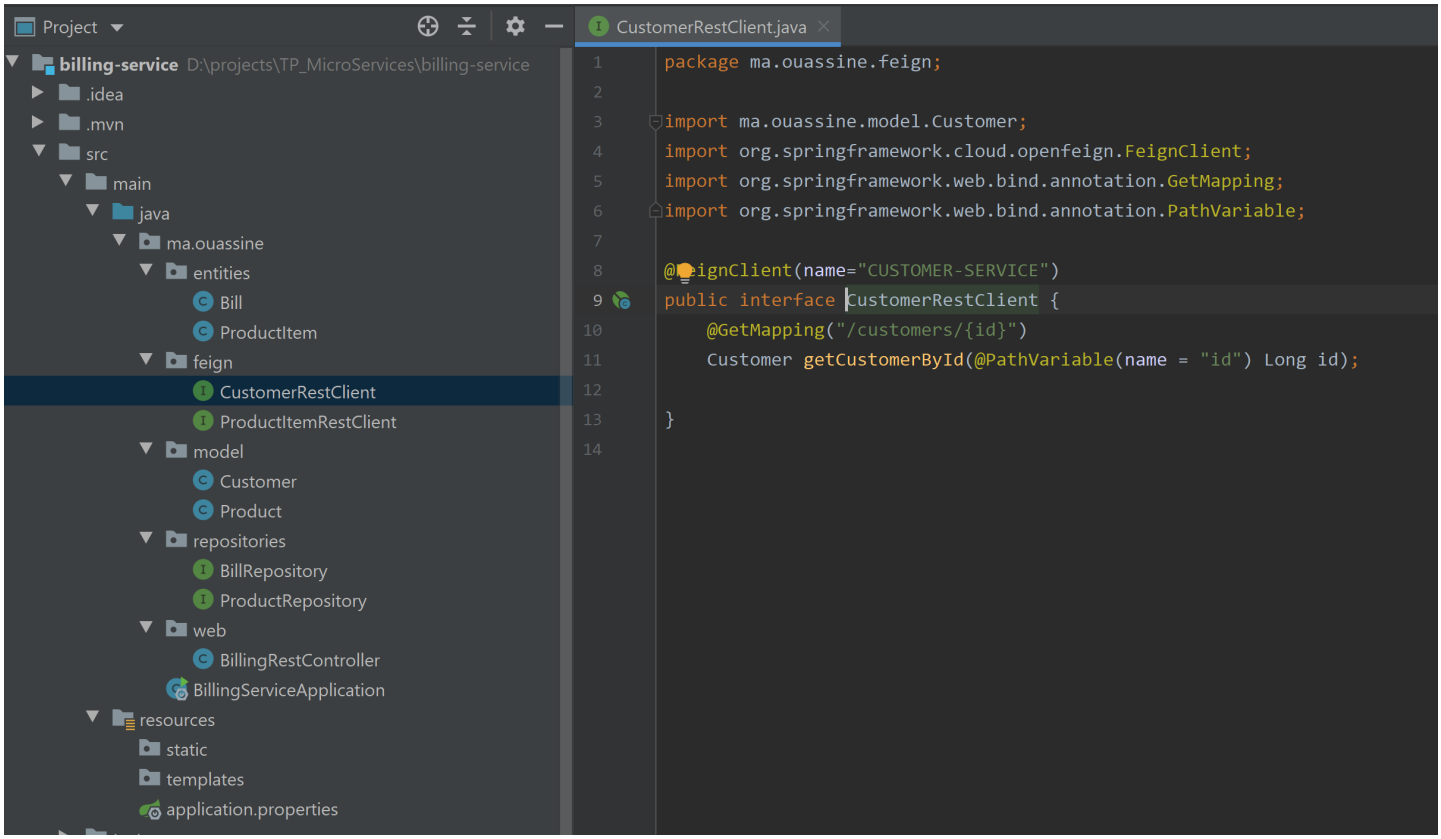


Bill Class

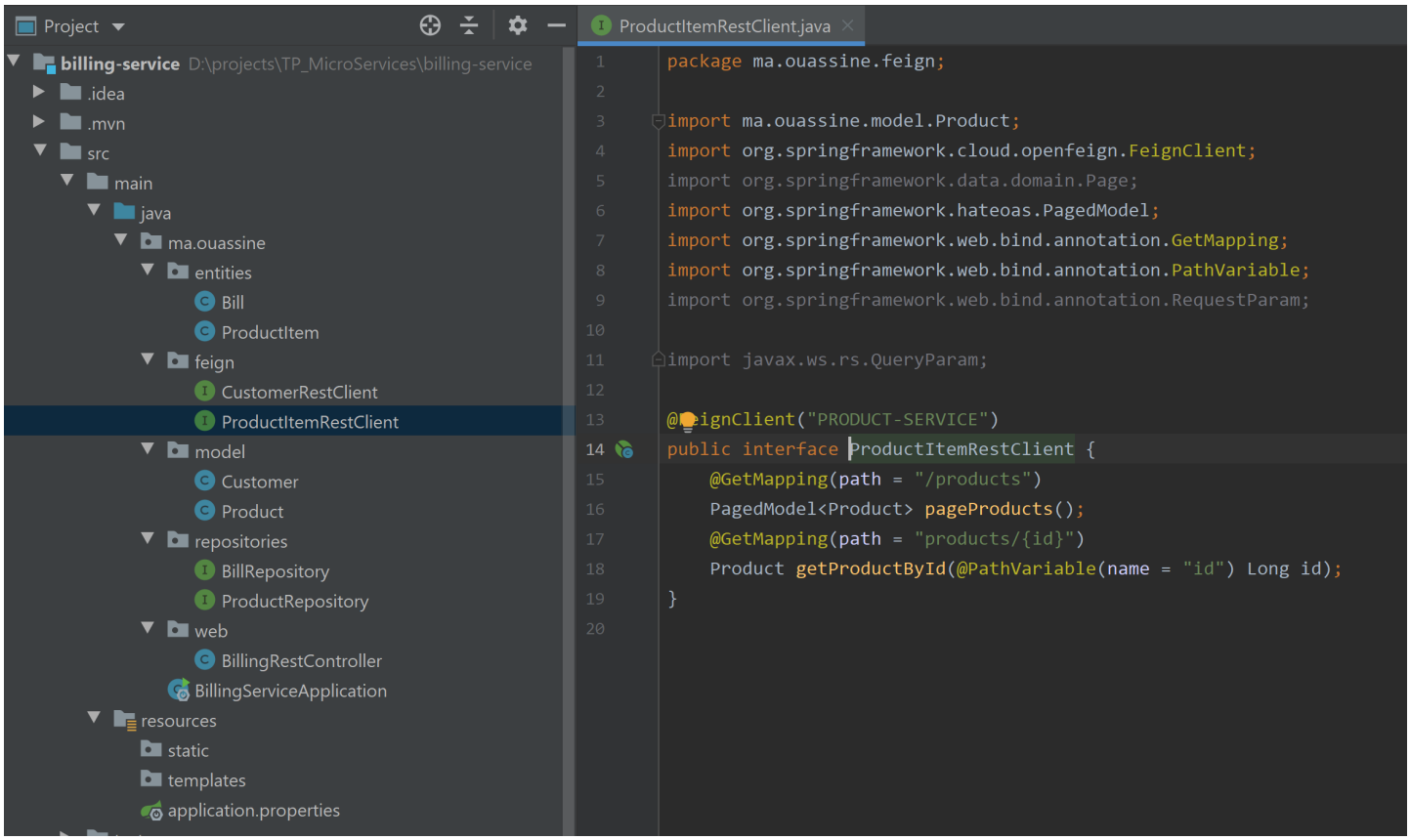


ProductItem Class

Package : Feign

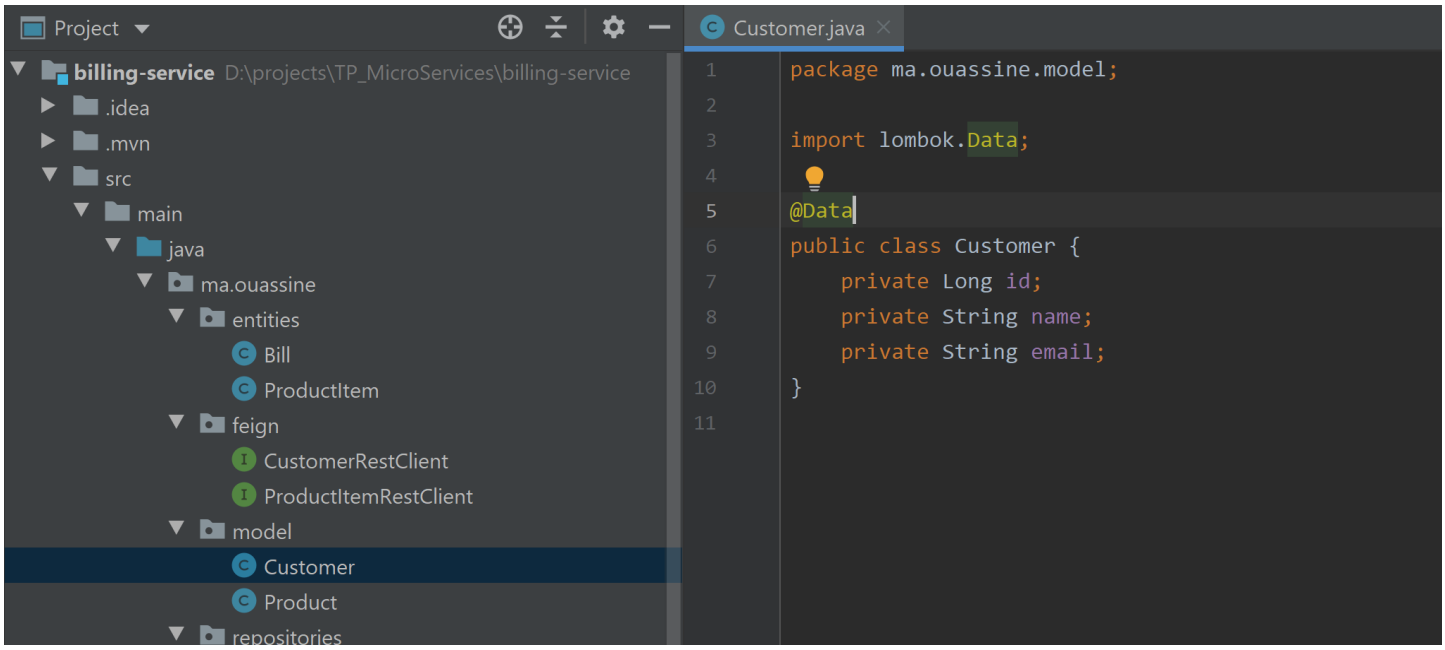


Interface CustomerRestClient

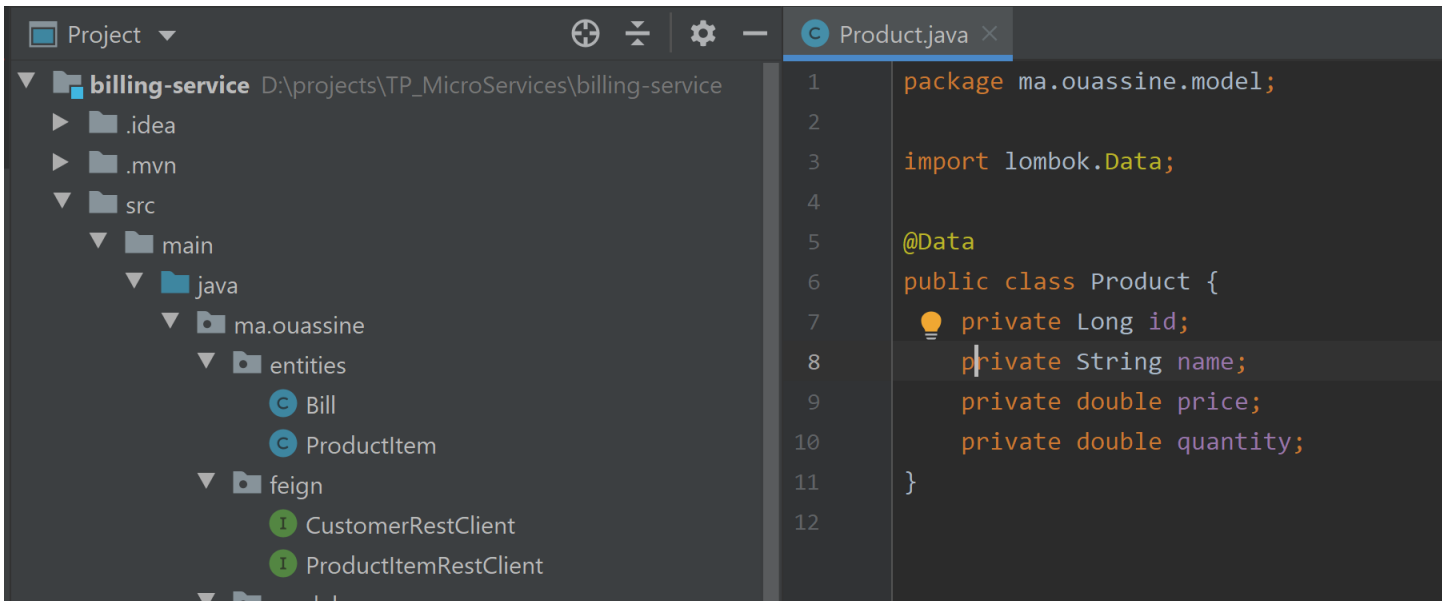


Interface ProductItemRestClient

Package : Model

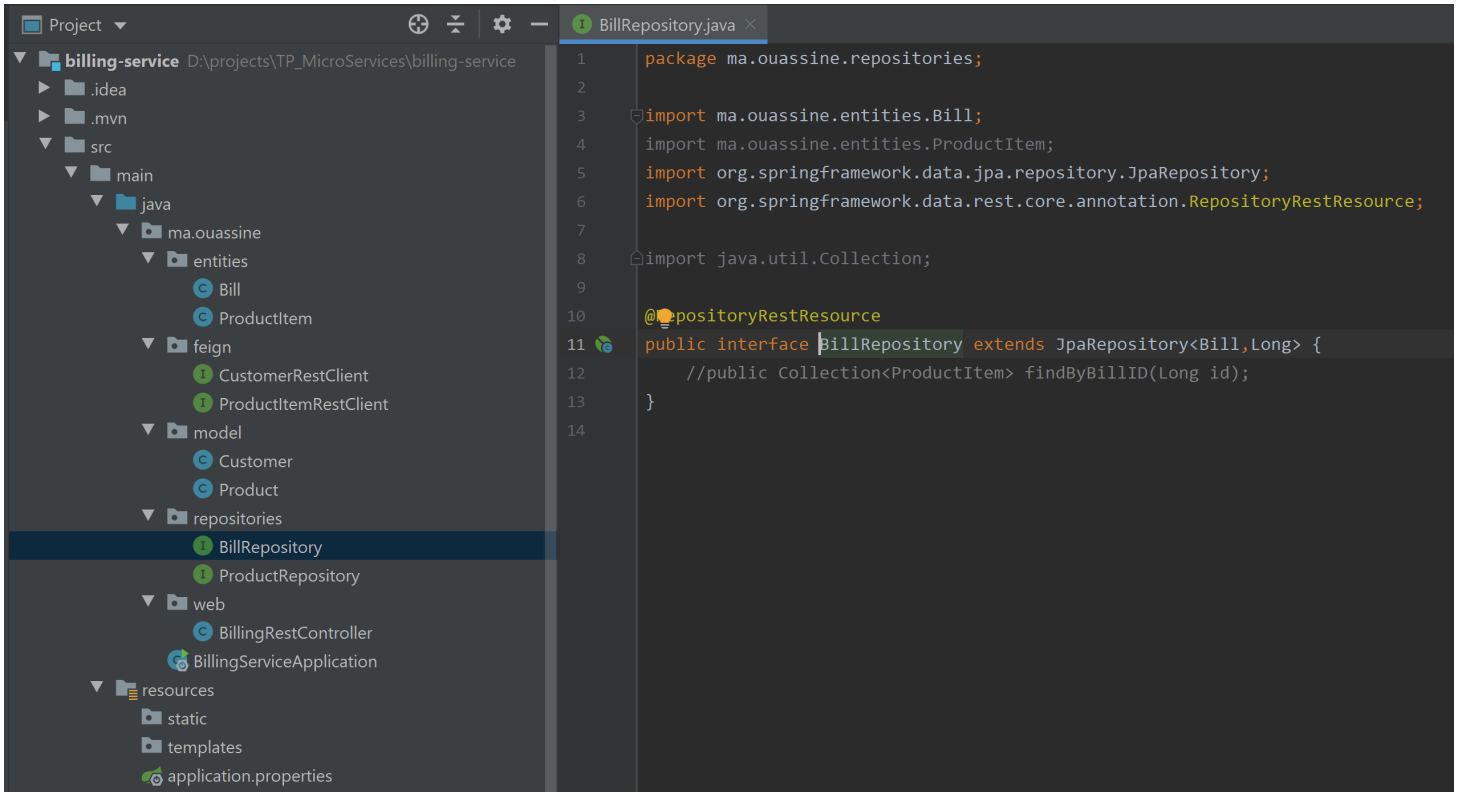


Class Customer

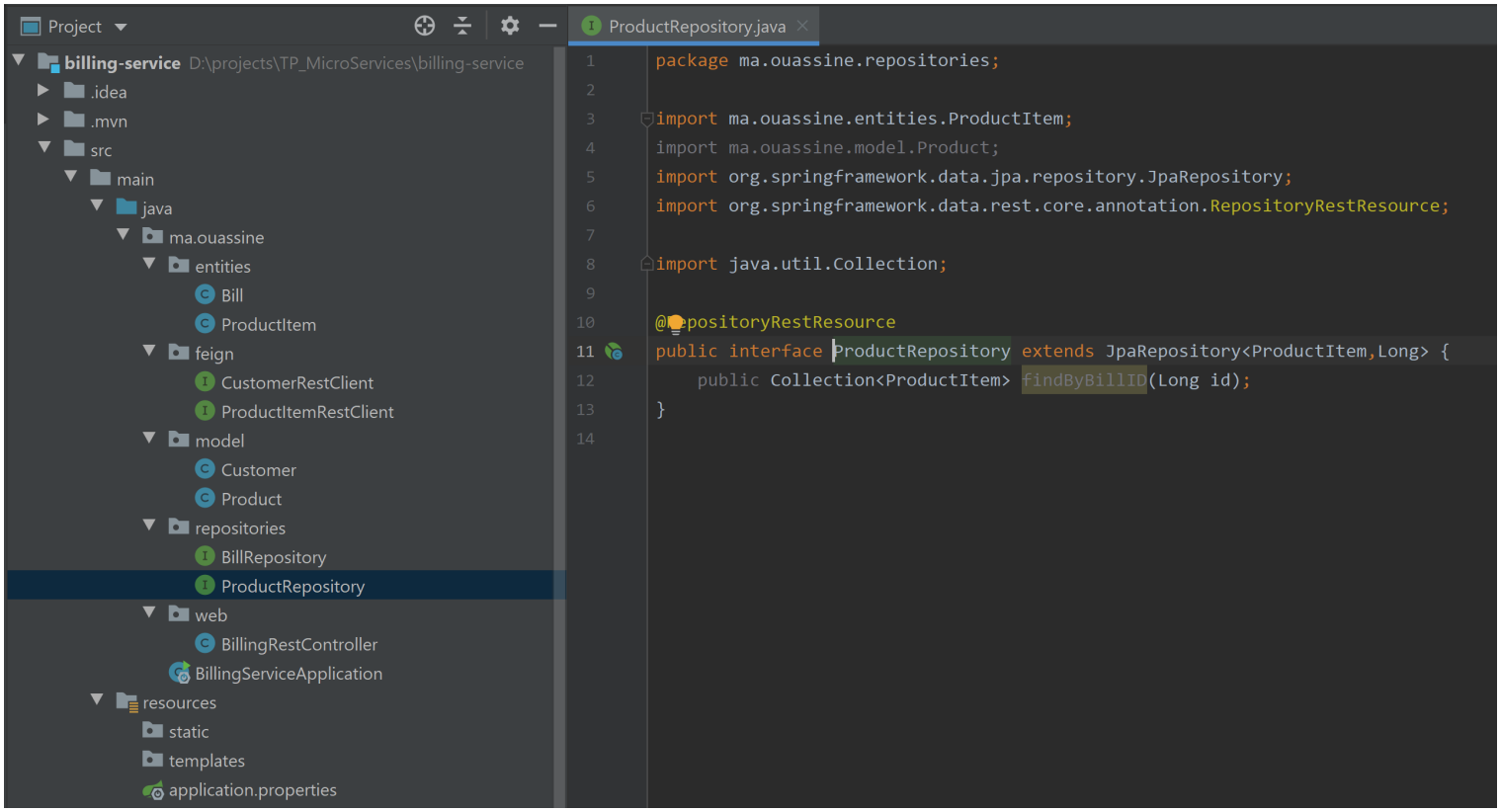


Class Product

Package : Repositories

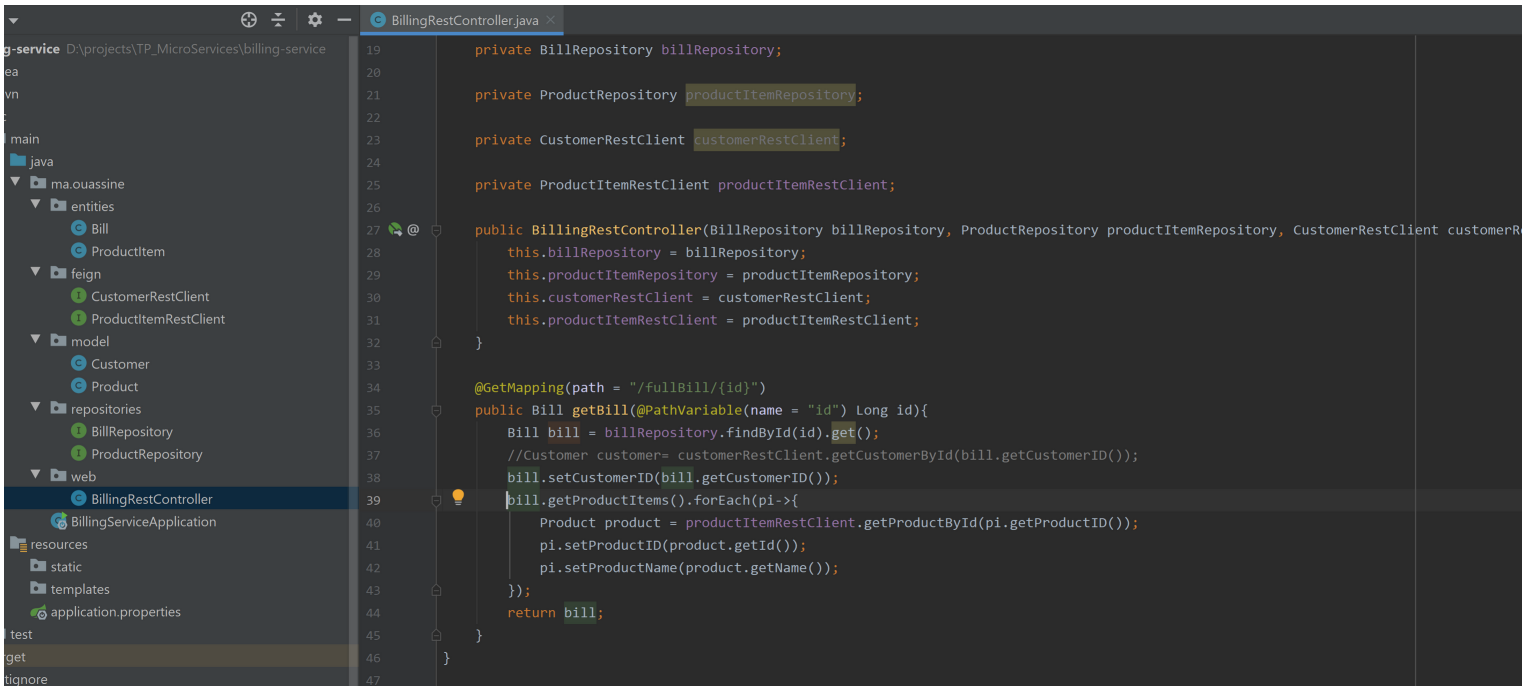


Interface BillRepository

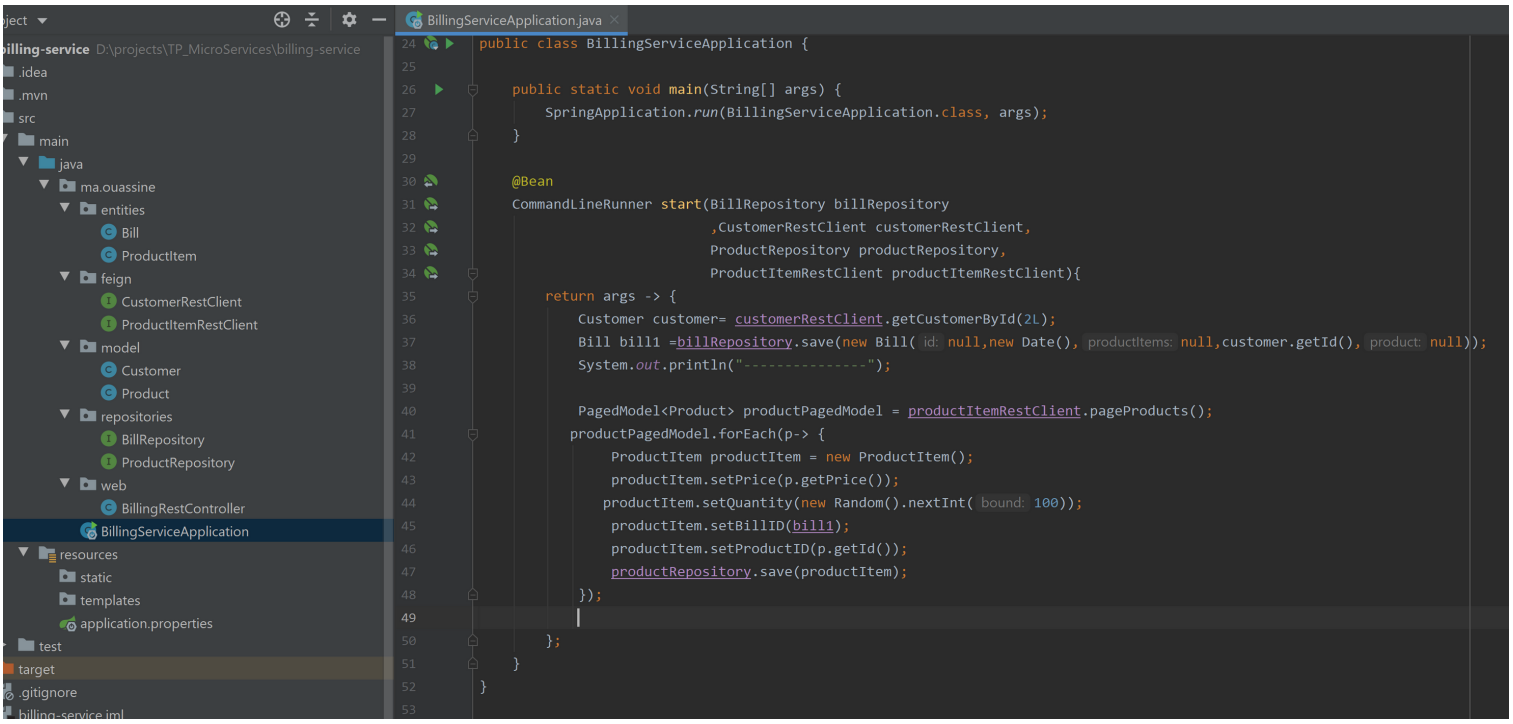


Interface ProductItemRepository

Package : Web

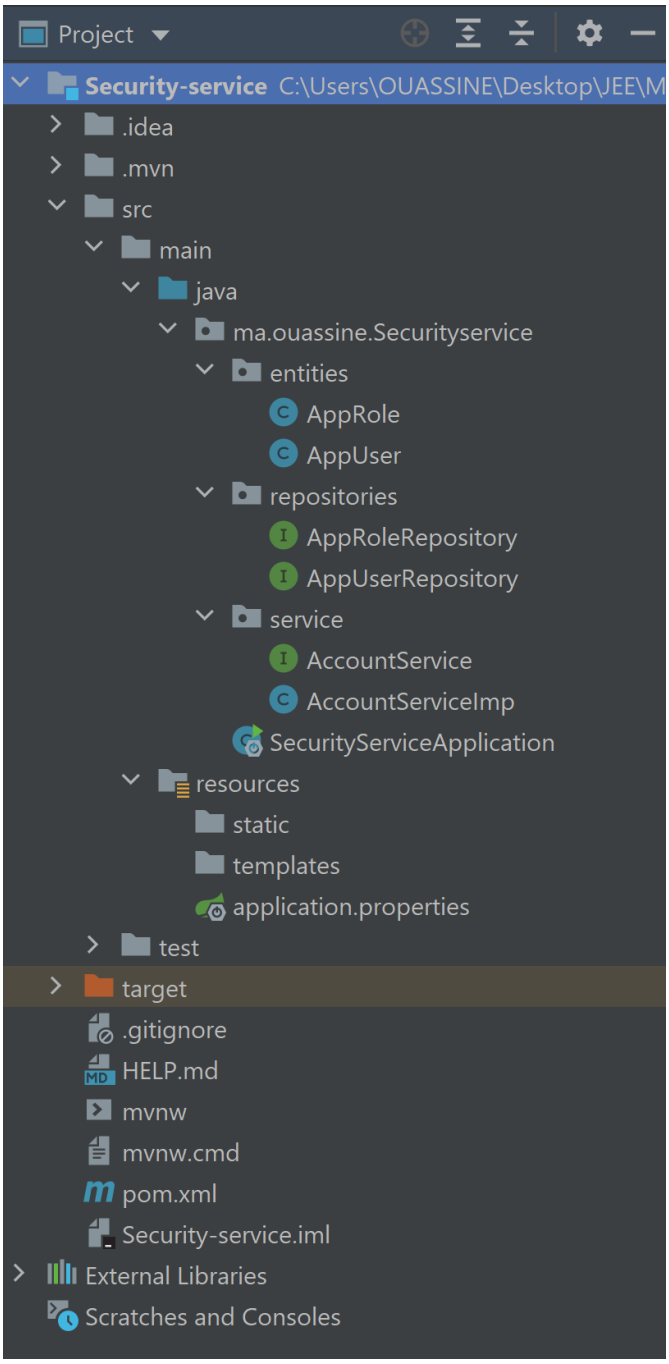


Class Billing RestController



Application Spring principale

security Service



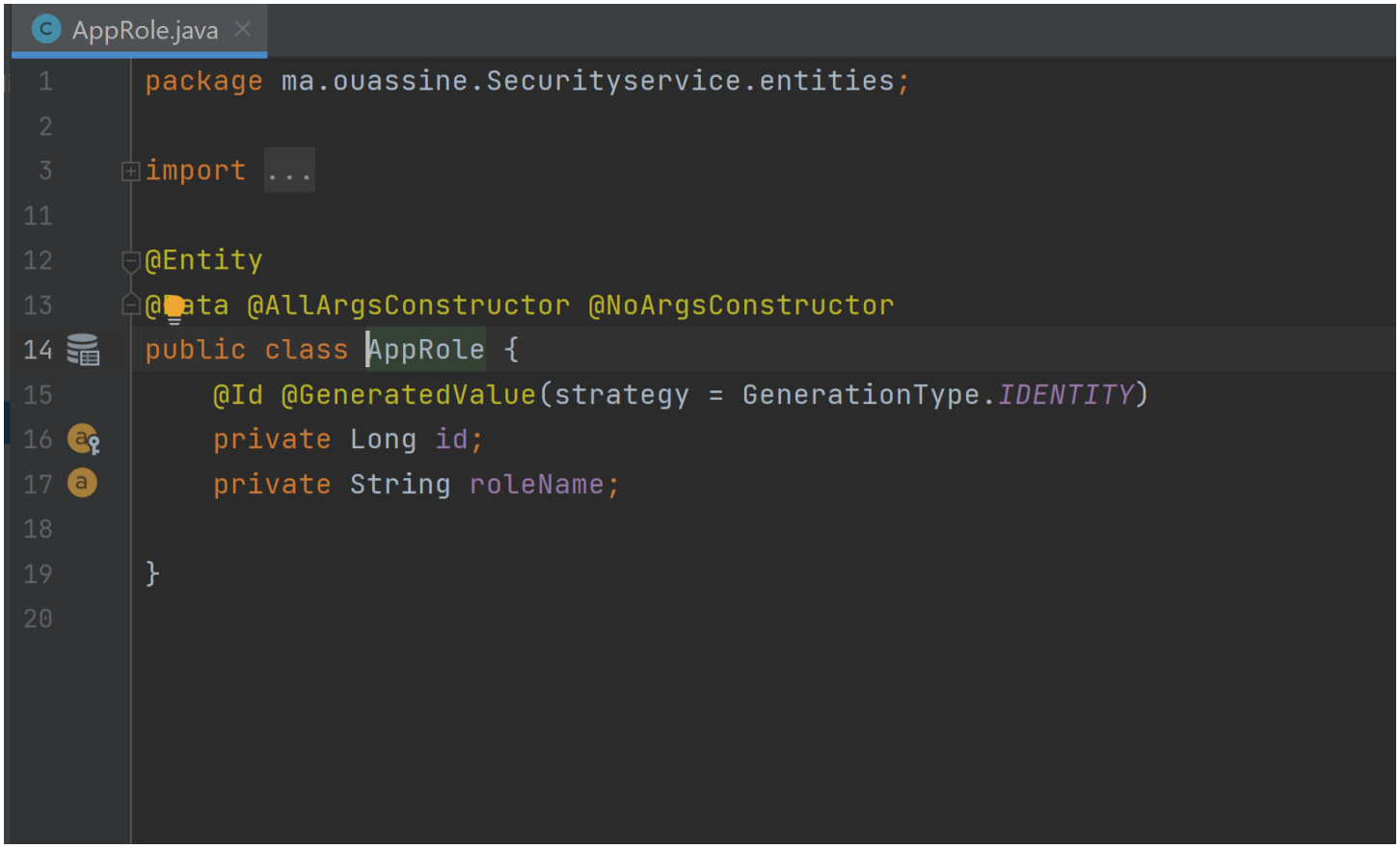
squelette du projet Billing service

Package : entities

AppUser.java

```java
package ma.ouassine.Securityservice.entities;

import ...
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class AppUser {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String username;
    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    private String password;
    @ManyToMany(fetch = FetchType.EAGER)
    private Collection<AppRole> appRoles = new ArrayList<>();

}
```

Class AppUser



Class AppRole

Package : repositories

AppUserRepository.java

```java
package ma.ouassine.Securityservice.repositories;

import ...

public interface AppUserRepository extends JpaRepository<AppUser,Long> {
    AppUser findByUsername(String username);
}
```

L'interface AppUserRepository

L'interface AppRoleRepository

Package : service

AccountService.java

```java
package ma.ouassine.Securityservice.service;

import ...

public interface AccountService {
    AppUser addNewUser(AppUser appUser);
    AppRole addNewRole(AppRole appRole);
    void addRoleToUser(String userName,String roleName);
    AppUser loadUserByUsername(String username);
    List<AppUser> listUsers();

}
```

L'interface Account Service

AccountServiceImp.java

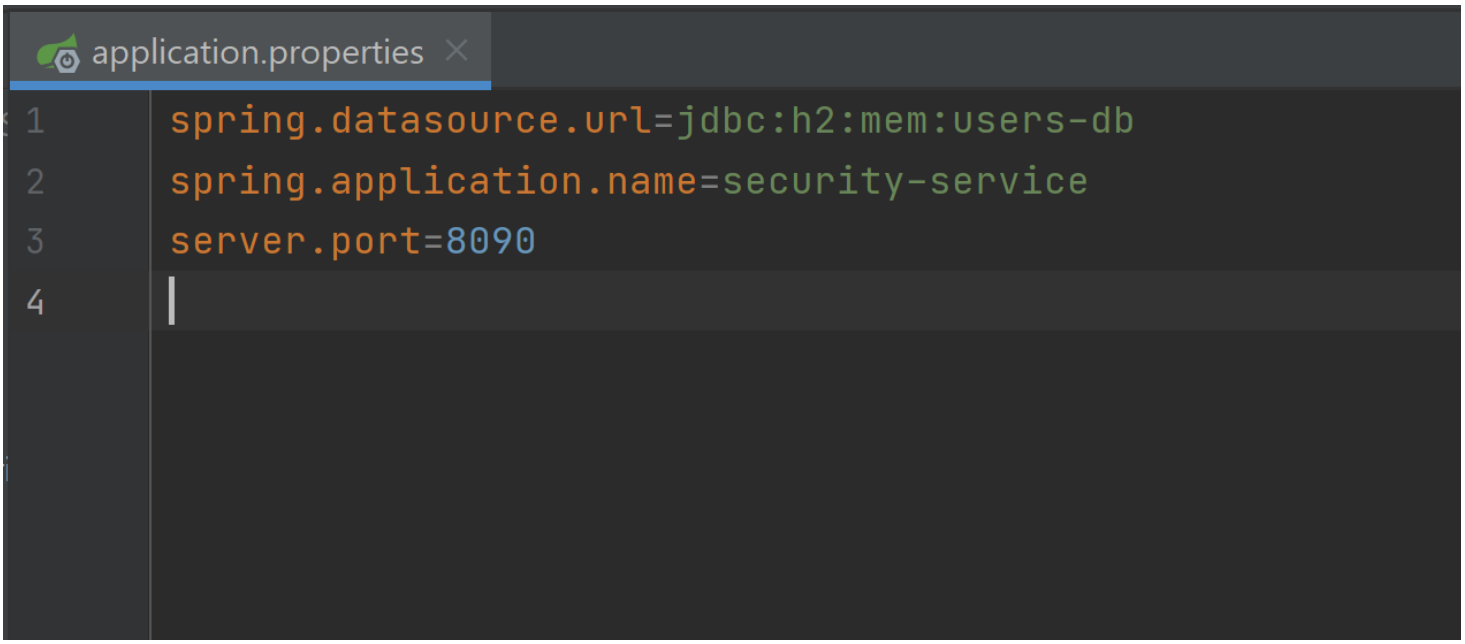
```java
@Service
@Transactional
public class AccountServiceImp implements AccountService  {
    private AppRoleRepository appRoleRepository;
    private AppUserRepository appUserRepository;

    private PasswordEncoder passwordEncoder;
    public AccountServiceImp(AppRoleRepository appRoleRepository, AppUserRepository appUserRepository, PasswordEncoder
        this.appRoleRepository = appRoleRepository;
        this.appUserRepository = appUserRepository;
        this.passwordEncoder = passwordEncoder;
    }

    @Override
    public AppUser addNewUser(AppUser appUser) {
        String pwd= appUser.getPassword();
        appUser.setPassword(passwordEncoder.encode(pwd));
        return appUserRepository.save(appUser);
    }

    @Override
    public AppRole addNewRole(AppRole appRole) { return appRoleRepository.save(appRole); }

    @Override
    public void addRoleToUser(String userName, String roleName) {
        AppUser appUser= appUserRepository.findByUsername(userName);
        AppRole appRole = appRoleRepository.findByRoleName(roleName);
        appUser.getAppRoles().add(appRole);
    }

    @Override
    public AppUser loadUserByUsername(String username) { return appUserRepository.findByUsername(username); }
```

Class AccountserviceImplementation

SecurityServiceApplication.java

```java
@SpringBootApplication
public class SecurityServiceApplication {

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

    PasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder(); }

    @Bean
    CommandLineRunner start(AccountService accountService){
        return args ->{
            accountService.addNewRole(new AppRole( id: null, roleName: "USER"));
            accountService.addNewRole(new AppRole( id: null, roleName: "ADMIN"));
            accountService.addNewRole(new AppRole( id: null, roleName: "CUSTOMER-MANAGER"));
            accountService.addNewRole(new AppRole( id: null, roleName: "PRODUCT-MANAGER"));
            accountService.addNewRole(new AppRole( id: null, roleName: "BILLS-MANAGER"));

            accountService.addNewUser(new AppUser( id: null, username: "user1", password: "1234",new ArrayList<>()));
            accountService.addNewUser(new AppUser( id: null, username: "user2", password: "abcd",new ArrayList<>()));
            accountService.addNewUser(new AppUser( id: null, username: "user3", password: "azerty",new ArrayList<>()));

            accountService.addRoleToUser( userName: "user1", roleName: "USER");
            accountService.addRoleToUser( userName: "user2", roleName: "ADMIN");
            accountService.addRoleToUser( userName: "user3", roleName: "PRODUCT-MANAGER");


        };
    }
}
```

L'application spring principale



appliation.properties file

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

Sécurisation d'Inventory Service

KeycloakConfig.java

```java
package ma.ouassine.securite;

import ...

@Configuration
public class KeycloakConfig {
    @Bean
    KeycloakSpringBootConfigResolver configResolver() { return new KeycloakSpringBootConfigResolver(); }


}

```

@Configuration

KeycloakSpringSecuriteConfig.java | KeycloakConfig.java

```java
package ma.ouassine.securite;

import ...

@KeycloakConfiguration
public class KeycloakSpringSecuriteConfig extends KeycloakWebSecurityConfigurerAdapter {

    @Override
    protected SessionAuthenticationStrategy sessionAuthenticationStrategy() {
        return new RegisterSessionAuthenticationStrategy(new SessionRegistryImpl());
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.authenticationProvider(keycloakAuthenticationProvider());
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        super.configure(http);
        http.authorizeRequests().antMatchers( ...antPatterns: "/produits/**").authenticated();
    }



}
```

@KeycloackConfiguration

Sécurisation du Customer Service

KeycloakConfig.java

```java
package com.yassa.supplierService.securite;

import ...

@Configuration
public class KeycloakConfig  {
    @Bean
    KeycloakSpringBootConfigResolver configResolver() { return new KeycloakSpringBootConfigResolver(); }

}
```

@Configuration

KeycloakSpringSecuriteConfig.java

```java
package com.yassa.supplierService.securite;

import ...

@KeycloakConfiguration
public class KeycloakSpringSecuriteConfig extends KeycloakWebSecurityConfigurerAdapter {

    @Override
    protected SessionAuthenticationStrategy sessionAuthenticationStrategy() {
        return new RegisterSessionAuthenticationStrategy(new SessionRegistryImpl());
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.authenticationProvider(keycloakAuthenticationProvider());
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        super.configure(http);
        http.authorizeRequests().antMatchers( ...antPatterns: "/suppliers/**").hasAuthority("app-manager");
    }
}
```

@KeycloackConfiguration

Sécurisation du Billing Service

KeycloakConfig.java

```java
package org.sid.billingservice.securite;

import ...

@Configuration
public class KeycloakConfig  {
    @Bean
    KeycloakSpringBootConfigResolver configResolver() { return new KeycloakSpringBootConfigResolver(); }

}
```

@Configuration

KeycloakSpringSecuriteConfig.java

```java
package org.sid.billingservice.securite;

import ...

@KeycloakConfiguration
public class KeycloakSpringSecuriteConfig extends KeycloakWebSecurityConfigurerAdapter {

    @Override
    protected SessionAuthenticationStrategy sessionAuthenticationStrategy() {
        return new RegisterSessionAuthenticationStrategy(new SessionRegistryImpl());
    }

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.authenticationProvider(keycloakAuthenticationProvider());
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        super.configure(http);
        http.authorizeRequests().antMatchers( ...antPatterns: "/bills/**").hasAuthority("app-manager");
    }
}
```

@KeycloackConfiguration