Ecole Normale Supérieure de l'Enseignement Technique Mohammedia
Université Hassan II de Casablanca
ENSET
المدرسة العليا لأساتذة التعليم التقني المحمدية
جامعة الحسن الثاني بالدار البيضاء

Département Mathématiques et Informatique

« Ingénierie Informatique : Big Data et Cloud Computing »

II-BDCC

Module : Systèmes Distribués basés sur les Micro services

TAF : TP Micro Services

Réalisé par :

Hamza El Anssari

Année Universitaire : 2019-2020

Introduction :

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, Python,...)

• 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 de

données effectué à travers les différentes APIs qu'ils exposent. (SOAP, REST, RMI, CORBA, JMS, MQP, ...)

• Lorsqu'on les combine, ces micro services peuvent réaliser des opérations très complexes.

• Ils sont faiblement couplés puisque chaque micro service est physiquement séparé des autres,

• Indépendance relative entre les différentes équipes qui développent les différents micro services.

• Facilité des tests et du déploiement

• Livraison continue.

• S'apprête bien à au processus du GL : TDD (Test Driver Développement) et les méthodes agiles

• Comme pour le cas d'une application monolithique, un micro service peut être composé de plusieurs très petites couches:

- Couche DAO
- Couche Métier,
- Couches Techniques (REST, SOAP, RMI, JMS, AMQP, Sécurité, etc...)

Travail à faire :

1. Créer le micro service Customer-service
    - Créer l'entité Customer
    - Créer l'interface Custom Repository 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 Product Repository 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 Discovery 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
6. Créer le service Billing Service
7. Créer un service d'authentification Stateless 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

I. Créer le micro service Customer-service :

➢ Entité et l'interface :

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

Customer Entité

```
@RepositoryRestResource
interface CustomerRepository extends JpaRepository<Customer,Long> {
    @RestResource(path = "/byName")
    Page<Customer> findByNameContains(@Param("kw") String name, Pageable pageable);
}
@Projection(name = "FullCustomer",types = Customer.class)
interface FullCustomerProjection extends Projection{
    Long getId();
    String getName();
    String getEmail();
}

@Projection(name = "NameCustomer",types = Customer.class)
interface NameCustomerProjection extends Projection{
    String getName();
}
```

interface Custom Repository

```
spring.cloud.discovery.enabled=true
server.port=8081
spring.application.name=customer-service
#management.endpoints.web.exposure.include=*
eureka.client.service-url.defaultZone=http://localhost:8765/eureka
```

Application properties

➢ **Code :**

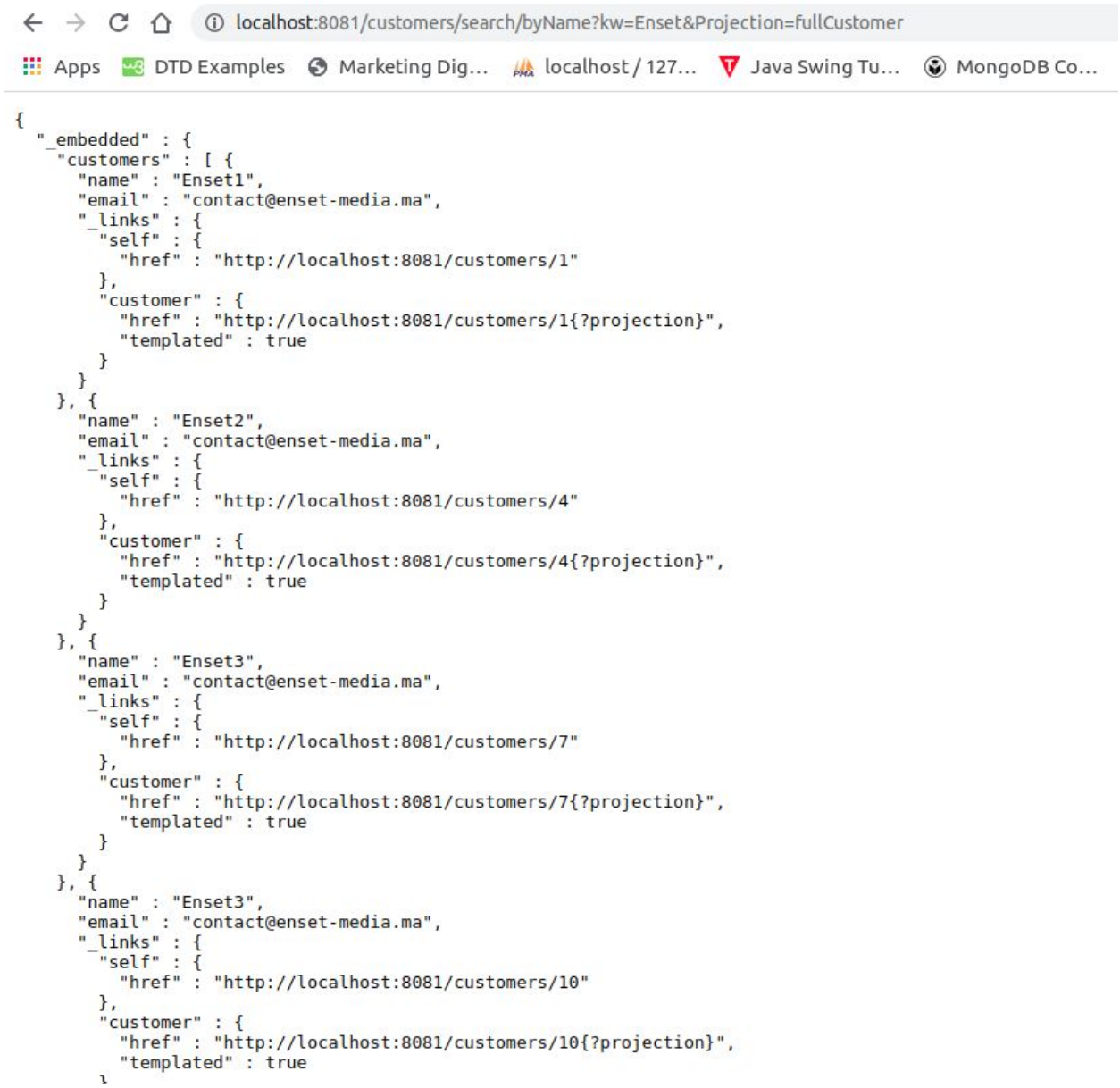
```
package org.sid.customerservice;
import lombok.AllArgsConstructor; import lombok.Data; import lombok.NoArgsConstructor;
import lombok.ToString; import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.repository.query.Param;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;
import org.springframework.data.rest.core.annotation.RestResource;
import org.springframework.data.rest.core.config.Projection;
import javax.persistence.Entity; import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType; import javax.persistence.Id;

@SpringBootApplication
public class CustomerServiceApplication {
  public static void main(String[] args) {
    SpringApplication.run(CustomerServiceApplication.class, args); }
  @Bean
  CommandLineRunner start(CustomerRepository customerRepository){
    return args -> {
      customerRepository.save(new Customer(null,"Enset1","contact@enset-media.ma"));
      customerRepository.save(new Customer(null,"FSTM1","contact@fstm.ma"));
      customerRepository.save(new Customer(null,"ENSAM1","contact@ensam.ma"));
      customerRepository.save(new Customer(null,"Enset2","contact@enset-media.ma"));
```

```
        customerRepository.save(new Customer(null,"FSTM2","contact@fstm.ma"));
        customerRepository.save(new Customer(null,"ENSAM2","contact@ensam.ma"));
        customerRepository.save(new Customer(null,"Enset3","contact@enset-media.ma"));
        customerRepository.save(new Customer(null,"FSTM3","contact@fstm.ma"));
        customerRepository.save(new Customer(null,"ENSAM3","contact@ensam.ma"));
        customerRepository.save(new Customer(null,"Enset3","contact@enset-media.ma"));
        customerRepository.save(new Customer(null,"FSTM3","contact@fstm.ma"));
        customerRepository.save(new Customer(null,"ENSAM3","contact@ensam.ma"));
        customerRepository.save(new Customer(null,"Enset4","contact@enset-media.ma"));
        customerRepository.save(new Customer(null,"FSTM4","contact@fstm.ma"));
        customerRepository.save(new Customer(null,"ENSAM4","contact@ensam.ma"));
        customerRepository.save(new Customer(null,"Enset5","contact@enset-media.ma"));
        customerRepository.save(new Customer(null,"FSTM5","contact@fstm.ma"));
        customerRepository.save(new Customer(null,"ENSAM5","contact@ensam.ma"));
        customerRepository.findAll().forEach(System.out::println);
    };}}
@Entity @Data @NoArgsConstructor @AllArgsConstructor @ToString
class Customer{
  @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
  private Long id; private String name; private String email; }
@RepositoryRestResource
interface CustomerRepository extends JpaRepository<Customer,Long> {
  @RestResource(path = "/byName")
  Page<Customer> findByNameContains(@Param("kw") String name, Pageable pageable);}
@Projection(name = "FullCustomer",types = Customer.class)
interface FullCustomerProjection extends Projection{
  Long getId();
  String getName();
```

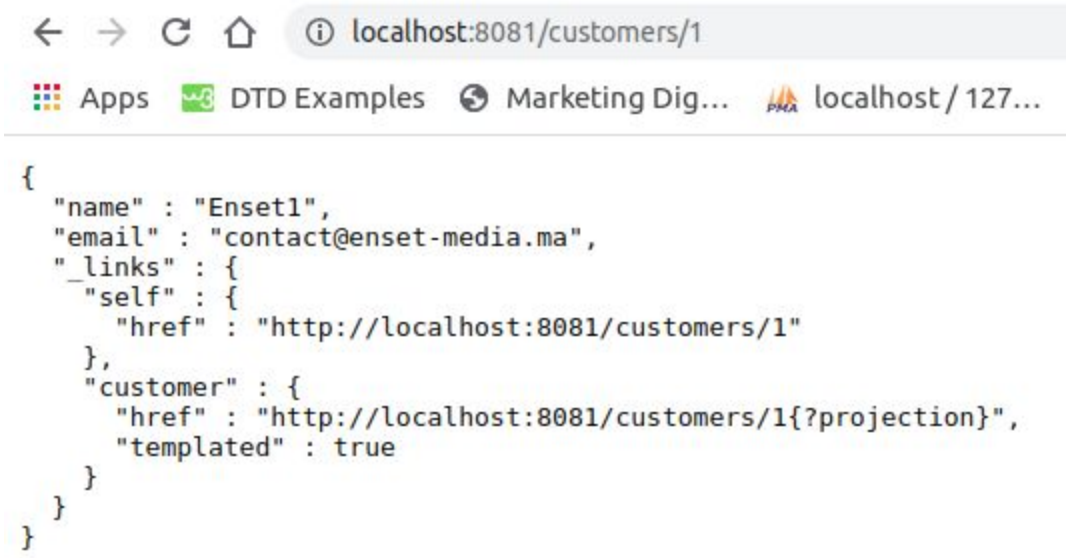
```
    String getEmail();}
@Projection(name = "NameCustomer",types = Customer.class)
interface NameCustomerProjection extends Projection{
    String getName(); }
```

➢ **Implémentation (localhost:8081):**

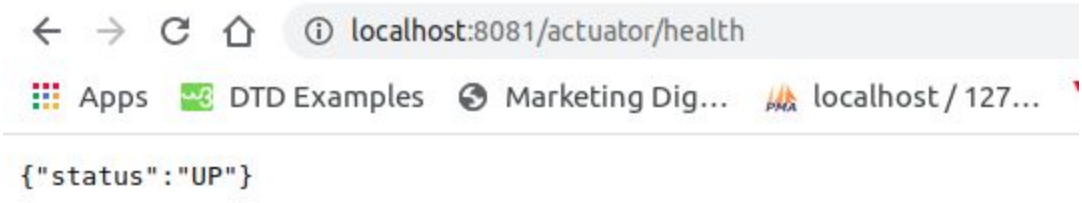
All Customers



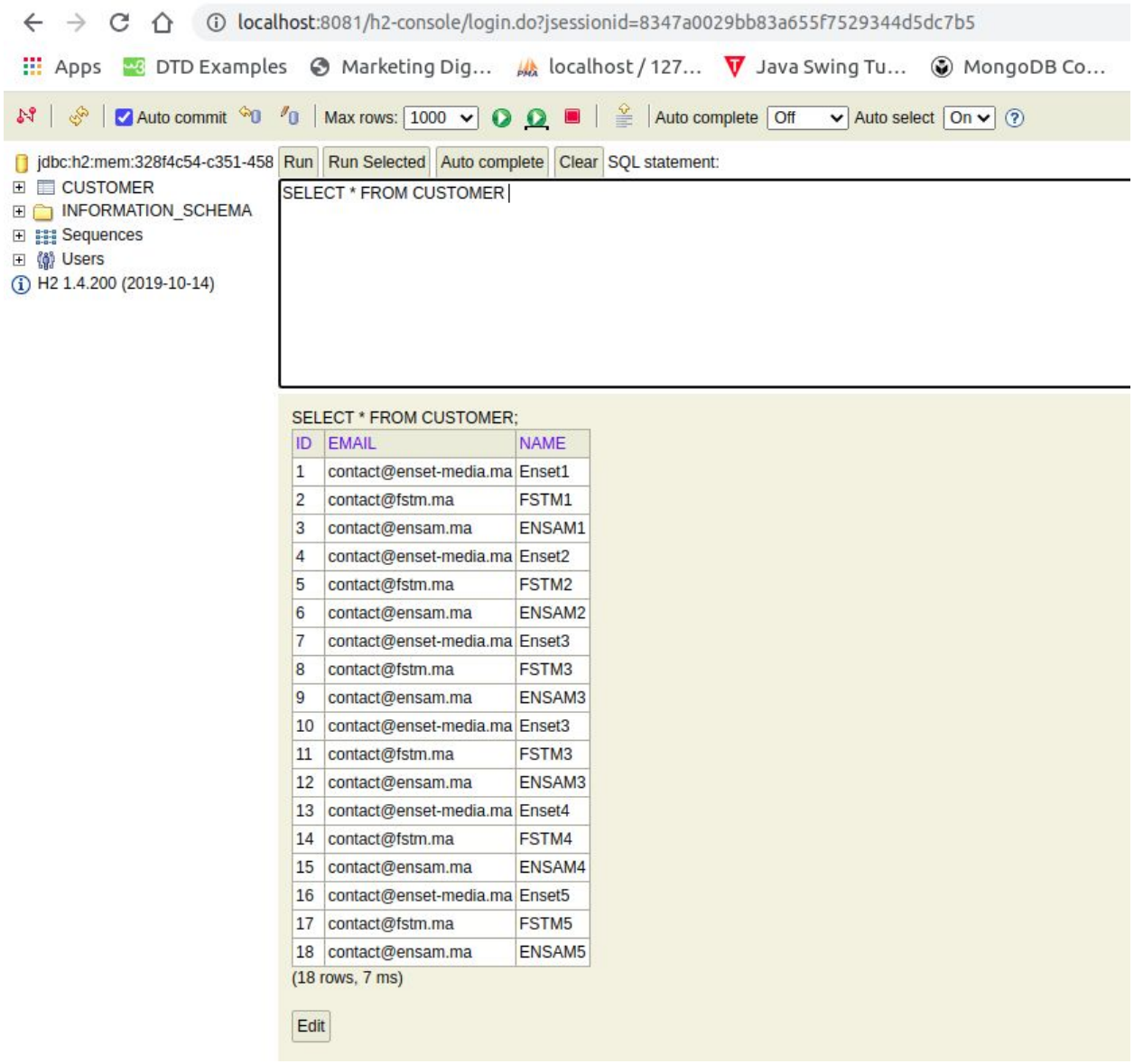
Full Customers avec projection



Customer id=1



Actuator health


localhost:8081/h2-console/login.do?jsessionid=8347a0029bb83a655f7529344d5dc7b5
Apps
DTD Examples
Marketing Dig...
localhost / 127...
Java Swing Tu...
MongoDB Co...
Auto commit
Max rows: 1000
Auto complete Off
Auto select On
jdbc:h2:mem:328f4c54-c351-458
CUSTOMER
INFORMATION_SCHEMA
Sequences
Users
H2 1.4.200 (2019-10-14)
Run
Run Selected
Auto complete
Clear
SQL statement:
SELECT * FROM CUSTOMER
SELECT * FROM CUSTOMER;
ID EMAIL NAME
1 contact@enset-media.ma Enset1
2 contact@fstm.ma FSTM1
3 contact@ensam.ma ENSAM1
4 contact@enset-media.ma Enset2
5 contact@fstm.ma FSTM2
6 contact@ensam.ma ENSAM2
7 contact@enset-media.ma Enset3
8 contact@fstm.ma FSTM3
9 contact@ensam.ma ENSAM3
10 contact@enset-media.ma Enset3
11 contact@fstm.ma FSTM3
12 contact@ensam.ma ENSAM3
13 contact@enset-media.ma Enset4
14 contact@fstm.ma FSTM4
15 contact@ensam.ma ENSAM4
16 contact@enset-media.ma Enset5
17 contact@fstm.ma FSTM5
18 contact@ensam.ma ENSAM5
(18 rows, 7 ms)
Edit

Database

II. Créer le micro service Inventory-service :

➢ Entité et l'interface :

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

Product Entité

```
@RepositoryRestResource
interface ProductRepository extends JpaRepository<Product,Long> {
    @RestResource(path = "/byName")
    Page<Product> findByNameContains(@Param("kw") String name, Pageable pageable);
}


@Projection(name = "FullProduct",types = Product.class)
interface FullCustomerProjection extends Projection{
    Long getId();
    String getName();
    String getPrice();
}

@Projection(name = "NameProduct",types = Product.class)
interface NameCustomerProjection extends Projection{
    String getName();
}
```

interface Product Repository

```
spring.application.name=inventory-service
spring.cloud.discovery.enabled=true
server.port=8082
eureka.client.service-url.defaultZone=http://localhost:8765/eureka
```

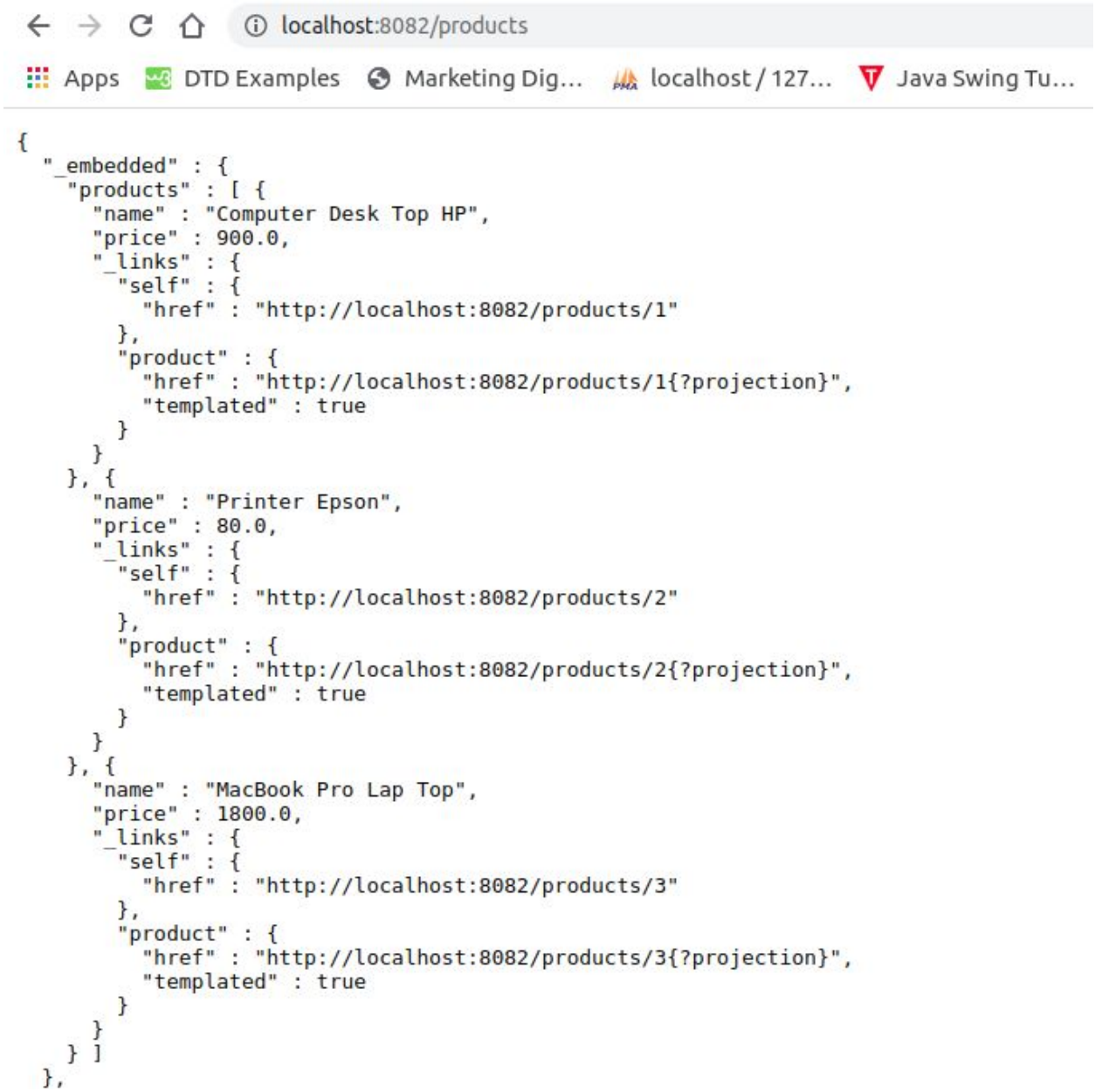
Application properties

➢ **Code :**

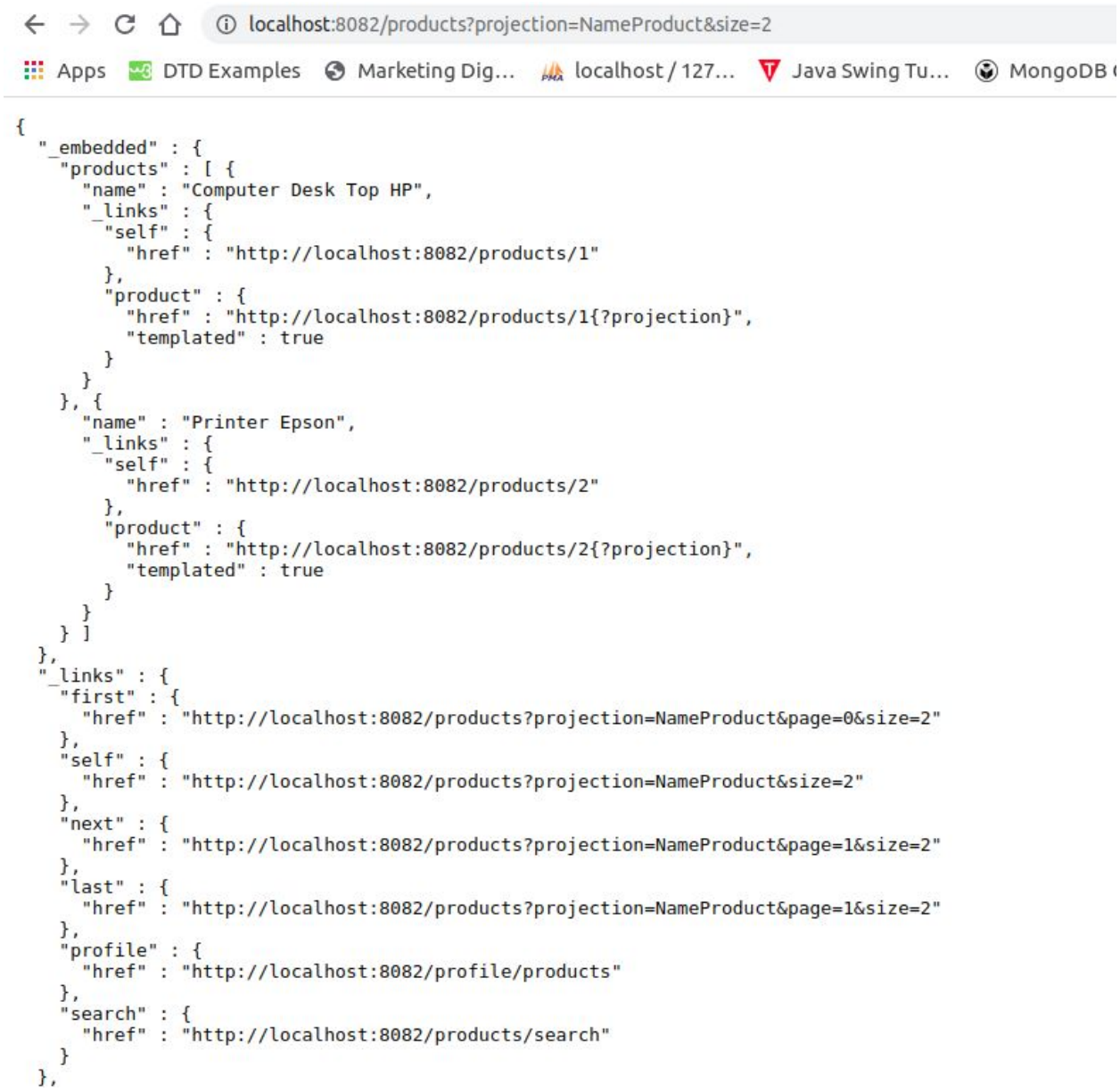
```
package org.sid.customerservice;

import lombok.AllArgsConstructor;
import lombok.Data; import lombok.NoArgsConstructor;
import lombok.ToString;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.repository.query.Param;
import
org.springframework.data.rest.core.annotation.RepositoryRestResource;
import org.springframework.data.rest.core.annotation.RestResource;
import org.springframework.data.rest.core.config.Projection;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;

@SpringBootApplication
public class InventoryServiceApplication {
    public static void main(String[] args) {
        SpringApplication.run(InventoryServiceApplication.class, args); }
    @Bean
    CommandLineRunner start(ProductRepository productRepository){
        return args -> {
```

```
            productRepository.save(new Product(null,"Computer Desk Top
HP",900));
            productRepository.save(new Product(null,"Printer Epson",80));
            productRepository.save(new Product(null,"MacBook Pro Lap
Top",1800));
            productRepository.findAll().forEach(System.out::println);
        }; } }
@Entity @Data @NoArgsConstructor @AllArgsConstructor @ToString
class Product{
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id; private String name;private double price; }

@RepositoryRestResource
interface ProductRepository extends JpaRepository<Product,Long> {
    @RestResource(path = "/byName")
    Page<Product> findByNameContains(@Param("kw") String name, Pageable
pageable); }

@Projection(name = "FullProduct",types = Product.class)
interface FullCustomerProjection extends Projection{
    Long getId();
    String getName();
    String getPrice(); }

@Projection(name = "NameProduct",types = Product.class)
interface NameCustomerProjection extends Projection{
    String getName(); }
```

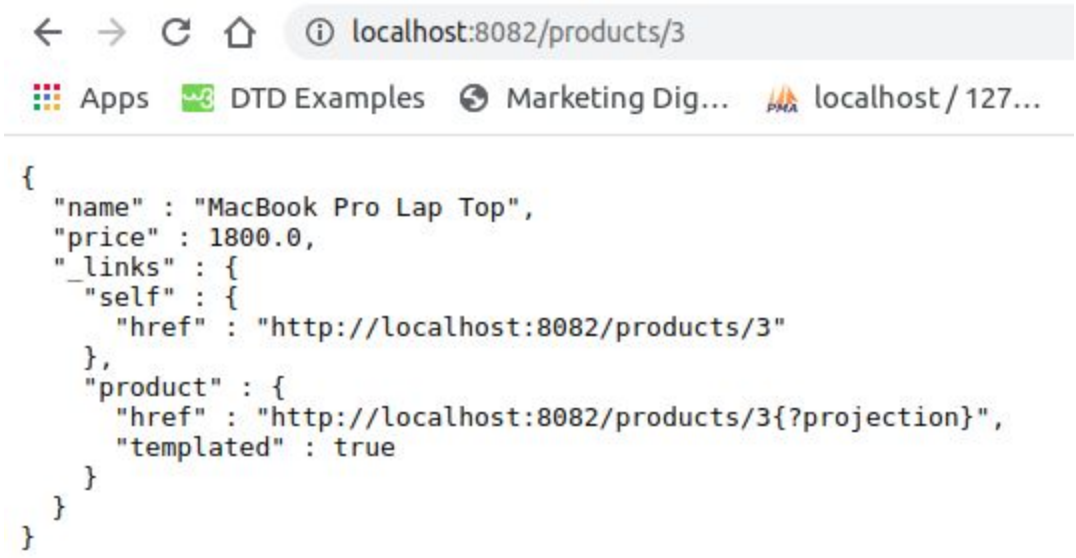
➢ **Implémentation (localhost:8082):**



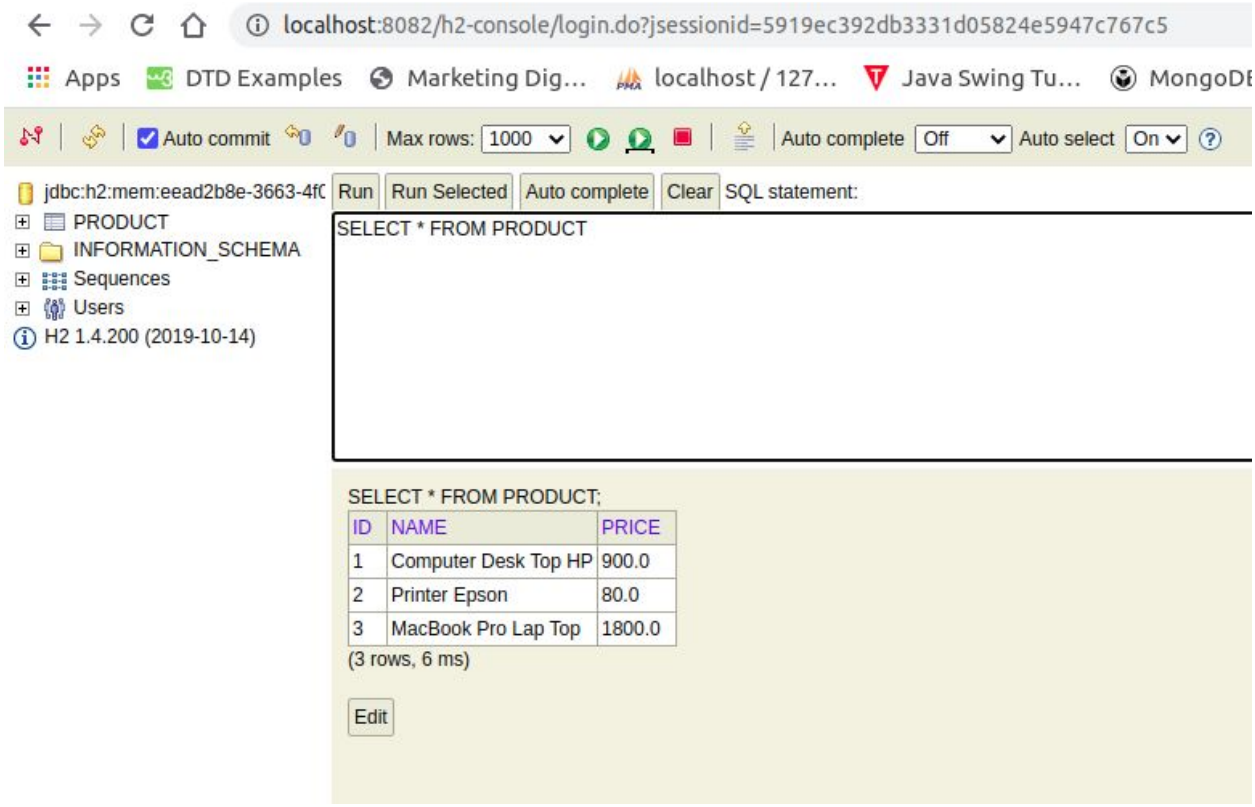
All Products



Affiche le nom de chaque produits (Projection)



Product id=3



Database

III. Créer la Gateway service en utilisant Spring Cloud Gateway :

Statique Routes Configuration

GatewayServiceApplication.java | 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/**
      discovery:
        enabled: false
server:
  port: 8088
```

Application properties

. .

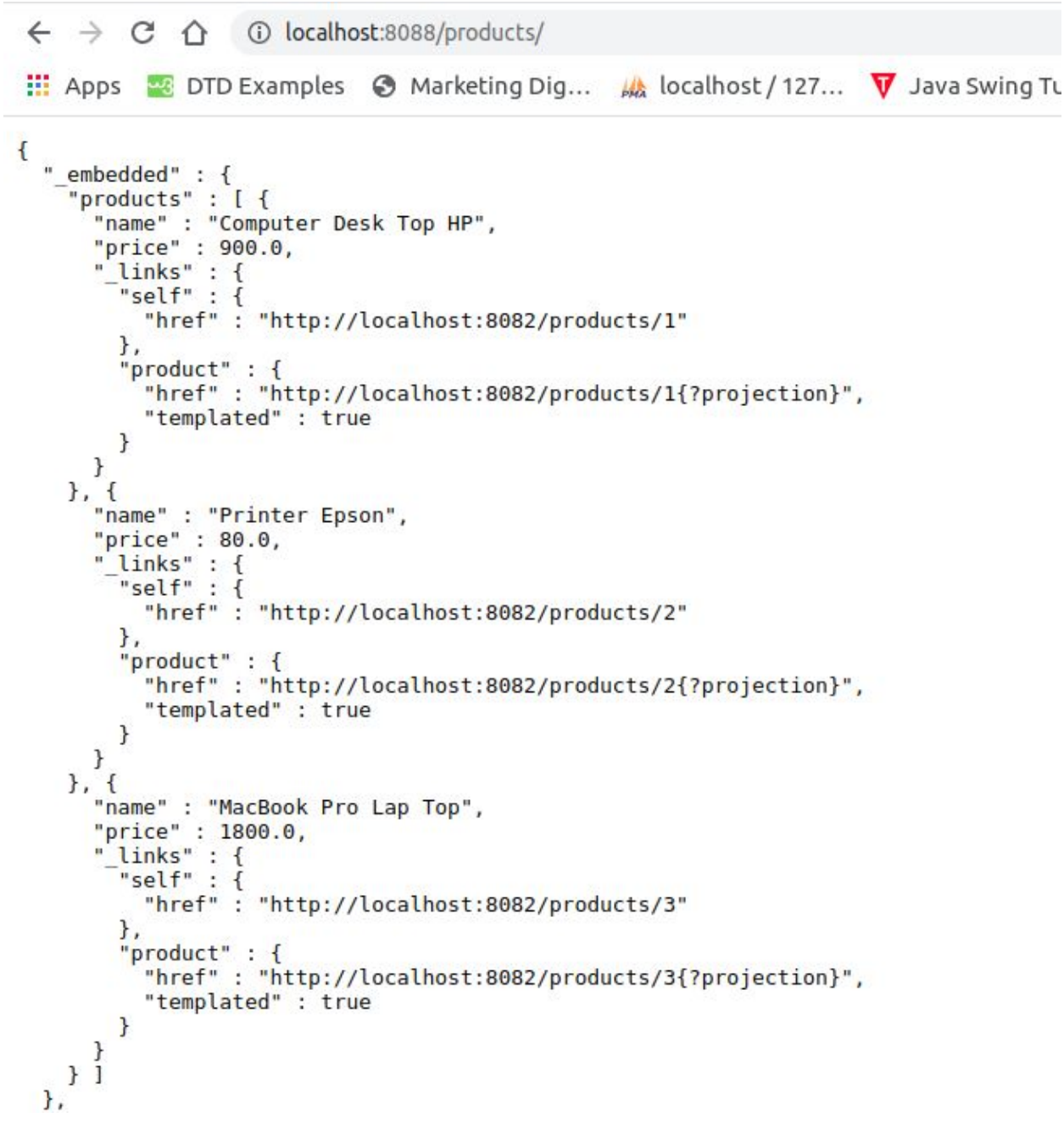
```java
@Bean
RouteLocator gatewayRoutes(RouteLocatorBuilder builder){
    return builder.routes()
            .route(r->r.path( ...patterns: "/customers/**").uri("http://localhost:8081/").id("r1"))
            .route(r->r.path( ...patterns: "/products/**").uri("http://localhost:8082/").id("r2")).build();}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

Statique Configuration

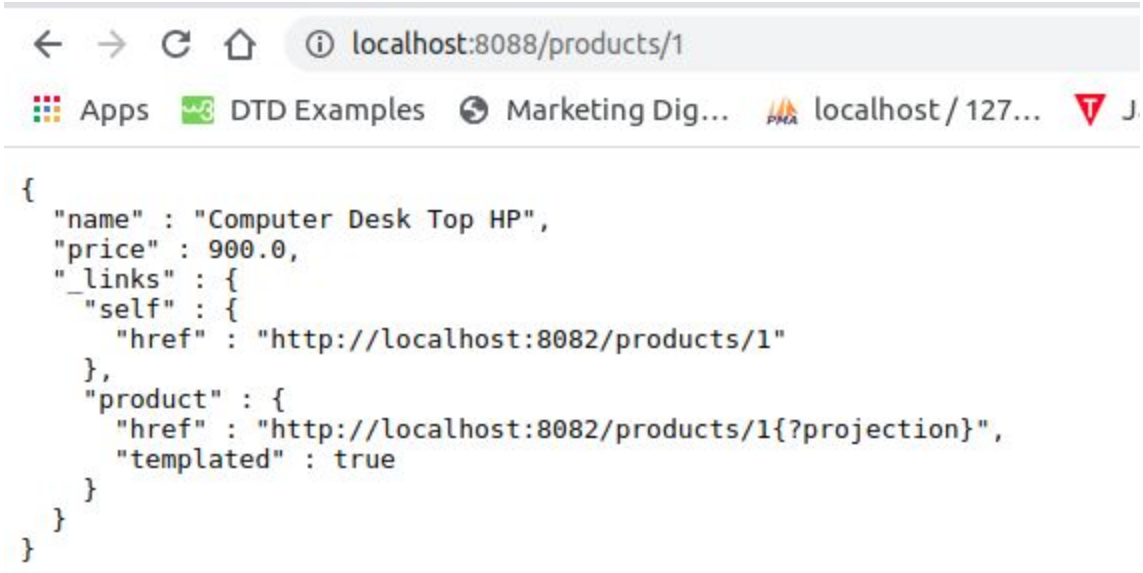
➢ **Code :**

```
package org.sid.gatewayservice;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.gateway.route.RouteLocator;
import
org.springframework.cloud.gateway.route.builder.RouteLocatorBuilder;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
public class GatewayServiceApplication {

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

    @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();
    }
}
```

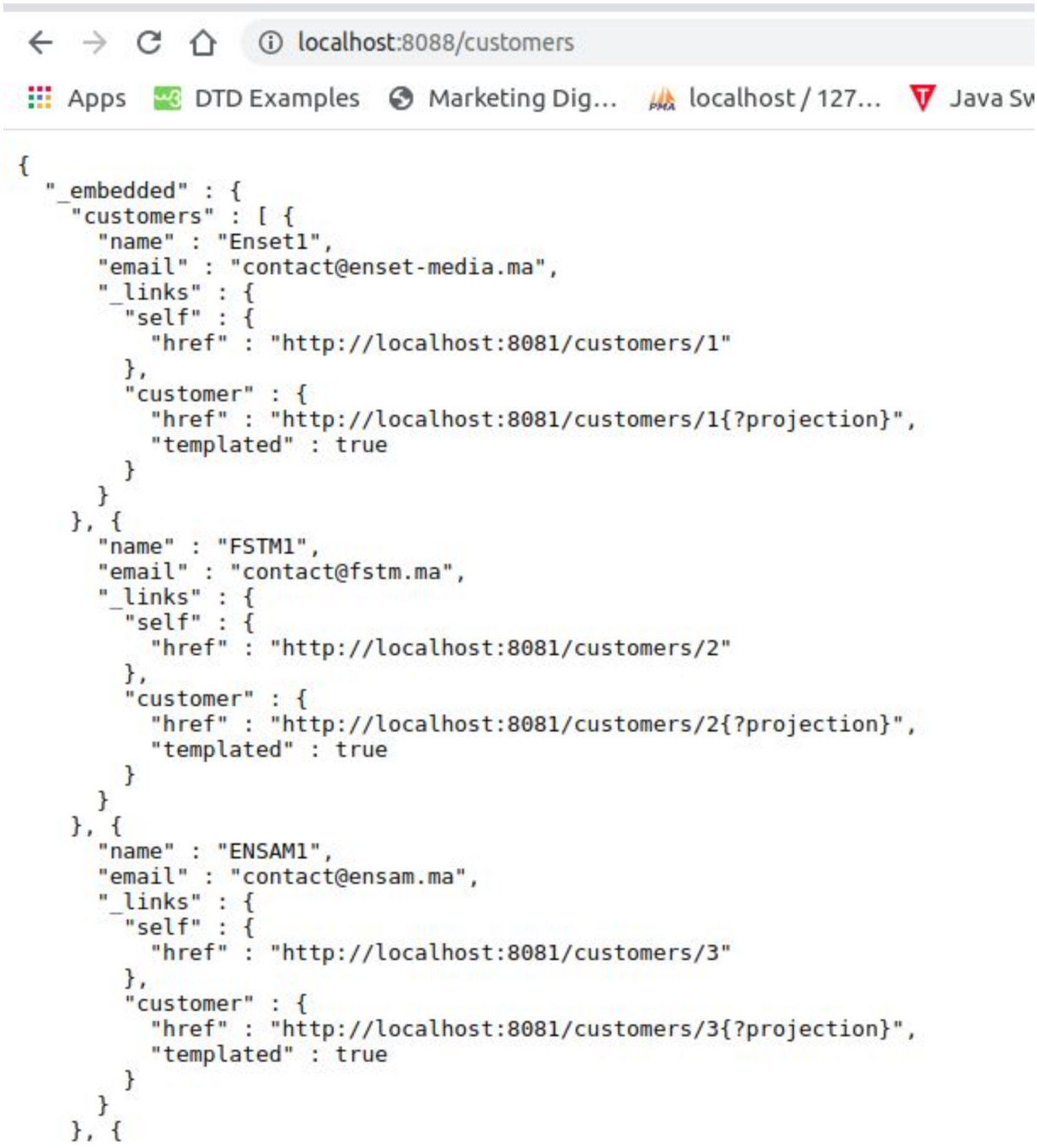
➢ **Implémentation (localhost:8088):**



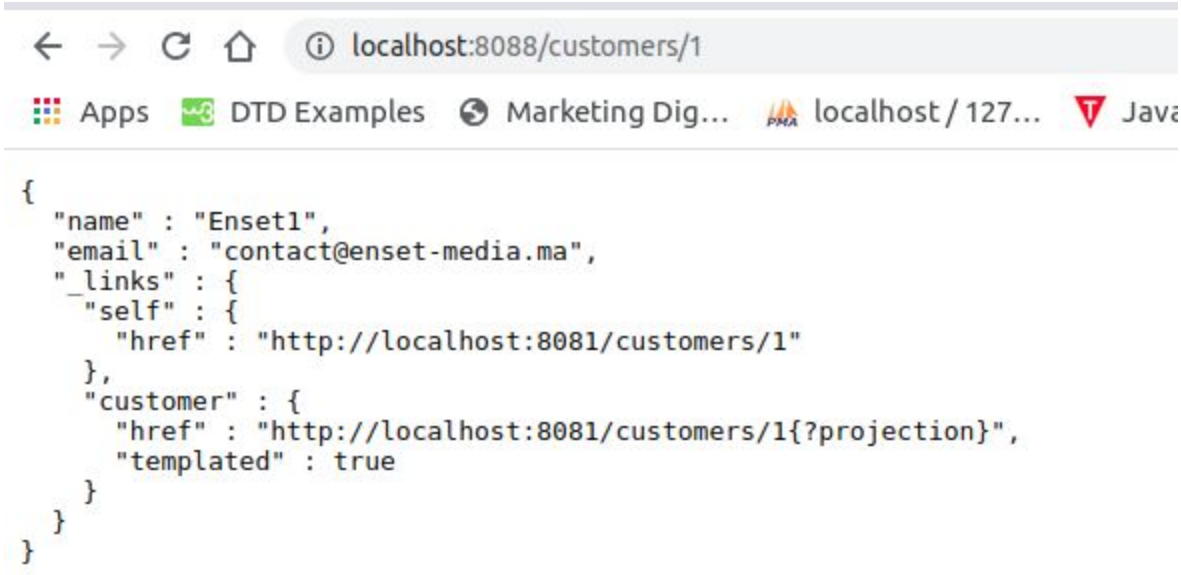
All Products


localhost:8088/products/1
Apps
DTD Examples
Marketing Dig...
localhost / 127...
{
"name" : "Computer Desk Top HP",
"price" : 900.0,
"_links" : {
"self" : {
"href" : "http://localhost:8082/products/1"
},
"product" : {
"href" : "http://localhost:8082/products/1{?projection}",
"templated" : true
}
}
}

Produit id=1



All Customers



Customer id=1

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

Dynamique Routes Configuration

DiscoveryServiceApplication.java | application.properties

```
server.port=8765
application.properties
eureka.client.fetch-registry=false
eureka.client.register-with-eureka=false
```

Application properties

application.properties | CustomerServiceApplication.java

```
spring.cloud.discovery.enabled=true
server.port=8081
spring.application.name=customer-service
#management.endpoints.web.exposure.include=*
eureka.client.service-url.defaultZone=http://localhost:8765/eureka
```

Customer application

application.properties | InventoryServiceApplication.java

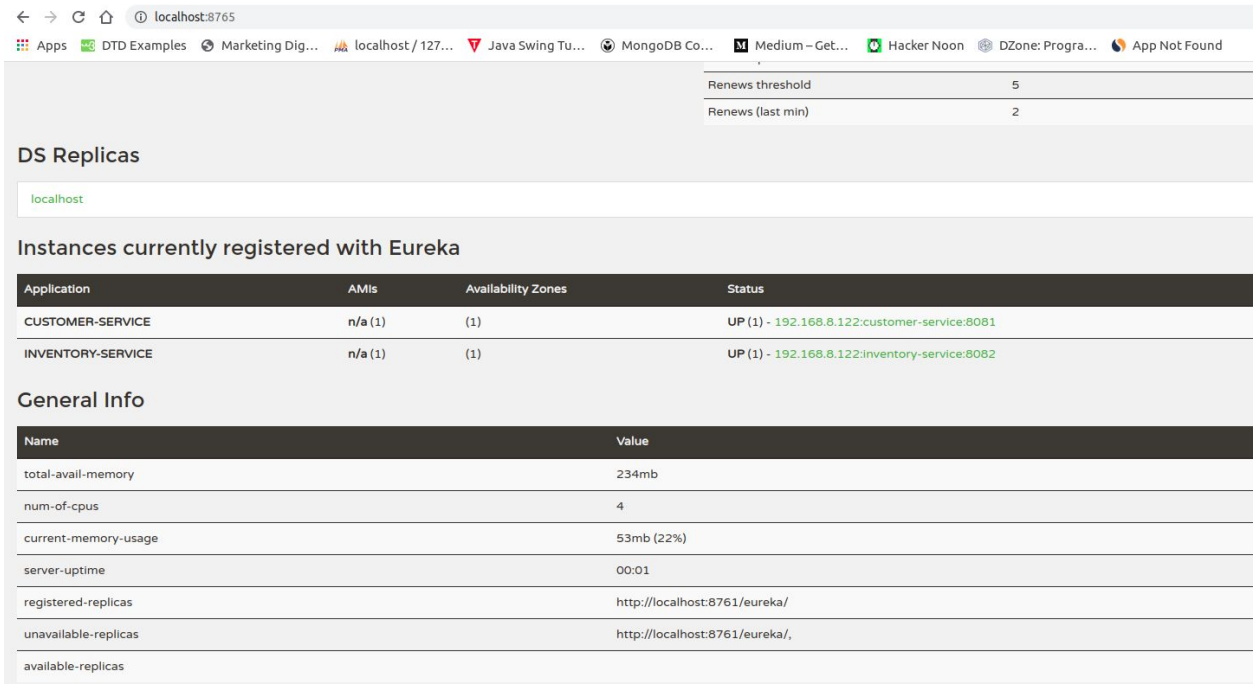
```
spring.application.name=inventory-service
spring.cloud.discovery.enabled=true
server.port=8082
eureka.client.service-url.defaultZone=http://localhost:8765/eureka
```

Inventory application

➢ Code :

```
package or.sid.discoveryservice;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import
org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;

@SpringBootApplication

@EnableEurekaServer

public class DiscoveryServiceApplication {

   public static void main(String[] args) {

       SpringApplication.run(DiscoveryServiceApplication.class, args);

   }

}
```

➢ Implémentation :



Localhost:8765

V. Créer le service Billing Service :

➢ **Code :**

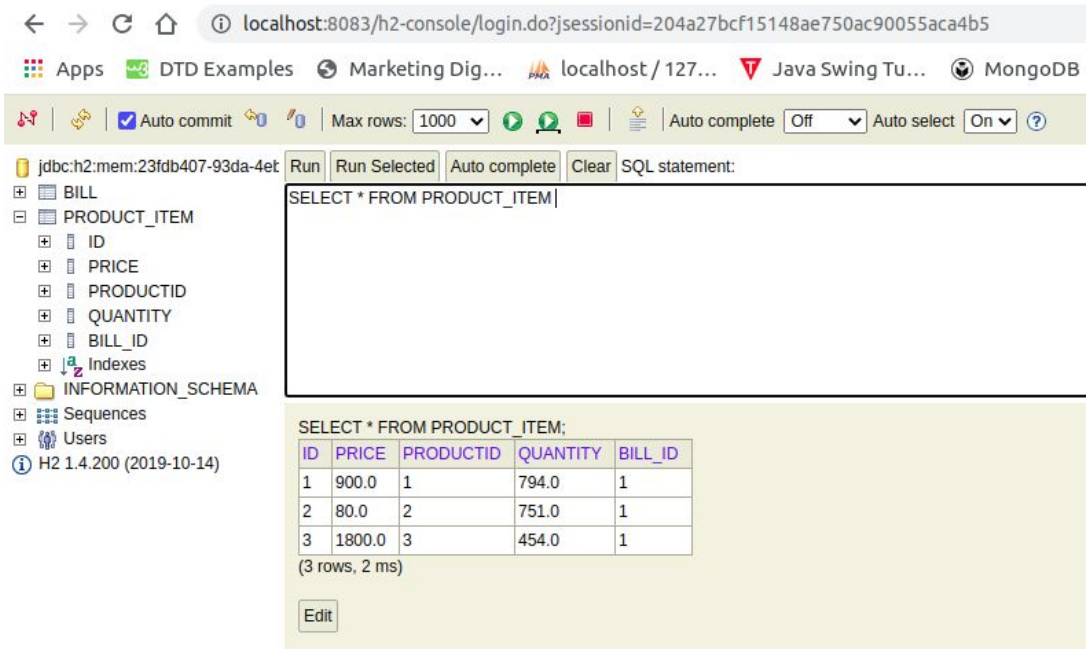
```
package org.sid.billingservice;


import com.fasterxml.jackson.annotation.JsonProperty;

import lombok.AllArgsConstructor;import lombok.Data; import
lombok.NoArgsConstructor;

import org.springframework.beans.factory.annotation.Autowired;

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.cloud.openfeign.FeignClient;

import org.springframework.context.annotation.Bean;

import org.springframework.data.jpa.repository.JpaRepository;

import
org.springframework.data.rest.core.annotation.RepositoryRestResource;

import org.springframework.hateoas.PagedModel;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.RestController;

import javax.persistence.*;import java.util.Collection; import
java.util.Date;import java.util.List;


@SpringBootApplication

@EnableFeignClients

public class BillingServiceApplication {

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

   @Bean

   CommandLineRunner start(BillRepository billRepository,
ProductItemRepository productItemRepository, InventoryServiceClient
inventoryServiceClient, CustomerServiceClient customerServiceClient){
```

```
        return args -> {
            Bill bill=new Bill();
            bill.setBillingDate(new Date());
            Customer customer=customerServiceClient.findCustomerById(1L);
            bill.setCustomerID(customer.getId());
            billRepository.save(bill);
            inventoryServiceClient.findAll().getContent().forEach(p->{
                productItemRepository.save(new
ProductItem(null,null,p.getId(),p.getPrice(),(int)(1+Math.random()*1000)
,bill));
            });
        };
    }
}
@Entity
@Data
@NoArgsConstructor
@AllArgsConstructor
class Bill{
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id; private Date billingDate;
    @Transient
    @OneToMany(mappedBy = "bill")
    private Collection<ProductItem> productItems;
    @Transient private Customer customer;
    private long customerID;
}
@RepositoryRestResource
interface BillRepository extends JpaRepository<Bill,Long> {}


@Entity @Data @NoArgsConstructor @AllArgsConstructor
```

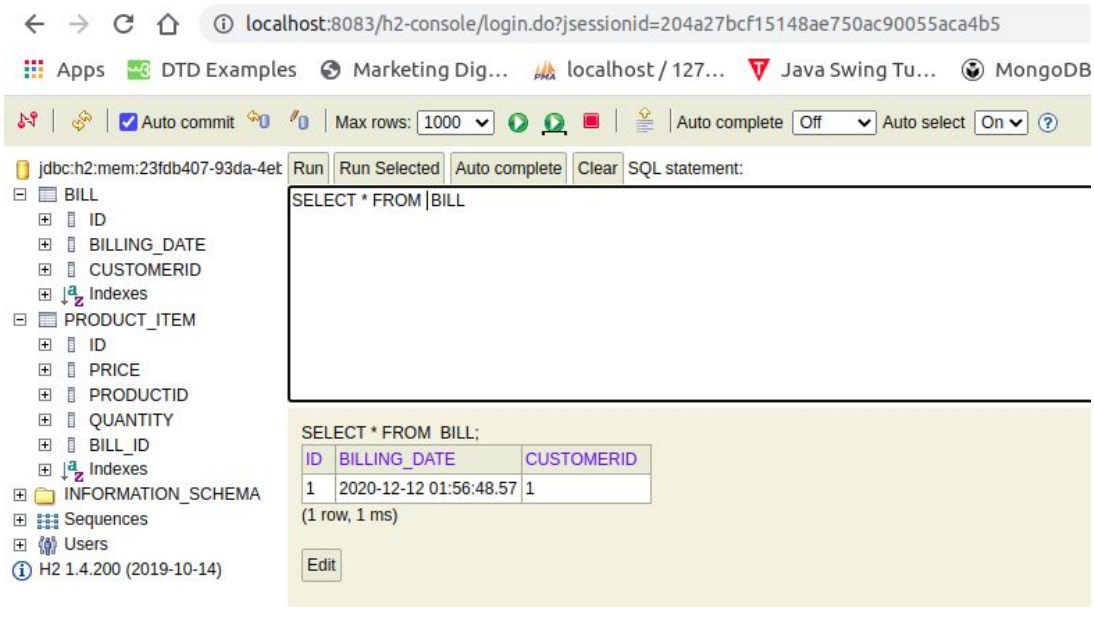
```
class ProductItem{
   @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
   private Long id;
   @Transient
   private Product product; private long productID;
   private double price; private double quantity;
   @ManyToOne
   @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
   private Bill bill;
}
@RepositoryRestResource
interface ProductItemRepository extends
       JpaRepository<ProductItem,Long>{
   List<ProductItem> findByBillId(Long billID);
}


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

@Data
class Customer{
   private Long id; private String name; private String email;
}

@FeignClient(name="inventory-service")
interface InventoryServiceClient{
   @GetMapping("/products/{id}?projection=FullProduct")
   Product findProductById(@PathVariable("id") Long id);
```

```
    @GetMapping("/products?projection=FullProduct")
    PagedModel<Product> findAll();
}

@FeignClient(name="customer-service")
interface CustomerServiceClient{
    @GetMapping("/customers/{id}?projection=FullCustomer")
    Customer findCustomerById(@PathVariable("id") Long id);
}

@RestController
class BillRestController{
    @Autowired private BillRepository billRepository;
    @Autowired private ProductItemRepository productItemRepository;
    @Autowired private CustomerServiceClient customerServiceClient;
    @Autowired private InventoryServiceClient inventoryServiceClient;
    @GetMapping("/bills/Full/{id}")
    Bill getBill(@PathVariable(name="id") Long id){
        Bill bill=billRepository.findById(id).get();
bill.setCustomer(customerServiceClient.findCustomerById(bill.getCustomer
ID()));
        bill.setProductItems(productItemRepository.findByBillId(id));
        bill.getProductItems().forEach(pi->{
pi.setProduct(inventoryServiceClient.findProductById(pi.getProductID()))
;
        });
        return bill; }
}
```

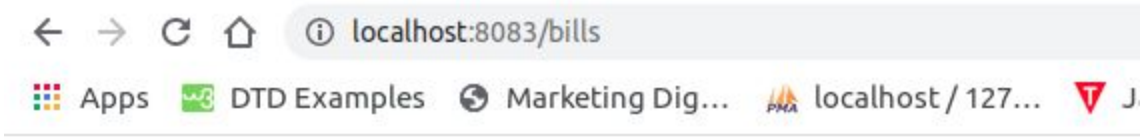
➢ **Implémentation :**



Products Table

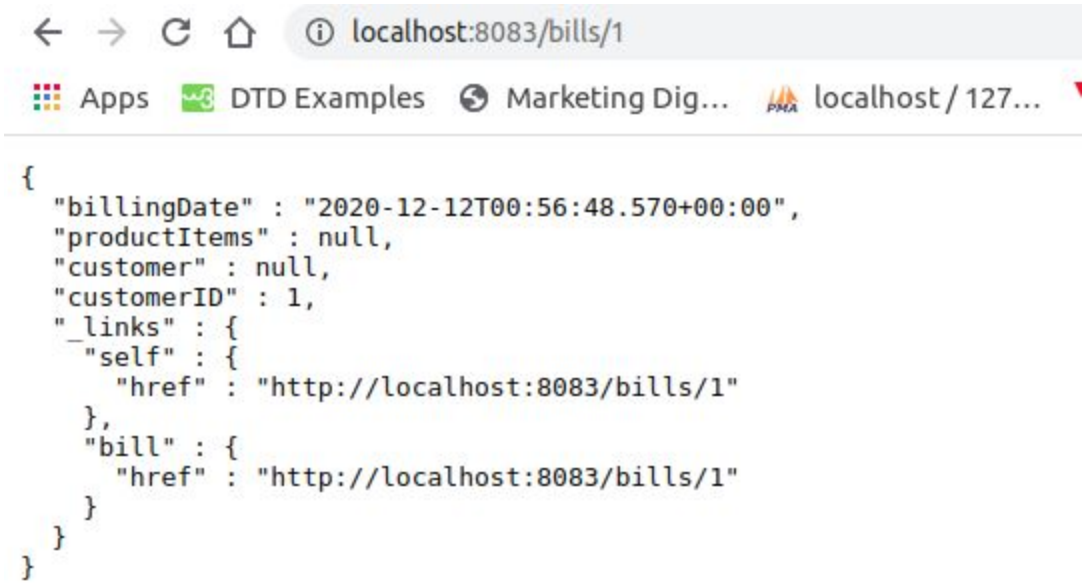


Bills Table

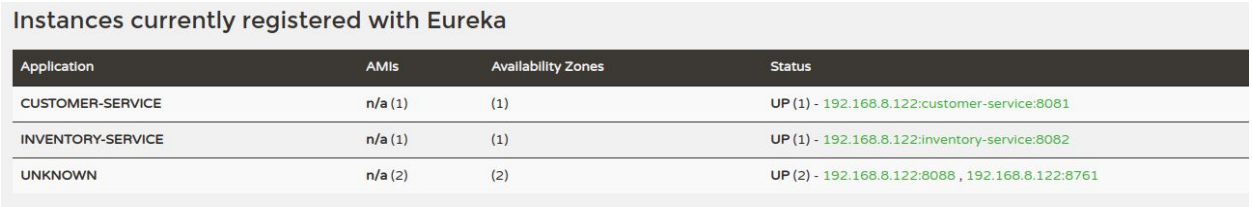



```
{
  "_embedded" : {
    "bills" : [ {
      "billingDate" : "2020-12-12T00:56:48.570+00:00",
      "productItems" : null,
      "customer" : null,
      "customerID" : 1,
      "_links" : {
        "self" : {
          "href" : "http://localhost:8083/bills/1"
        },
        "bill" : {
          "href" : "http://localhost:8083/bills/1"
        }
      }
    } ]
  },
  "_links" : {
    "self" : {
      "href" : "http://localhost:8083/bills"
    },
    "profile" : {
      "href" : "http://localhost:8083/profile/bills"
    }
  },
  "page" : {
    "size" : 20,
    "totalElements" : 1,
    "totalPages" : 1,
    "number" : 0
  }
}
```


All Bills



Bill id=1



Eureka

Bill id=1 Full Customer

VI. service d'authentification Stateless :

➢ **Code de billing service après l'ajout de service d'authentification :**

```
package org.sid.billingservice;


import com.fasterxml.jackson.annotation.JsonProperty;

import com.fasterxml.jackson.databind.ObjectMapper;

import io.jsonwebtoken.Claims;

import io.jsonwebtoken.Jwts;

import io.jsonwebtoken.SignatureAlgorithm;

import lombok.AllArgsConstructor;import lombok.Data; import
lombok.NoArgsConstructor;

import org.springframework.beans.factory.annotation.Autowired;

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.cloud.openfeign.FeignClient;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.data.jpa.repository.JpaRepository;

import
org.springframework.data.rest.core.annotation.RepositoryRestResource;

import org.springframework.hateoas.PagedModel;

import org.springframework.http.HttpMethod;

import
org.springframework.security.authentication.AuthenticationManager;

import
org.springframework.security.authentication.UsernamePasswordAuthenticati
onToken;

import
org.springframework.security.config.annotation.authentication.builders.A
uthenticationManagerBuilder;
```

```
import
org.springframework.security.config.annotation.web.builders.HttpSecurity
;

import
org.springframework.security.config.annotation.web.configuration.EnableW
ebSecurity;

import
org.springframework.security.config.annotation.web.configuration.WebSecu
rityConfigurerAdapter;

import org.springframework.security.config.http.SessionCreationPolicy;

import org.springframework.security.core.Authentication;

import org.springframework.security.core.AuthenticationException;

import org.springframework.security.core.GrantedAuthority;

import
org.springframework.security.core.authority.SimpleGrantedAuthority;

import org.springframework.security.core.context.SecurityContextHolder;

import org.springframework.security.core.userdetails.User;

import org.springframework.security.core.userdetails.UserDetails;

import org.springframework.security.core.userdetails.UserDetailsService;

import
org.springframework.security.core.userdetails.UsernameNotFoundException;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import
org.springframework.security.web.authentication.UsernamePasswordAuthenti
cationFilter;

import org.springframework.stereotype.Service;

import org.springframework.ui.Model;

import org.springframework.validation.BindingResult;

import org.springframework.web.bind.annotation.*;

import org.springframework.web.filter.OncePerRequestFilter;

import org.springframework.web.servlet.ModelAndView;

import org.springframework.web.servlet.view.RedirectView;


import javax.persistence.*;

import javax.servlet.FilterChain;
```

```
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.transaction.Transactional;
import javax.validation.Valid;
import java.io.IOException;
import java.util.*;


@SpringBootApplication
@EnableFeignClients
public class BillingServiceApplication {


    public static void main(String[] args) {
        SpringApplication.run(BillingServiceApplication.class, args);
    }
    @Bean
    public BCryptPasswordEncoder bCryptPasswordEncoder() {
        return new BCryptPasswordEncoder();
    }
    @Bean
    CommandLineRunner start(AccountService accountService,BillRepository
billRepository, ProductItemRepository productItemRepository,
InventoryServiceClient inventoryServiceClient, CustomerServiceClient
customerServiceClient) {
        return args -> {
            Bill bill = new Bill();
            bill.setBillingDate(new Date());
            Customer customer =
customerServiceClient.findCustomerById(1L);
            System.out.println(customer.toString());
```

```
            bill.setCustomerID(customer.getId());
            billRepository.save(bill);
            inventoryServiceClient.findAll().getContent().forEach(p -> {
                productItemRepository.save(new ProductItem(null, null,
p.getId(), p.getPrice(), (int) (1 + Math.random() * 1000), bill));
            });
            accountService.saveRole(new AppRole(null,"USER"));
            accountService.saveRole(new AppRole(null,"ADMIN"));
            accountService.saveUser(new AppUser(null,"user","1234",
null));
            accountService.saveUser(new AppUser(null,"admin","1234",
null));
            accountService.addRoleToUser("user", "USER");
            accountService.addRoleToUser("admin", "USER");
            accountService.addRoleToUser("admin", "ADMIN");
        };
    }
}
```

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

```
    private long customerID;
}
@RepositoryRestResource
interface BillRepository extends JpaRepository<Bill,Long> {}

@Entity @Data @NoArgsConstructor @AllArgsConstructor
class ProductItem{
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Transient
    private Product product; private long productID;
    private double price; private double quantity;
    @ManyToOne
    @JsonProperty(access = JsonProperty.Access.WRITE_ONLY)
    private Bill bill;
}
@RepositoryRestResource
interface ProductItemRepository extends
        JpaRepository<ProductItem,Long>{
    List<ProductItem> findByBillId(Long billID);
}


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

@Data
class Customer{
    private Long id; private String name; private String email;
```

```
}

@FeignClient(name="inventory-service")
interface InventoryServiceClient{
    @GetMapping("/products/{id}?projection=FullProduct")
    Product findProductById(@PathVariable("id") Long id);
    @GetMapping("/products?projection=FullProduct")
    PagedModel<Product> findAll();
}

@FeignClient(name="customer-service")
interface CustomerServiceClient{
    @GetMapping("/customers/{id}?projection=FullCustomer")
    Customer findCustomerById(@PathVariable("id") Long id);
}

@RestController
class BillRestController{
    @Autowired private BillRepository billRepository;
    @Autowired private ProductItemRepository productItemRepository;
    @Autowired private CustomerServiceClient customerServiceClient;
    @Autowired private InventoryServiceClient inventoryServiceClient;

    @GetMapping("/bills")
    public List<Bill> listTasks(){
        return billRepository.findAll();
    }
    @PostMapping("/bills")
    public Bill save(@RequestBody Bill task){
        return billRepository.save(task);
    }
```

```
    @GetMapping("/bills/Full/{id}")
    Bill getBill(@PathVariable(name="id") Long id){
        Bill bill=billRepository.findById(id).get();

bill.setCustomer(customerServiceClient.findCustomerById(bill.getCustomer
ID()));
        bill.setProductItems(productItemRepository.findByBillId(id));
        bill.getProductItems().forEach(pi->{

pi.setProduct(inventoryServiceClient.findProductById(pi.getProductID()))
;
        });
        return bill; }
    @RequestMapping(value = "/admin/bills/delete",method=
RequestMethod.GET)
    public RedirectView delete(Model model,
@RequestParam(name="id",required = true)Long id){
        billRepository.deleteById(id);
        return new RedirectView("/bills");
    }

    @RequestMapping(value="/admin/form",method= RequestMethod.GET)
    public ModelAndView form(Model model,
@RequestParam(name="id",defaultValue = "0") Long id){
        Bill p = (billRepository.existsById(id))?
billRepository.getOne(id):new Bill();
        model.addAttribute("bill", p);
        return new ModelAndView("form");
    }
    @RequestMapping(value="/admin/save",method=RequestMethod.POST)
    public ModelAndView save(Model model, @Valid Bill p, BindingResult
bindingResult){
        if(bindingResult.hasErrors()) return new ModelAndView("form");
```

```
        billRepository.save(p);
        model.addAttribute("bill", p);
        return new ModelAndView("confirmation");
    }
    @RequestMapping(value="/403",method= RequestMethod.GET)
    public ModelAndView error(){
        return new ModelAndView("error/403");
    }
}



/* Security */

@Entity
@Data @AllArgsConstructor @NoArgsConstructor
class AppRole {
    @Id @GeneratedValue
    private Long id;
    private String role;
}


@Entity
@Data @AllArgsConstructor @NoArgsConstructor
class AppUser {
    @Id @GeneratedValue
    private Long id;
    private String username;
    private String password;
    @ManyToMany(fetch=FetchType.EAGER)
```

```
    private Collection<AppRole> roles=new ArrayList<>();
}

interface AppUserRepository extends JpaRepository<AppUser, Long> {
    AppUser findByUsername(String username);
}
interface AppRoleRepository extends JpaRepository<AppRole,Long>{
    AppRole findByRole(String role);
}
interface AccountService {
    AppUser saveUser(AppUser u);
    AppRole saveRole(AppRole r);
    AppUser findUserByUsername(String username);
    void addRoleToUser(String username,String role);
}

@Service
@Transactional
class AccountServiceImpl implements AccountService {
    @Autowired
    private AppUserRepository userRepository;
    @Autowired
    private AppRoleRepository roleRepository;
    @Autowired
    private BCryptPasswordEncoder bCryptPasswordEncoder;

    @Override
    public AppUser saveUser(AppUser u) {
        u.setPassword(bCryptPasswordEncoder.encode(u.getPassword()));
        return userRepository.save(u);
    }
```

```
    @Override
    public AppRole saveRole(AppRole r) {
        return roleRepository.save(r);
    }
    @Override
    public AppUser findUserByUsername(String username) {
        return userRepository.findByUsername(username);
    }
    @Override
    public void addRoleToUser(String username, String roleName) {
        AppUser user=userRepository.findByUsername(username);
        AppRole role=roleRepository.findByRole(roleName);
        user.getRoles().add(role);
    }
}

@Data @AllArgsConstructor @NoArgsConstructor
class RegistrationForm {
    private String username;
    private String password;
    private String repassword;
}

@RestController
class UserController {
    @Autowired
    private AccountService accountService;
    @PostMapping("/users")
    public AppUser signUp(@RequestBody RegistrationForm data) {
        String username=data.getUsername();
        AppUser user=accountService.findUserByUsername(username);
```

```
        if(user!=null) throw new RuntimeException("This user already
exists, Try with an other username");

                String password=data.getPassword(); String
repassword=data.getRepassword();

        if(!password.equals(repassword))

            throw new RuntimeException("You must confirm your password");

        AppUser u=new AppUser(); u.setUsername(username);
u.setPassword(password);

        accountService.saveUser(u);

        accountService.addRoleToUser(username, "USER");

        return (u);

    }

}

@Configuration

@EnableWebSecurity

class SecurityConfig extends WebSecurityConfigurerAdapter {

    @Autowired

    private UserDetailsService userDetailsService;

    @Autowired

    private BCryptPasswordEncoder bCryptPasswordEncoder;


    @Override

    protected void configure(AuthenticationManagerBuilder auth) throws
Exception {

        auth.userDetailsService(userDetailsService)

                .passwordEncoder(bCryptPasswordEncoder);

    }


    @Override

    protected void configure(HttpSecurity http) throws Exception {

        http.csrf().disable()

// don't create session
```

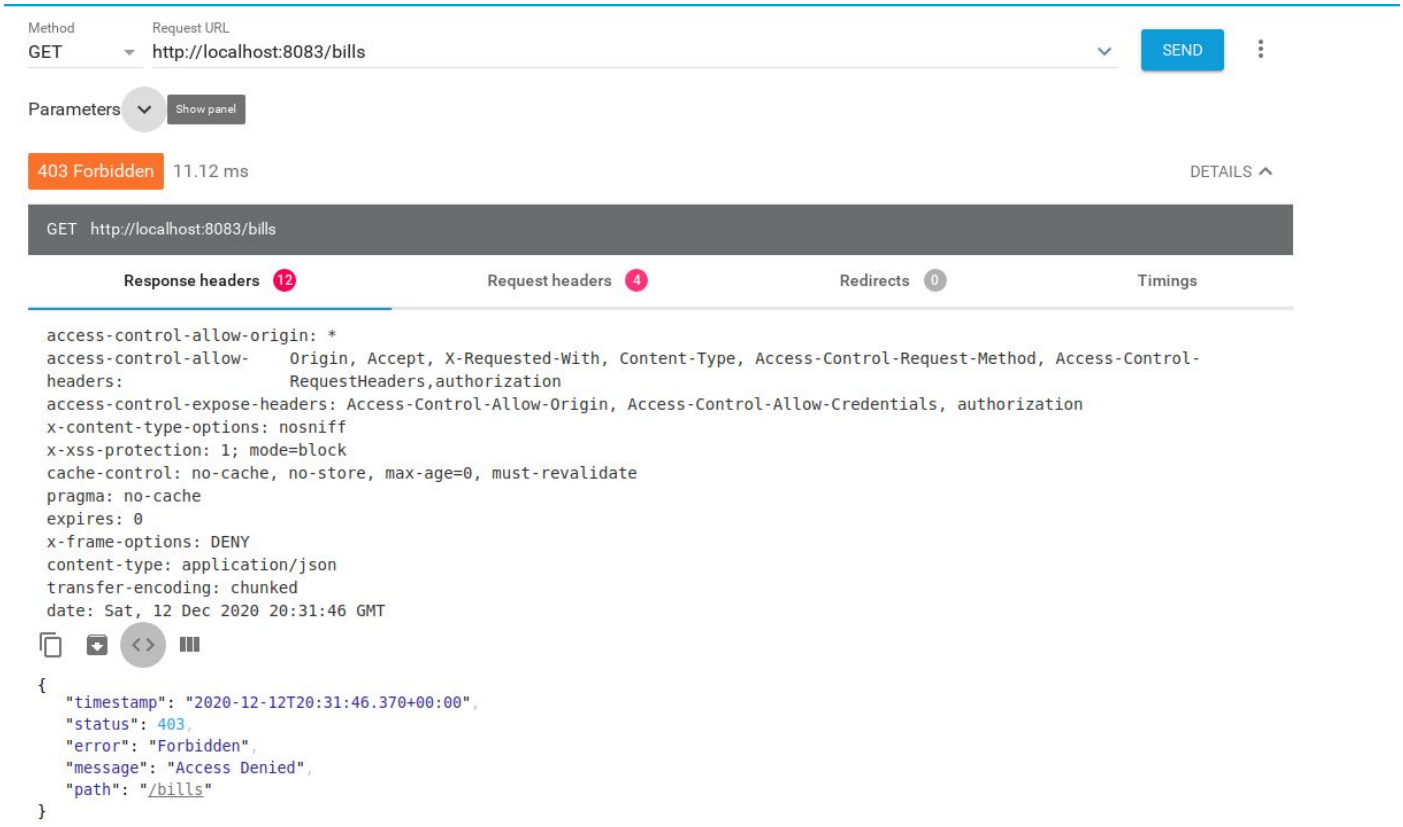
```
.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)
                .and()
                .authorizeRequests()
                .antMatchers("/users/**", "/login/**")
                .permitAll()
                .antMatchers(HttpMethod.POST, "/bills/**").hasAuthority("ADMIN")
                .anyRequest().authenticated()
                .and()
                .addFilter(new JWTAuthenticationFilter(authenticationManager()))
                .addFilterBefore(new JWTAuthorizationFilter(),
                        UsernamePasswordAuthenticationFilter.class);
    }
}
class SecurityConstants {
    public static final String SECRET = "elanssarihamza@gmail.com";
    public static final long EXPIRATION_TIME = 864_000_000;
    public static final String TOKEN_PREFIX = "Bearer ";
    public static final String HEADER_STRING = "Authorization";
}


@Service
class UserDetailsServiceImpl implements UserDetailsService {
    @Autowired
    private AccountService accountService;
    @Override
    public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {
        AppUser u=accountService.findUserByUsername(username);
        if(u==null) throw new UsernameNotFoundException(username);
```

```
        Collection<GrantedAuthority> authorities=new ArrayList<>();
        u.getRoles().forEach(r->{
            authorities.add(new SimpleGrantedAuthority(r.getRole()));
        });
        return new User(u.getUsername(), u.getPassword(), authorities);
    }
}


class JWTAuthenticationFilter extends
UsernamePasswordAuthenticationFilter {
    private AuthenticationManager authenticationManager;
    public JWTAuthenticationFilter(AuthenticationManager
authenticationManager) {
        super();
        this.authenticationManager = authenticationManager;
    }
    @Override
    public Authentication attemptAuthentication(HttpServletRequest
request,
                                                HttpServletResponse
response) throws AuthenticationException {
        AppUser user=null;
        try {
            user = new ObjectMapper().readValue(request.getInputStream(),
AppUser.class);
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
        return authenticationManager.authenticate(
                new UsernamePasswordAuthenticationToken(
                        user.getUsername(),
                        user.getPassword()
                ));
```

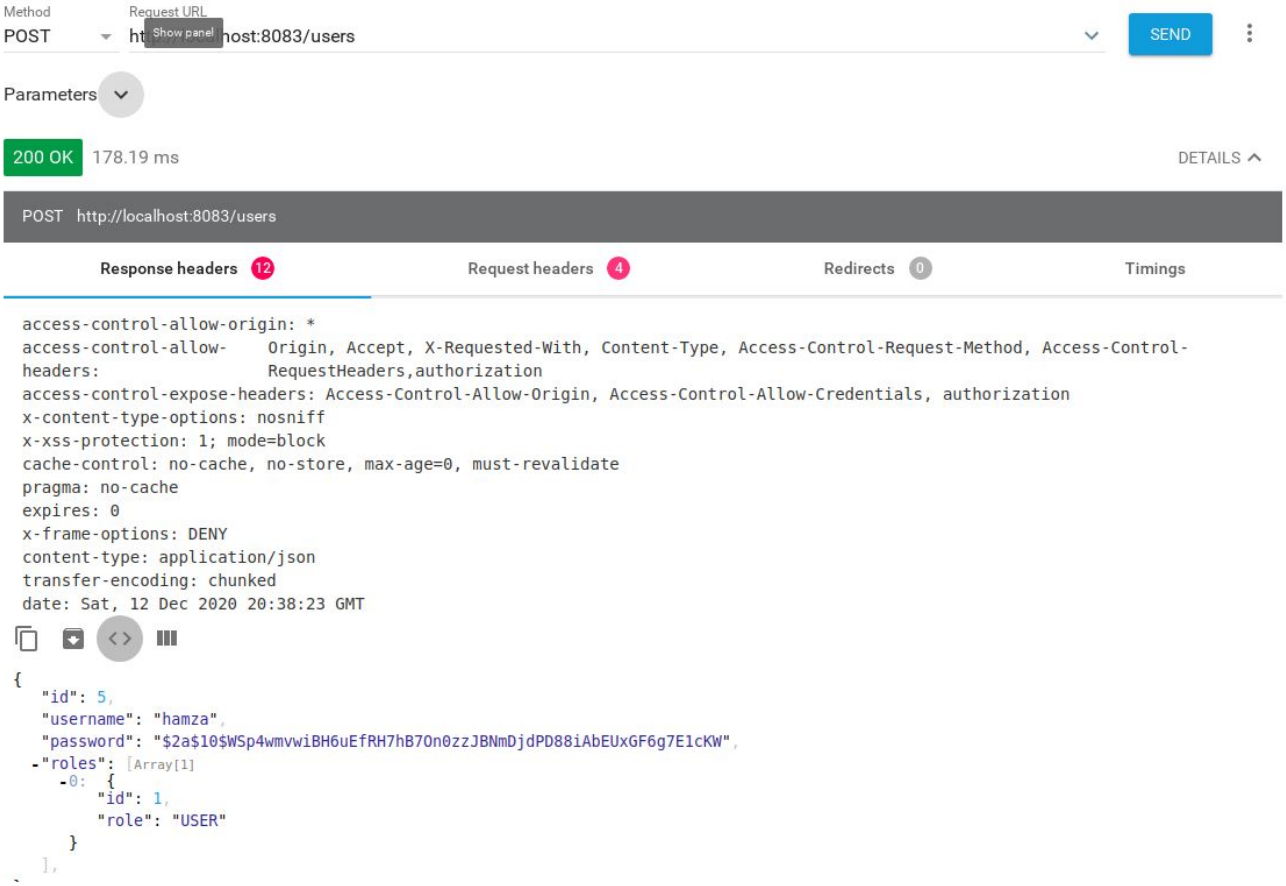
```
    }
    @Override
    protected void successfulAuthentication(HttpServletRequest request,
HttpServletResponse
            response, FilterChain chain,
                                                Authentication authResult)
throws IOException, ServletException {
        User springUser=(User)authResult.getPrincipal();
        String jwtToken= Jwts.builder()
                .setSubject(springUser.getUsername())
                .setExpiration(new

Date(System.currentTimeMillis()+SecurityConstants.EXPIRATION_TIME))
                .signWith(SignatureAlgorithm.HS512,
SecurityConstants.SECRET)
                .claim("roles", springUser.getAuthorities())
                .compact();
        response.addHeader(SecurityConstants.HEADER_STRING,
                SecurityConstants.TOKEN_PREFIX+jwtToken);
    }
}

class JWTAuthorizationFilter extends OncePerRequestFilter {
    @Override
    protected void doFilterInternal(HttpServletRequest request,
HttpServletResponse response,
                                    FilterChain chain)
            throws IOException, ServletException {
        response.addHeader("Access-Control-Allow-Origin", "*");
        response.addHeader("Access-Control-Allow-Headers", "Origin,
Accept, X-Requested-With, Content-Type, Access-Control-Request-Method,
Access-Control-RequestHeaders,authorization");
```

```
                response.addHeader("Access-Control-Expose-Headers",
"Access-Control-Allow-Origin, Access-Control-Allow-Credentials,
authorization");

        if(request.getMethod().equals("OPTIONS")){

            response.setStatus(HttpServletResponse.SC_OK);

        }

        else {

            String
jwtToken=request.getHeader(SecurityConstants.HEADER_STRING);

            if(jwtToken==null ||
!jwtToken.startsWith(SecurityConstants.TOKEN_PREFIX)) {

                chain.doFilter(request, response); return;

            }

            Claims claims=Jwts.parser()

                    .setSigningKey(SecurityConstants.SECRET)


.parseClaimsJws(jwtToken.replace(SecurityConstants.TOKEN_PREFIX,""))

                    .getBody();

            String username=claims.getSubject();

            ArrayList<Map<String, String>> roles=(ArrayList<Map<String,
String>>)

                    claims.get("roles");

            Collection<GrantedAuthority> authorities=new ArrayList<>();

            roles.forEach(r->{

                authorities.add(new
SimpleGrantedAuthority(r.get("authority")));

            });

            UsernamePasswordAuthenticationToken authenticationToken=

                    new UsernamePasswordAuthenticationToken(username,
null,authorities);


SecurityContextHolder.getContext().setAuthentication(authenticationToken
);

            chain.doFilter(request, response);

        }}}
```

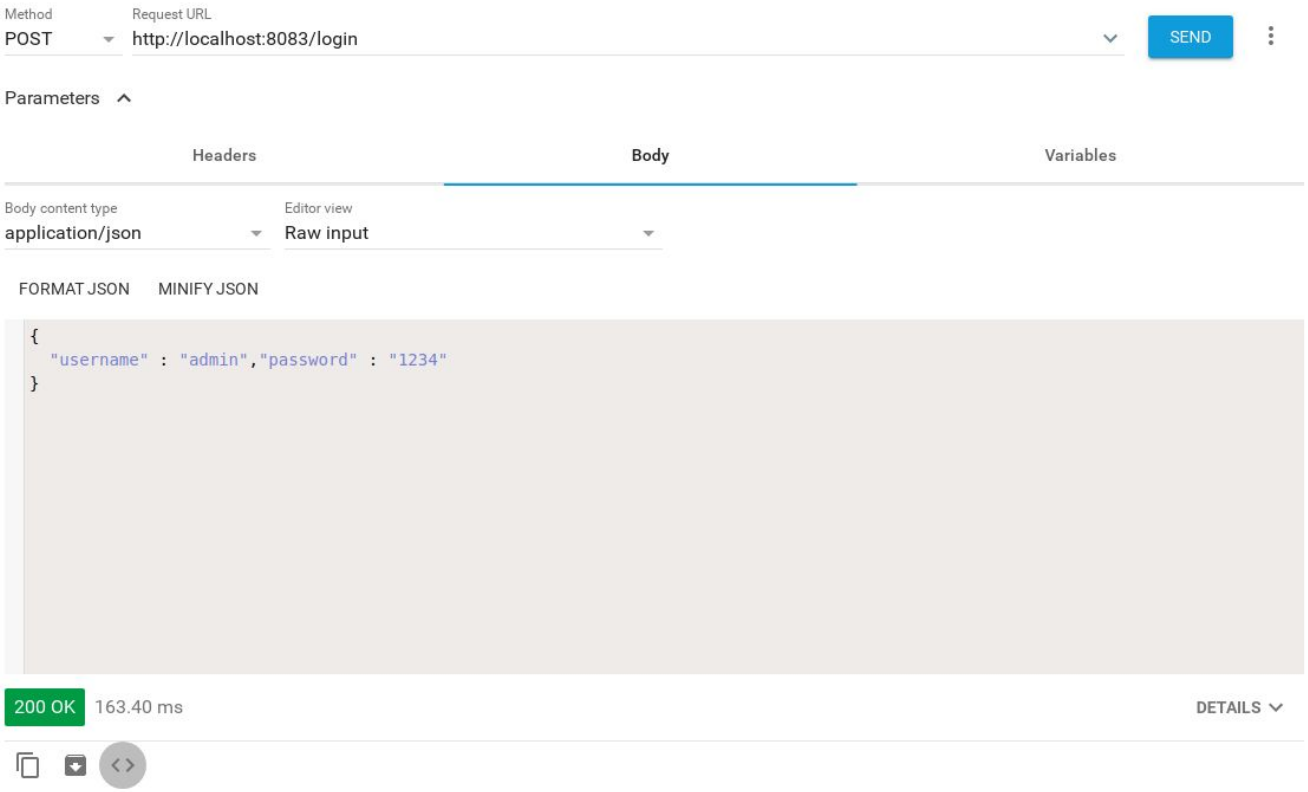
➢ **Implémentation :**



Test : Consulter les listes des bills



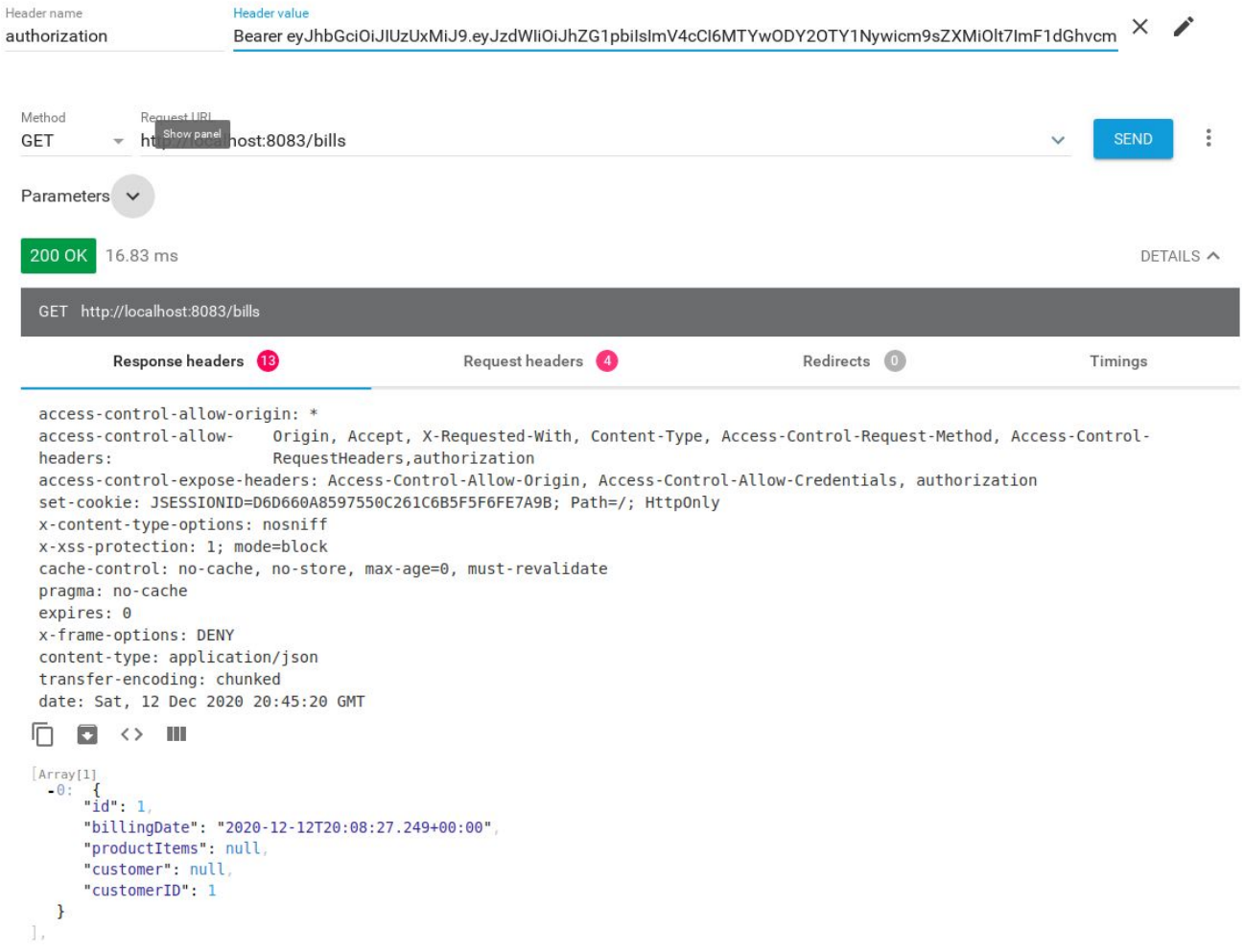
Ajouter un utilisateur



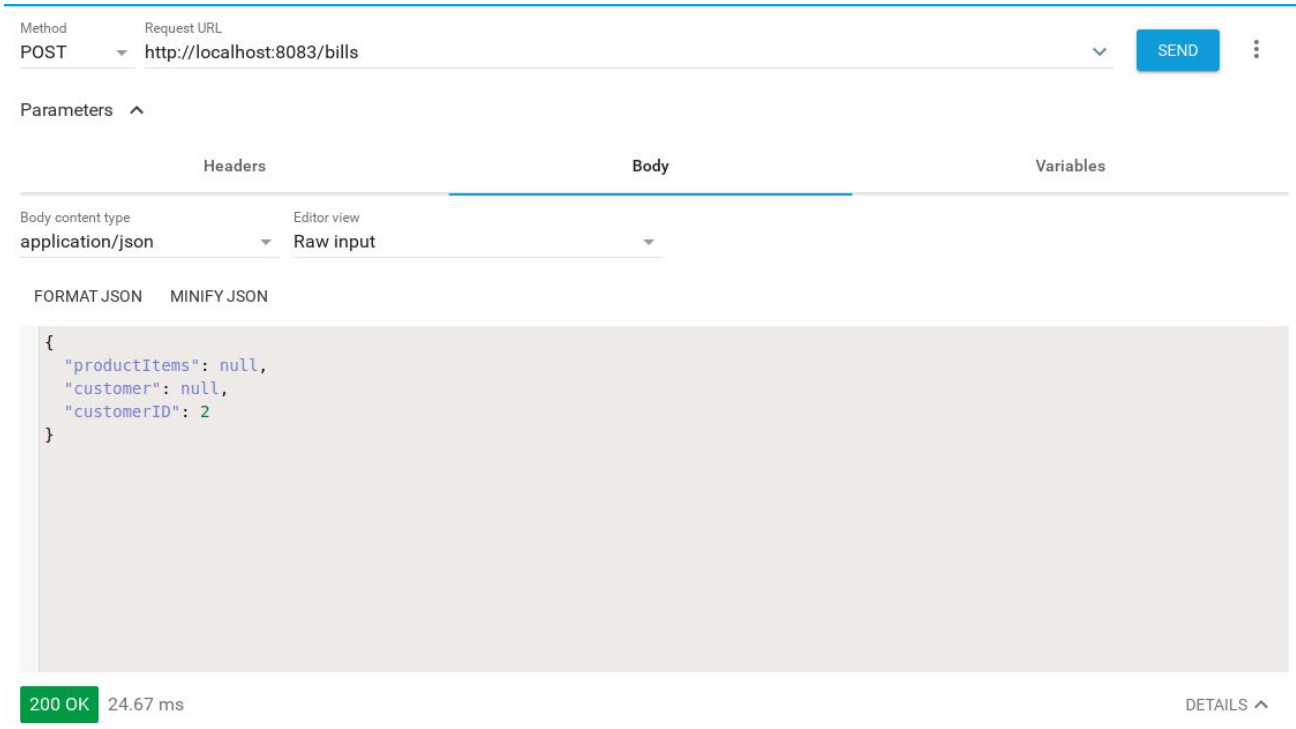
Authentification avec le rôle admin

```
authori Bearer
zation: eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJhZG1pbiIsImV4cCI6MTYwODY2OTY1Nywicm9sZXMiOlt7ImF1dGhvcml0eSI6IkFETUlOIn0seyJhdXRob3Jpd
        HkiOiJVU0VSIn1dfQ.UfN3tOReZP5mR3bT8V6PLC-Fuj99HlrXCsV56RQ_UlAA-heGoVZPX9QK2ZRZq9aDdGM34LXYLsujyaO5kcEF7Q
```

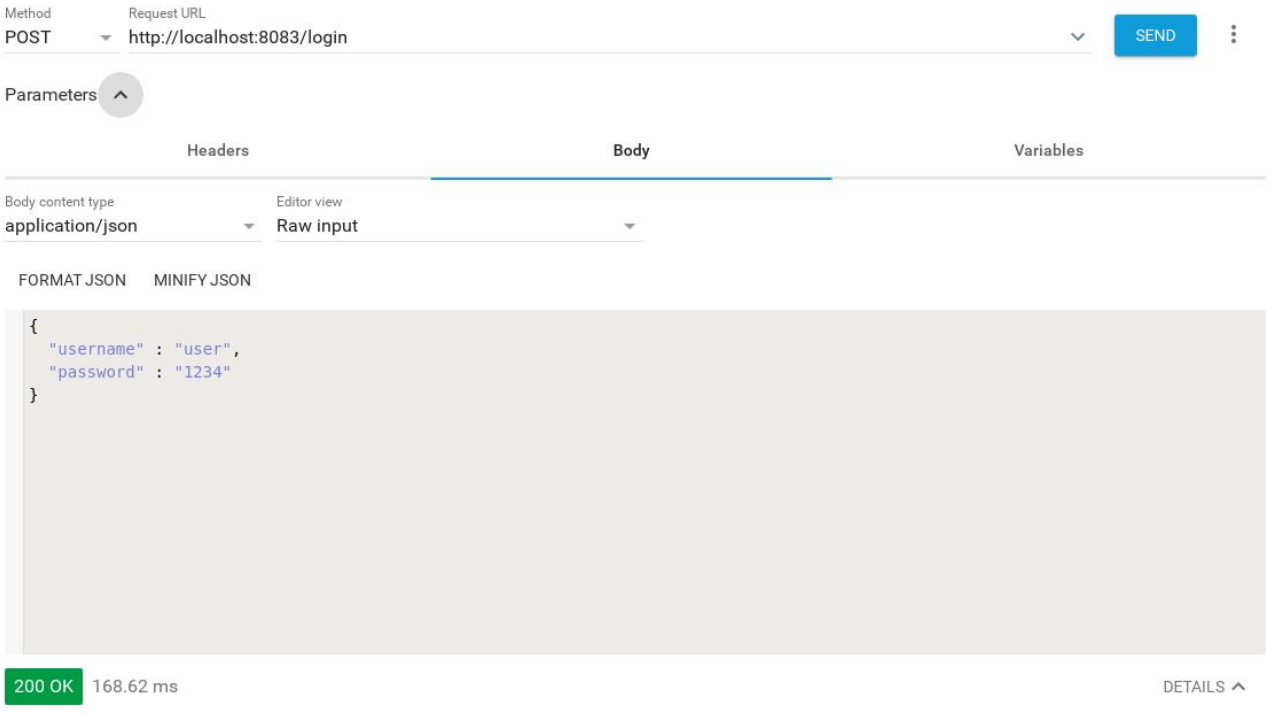
Code d'authentification



Liste des bills



Ajouter un bill



Authentification avec le rôle user

Code d'authentification

Conclusion :

Ce TP est très important pour moi car j'ai utilisé le Framework Spring (Spring Cloud Gateway, Spring Security et Spring Boot) et Eureka Server, aussi la différence entre la configuration Statique qui basée sur le fichier application.yml et la configuration dynamique des routes avec Eureka.cela m'a donné une idée sur les micro services et Spring Cloud, aussi tester et améliorer mes connaissances dans le Framework Spring.