EMSI
Ecole Marocaine des Sciences de l'Ingénieur
المدرسة المغربية لعلوم المهندس

RAPPORT DU PROJET

SPÉCIALITÉ : GÉNIE INFORMATIQUE

Réalisé par : ibtissam kasmi

Table des matières

INTRODUCTION GENERALE

A la fin de ce module,les étudiants ont l'opportunité de réaliser un projet qui a était proposé par notre professeur Mr Mohamed Youssfi qui consiste à développer une application web qui permet de voir l'utilité de spring cloud gateway.

La réalisation de ce projet à pour buts :

- Évaluer les étudiants sous plusieurs éléments tels que l'organisation du projet, le travail fournis depuis les connaissances acquises au cours de la formation d'ingénieur ou encore le respect du cahier des charges.
-Permettre aux étudiants d'acquérir de nouvelles connaissances et ainsi obtenir de nouvelles compétences relatives à leur futur métier.
Dans ce ce Rapport ,vous découvrirez un exemple très simplifié de l'utilisation de Spring cloud gateway .
Selon le cahier des charges défini au début de ce projet,L'application se compose de 3 Parties : La partie backend et la partie Frontend,ainsi la partie security

Ce rapport va s'articuler autour de trois chapitres.

Il fait la synthèse de mon travail pour la période du 3 semaines .
Dans le premier chapitre, s'articule autour d'une présentation générale de l'étude Préalable d'application.
le deuxième chapitre , nous traitons la partie conception du projet .
le troisième chapitre, Cette partie aborde la partie backend du projet backend basé sur ensemble des micro services :Customer-service,Inventory-service,Billing-service et l'orchestration des services se fait via les services techniques de Spring Cloud :
Spring Cloud Gateway Service comme service proxy et Registry Eureka Service comme annuaire d'enregistrement et Service d'authentification avec Spring Security et JWT,et à la Partie Frontend à base d'Angular et enfin, je présente notre application dédiée.
Ce projet est réalisée en suivant l'ensemble des vidéos proposé par Mr Mohamed Youssfi

Chapitre 1: Contexte général

1-1 Introduction:

Dans ce chapitre, On commence par une présentation de contexte du projet
Ensuite, on détermine le cahier de charge et les objectifs à atteindre de ce projet

1-2 ETUDE DE L'EXISTANT

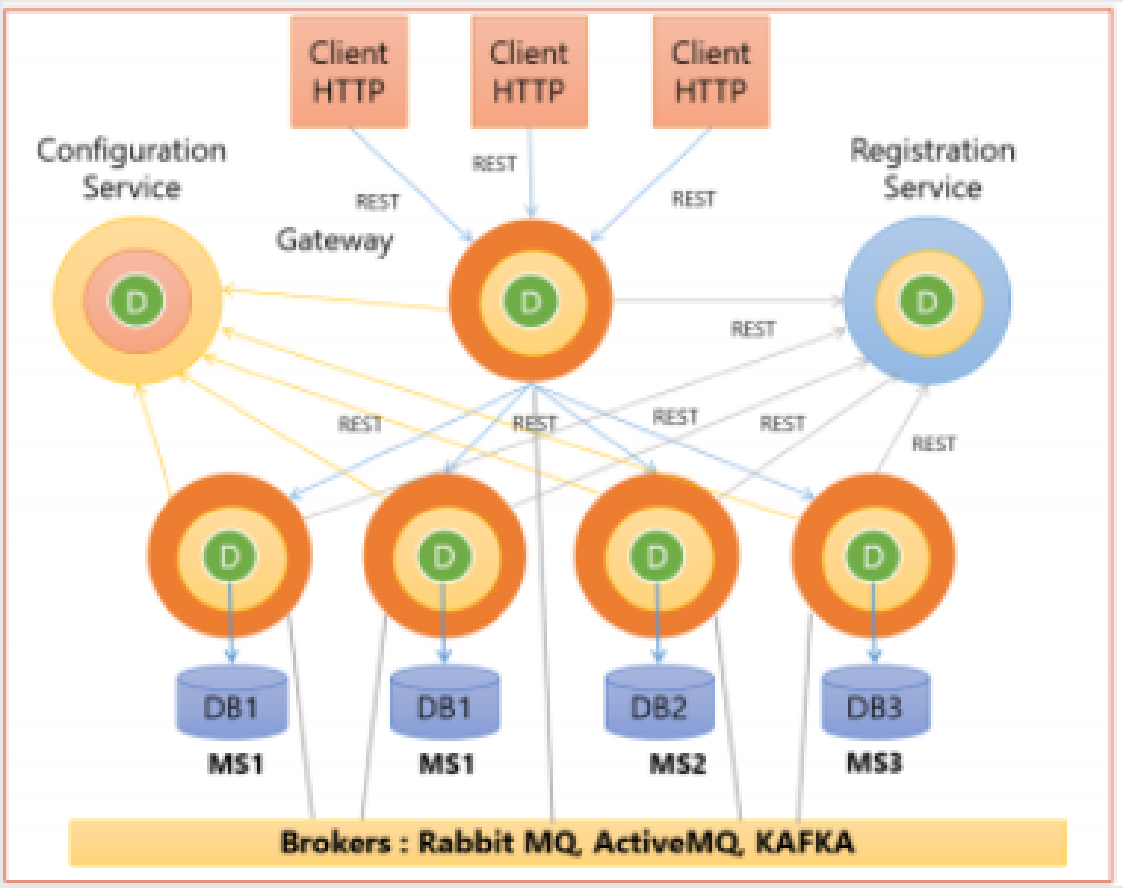
a-problème

Principaux Problèmes d'une approche monolithique

Une application monolithique est une application qui est développée en un seul bloc (war, jar, Ear, dll),avec une même technologie et déployée dans un serveur d'application

b. Solution:

Approche Micro services

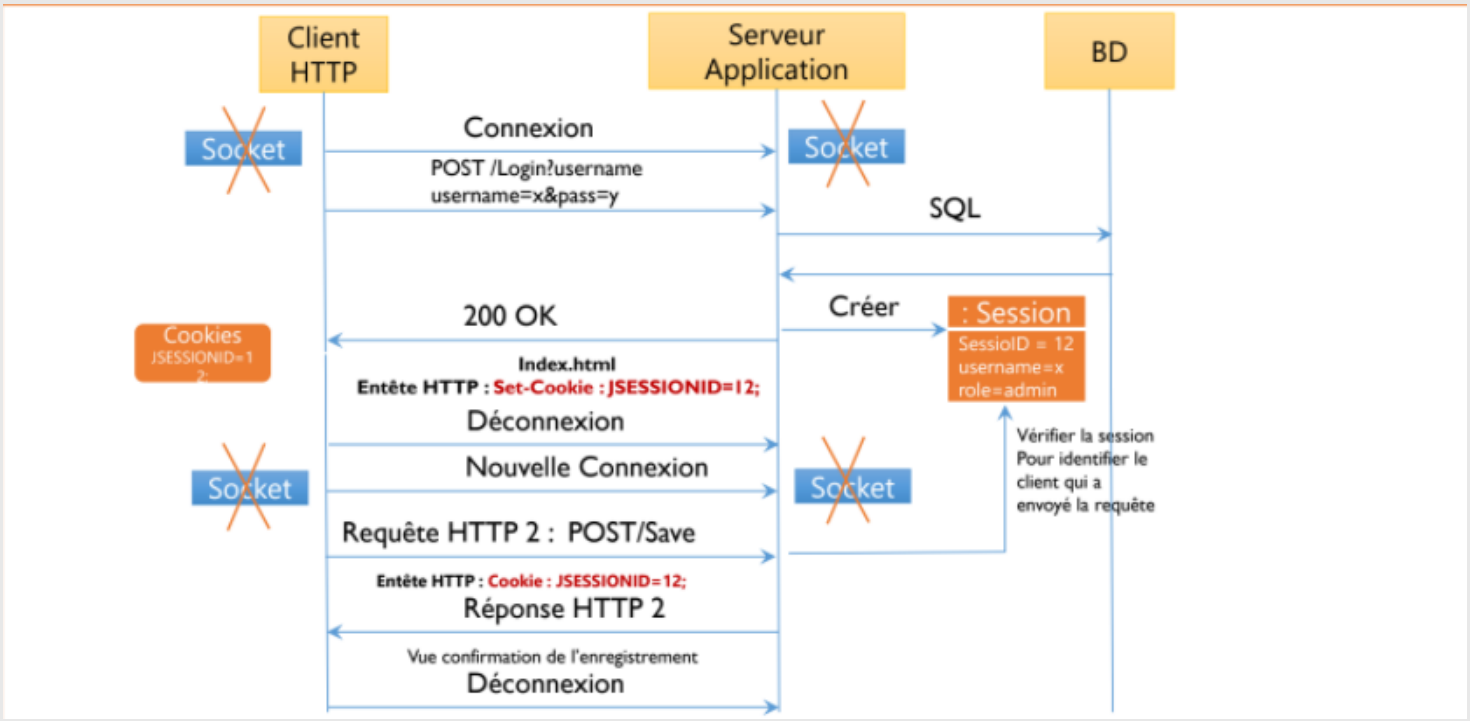


• Les micro services sont une approche d'architecture et de développement d'une application composées de petits services.
• L'idée étant de découper un grand problème en petites unités implémentée sous forme de micro-services
• Chaque service est responsable d'une fonctionnalité,
• Chaque micro-service est développé, testé et déployé séparément des autres.
• Chaque micro service est développé en utilisant une technologie qui peut être différente des autres. (Java, C++, C#, PHP, NodeJS, Pyton, ...)
• Chaque service tourne dans un processus séparé.
• Utilisant des mécanismes de communication légers (REST)
• La seule relation entre les différents micro services est l'échange dedonnées effectué à travers les différentes APIs qu'ils exposent. (SOAP, REST, RMI, CORBA, JMS, MQP, ...)
• Lorsqu'on les combinent, ces micro services peuvent réaliser desopérations très complexes.

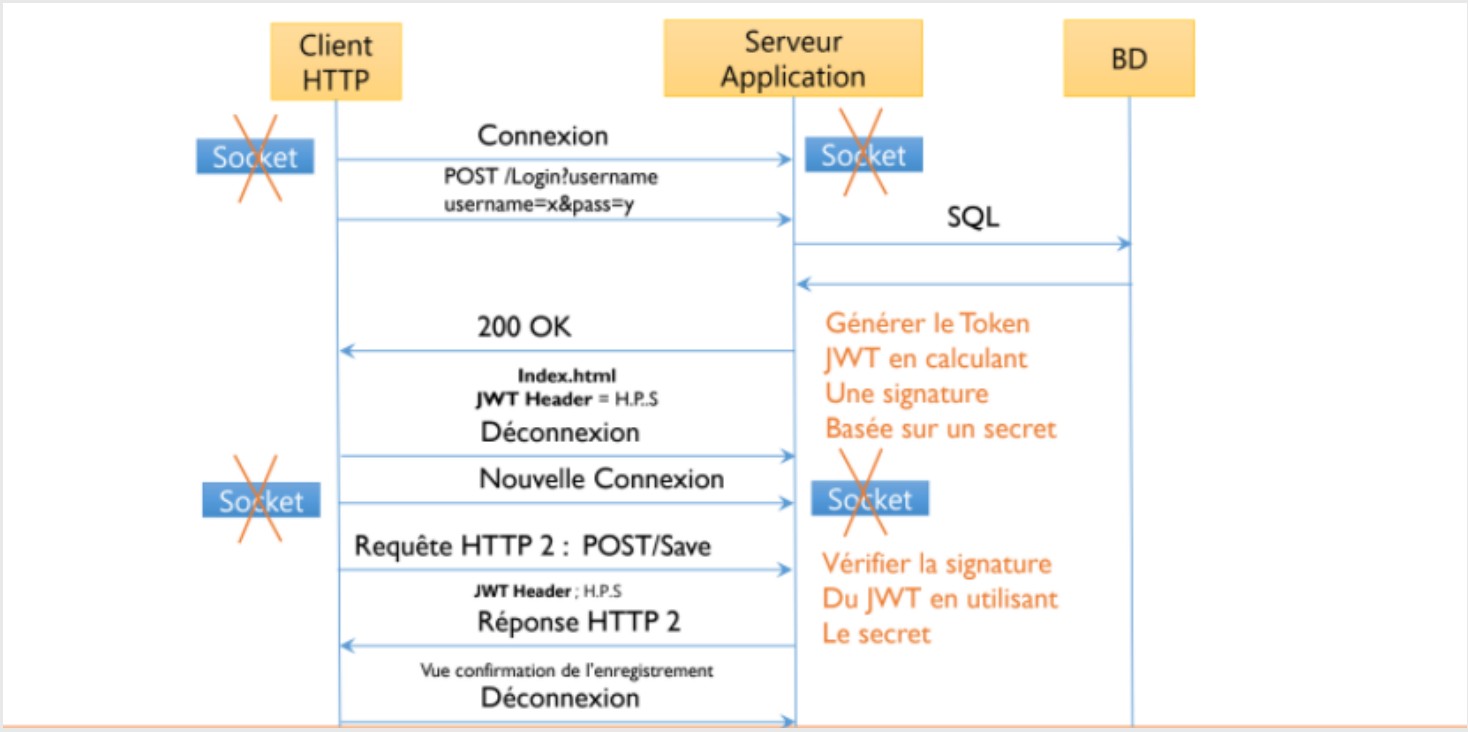
Sécurité :

Deux types de modèles d'authentification :
• Statful : Les données de la session sont enregistrés coté serveur d'authentification
• Statless : les données de la session sont enregistrés dans un jeton d'authentification délivré au client.

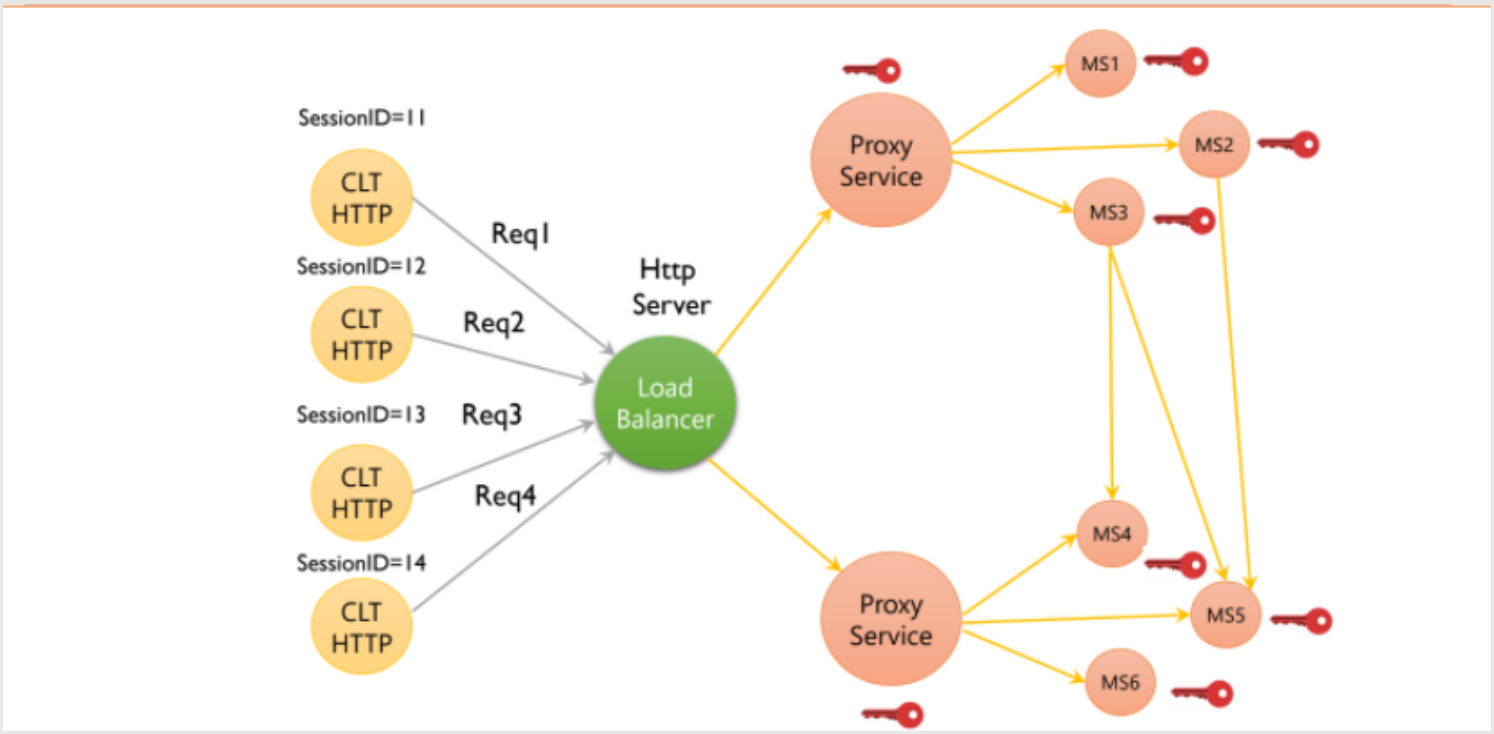
Authentification Statful basée sur les Sessions et Cookies



Authentification Statless basée sur les Sessions et Cookies



Bonne solution pour les architectures distribuées basées sur les micro services



1-3 Les exigences fonctionnelles de l'application

Créer une application basée sur deux services métiers:

- Service des clients
- Service d'inventaire
- Service facturation
- Service Externes: RapidAPI
- l'orchestration des services se fait via les services techniques despring cloud:
- Spring Cloud Gateway Service comme service proxy
- Registry Eureka Service comme annuaire d'enregistrement et de découverte des services de
- l'architecture
- Hystrix Circuit Breaker
- Hystrix DashBoard

- Créer un micro service d'authentification en utilisant Spring Security et JWT

a-Micro services

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:

- 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

- Tester la Service proxy en utilisant une configuration Statique basée sur le fichier application.yml
- Tester la Service proxy en utilisant une configuration Statique basée une configuration Java

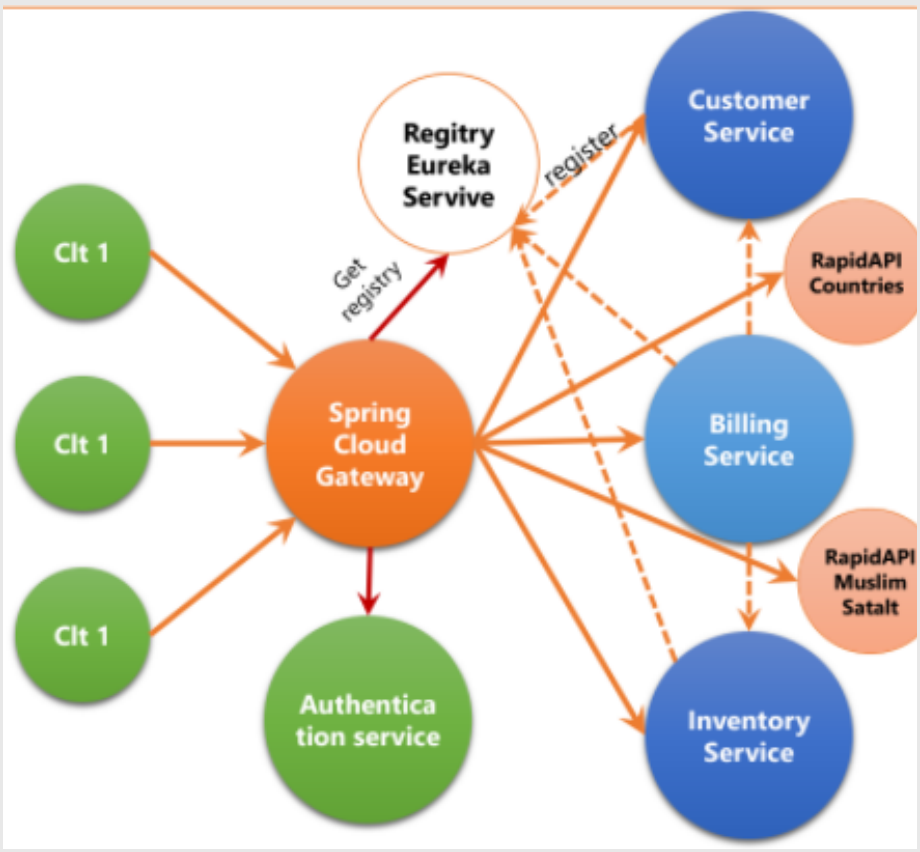
4. Créer l'annuaire Registry Service basé sur NetFlix Eureka Server

5. Tester le proxy en utilisant une configuration dynamique de Gestion des routes vers les micro services enregistrés dans l'annuaire Eureka Server

b-Partie Securité :

Service d'authentification avec Spring Security et JWT

-Créer un micro service d'authentification en utilisant Spring Security et JWT
-Ce services permet de gérer Les utilisateurs
-Les rôles (USER, ADMIN, CUSTOMER_MANAGER, PRODUCT_MANAGER, BILLS_MANAGER)
-Un utilisateur peut avoir plusieurs rôles et chaque rôle peut être affecté à plusieurs utilisateurs

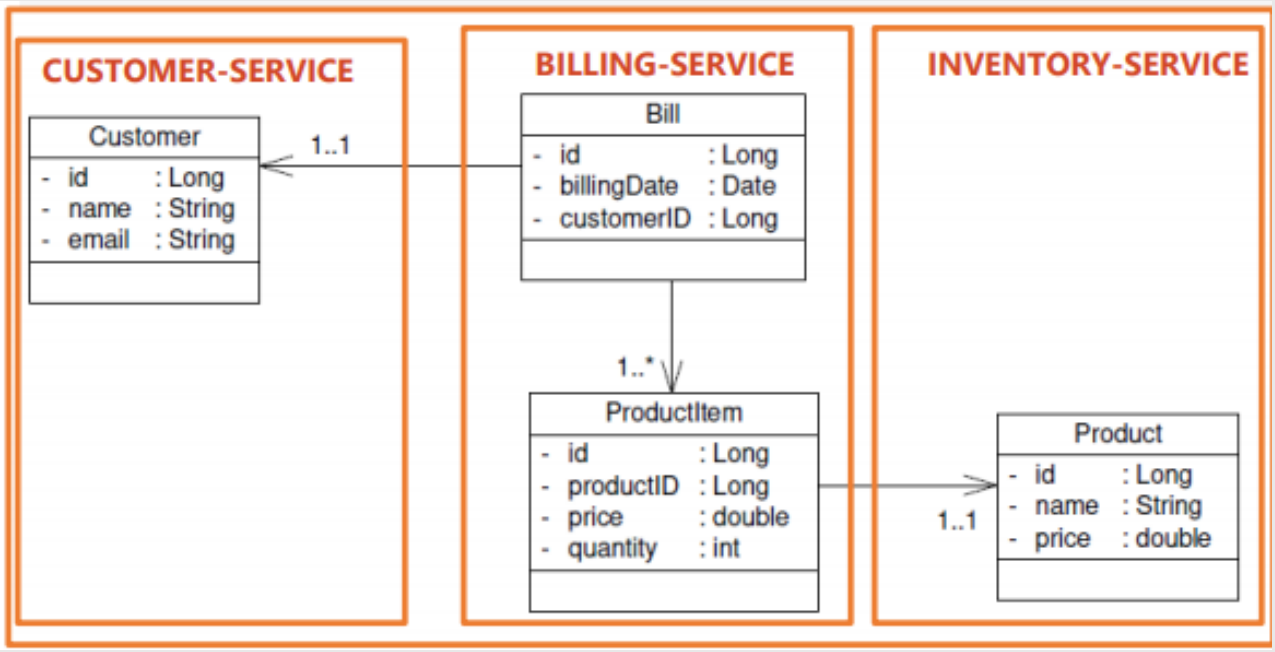


Chapitre2:Conception

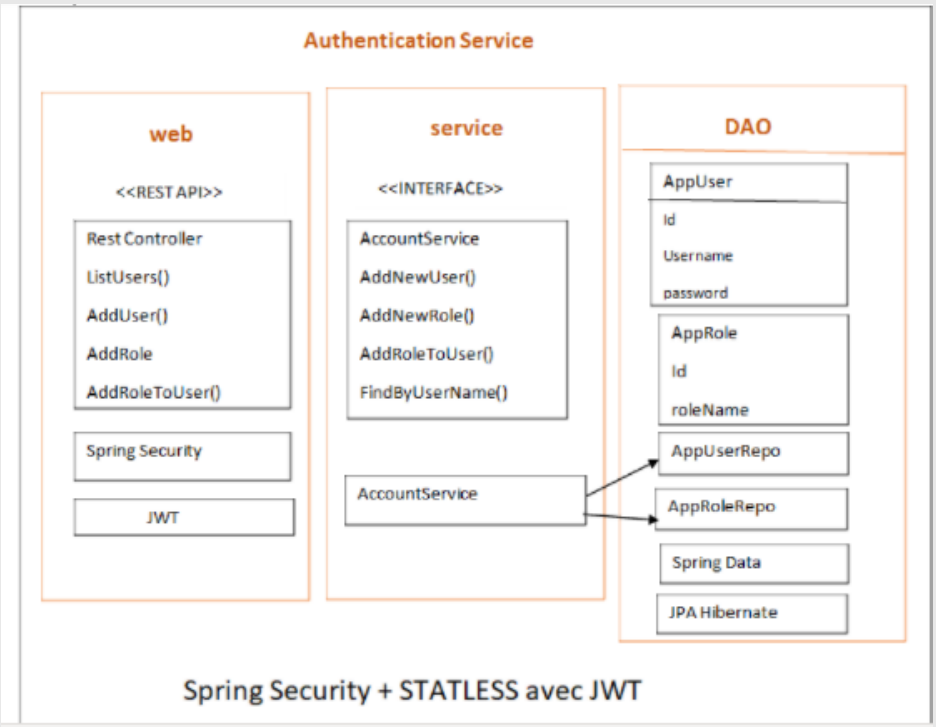
1- Diagramme de classes

Le diagramme de classes est un schéma utilisé en génie logiciel pour présenter les classes et les interfaces des systèmes ainsi que les différentes relations entre celles-ci. Ce diagramme fait partie de la partie statique d'UML car il fait abstraction des aspects temporels et dynamiques.
Les classes peuvent être liées entre elles grâce au mécanisme d'héritage qui permet de mettre en évidence des relations de parenté. D'autres relations sont possibles entre des classes, chacune de ces relations est représentée par un arc spécifique dans le diagramme de classes. Dans notre projet nous avons utilisé le diagramme de classe comme une transformation du MCD afin d'avoir une approche orienté objet

A-Micro-Services:



B-Sécurité

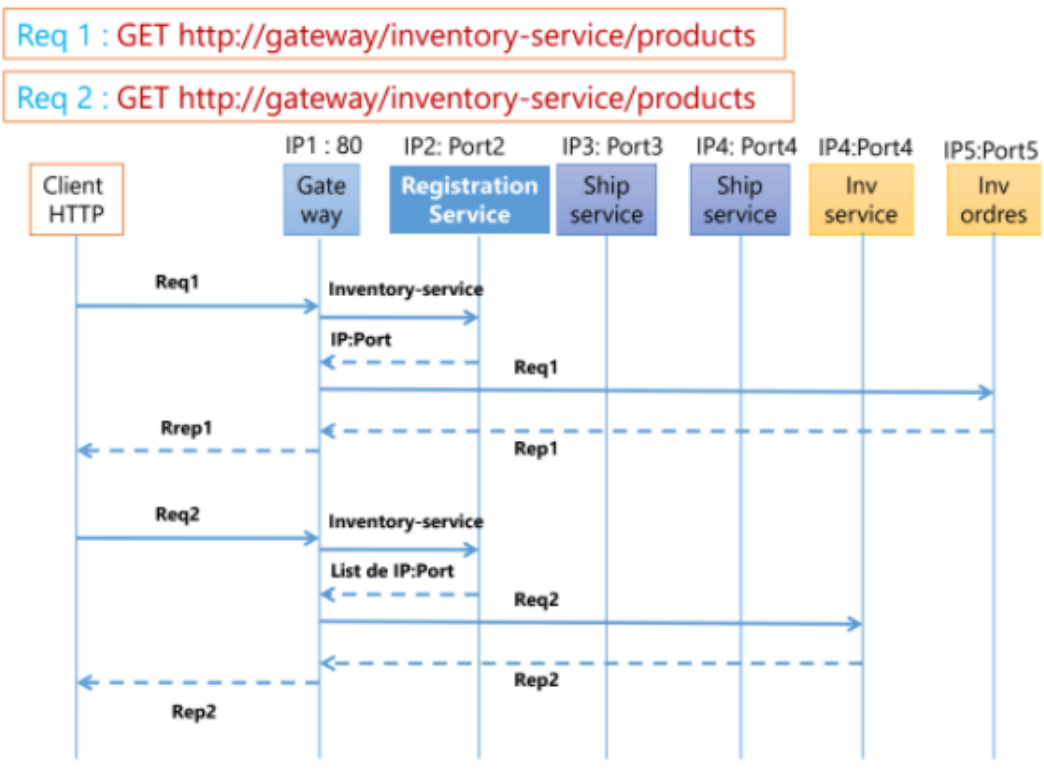


2-diagramme de séquence

Les diagrammes de séquences sont la représentation graphique des interactions entre les acteurs et le système selon un ordre chronologique dans la formulation Unified Modeling Language.

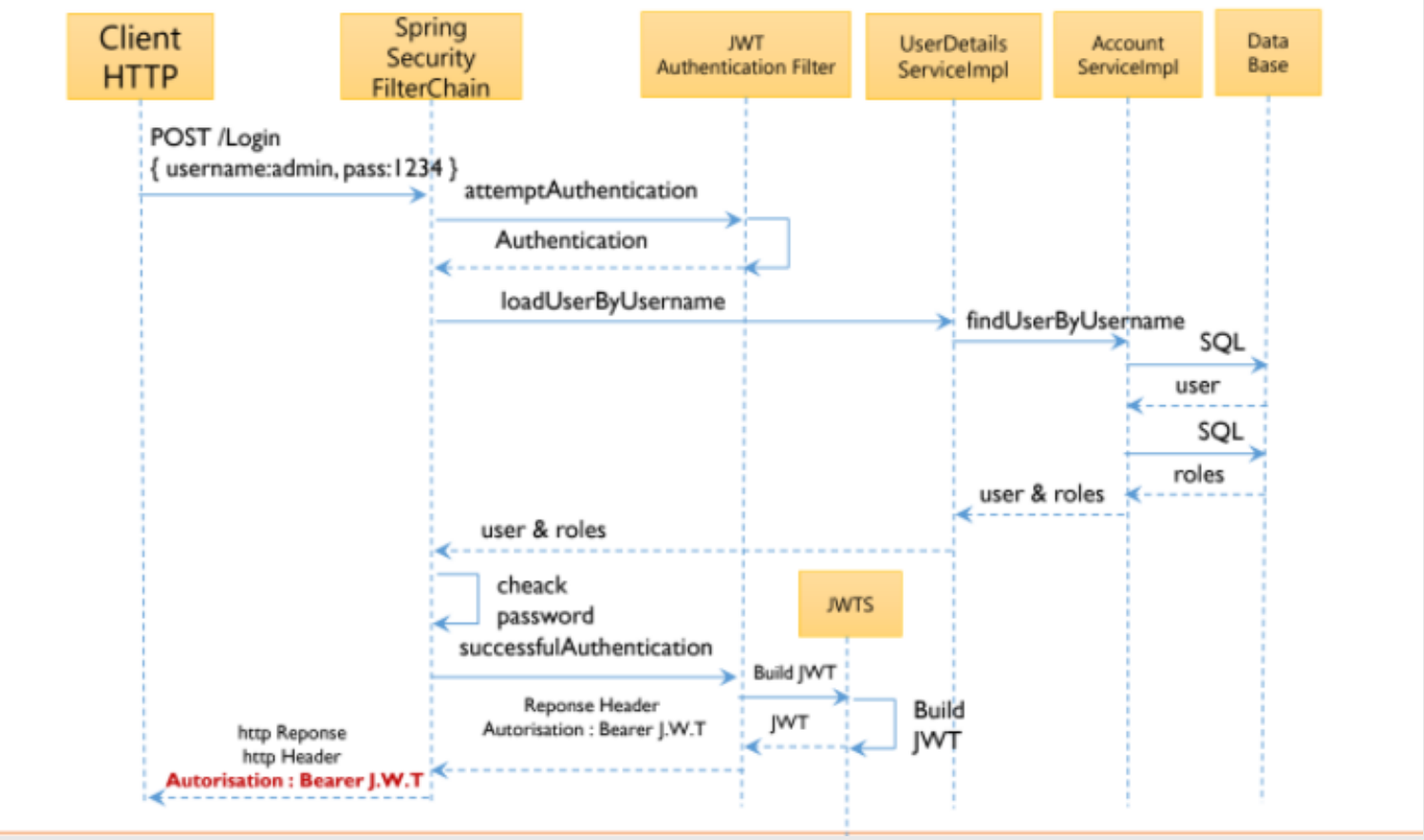
A-Micro-Services:

Consulter les services via le service proxy

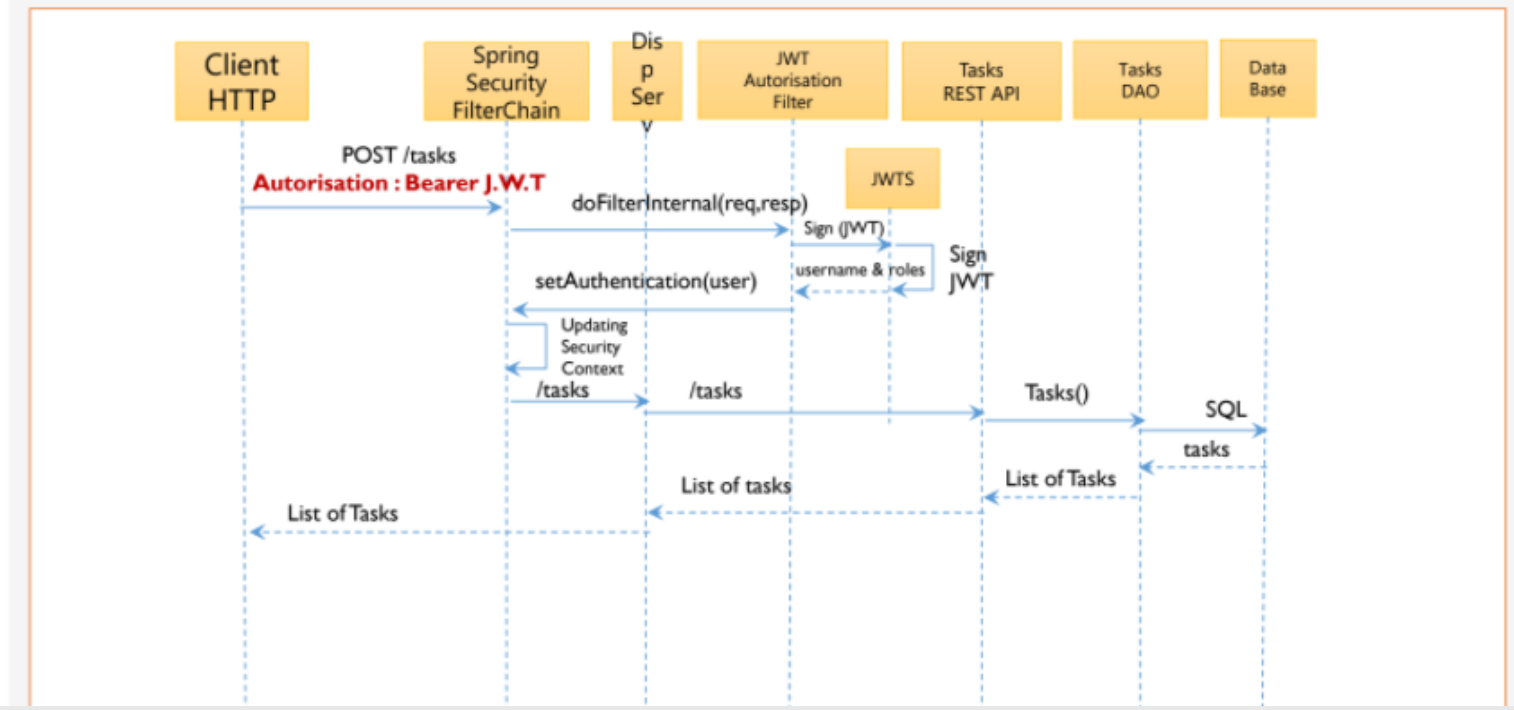


B-Sécurité

Authentification Spring Security avec JWT



Demander une ressource nécessitant l'authentification



Chapitre 3:Réalisation

1-Introduction

Après avoir effectué la conception de l'application, je vais passer à la phase d'implémentation. Cette partie présente le résultat du travail effectué durant la partie Backend de ce projet. je vais présenter aussi l'environnement matériel et les outils de développement utilisés.
Cette partie sera clôturée par quelques captures d'écran démontrent les fonctionnalités de l'application

2- Environnement de travail : Partie logiciel

a-Spring

Spring est un framework libre pour construire et définir l'infrastructure d'une application java, dont il facilite le développement et les tests.

b-angular

Angular est un framework Javascript côté client qui permet de réaliser des applications de type "Single Page Application". Il est basé sur le concept de l'architecture MVC (Model View Controller) qui permet de séparer les données, les vues et les différentes actions que l'on peut effectuer.

c-IntelliJ IDEA

IntelliJ IDEA également appelé « IntelliJ », « IDEA » ou « IDJ » est un environnement de développement intégré (en anglais Integrated Development Environment - IDE) de technologie Java destiné au développement de logiciels informatiques. Il est développé par JetBrains (anciennement « IntelliJ ») et disponible en deux versions, l'une communautaire, open source, sous licence Apache 2 et l'autre propriétaire, protégée par une licence commerciale. Tous deux supportent les langages de programmation Java, Kotlin, Groovy et Scala.

3-La partie backend

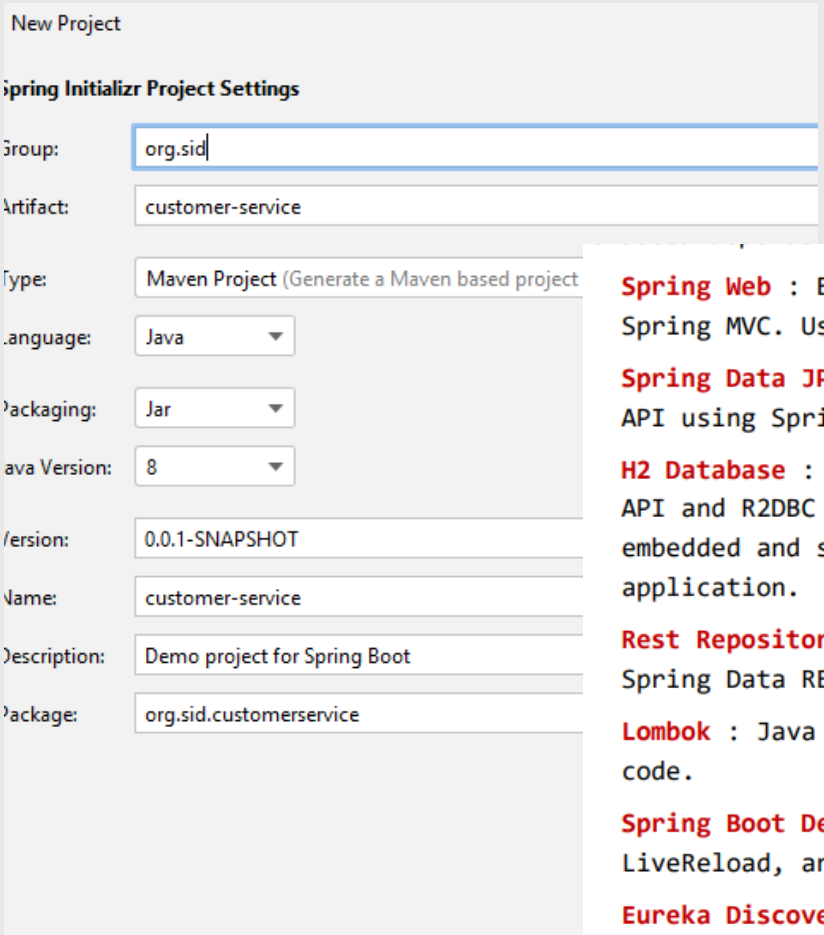
le micro service Customer service

Chaque Customer est défini par :
- Sa référence de type Long
- Sa désignation de type String
- Son email

01

CUSTOMER-SERVICE

creation de service Customer



dépendance

Spring Web : Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container

Spring Data JPA : Persist data in SQL stores with Java Persister API using Spring Data and Hibernate.

H2 Database : Provides a fast in-memory database that supports J API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.

Rest Repositories : Exposing Spring Data repositories over REST Spring Data REST.

Lombok : Java annotation library which helps to reduce boilerpla code.

Spring Boot DevTools : Provides fast application restarts, LiveReload, and configurations for enhanced development experien

Eureka Discovery Client : a REST based service for locating services for the purpose of load balancing and failover of middl tier servers.

Spring Boot Actuator : Supports built in (or custom) endpoints t let you monitor and manage your application - such as applicatio health, metrics, sessions, etc.

02

CUSTOMER-SERVICE : CUSTOMERSERVICEAPPLICATION.JAVA

class Customer

```java
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.ToString;

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

CustomerRepository

```java
package org.sid.servicecustomer.repository;

import org.sid.servicecustomer.entities.Customer;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;

@RepositoryRestResource
public interface CustomerRepository  extends JpaRepository<Customer,Long> {
}
```

class ServiceCustomerApplication

ServiceCustomerApplication.java | application.properties | pom.xml (service-customer) | Customer.java | CustomerRepository.java

```java
package org.sid.servicecustomer;

import ...

@SpringBootApplication
public class ServiceCustomerApplication {

    public static void main(String[] args) { SpringApplication.run(ServiceCustomerApplication.class, args); }
    @Bean
CommandLineRunner start(CustomerRepository customerRepository, RepositoryRestConfiguration repositoryRestConfiguration
        repositoryRestConfiguration.exposeIdsFor(Customer.class);
        return  args -> {
            customerRepository.save(new Customer( id: null, name: "sara", email: "sara@gmail.com"));
            customerRepository.save(new Customer( id: null, name: "iman", email: "iman@gmail.com"));
            customerRepository.save(new Customer( id: null, name: "inass", email: "inass@gmail.com"));
            customerRepository.findAll().forEach(c->{
                System.out.println(c.toString());
            });
        };
```

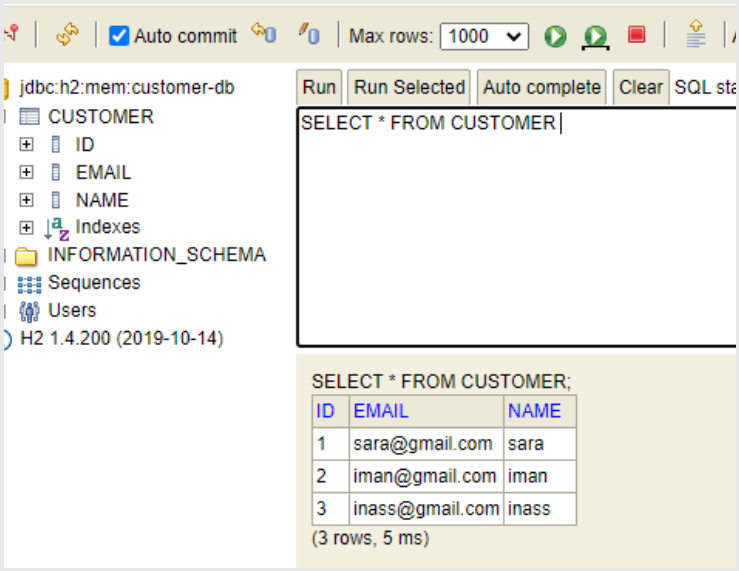
Application.properties

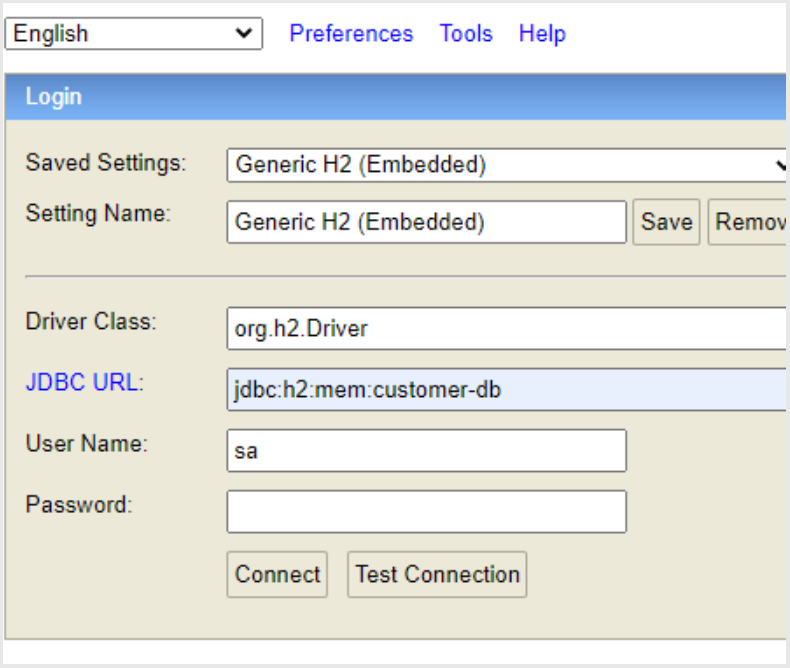
```
spring.application.name=customer-service
spring.datasource.url=jdbc:h2:mem:customer-db
server.port=8081
spring.cloud.discovery.enabled=false
#management.endpoints.web.exposure.include=*
```

03

CUSTOMER-SERVICE : BASE DE DONNÉES H2

http://localhost:8081/h2-console





04

CUSTOMER-SERVICE :

http://localhost:8081/customers


```
{
  "_embedded": {
    "customers": [
      {
        "name": "sara",
        "email": "sara@gmail.com",
        "_links": {
          "self": {
            "href": "http://localhost:8081/customers/1"
          },
          "customer": {
            "href": "http://localhost:8081/customers/1"
          }
        }
      },
      {
        "name": "iman",
        "email": "iman@gmail.com",
        "_links": {
          "self": {
            "href": "http://localhost:8081/customers/2"
          },
          "customer": {
            "href": "http://localhost:8081/customers/2"
          }
        }
```


```
2020-12-18 22:21:54.147  INFO 6768 --- [  restartedM
2020-12-18 22:21:54.346  INFO 6768 --- [  restartedM
Customer(id=1, name=sara, email=sara@gmail.com)
Customer(id=2, name=iman, email=iman@gmail.com)
Customer(id=3, name=inass, email=inass@gmail.com)
```

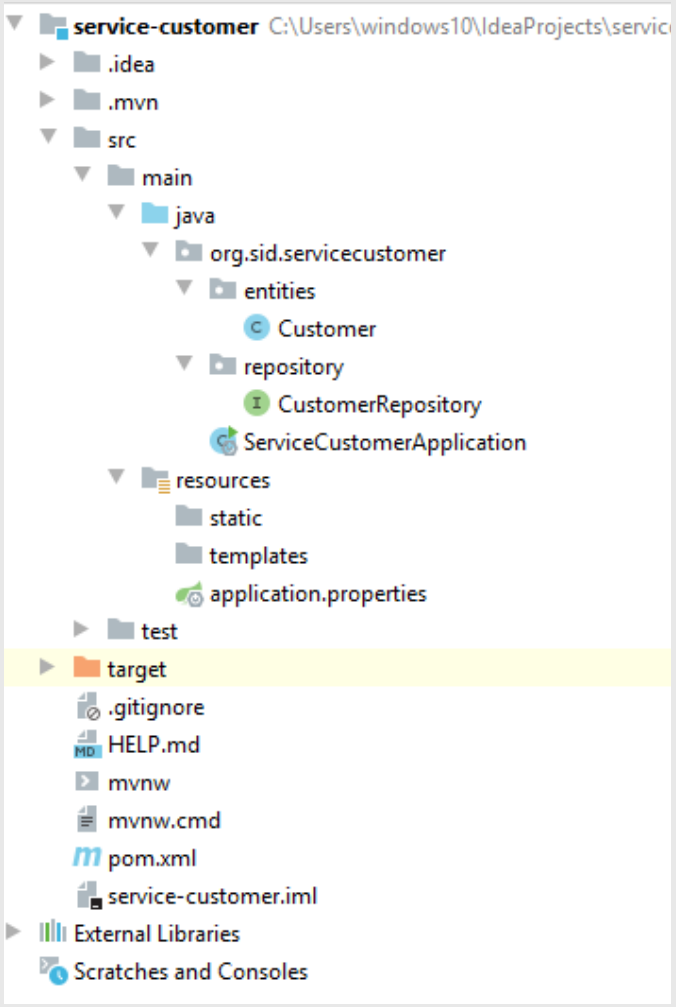
```
--------------------------

id: 1
Name: sara
Email: sara@gmail.com
--------------------------
```

http://localhost:8081/actuator


```
{
  "_links": {
    "self": {
      "href": "http://localhost:8081/actuator",
      "templated": false
    },
    "beans": {
      "href": "http://localhost:8081/actuator/beans",
      "templated": false
    },
    "caches-cache": {
      "href": "http://localhost:8081/actuator/caches/{cache}",
      "templated": true
    },
    "caches": {
      "href": "http://localhost:8081/actuator/caches",
      "templated": false
    },
    "health": {
      "href": "http://localhost:8081/actuator/health",
      "templated": false
```


Spring Booot Actuator ,créer pour collecter,superviser les informations de l'application.

architecture de service-customer



le micro service Inventory-service

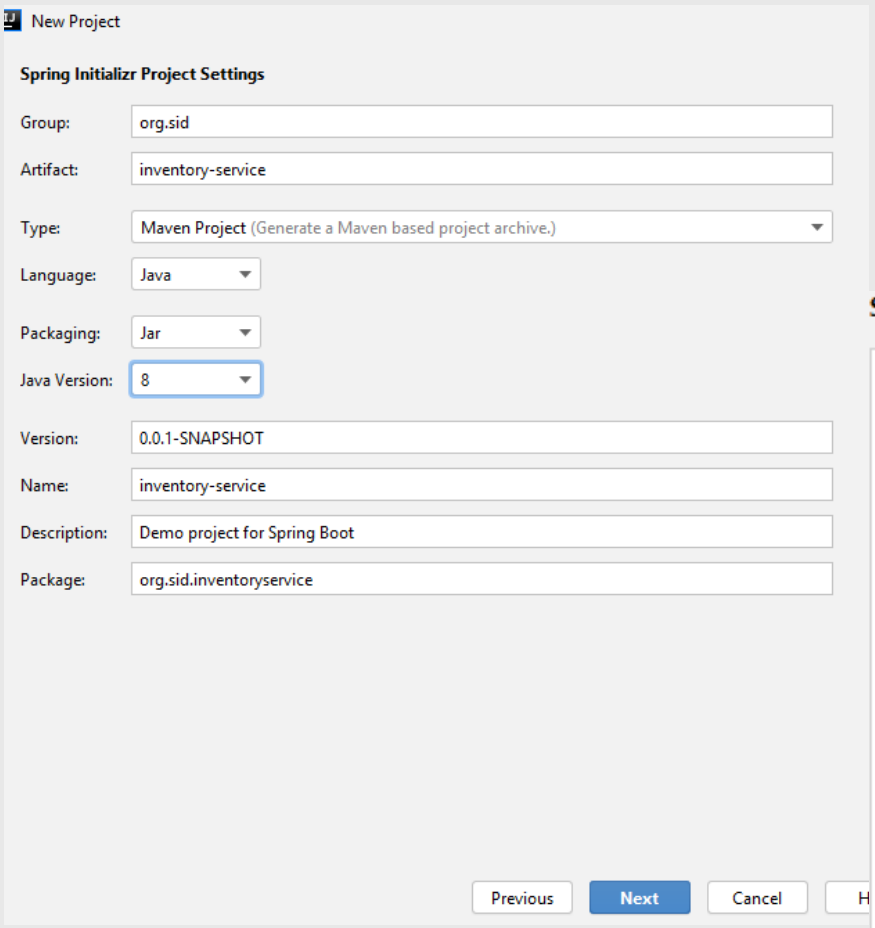
Chaque Product est défini par :

- Sa référence de type Long
- Sa désignation de type String
- Son prix
- Sa quantité

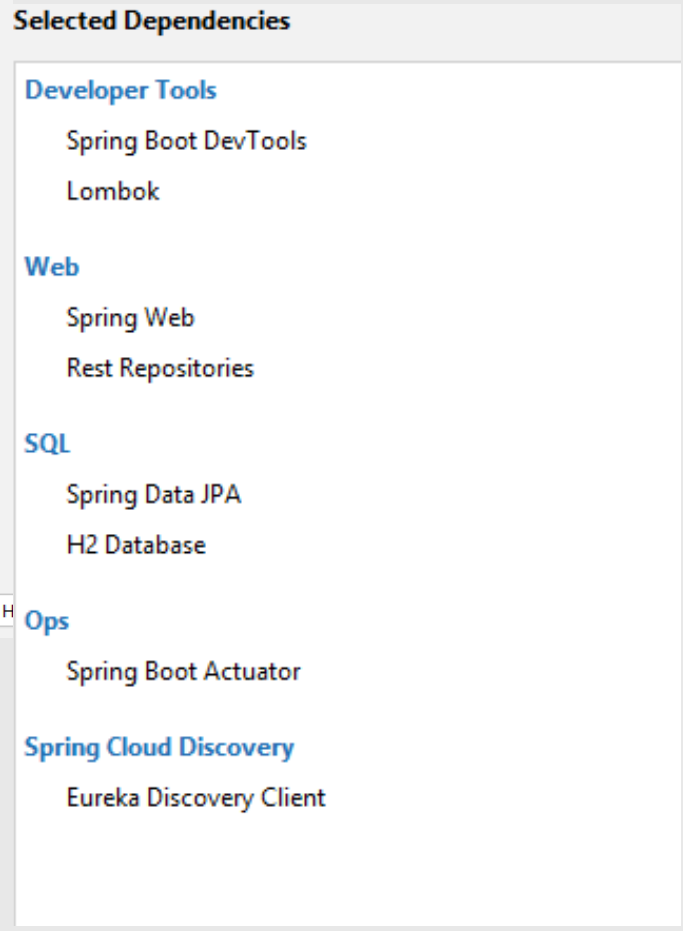
01

INVENTORY-SERVICE

creation de inventory-service



dépendance



02

INVONTORY-SERVICE : CUSTOMERSERVICEAPPLICATION.JAVA

class Product

```java
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.ToString;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
@Entity
@Data @AllArgsConstructor @NoArgsConstructor @ToString
public class Product {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private double price;
    private double quantité ;
```

ProductRepository

```java
package org.sid.inventoryservice.repository;

import org.sid.inventoryservice.entities.Product;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;

@RepositoryRestResource
public interface ProductRepository extends JpaRepository<Product,Long> {
}
```

class ServiceProductApplication

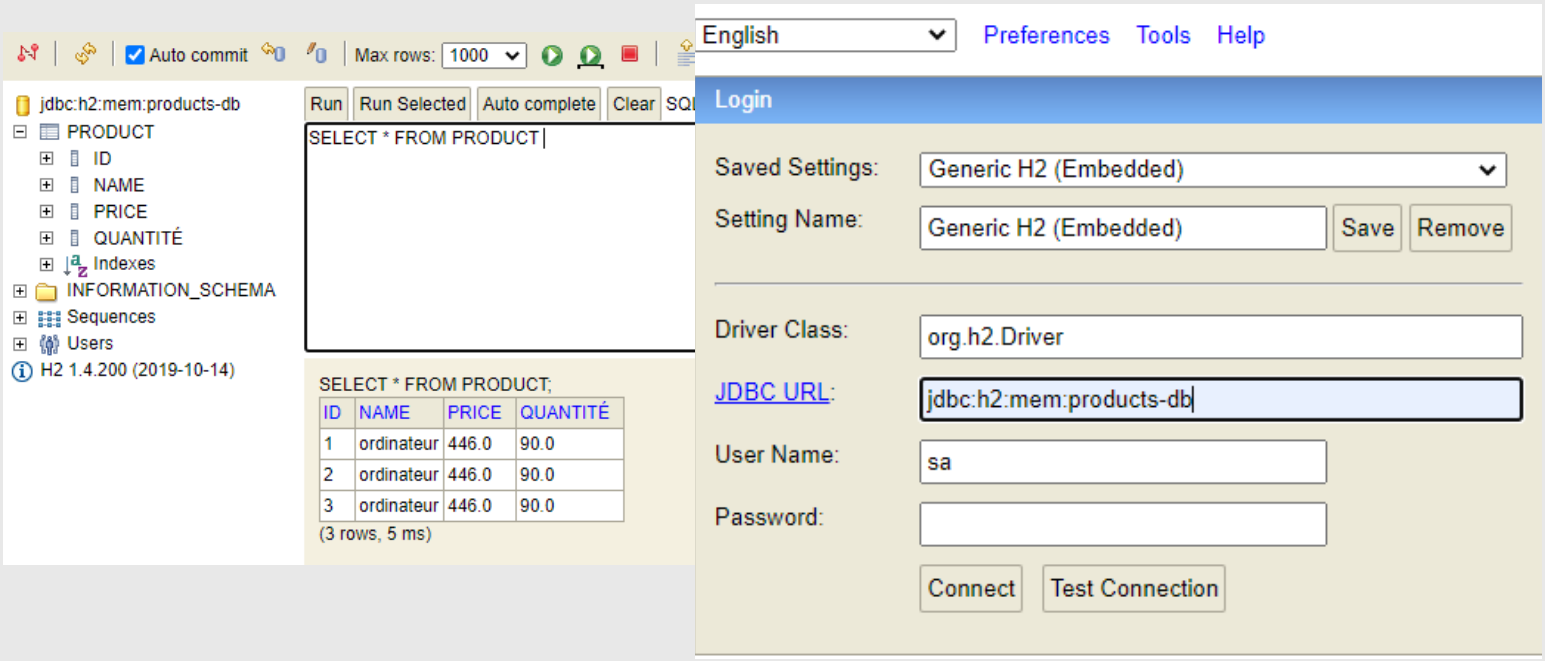
```java
package org.sid.inventoryservice;

import ...

@SpringBootApplication
public class InventoryServiceApplication {

    public static void main(String[] args) { SpringApplication.run(InventoryServiceApplication.class, args); }
    @Bean
CommandLineRunner start(ProductRepository productRepository, RepositoryRestConfiguration repositoryRestConfiguration){
        repositoryRestConfiguration.exposeIdsFor(Product.class);
        return args -> {
            productRepository.save(new Product( id: null, name: "ordinateur", price: 466, quantité: 90));
            productRepository.save(new Product( id: null, name: "phone", price: 446, quantité: 98));
            productRepository.save(new Product( id: null, name: "imprimante", price: 400, quantité: 9));
      productRepository.findAll().forEach(c->{
                System.out.println(c.toString());
          });


        };
}
}
```

Application.properties

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

03 INVONTORY-SERVICE : BASE DE DONNÉES H2

http://localhost:8082/h2-console



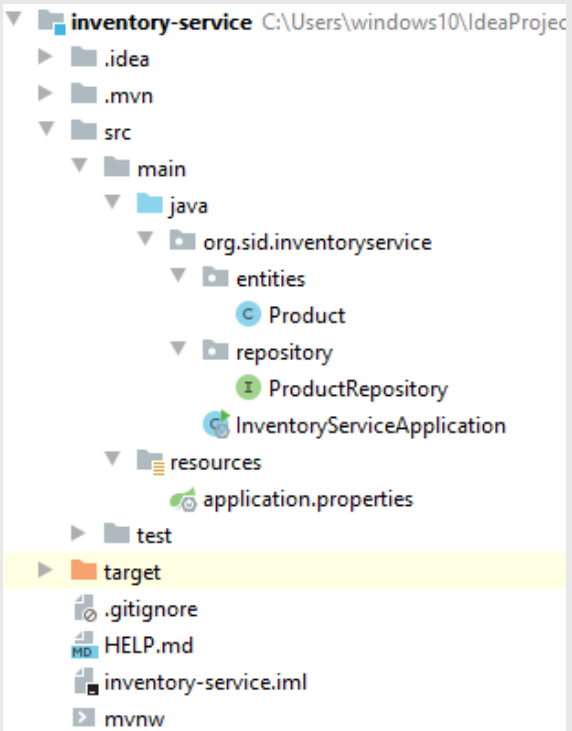
04 INVENTORY-SERVICE :

```
Product(id=1, name=ordinateur, price=466.0, quantité=90.0)
Product(id=2, name=phone, price=446.0, quantité=98.0)
Product(id=3, name=imprimante, price=400.0, quantité=9.0)
```

http://localhost:8082/Product


```
{
  "_embedded": {
    "products": [
      {
        "name": "ordinateur",
        "price": 446.0,
        "quantité": 90.0,
        "_links": {
          "self": {
            "href": "http://localhost:8082/products/1"
          },
          "product": {
            "href": "http://localhost:8082/products/1"
          }
        }
      },
      {
        "name": "ordinateur",
        "price": 446.0,
        "quantité": 90.0,
        "_links": {
          "self": {
            "href": "http://localhost:8082/products/2"
          },
          "product": {
            "href": "http://localhost:8082/products/2"
          }
```

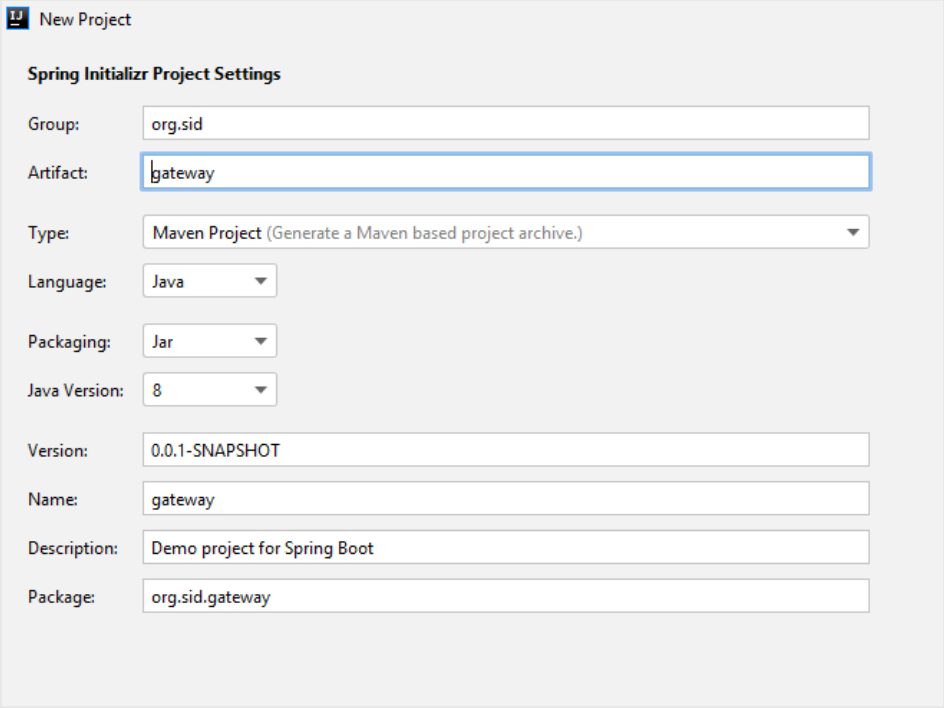

architecture de inventory-service



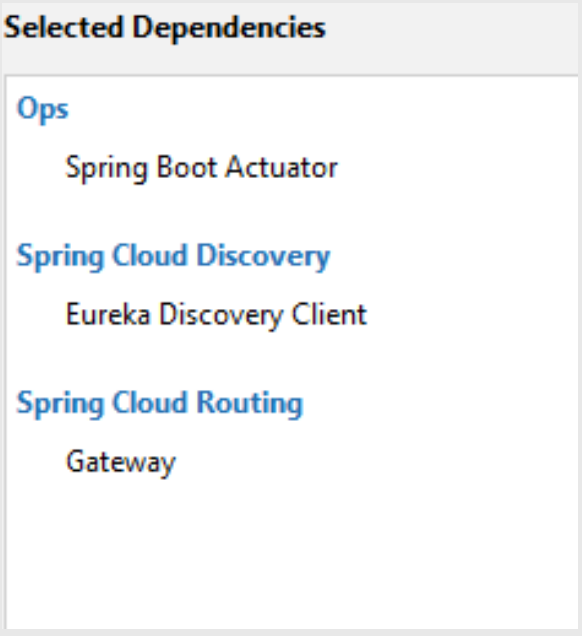
01

GATEWAY-SERVICE

creation de gateway-service



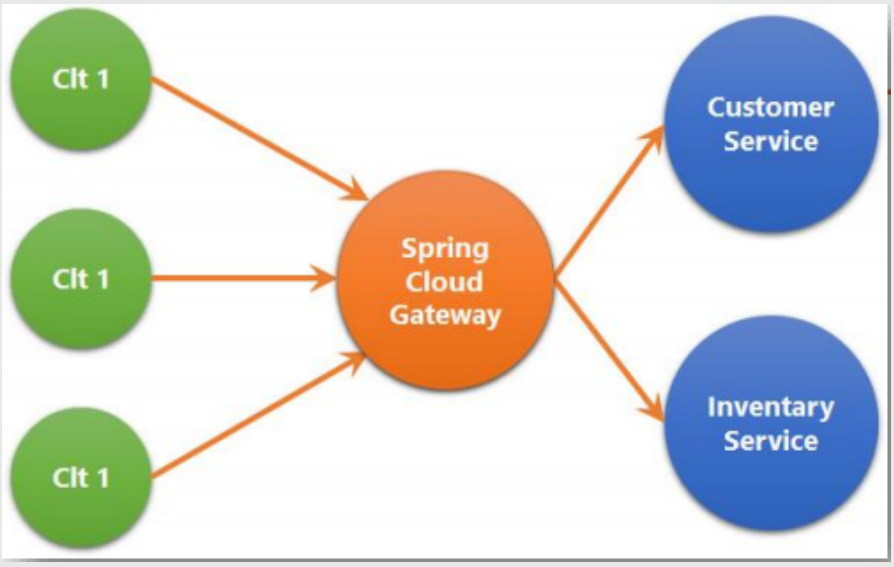
dépendance



Selected dependencies

- **Gateway** : Provides a simple, yet effective way to route to APIs and provide cross cutting concerns to them such as security, monitoring/metrics, and resiliency.
- **Spring Boot Actuator** : Supports built in (or custom) endpoints that let you monitor and manage your application - such as application health, metrics, sessions, etc.
- **Hystrix** : Circuit breaker with Spring Cloud Netflix Hystrix.
- **Eureka Discovery Client** : a REST based service for locating services for the purpose of load balancing and failover of middle-tier servers.

Spring Cloud Gateway Service comme service proxy



- configuration:application.yml

GatewayApplication.java | application.properties | application.yml

```yaml
spring:
  cloud:
    gateway:
      routes:
        - id : r1
          uri : http://localhost:8081/
          predicates:
            - Path= /customers/**

        - id: r2
          uri: http://localhost:8082/
          predicates:
            - Path= /products/**
```

- http://localhost:8888/customers

localhost:8888/customers

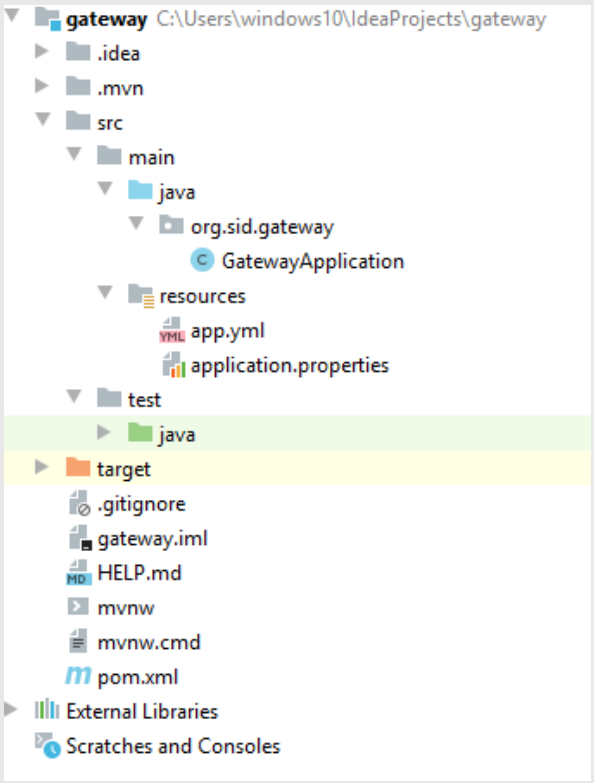
```
// 20201219002909
// http://localhost:8888/customers

{
  "_embedded": {
    "customers": [
      {
        "name": "sara",
        "email": "sara@gmail.com",
        "_links": {
          "self": {
            "href": "http://localhost:8081/customers/1"
          },
          "customer": {
            "href": "http://localhost:8081/customers/1"
          }
        }
      },
      {
        "name": "iman",
        "email": "iman@gmail.com",
        "_links": {
          "self": {
            "href": "http://localhost:8081/customers/2"
          },
          "customer": {
            "href": "http://localhost:8081/customers/2"
          }
```

Static routes configuration: Java Config Class

```
@Bean
RouteLocator gatewayRoutes(RouteLocatorBuilder builder){
        return builder.routes()
                        .route(r->r.path("/customers/**").uri("http://localhost:8081/").id("r1"))
                        .route(r->r.path("/products/**").uri("http://localhost:8082/").id("r2"))
                        .build();
}
```

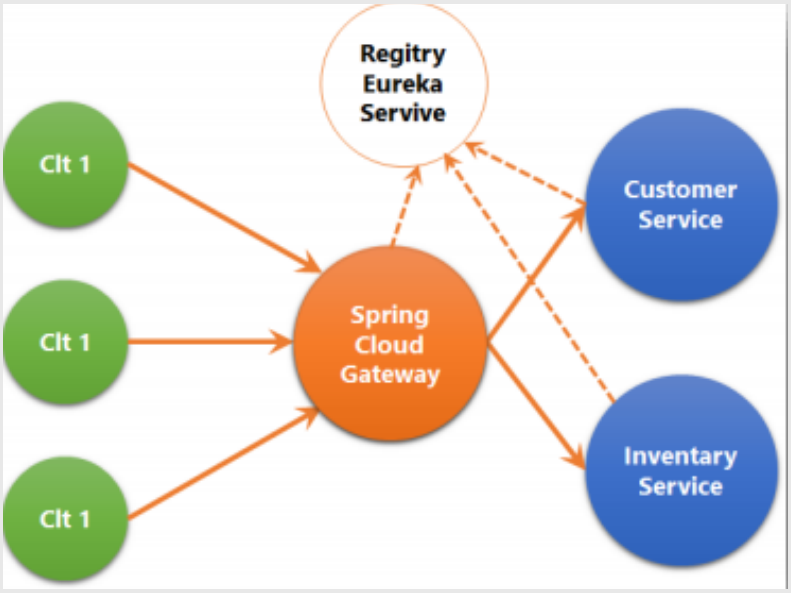
http://localhost:8888/customers/1


```
{
  "name": "sara",
  "email": "sara@gmail.com",
  "_links": {
    "self": {
      "href": "http://localhost:8081/customers/1"
    },
    "customer": {
      "href": "http://localhost:8081/customers/1"
    }
  }
}
```

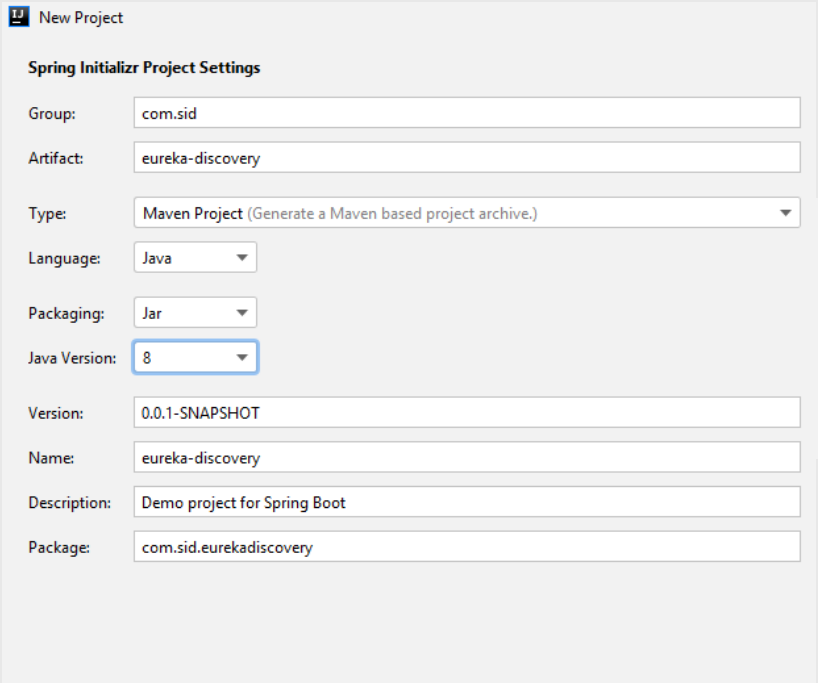



architecture de gateway-service

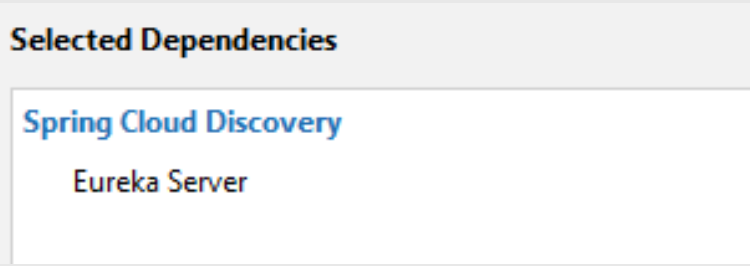
Eureka Discovery Service : Dynamic Routing



creation de eureka-discovery

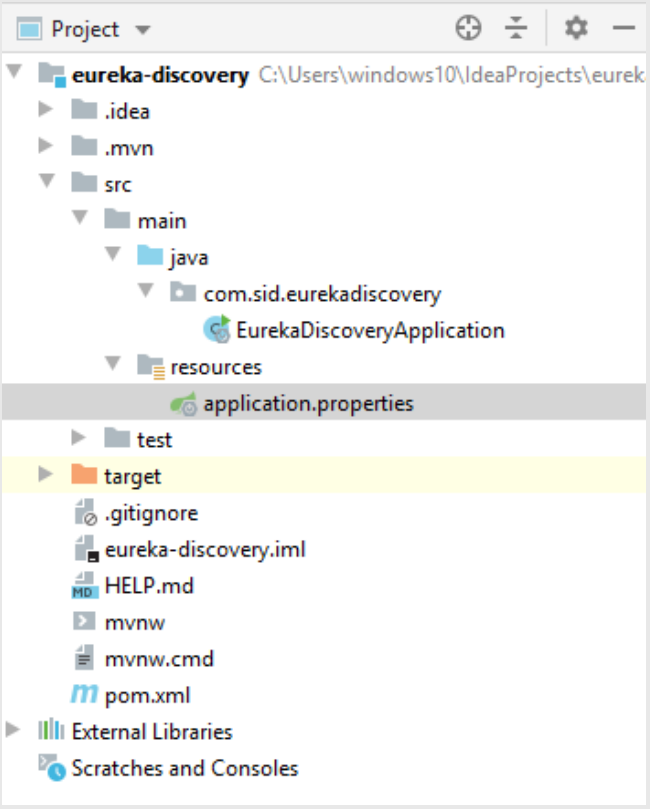


dépendance




```
package com.sid.eurekadiscovery;

import ...

@SpringBootApplication
@EnableEurekaServer
public class EurekaDiscoveryApplication {

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

}
```

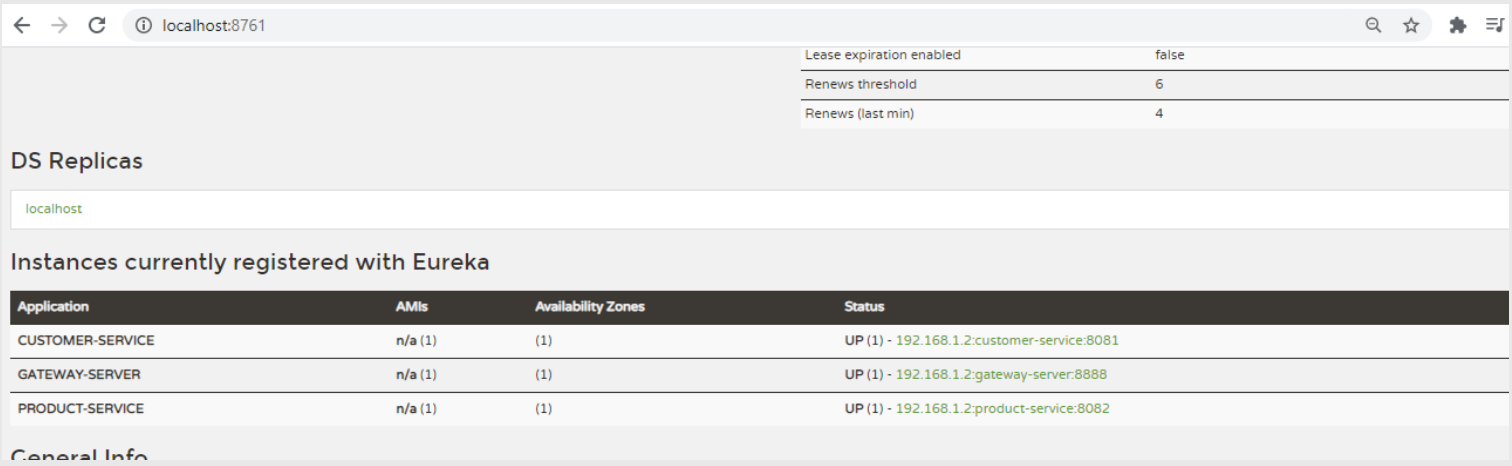
architecture de eureka-service



Application.properties

```
server.port=8761
eureka.client.fetch-registry=false
eureka.client.register-with-eureka=false
```

Permettre à Customer-service et Invotory-service de s'enregistrer chez Eureka server



Customer-service

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

Inventory-service

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

Static routes configuration with Discovery Service

```
@Bean
RouteLocator gatewayRoutes(RouteLocatorBuilder builder){
        return builder.routes()
                        .route(r->r.path("/customers/**").uri("lb://CUSTOMER-SERVICE") .id("r1"))
                        .route(r->r.path("/products/**").uri("lb://INVENTORY-SERVICE") .id("r2"))
                        .build();
}
```

Dynamic routes configuration with Discovery Service

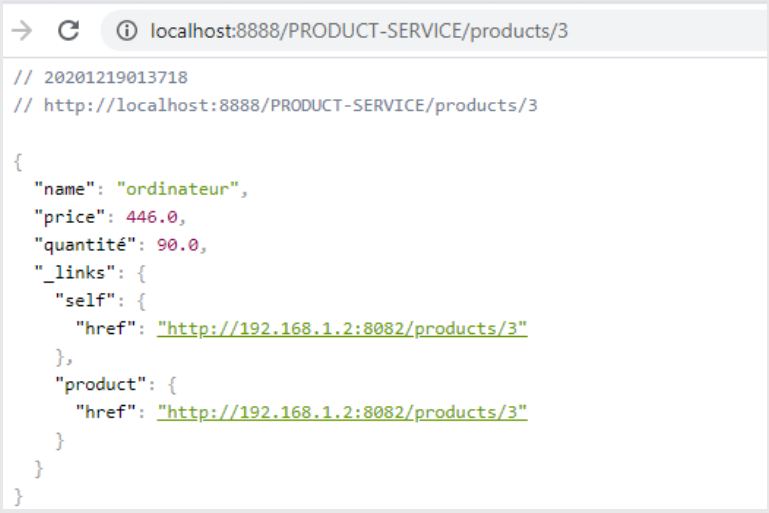
application.properties

```
server.port=8888
spring.application.name=gateway-server
spring.cloud.discovery.enabled=true
```

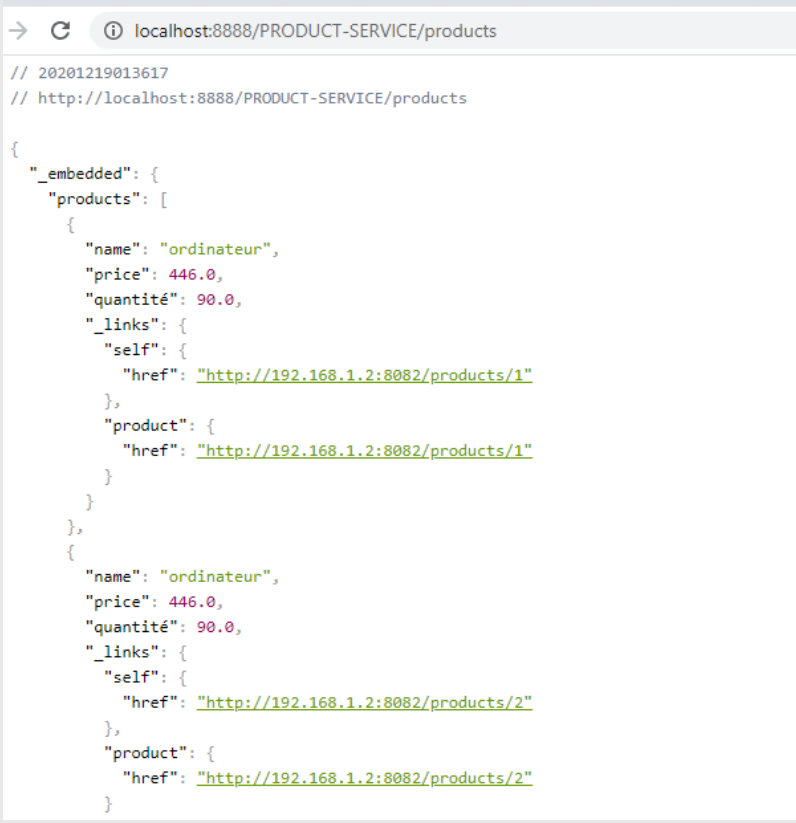
GatewayApplication

```
@Bean
    DiscoveryClientRouteDefinitionLocator definitionLocator(ReactiveDiscoveryClient rdc, DiscoveryLocatorProperties properties){
        return new DiscoveryClientRouteDefinitionLocator(rdc,properties);
```

http://localhost:8888/PRODUCT-SERVICE/products/3



http://localhost:8888/PRODUCT-SERVICE/products



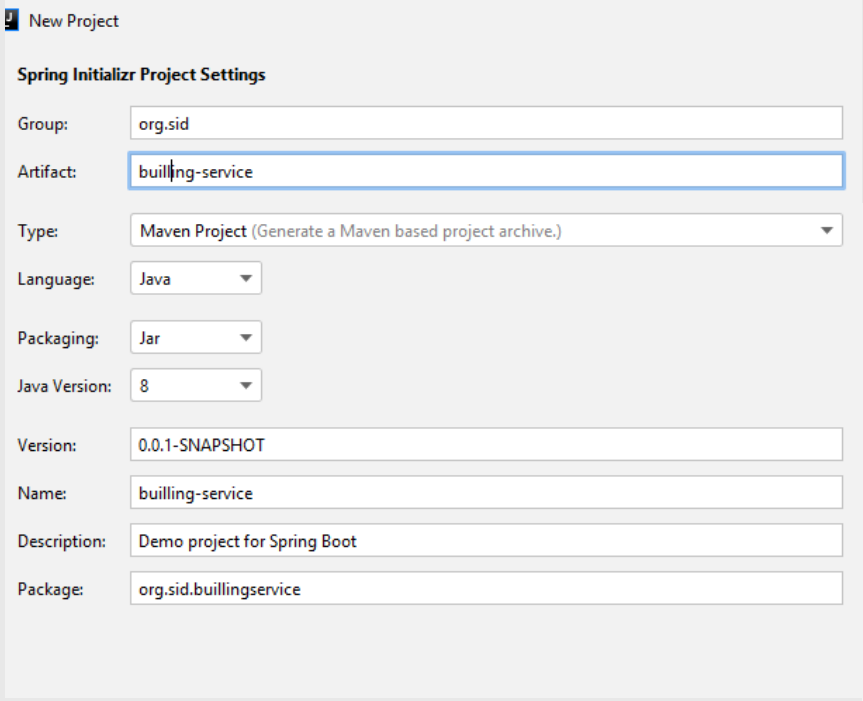
01

BILLING-SERVICE

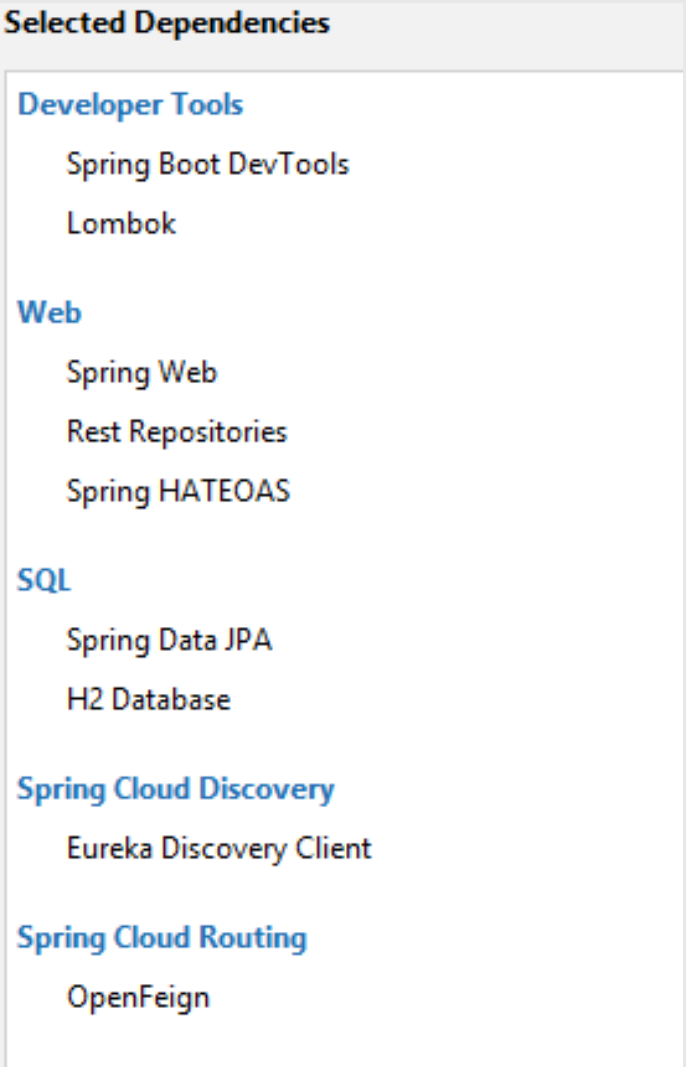
Ajouter un service de facturation (Billing Service), qui communique avec les services Clients et Inventaire en utilisant Spring cloud OpenFeign Rest Client

Billing-service

architecture de eureka-service



dépendance



Selected dependencies

- **Spring Web** : Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container
- **Spring Data JPA** : Persist data in SQL stores with Java Persisten API using Spring Data and Hibernate.
- **H2 Database** : Provides a fast in-memory database that supports J API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.
- **Rest Repositories** : Exposing Spring Data repositories over REST Spring Data REST.
- **Lombok** : Java annotation library which helps to reduce boilerpla code.
- **Spring Boot DevTools** : Provides fast application restarts, LiveReload, and configurations for enhanced development experience.
- **Eureka Discovery Client** : a REST based service for locating services for the purpose of load balancing and failover of middle-tier servers.
- **OpenFeign** : Declarative REST Client. OpenFeign creates a dynamic implementation of an interface decorated with JAX-RS or Spring MVC annotations.
- **Spring HATEOAS** : Eases the creation of RESTful APIs that follow the HATEOAS principle when working with Spring / Spring MVC.

```java
package org.sid.buillingservice;
import org.sid.buillingservice.Feign.CustomerRestClient;
import org.sid.buillingservice.Feign.ProductItemRestClient;
import org.sid.buillingservice.entities.Bill;
import org.sid.buillingservice.entities.ProductItem;
import org.sid.buillingservice.model.Customer;
import org.sid.buillingservice.model.Product;
import org.sid.buillingservice.repository.BillRepositery;
import org.sid.buillingservice.repository.ProductItemRepositery;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.openfeign.EnableFeignClients;
import org.springframework.context.annotation.Bean;
import org.springframework.hateoas.PagedModel;
import java.util.Collection;
import java.util.Date;
import java.util.Random;
```

class Bill

```java
package org.sid.buillingservice.entities;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.sid.buillingservice.model.Customer;

import javax.persistence.*;
import java.util.Collection;
import java.util.Date;
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class Bill {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private  Long id;
    private Date billingDate;
    @OneToMany(mappedBy ="bill")
    private Collection<ProductItem> productItems;
    private  Long customerID;
    @Transient
    private Customer customer;
```

class ProductItem

```java
package org.sid.buillingservice.entities;
import com.fasterxml.jackson.annotation.JsonProperty;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import org.sid.buillingservice.model.Product;

import javax.persistence.*;

@Entity
@Data  @NoArgsConstructor @AllArgsConstructor
public class ProductItem {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private double quantity;
    private double  price;
    private  long productID;
    @JsonProperty(access =  JsonProperty.Access.WRITE_ONLY)
    @ManyToOne
    private  Bill bill;
    @Transient
    private Product product;
}
```

class Customer

```java
package org.sid.buillingservice.model;

import lombok.Data;

@Data
public class Customer {
    private Long Id;
    private  String name;
    private String email;
}
```

class Product

```java
package org.sid.buillingservice.model;

import lombok.Data;

@Data
public class Product {
    private  Long id;
    private String name;
    private  double price;
    private  double quantity;
}
```

interface ProductItemRestClient

```java
package org.sid.buillingservice.Feign;

import ...

@FeignClient(name ="PRODUCT-SERVICE")
public interface ProductItemRestClient {
    @GetMapping(path = "/products")
    PagedModel<Product> pageProducts();
    @GetMapping(path = "/products/{id}")
    Product getProductById(@PathVariable( "id") Long id);
}
```

interface CustomerRestClient

```java
package org.sid.buillingservice.Feign;

import org.sid.buillingservice.model.Customer;
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;

@FeignClient(name ="CUSTOMER-SERVICE")
public interface CustomerRestClient {
    @GetMapping(path="/customers/{id}")
     Customer getCustomerById(@PathVariable("id") Long id);

}
```

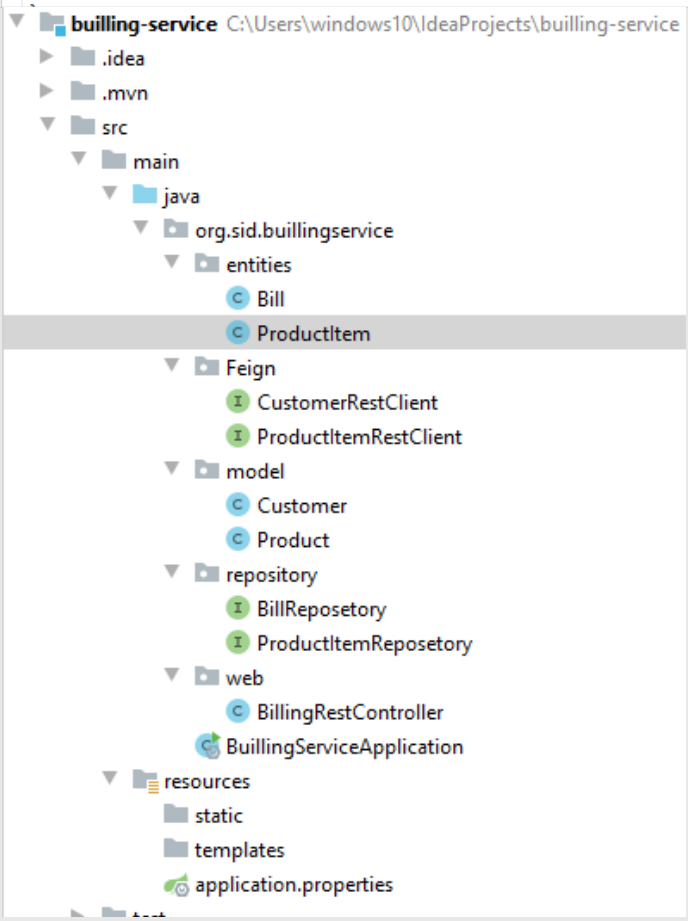
interface BillRepositoty

```java
package org.sid.buillingservice.repository;

import org.sid.buillingservice.entities.Bill;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;

@RepositoryRestResource
public interface BillRepositoty  extends JpaRepository<Bill,Long> {
}
```

interface BillRepository

```java
package org.sid.buillingservice.repository;

import org.sid.buillingservice.entities.Bill;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;

@RepositoryRestResource
public interface BillRepository  extends JpaRepository<Bill,Long> {
}
```

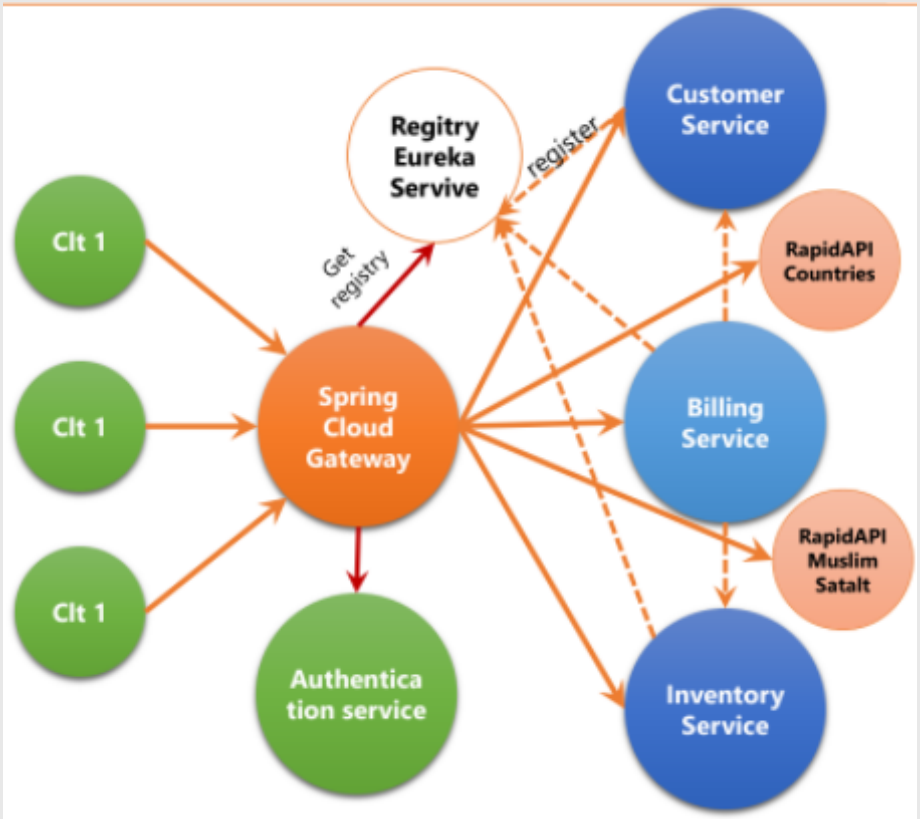
```java
@RestController
public class BillingRestController {
    private BillRepository billRepository;
    private ProductItemRepository productItemRepository;
    private CustomerRestClient customerRestClient;
    private ProductItemRestClient productItemRestClient;

 public BillingRestController(BillRepository billRepository, ProductItemRepository 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.setCustomer(customer);
        bill.getProductItems().forEach(pi ->{
            Product product=productItemRestClient.getProductById(pi.getProductID());
            pi.setProduct(product);
        } );

        return  bill;
```



architecture de eureka-service

01

SYSTÈMES D'AUTHENTIFICATION



Couche DAO

creation de sec -service

dépendance

New Project

Spring Initializr Project Settings

Group:	org.sid
Artifact:	sec-service
Type:	Maven Project (Generate a Maven based project archive.)
Language:	Java
Packaging:	Jar
Java Version:	8
Version:	0.0.1-SNAPSHOT
Name:	sec-service
Description:	Demo project for Spring Boot
Package:	org.sid.secservice

Developer Tools

- Spring Boot DevTools
- Lombok

Web

- Spring Web
- Rest Repositories

Security

- Spring Security

SQL

- Spring Data JPA
- H2 Database

Class AppRole

```java
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@Entity
@Data
@NoArgsConstructor @AllArgsConstructor

public class AppRole {
  @Id @GeneratedValue(strategy = GenerationTy
  private  Long id;
  private  String roleName;

}
```

ClassUser

```java
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import javax.persistence.*;
import java.util.ArrayList;
import java.util.Collection;

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

interface AppRoleRepository

```java
package org.sid.secservice.sec.repo;

import org.sid.secservice.sec.entities.AppRole;
import org.springframework.data.jpa.repository.JpaRepository;

public interface AppRoleRepository extends JpaRepository<AppRole,Long> {
    AppRole findByRoleName(String roleName);

}
```

interface AppUserRepository

```java
package org.sid.secservice.sec.repo;

import org.sid.secservice.sec.entities.AppUser;
import org.springframework.data.jpa.repository.JpaRepository;

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

Couche Service

interface AccountService

```java
package org.sid.secservice.sec.sec.service;

import org.sid.secservice.sec.entities.AppRole;
import org.sid.secservice.sec.entities.AppUser;

import java.util.List;

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

class AccountServiceImpl

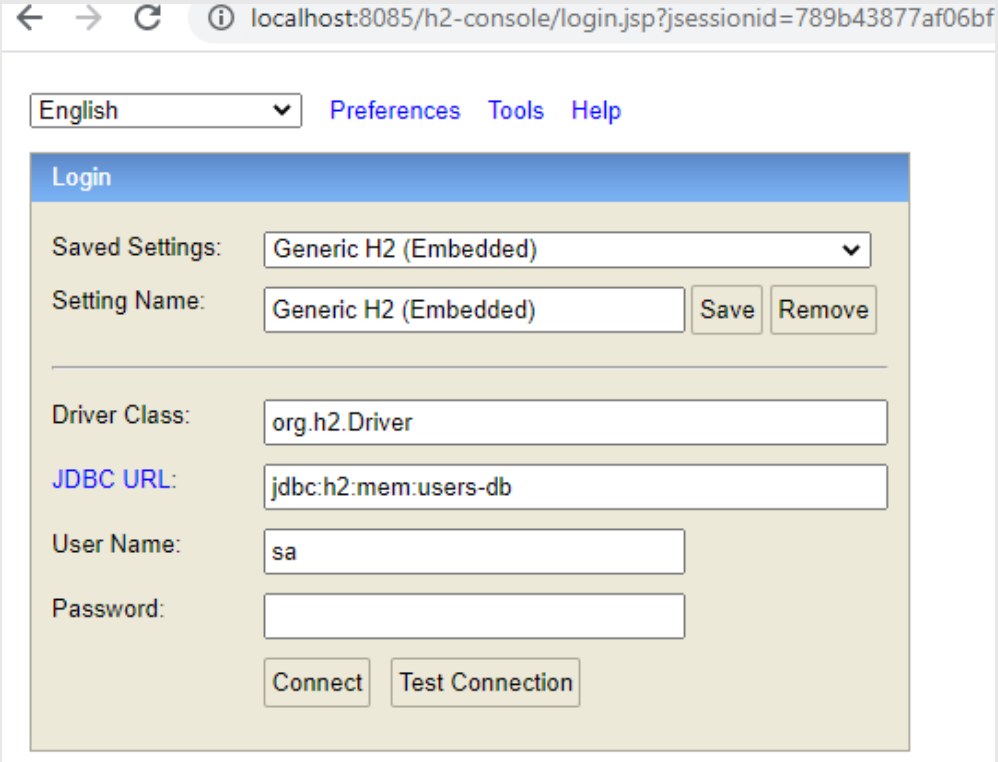
```java
@Service
@Transactional
public class AccountServiceImpl implements AccountService {
    private AppUserRepository appUserRepository;
    private AppRoleRepository appRoleRepository;

    public AccountServiceImpl(AppUserRepository appUserRepository, AppRoleRepository appRoleRepository) {
        this.appUserRepository = appUserRepository;
        this.appRoleRepository = appRoleRepository;
    }

    @Override
    public AppUser addNewUser(AppUser appUser) { 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);
    }

    @Override
    public List<AppUser> listUsers() {

        return appUserRepository.findAll();
    }
```

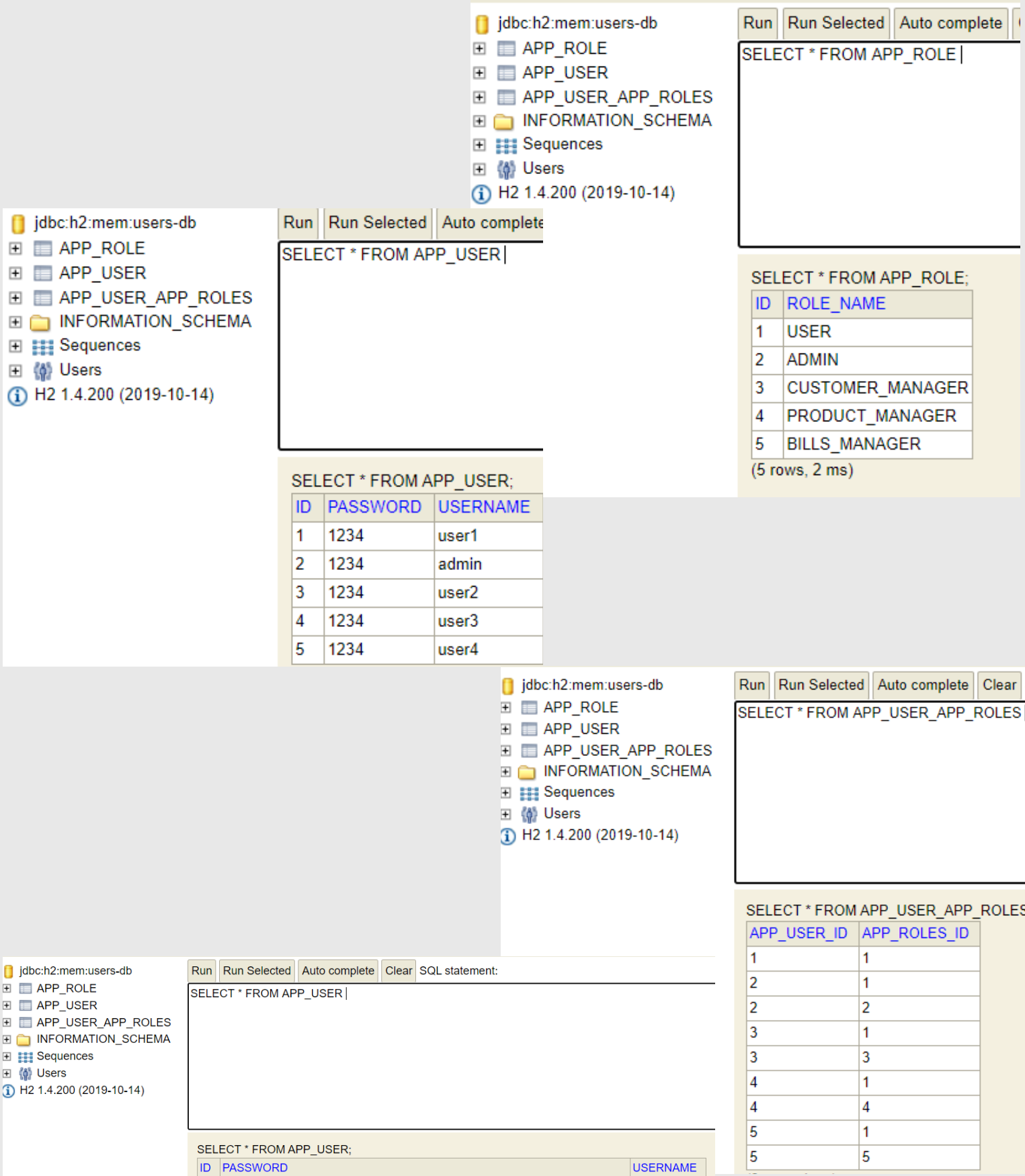
SecServiceApplication

```java
public class SecServiceApplication {
    public static void main(String[] args) { SpringApplication.run(SecServiceApplication.class, args); }
@Bean
    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: "admin", password: "1234",new ArrayList<>()));
            accountService.addNewUser(new AppUser( id: null, username: "user2", password: "1234",new ArrayList<>()));
            accountService.addNewUser(new AppUser( id: null, username: "user3", password: "1234",new ArrayList<>()));
            accountService.addNewUser(new AppUser( id: null, username: "user4", password: "1234",new ArrayList<>()));

            accountService.addRoleToUser( username: "user1", roleName: "USER");
            accountService.addRoleToUser( username: "admin", roleName: "USER");
            accountService.addRoleToUser( username: "user2", roleName: "USER");
            accountService.addRoleToUser( username: "user3", roleName: "USER");
            accountService.addRoleToUser( username: "user4", roleName: "USER");
            accountService.addRoleToUser( username: "admin", roleName: "ADMIN");
            accountService.addRoleToUser( username: "user2", roleName: "CUSTOMER_MANAGER");
            accountService.addRoleToUser( username: "user3", roleName: "PRODUCT_MANAGER");
            accountService.addRoleToUser( username: "user4", roleName: "BILLS_MANAGER");
        };
```

02

SEC-SERVICE : BASE DE DONNÉES H2




jdbc:h2:mem:users-db
APP_ROLE
APP_USER
APP_USER_APP_ROLES
INFORMATION_SCHEMA
Sequences
Users
H2 1.4.200 (2019-10-14)
Run
Run Selected
Auto complete
SELECT * FROM APP_ROLE
SELECT * FROM APP_ROLE;
ID ROLE_NAME
1 USER
2 ADMIN
3 CUSTOMER_MANAGER
4 PRODUCT_MANAGER
5 BILLS_MANAGER
(5 rows, 2 ms)
SELECT * FROM APP_USER
SELECT * FROM APP_USER;
ID PASSWORD USERNAME
1 1234 user1
2 1234 admin
3 1234 user2
4 1234 user3
5 1234 user4
Clear
SELECT * FROM APP_USER_APP_ROLES
SELECT * FROM APP_USER_APP_ROLES;
APP_USER_ID APP_ROLES_ID
1 1
2 1
2 2
3 1
3 3
4 1
4 4
5 1
5 5
SQL statement:
SELECT * FROM APP_USER;
ID PASSWORD USERNAME
1 $2a$10$1YsqymABXC.IWLfZTXtLWOQhyj3.4AR.NJ2S53ESpyJEM6kAFqZQi user1
2 $2a$10$.CAyqp61L13COPVM84TmKOpULuWQ.nsS9A6mpwd.5ZfZ5lpkOHkdO admin
3 $2a$10$njPJNvDF1BrH.Ae0EakI7efSSIOD91I..KR4ugy9hBMZnj/AizFDS user2
4 $2a$10$7rqUzvsn9wa1oQFc5W6nSeCn6GgVKnroCqM9Lrtj4FUE208JRHUlq user3
5 $2a$10$0VQ8mPUAMjoQrmjY3zM4xuTCRVQNHM31C4ldZfhkbfIbv8jdcGxTK user4
(5 rows, 5 ms)

02 COUCHE WEB : REST API POUR LA GESTION DES UTILISATEURS ET LES GROUPES

```java
@RestController
public class AccountRestController {
    private  AccountService accountService;
    public AccountRestController(AccountService accountService){
        this.accountService= accountService; }
    @GetMapping(path = "/users")
    public List<AppUser> appUsers(){
        return  accountService.listUsers(); }
        @PostMapping (path = "/users")
    public AppUser saveUser(@RequestBody AppUser appUser) { return accountService.addNewUser(appUser); }
    @PostMapping(path = "/roles")
    public AppRole saveRole(@RequestBody AppRole appRole) { return accountService.addNewRole(appRole); }
    @PostMapping(path = "/addRoleToUser")
    public void addRoleToUser(@RequestBody RoleUserForm roleUserForm) {
        accountService.addRoleToUser(roleUserForm.getUsername(), roleUserForm.getRoleName());
```

```java
@Data
class RoleUserForm{

    private String username;
    private String roleName;
}
```

03 COUCHE WEB : ENDPOINT DE REFRESH TOKEN

```java
@GetMapping(path ="/refreshToke")
public  void  refreshToken(HttpServletRequest request, HttpServletResponse response) throws Exception{
    String auhToken=request.getHeader( s: "Authorization");
    if (auhToken!=null && auhToken.startsWith("Bearer")) {
        try {
            String jwt = auhToken.substring(7);
            Algorithm algorithm = Algorithm.HMAC256("mySecret1234");
            JWTVerifier jwtVerifier = JWT.require(algorithm).build();
            DecodedJWT decodedJWT = jwtVerifier.verify(jwt);
            String username = decodedJWT.getSubject();
            AppUser appUser=accountService.loadUserByUsername(username);

            String jwtAccessToken = JWT.create()
                    .withSubject(appUser.getUsername())
                    .withExpiresAt(new Date(System.currentTimeMillis() + 1 * 60 * 1000))
                    .withIssuer(request.getRequestURL().toString())
                    .withClaim( name: "roles",appUser.getAppRoles().stream().map(r->r.getRoleName())
                            .collect(Collectors.toList()))
                    .sign(algorithm);
            Map<String, String> idToken = new HashMap<>();
            idToken.put("access-token", jwtAccessToken);
            idToken.put("refresh-token", jwt);
            response.setContentType("application/json");
            new ObjectMapper().writeValue(response.getOutputStream(),idToken);

        } catch (Exception e) {
            throw e;
```

Spring Security Configuration

```java
public class SecurityConfig extends WebSecurityConfigurerAdapter {
    @Autowired
    private AccountService accountService;
    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.userDetailsService(new UserDetailsService() {
            @Override
            public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
                AppUser appUser=accountService.loadUserByUsername(username);
                Collection<GrantedAuthority> authorities=new ArrayList<>();
                appUser.getAppRoles().forEach(
                        r->{
                            authorities.add(new SimpleGrantedAuthority(r.getRoleName()));
                        });
                return new User(appUser.getUsername(),appUser.getPassword(),authorities); } }); }
    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.csrf().disable();
        http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
        http.headers().frameOptions().disable();
        http.authorizeRequests().antMatchers( ...antPatterns: "/h2-console/**").permitAll();
        http.authorizeRequests().anyRequest().authenticated();
        http.addFilter(new JwtAuthticationFilter(authenticationManager()));
        http.addFilterBefore(new JwtAuthorizationFilter(), UsernamePasswordAuthenticationFilter.class); }
    @Bean
    @Override
    public AuthenticationManager authenticationManagerBean() throws Exception {
        return super.authenticationManagerBean(); }
```

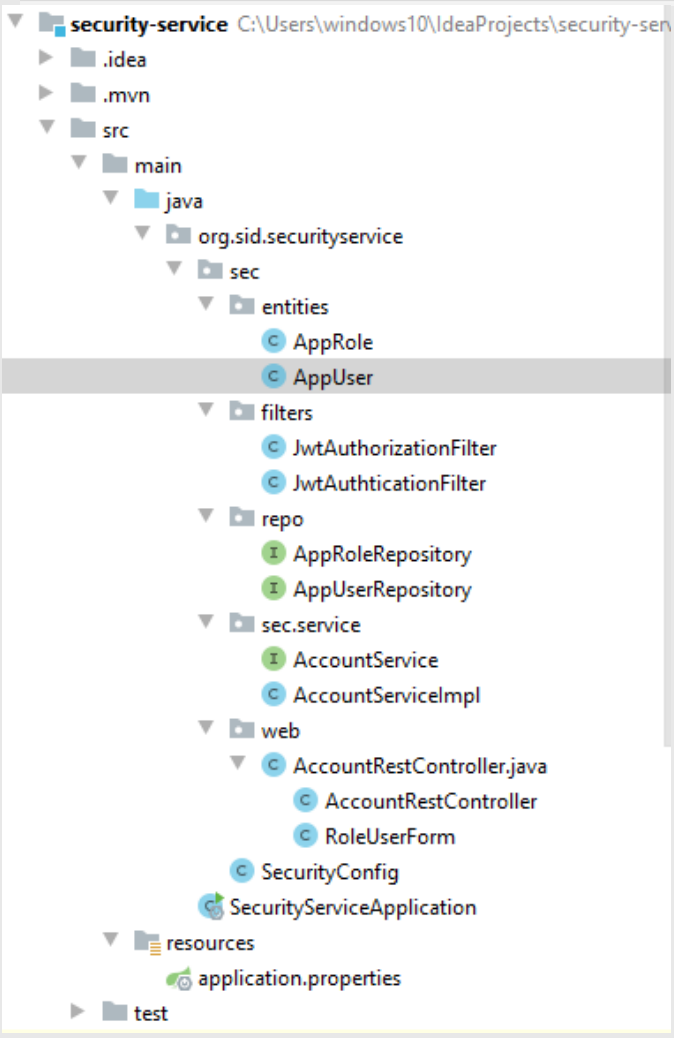
Spring Security Configuration : JWT Authentication Filter

```
public class JwtAuthticationFilter extends UsernamePasswordAuthenticationFilter {
    private AuthenticationManager authenticationManager;
    public JwtAuthticationFilter(AuthenticationManager authenticationManager) {
        this.authenticationManager = authenticationManager; }
    @Override
    public Authentication attemptAuthentication(HttpServletRequest request, HttpServletResponse response)
            throws AuthenticationException {
    System.out.println("attemptAuthentication");
     String username=request.getParameter( s: "username");
     String password=request.getParameter( s: "password");
     System.out.println(username);
     System.out.println(password);
   UsernamePasswordAuthenticationToken authenticationToken=new UsernamePasswordAuthenticationToken(
           username,password);
      return  authenticationManager.authenticate(authenticationToken);
    }
    @Override
    protected void successfulAuthentication(HttpServletRequest request, HttpServletResponse response,
                                            FilterChain chain, Authentication authResult)
            throws IOException, ServletException {
        System.out.println("successfullAuthentification");
        User user = (User) authResult.getPrincipal();
        Algorithm algo1 = Algorithm.HMAC256("mySecret1234");
        String jwtAccessToken = JWT.create()
        .withSubject(user.getUsername())
     .withExpiresAt(new Date(System.currentTimeMillis() + 1 * 60 * 1000))
```

```
      .withIssuer(request.getRequestURL().toString())
 .withClaim( name: "roles",user.getAuthorities().stream().map(ga->ga.getAuthority())
          .collect(Collectors.toList()))
              .sign(algo1);
     // response.setHeader("Authorization",jwtAccessToken);
 String jwtRefreshTokenToken = JWT.create()
.withSubject(user.getUsername())
.withExpiresAt(new Date(System.currentTimeMillis() + 15 * 60 * 1000))
  .withIssuer(request.getRequestURL().toString())
    .sign(algo1);
     Map<String, String> idToken = new HashMap<>();
     idToken.put("access-token", jwtAccessToken);
     idToken.put("refresh-token", jwtRefreshTokenToken);
     response.setContentType("application/json");
     new ObjectMapper().writeValue(response.getOutputStream(),idToken);

 }
```

Spring Security Configuration : JWT Authorization Filter

```
public class JwtAuthorizationFilter extends OncePerRequestFilter {
    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response
            , FilterChain filterChain) throws ServletException, IOException {
String authorizationToken=request.getHeader( s: "Authorization");
if (authorizationToken!=null && authorizationToken.startsWith("Bearer")){
  try {
    String jwt=authorizationToken.substring(7);
    Algorithm algorithm=Algorithm.HMAC256("mySecret1234");
    JWTVerifier jwtVerifier = JWT.require(algorithm).build();
     DecodedJWT decodedJWT =jwtVerifier.verify(jwt);
     String username=decodedJWT.getSubject();
     String[] roles=decodedJWT.getClaim( s: "roles").asArray(String.class);
      Collection<GrantedAuthority> authorities =new ArrayList<>();
      for (String r :roles){
          authorities.add(new SimpleGrantedAuthority(r)); }
      UsernamePasswordAuthenticationToken authenticationToken=
      new UsernamePasswordAuthenticationToken(username, credentials: null,authorities);
      SecurityContextHolder.getContext().setAuthentication(authenticationToken);
      filterChain.doFilter(request,response); }
  catch (Exception e){
      response.setHeader( s: "error-message",e.getMessage());
      response.sendError(HttpServletResponse.SC_FORBIDDEN); } }
else {
    filterChain.doFilter(request,response); }
    }}
```



architecture de security-service

Tests : Authentification

localhost:8085/users

```
// 20201225013310
// http://localhost:8085/users

[
  {
    "id": 1,
    "username": "user1",
    "password": "1234",
    "appRoles": [
      {
        "id": 1,
        "roleName": "USER"
      }
    ]
  },
  {
    "id": 2,
    "username": "admin",
    "password": "1234",
    "appRoles": [
      {
        "id": 1,
        "roleName": "USER"
      },
      {
        "id": 2,
        "roleName": "ADMIN"
      }
    ]
  },
  {
    "id": 3,
    "username": "user2",
    "password": "1234",
    "appRoles": [
      {
        "id": 1,
        "roleName": "USER"
      },
      {
```

localhost:8085/users

```
// 20201226164719
// http://localhost:8085/users

[
  {
    "id": 1,
    "username": "user1",
    "appRoles": [
      {
        "id": 1,
        "roleName": "USER"
      }
    ]
  },
  {
    "id": 2,
    "username": "admin",
    "appRoles": [
      {
        "id": 1,
        "roleName": "USER"
      },
      {
        "id": 2,
        "roleName": "ADMIN"
      }
```

```
{
  "id": 1,
  "username": "user1",
  "password": "$2a$10$1YsqymABXC.IWLfZTXtLWOQhyj3.4AR.NJ2S53ESpyJEM6kAFqZQi",
  "appRoles": [
    {
      "id": 1,
      "roleName": "USER"
    }
  ]
},
{
  "id": 2,
  "username": "admin",
  "password": "$2a$10$.CAyqp61L13COPVM84TmKOpULuWQ.nsS9A6mpwd.5ZfZ5IpkOHkdO",
  "appRoles": [
    {
      "id": 1,
      "roleName": "USER"
    },
    {
      "id": 2,
      "roleName": "ADMIN"
```

Method: POST

Request URL: http://localhost:8085/users

```
{
"username":"toto",
"password": 2223

}
```

200 OK 161.61 ms DETAILS

```
{
   "id": 6,
   "username": "toto",
   "appRoles": [Array[0]],
}
```

Method: POST

Request URL: http://localhost:8085/roles

```
{
"roleName":"ROLEX"

}
```

Method: POST

Request URL: http://localhost:8085/addRoleToUser

Parameters

Headers | Body

Body content type: application/json

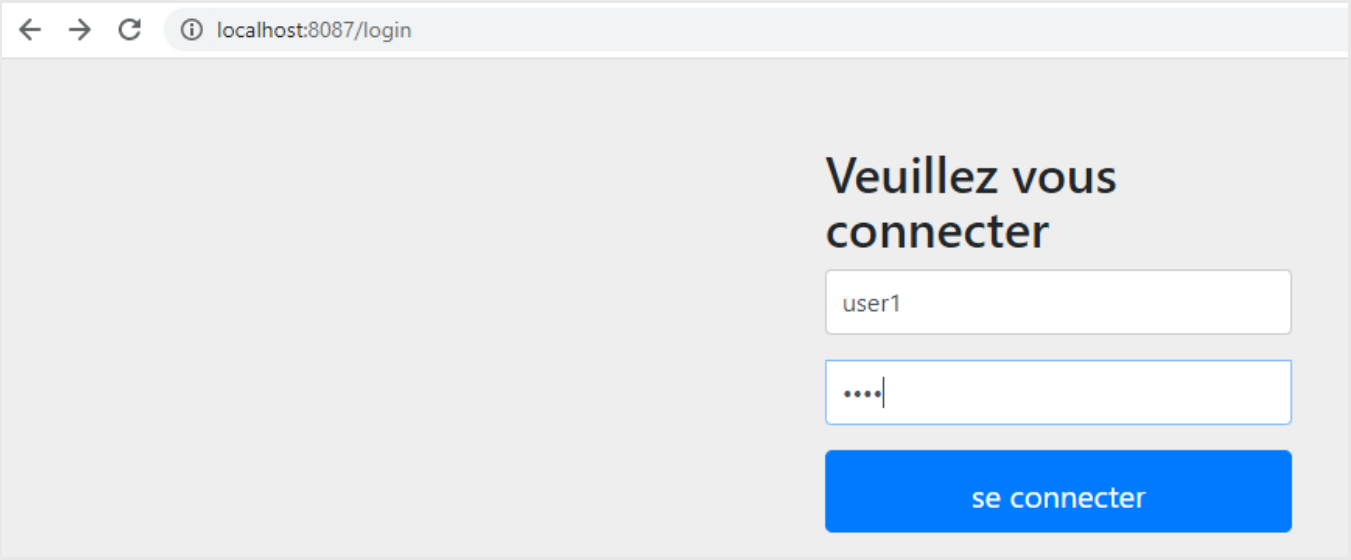
Editor view: Raw input

FORMAT JSON MINIFY JSON

```
{"username":"toto","roleName":"ROLEX"}
```

200 OK 12.01 ms

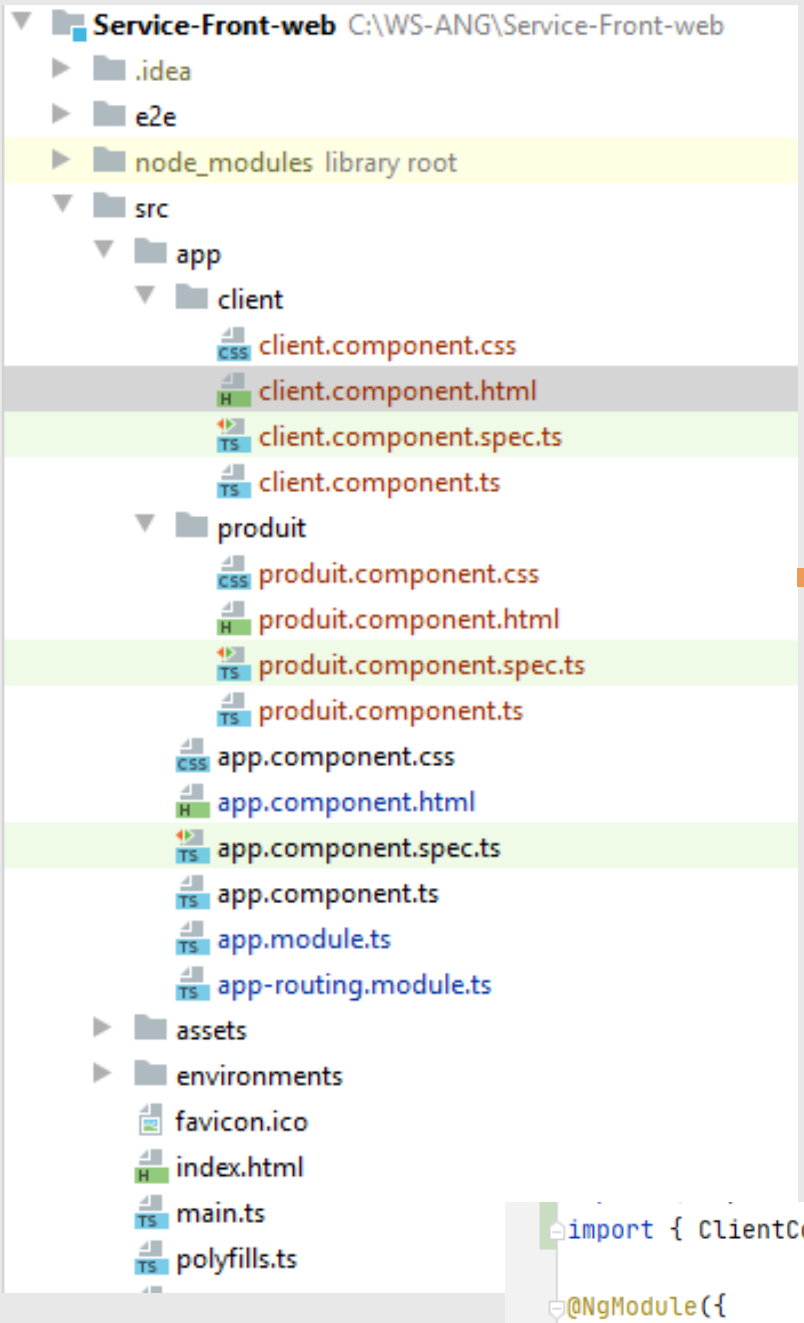
```
{
   "id": 9,
   "roleName": "ROLEX"
}
```


localhost:8087/login
Veuillez vous connecter
user1
se connecter


localhost:8085/users
]
},
{
"id": 5,
"username": "user4",
"appRoles": [
{
"id": 1,
"roleName": "USER"
},
{
"id": 5,
"roleName": "BILLS
}
]
},
{
"id": 6,
"username": "toto",
"appRoles": [
{
"id": 6,
"roleName": "ROLEX
}


{
"_links": {
"appUsers": {
"href": "http://localhost:8087/appUsers{?page,size,sort}",
"templated": true
},
"appRoles": {
"href": "http://localhost:8087/appRoles{?page,size,sort}",
"templated": true
},
"profile": {
"href": "http://localhost:8087/profile"
}
}
}

Partie Frontend

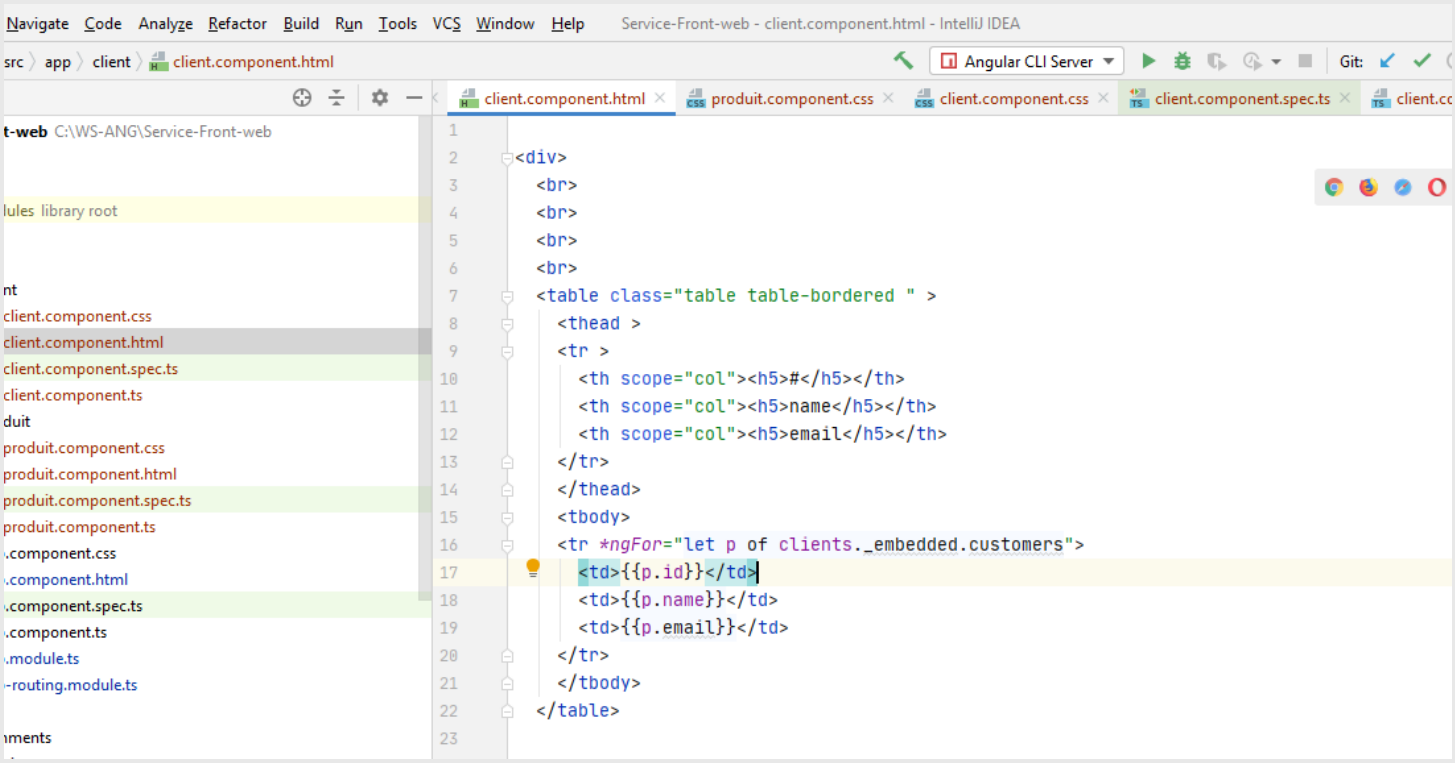


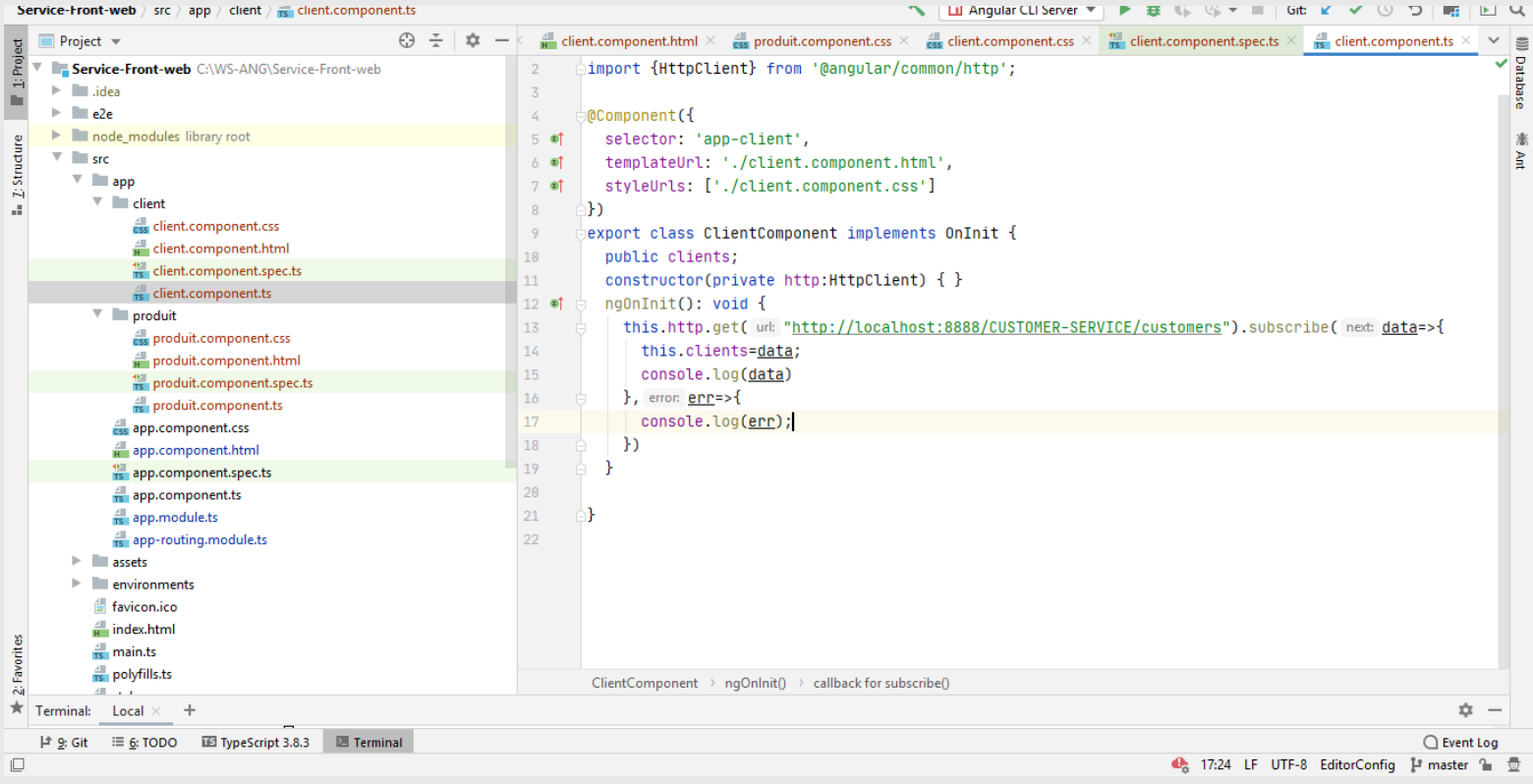
architecture de projet service front-web

AppRoutingModule

```
import { ClientComponent } from './client/client.component';

@NgModule({
  declarations: [
    AppComponent,
    ProduitComponent,
    ClientComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule, HttpClientModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Page clients:





Page produit:

produit.component.html | client.component.html | produit.component.css | client.component

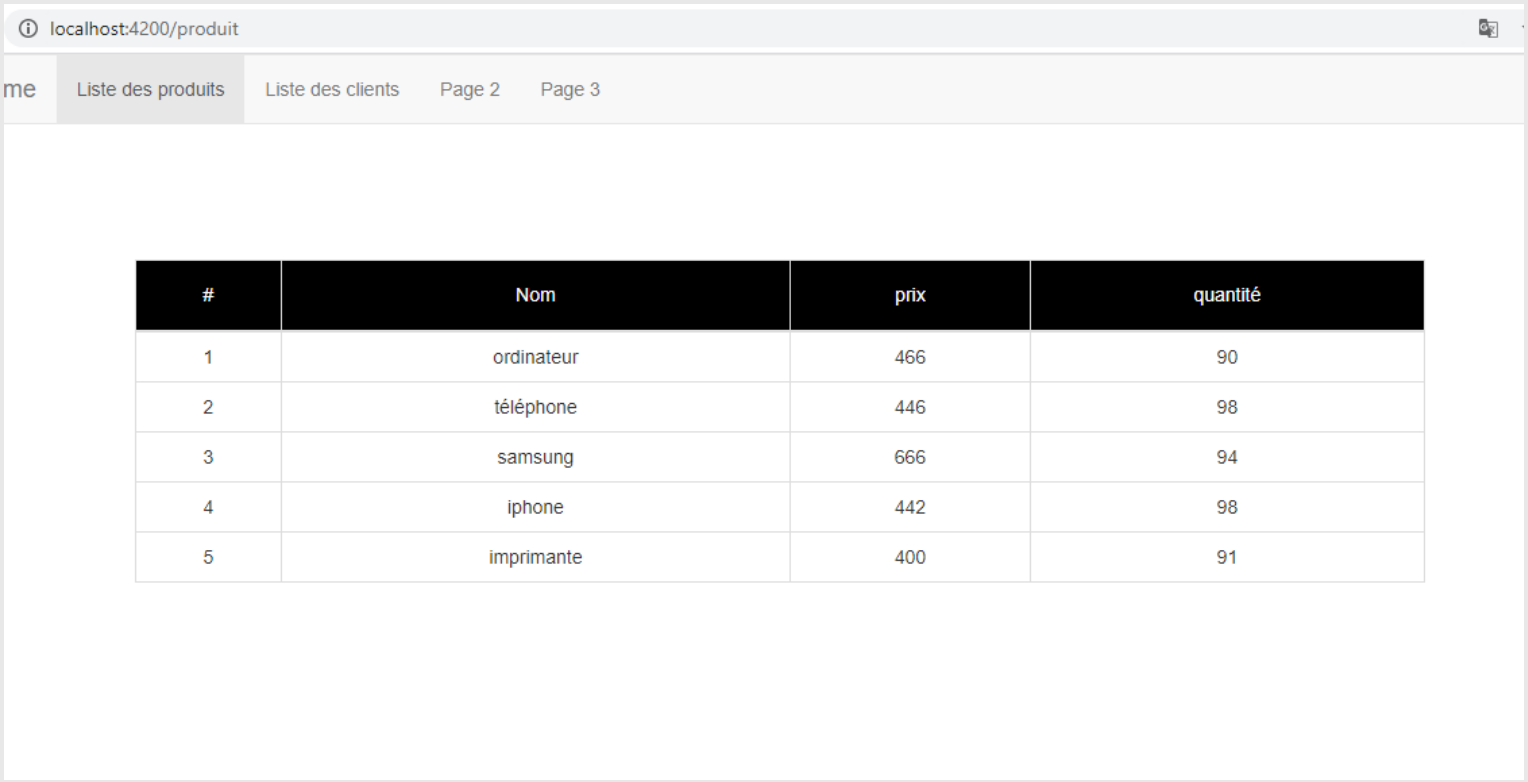
```
<br>
<table class="table table-bordered " >
    <thead >
    <tr >
      <th scope="col"><h5>#</h5></th>
      <th scope="col"><h5>name</h5></th>
      <th scope="col"><h5>price</h5></th>
      <th scope="col"><h5>quantity</h5></th>
    </tr>
    </thead>
    <tbody>
    <tr *ngFor="let p of products._embedded.products">
              <td>{{p.id}}</td>
              <td>{{p.name}}</td>
            <td>{{p.price}}</td>
              <td>{{p.quantity}}</td>
    </tr>
    </tbody>
  </table>
```

div

produit.component.css | client.component.css | client.component.spec.ts | client.component.ts | produit.component.ts

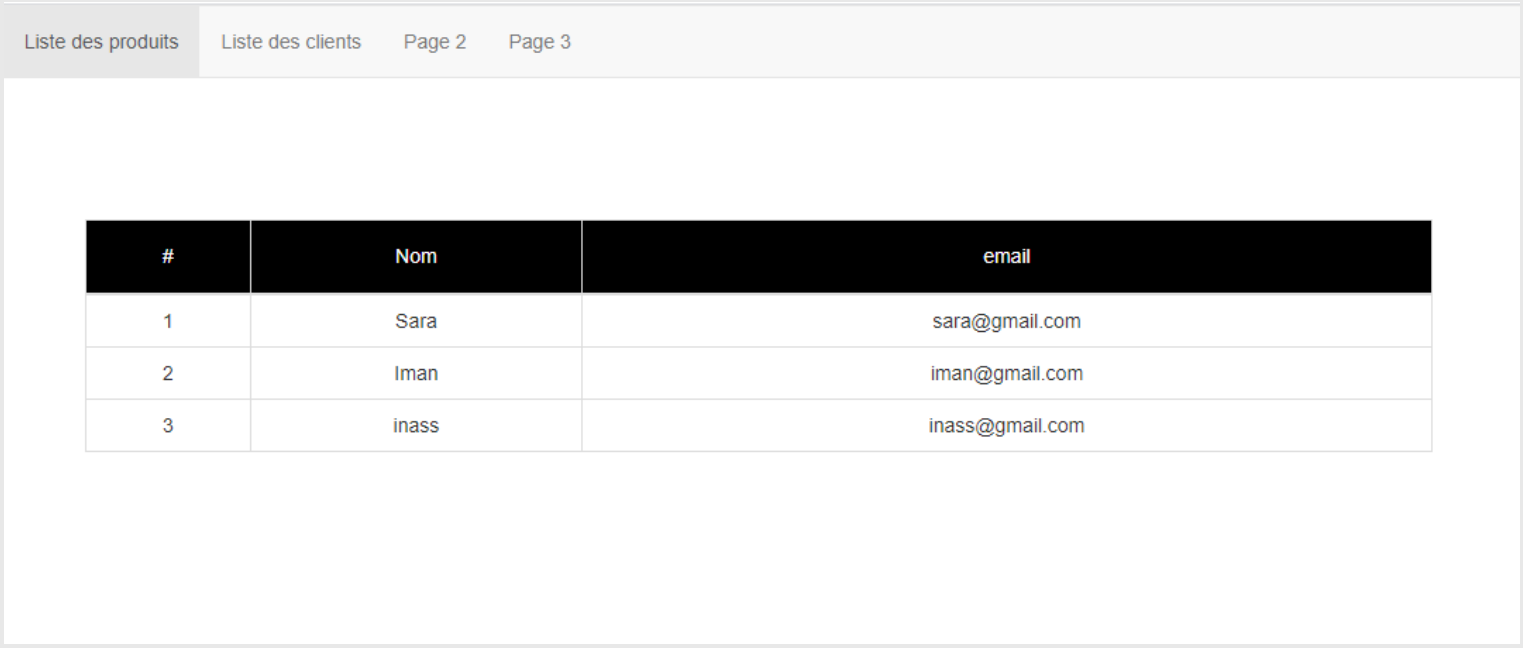
```
@Component({
  selector: 'app-produit',
  templateUrl: './produit.component.html',
  styleUrls: ['./produit.component.css']
})
export class ProduitComponent implements OnInit {
  public products;
  constructor(private http:HttpClient) { }
  ngOnInit(): void {
    this.http.get( url: "http://localhost:8888/PRODUCT-SERVICE/products").subscribe( next: data=>{
      this.products=data;
      console.log(data)
    }, error: err=>{
      console.log(err);
    })
  }
  }
```

interface des produits



#	Nom	prix	quantité
1	ordinateur	466	90
2	téléphone	446	98
3	samsung	666	94
4	iphone	442	98
5	imprimante	400	91

interface des clients



#	Nom	email
1	Sara	sara@gmail.com
2	Iman	iman@gmail.com
3	inass	inass@gmail.com

Conclusion

l'objectif de L’application est de guider nos premiers pas dans Approche Micro services et l’orchestration des services se fait via les services techniques de Spring Cloud

 À la fin du projet, je serai capables de créer une application web qui respecte les standards reconnus dans le domaine et je disposerai des bases nécessaires pour utiliser la plupart des technologies se basant sur les micro services