


RAPPORT DU PROJET

Rhouny Mohamed Amine




**ECOLE MAROCAINE DES
SCIENCES DE L'INGENIEUR**
Membre de
HONORIS UNITED UNIVERSITIES





Micro Services avec Spring Cloud


- Spring Cloud Gateway
- Eureka Discovery
- Open Feign Rest Client
- Hystrix DashBoard



Mohamed Youssfi
Laboratoire Signaux Systèmes Distribués et Intelligence Artificielle (SSDIA)
ENSET, Université Hassan II Casablanca, Maroc
Email : med@youssfi.net
Supports de cours : <http://fr.slideshare.net/mohamedyoussfi9>
Chaîne vidéo : <http://youtube.com/mohamedYoussfi>
Recherche : http://www.researchgate.net/profile/Youssfi_Mohamed/publications

Sécurité des application Web Mobiles et Systèmes Distribués Micro services



Mohamed Youssfi
Laboratoire Signaux Systèmes Distribués et Intelligence Artificielle (SSDIA)
ENSET, Université Hassan II Casablanca, Maroc
Email : med@youssfi.net
Supports de cours : <http://fr.slideshare.net/mohamedyoussfi9>
Chaîne vidéo : <http://youtube.com/mohamedYoussfi>
Recherche : http://www.researchgate.net/profile/Youssfi_Mohamed/publications

1. Introduction	4
Micro-service:	4
Sécurité:	4
2. Besoins et objectifs du projet	5
a. Critique de L'existant	5
Micro-Services :	5
Sécurité :	5
b. Solution:	6
Micro services	6
Sécurité:	6
3. Gestion du projet	7
a. Exigences:	7
Partie Micro Service :	7
Partie Sécurité :	7
b. Conception	8
Micro-Services:	8
Sécurité	9
c. Les exigences fonctionnelles de l'application sont :	10
1. Partie Backend	10
2. Partie Frontend	10
4. Développement technique	11
a. Logiciel utilisé:	11
5. Réalisation	13
1. Partie Backend	13
Micro-services:	13
Customer-service	13
Inventory-service	16
Gateway-service	18
Eureka Discovery Service : Dynamic Routing	19
Billing-service	21
Sécurité	26
Code du Micro Service d'authentification : Couche DAO	27

Couche Service	28
Couche Web : Rest API pour la gestion des utilisateurs et les groupes	30
Spring Security Configuration : JWT Authentication Filter	30
Spring Security Configuration : JWT Authorization Filter	32
Tests : Authentification	33
Tests : Consulter les utilisateurs sans JWT	33
Tests : Consulter les utilisateurs avec JWT qui a expiré	34
Tests : Consulter les utilisateurs avec un JWT Valide	34
Tests : Ajouter un utilisateur avec le Rôle USER	35
Tests : Ajouter un utilisateur avec le Rôle ADMIN	35
Refresh token	36
2.Partie Frontend:	37
Page produits:	38
Page Customers :	40
Page Affichage Bill :	41
Page Ajout Bill :	43
Conclusion	46

1. Introduction

Micro-service:

Les architectures de microservices sont la «nouvelle norme». La création d'applications petites, autonomes et prêtes à être exécutées peut apporter une grande flexibilité et une meilleure résilience à votre code. Les nombreuses fonctionnalités spécialement conçues de Spring Boot facilitent la création et l'exécution de vos microservices en production à grande échelle. Et n'oubliez pas qu'aucune architecture de micro service n'est complète sans Nuage de printemps - faciliter l'administration et booster votre tolérance aux pannes.

Sécurité:

Spring Security est un cadre d'authentification et de contrôle d'accès puissant et hautement personnalisable. C'est la norme de facto pour sécuriser les applications Spring.

Spring Security est un framework qui se concentre sur la fourniture à la fois d'authentification et d'autorisation aux applications Java. Comme tous les projets Spring, la vraie puissance de Spring Security réside dans la facilité avec laquelle il peut être étendu pour répondre aux exigences personnalisées

2. Besoins et objectifs du projet

a. Critique de L'existant

Micro-Services :

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

- Elles centralisent tous les besoins fonctionnels
- Elles sont réalisées dans une seule technologie.
- Chaque modification nécessite de :
- Tester les régressions
- Redéployer toute l'application
- Difficile à faire évoluer au niveau fonctionnel
- Livraison en bloc (Le client attend beaucoup de temps pour commencer à voir les premières versions)

Sécurité :

81% des entreprises françaises ont été visées par des attaques en 2015. Le coût moyen d'une violation de sécurité est estimé à 800 000 euros. Et pour réparer les dégâts, 9 semaines en moyenne sont nécessaires.

La sécurité des applications web est encore plus alarmante. D'après un rapport de Positive Technologies, 100% des applications analysées contiennent des failles de sécurité dont 85% peuvent toucher les utilisateurs.

b. Solution:

Micro services

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, Node JS, 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 combinent, ces micro services peuvent réaliser des opérations très complexes.

Securité:

Deux types de modèles d'authentification :

- Statful : Les données de la session sont enregistrés coté serveur d'authentification
- Stateless : les données de la session sont enregistrés dans un jeton d'authentification délivré au client.

3. Gestion du projet

a. Exigences:

Partie Micro Service :

- Créer une application basée sur deux services métiers:
 - Service des clients
 - Service d'inventaire
 - Service Facturation
 - Services Externes : RapidAPI
 - L'orchestration des services se fait via les services techniques de Spring 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

Partie Sécurité :

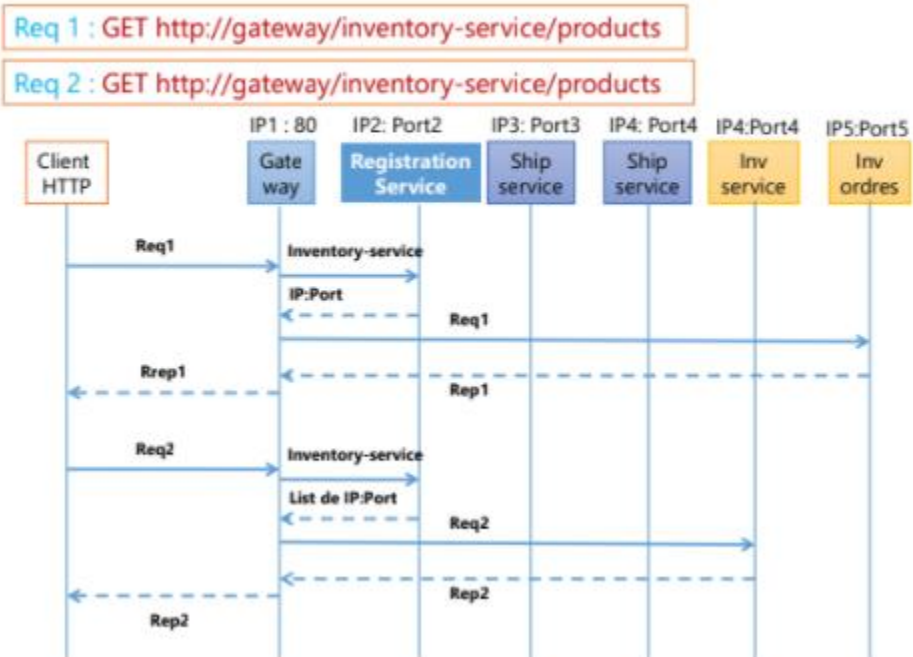
Créer un micro service d'authentification en utilisant Spring Security et JWT

- 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

b. Conception

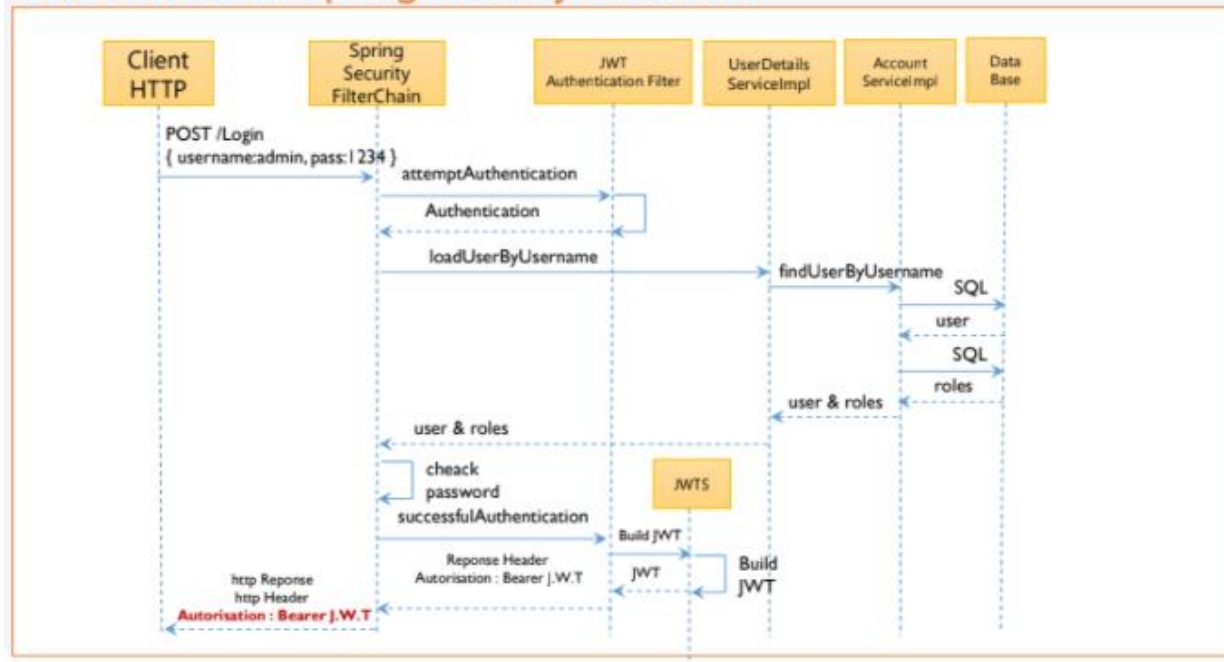
Micro-Services:

Consulter les services via le service proxy

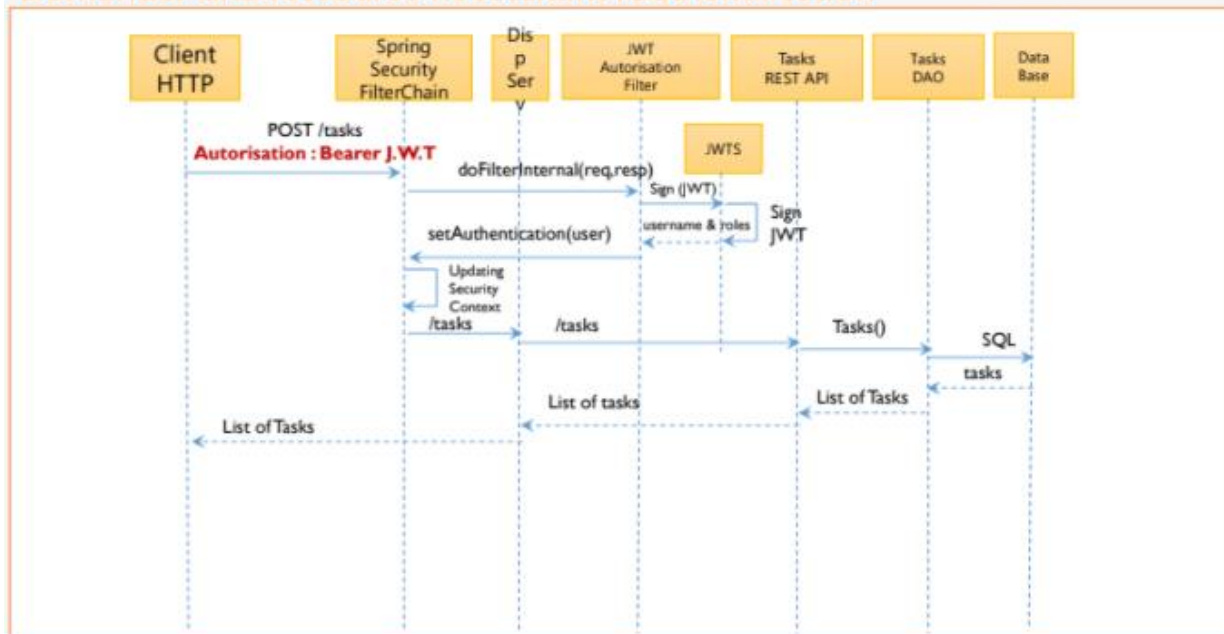


Sécurité

Authentification Spring Security avec JWT



Demander une ressource necessitant l'authentification



c. Les exigences fonctionnelles de l'application sont :

1. Partie Backend

La partie backend est basée sur Spring et se compose de plusieurs micro services, des couches DAO, Service et Web

- La couche DAO est basée sur Spring Data, JPA, Hibernate
- La couche Métier est définie par une interface et une implémentation quelques spécifications fonctionnelles qui nécessitent des calculs ou des traitements particuliers
- La couche Web est basée sur des API Restful basée sur Spring Data Rest ou un RestController

2. Partie Frontend

la partie Frontend est réalisée en utilisant angular et se compose de partie html et partie services

- S'occupe de la présentation des IHM côté client utilise le langage HTML, CSS, JavaScript
- la communication entre la partie Frontend et la partie Backend se fait en utilisant le protocole HTTP

4. Développement technique

a. Logiciel utilisé:

Spring



Spring Boot est un framework qui facilite le développement d'applications fondées sur Spring en offrant des outils permettant d'obtenir une application packagée en *jar* , totalement autonome.

Spring Boot est un framework Java open source utilisé pour créer un micro-service. Il est développé par Pivotal Team et est utilisé pour créer des applications de printemps autonomes et prêtes pour la production. Ce chapitre vous donnera une introduction à Spring Boot et vous familiarisera avec ses concepts de base.

AngularJS



AngularJS vous permet de mieux organiser votre code Javascript, en vue de créer des sites web dynamiques (bien qu'à la base, AngularJS n'avait pas été pensé pour du contenu dynamique). Basé du côté client, il vous permet de créer de l'HTML interactif. A part la librairie

d'AngularJS, il n'est pas nécessaire de créer du Javascript.

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

IntelliJ



IntelliJ IDEA est un environnement de développement intégré (IDE) pour les langages JVM conçu pour maximiser la productivité des développeurs. Il effectue les tâches routinières et répétitives pour vous en fournissant une saisie intelligente du code, une analyse statique du code et des refactorisations, et vous permet de vous concentrer sur le côté positif du développement logiciel, ce qui le rend non seulement productif mais également une expérience agréable.

5. Réalisation

1.Partie Backend

Micro-services:

Customer-service

```
import ...

@Entity
@Data
@AllArgsConstructor
@NoArgsConstructor
@ToString
public class Customer {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String name;
    private String email;
}
```

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

import ...

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

```

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: "gg", email: "gg@gmail.com"));
            customerRepository.save(new Customer( id: null, name: "ii", email: "ii@gmail.com"));
            customerRepository.save(new Customer( id: null, name: "rr", email: "rr@gmail.com"));
            customerRepository.findAll().forEach(c -> {
                System.out.println(c.toString());
            });
        };
    }
}

```

```

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

```

```

{
  - _embedded: {
    - customers: [
      - {
        id: 1,
        name: "gg",
        email: "gg@gmail.com",
        - _links: {
          - self: {
            href: "http://localhost:8081/customers/1"
          },
          - customer: {
            href: "http://localhost:8081/customers/1"
          }
        }
      },
      - {
        id: 2,
        name: "ii",
        email: "ii@gmail.com",
        - _links: {
          - self: {
            href: "http://localhost:8081/customers/2"
          },
          - customer: {
            href: "http://localhost:8081/customers/2"
          }
        }
      },
      - {
        id: 3,
        name: "rr",
        email: "rr@gmail.com",
        - _links: {
          - self: {
            href: "http://localhost:8081/customers/3"
          },
          - customer: {
            href: "http://localhost:8081/customers/3"
          }
        }
      }
    ]
  }
}

```

Customer-service : Base de données H2 (<http://localhost:8081/h2-console>)

jdbc:h2:mem:customer-db
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	gg@gmail.com	gg
2	ii@gmail.com	ii
3	rr@gmail.com	rr

(3 rows, 9 ms)

Inventory-service

```
package org.sid.inventoryservice.entities;

import ...

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

```
@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: "Ord", price: 7000, quantity: 100));
            productRepository.save(new Product( id: null, name: "Imp", price: 584, quantity: 48));
            productRepository.save(new Product( id: null, name: "Phone", price: 40000, quantity: 15));
            productRepository.findAll().forEach(c -> {
                System.out.println(c.toString());
            });
        };
    }
}
```

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

import ...

@RepositoryRestResource
@CrossOrigin("*")
public interface ProductRepository extends JpaRepository<Product, Long> {
}
```



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

```
{
  "_embedded": {
    "products": [
      {
        "id": 1,
        "name": "Ord",
        "price": 7000,
        "quantity": 100,
        "_links": {
          "self": {
            "href": "http://localhost:8082/products/1"
          },
          "product": {
            "href": "http://localhost:8082/products/1"
          }
        }
      },
      {
        "id": 2,
        "name": "Imp",
        "price": 584,
        "quantity": 48,
        "_links": {
          "self": {
            "href": "http://localhost:8082/products/2"
          }
        }
      }
    ]
  }
}
```

jdbc:h2:mem:products-db

- PRODUCT
- INFORMATION_SCHEMA
- Sequences
- Users
- H2 1.4.200 (2019-10-14)

Run Run Selected Auto complete Clear SQL statement:

SELECT * FROM PRODUCT

SELECT * FROM PRODUCT;

ID	NAME	PRICE	QUANTITY
1	Ord	7000.0	100.0
2	Imp	584.0	48.0
3	Phone	40000.0	15.0

(3 rows, 3 ms)

Gateway-service

```
import org.springframework.context.annotation.Bean;

@SpringBootApplication
public class GatewayApplication {

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

    // @Bean
    /**RouteLocator routeLocator(RouteLocatorBuilder builder){
        return builder.routes()
            .route((r)->r.path("/customers/**").uri("lb://CUSTOMER-SERVICE"))
            .route((r)->r.path("/products/**").uri("lb://PRODUCT-SERVICE"))
            .build();
    }*/

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

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

Static routes configuration: application.yml

```
spring:
  cloud:
    gateway:
      routes:
        - id : r1
          uri : http://localhost:8081/
          predicates:
            - Path= /customers/**
        - id : r2
          uri : http://localhost:8082/
          predicates:
            - Path= /products/**
```

Static routes configuration with Discovery Service

```
public static void main(String[] args) { SpringApplication.run(GatewayApplication.class, args); }  
//@Bean  
/*RouteLocator routeLocator(RouteLocatorBuilder builder){  
    return builder.routes()  
        .route((r)->r.path("/customers/**").uri("lb://CUSTOMER-SERVICE"))  
        .route((r)->r.path("/products/**").uri("lb://PRODUCT-SERVICE"))  
        .build();  
}*/
```

Dynamic routes configuration with Discovery Service

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

Eureka Discovery Service : Dynamic Routing

```
package com.sid.eurekadiscovery;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.cloud.netflix.eureka.server.EnableEurekaServer;  
  
@SpringBootApplication  
@EnableEurekaServer  
public class EurekaDiscoveryApplication {  
  
    public static void main(String[] args) { SpringApplication.run(EurekaDiscoveryApplication.class, args); }  
  
}
```

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

The screenshot shows the Spring Eureka web interface. The header includes the 'spring Eureka' logo and navigation links for 'HOME' and 'LAST 1000 SINCE STARTUP'. The main content area is divided into sections: 'System Status', 'DS Replicas', and 'Instances currently registered with Eureka'.

System Status

Environment	N/A	Current time	2021-01-08T17:15:18 +0100
Data center	N/A	Uptime	22:37
		Lease expiration enabled	true
		Renews threshold	8
		Renews (last min)	16

DS Replicas

localhost

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
-------------	------	--------------------	--------

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

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

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

Billing-service

```
import org.sid.billingService.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;  
}
```

```

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;
    @Transient
    private String productName;
}

```

```

package org.sid.billingservice.model;

import lombok.Data;

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

```

```

package org.sid.billingservice.model;

import lombok.Data;

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

```

```

package org.sid.billingservice.feign;

import org.sid.billingservice.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(name = "id") Long id);
}

```

```

package org.sid.billingservice.feign;

import ...

@FeignClient(name = "PRODUCT-SERVICE")
public interface ProductItemRestClient {
    /*@GetMapping(name = "/products")
    PagedModel<Product> pageProducts(@RequestParam(value = "page") int page, @RequestParam(value = "size") int size);*/

    @GetMapping(path = "/products")
    PagedModel<Product> pageProducts();
    @GetMapping(path = "/products/{id}")
    Product getProductById(@PathVariable(name = "id") Long id);
}

```

```

package org.sid.billingservice.repository;

import org.sid.billingservice.entities.Bill;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.rest.core.annotation.RepositoryRestResource;
import org.springframework.web.bind.annotation.CrossOrigin;

@RepositoryRestResource
@CrossOrigin("*")
public interface BillRepository extends JpaRepository<Bill, Long> {
}

```

```

@RestController
@CrossOrigin("")
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());
    }
}

```



```

@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);
        System.out.println(product.getName());
        pi.setProductName(product.getName());
    });
    return bill;
}

```

```

@SpringBootApplication
@EnableFeignClients
public class BillingServiceApplication {

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

    @Bean
    CommandLineRunner start(
        BillRepository billRepository,
        ProductItemRepository productItemRepository,
        CustomerRestClient customerRestClient,
        ProductItemRestClient productItemRestClient
    ){
        return args -> {
            Customer customer=customerRestClient.getCustomerById(1L);
            Bill bill1= billRepository.save(new Bill( id: null,new Date(), productItems: null,customer.getId(), customer: null));
            /* System.out.println("-----");
            System.out.println(customer.getId());
            System.out.println(customer.getName());
            System.out.println(customer.getEmail());*/
            PagedModel<Product> productPagedModel=productItemRestClient.pageProducts();
            System.out.println(productPagedModel.getContent().size());
            productPagedModel.forEach(p->{
                ProductItem productItem=new ProductItem();
                productItem.setPrice(p.getPrice());
                productItem.setQuantity(1+new Random().nextInt( bound: 100));
                productItem.setBill(bill1);
                productItem.setProductID(p.getId());
                System.out.println(productItem);
                productItemRepository.save(productItem);
            });
        };
    }
}

```

```

spring.application.name=billing-service
spring.datasource.url=jdbc:h2:mem:billing-db
server.port=8083
spring.cloud.discovery.enabled=true
eureka.instance.preferIpAddress=true
#management.endpoints.web.exposure.include=*

```

```

{
  _embedded: {
    bills: [
      {
        billingDate: "2021-01-07T23:07:46.459+00:00",
        customerID: 1,
        customer: null,
        _links: {
          self: {
            href: "http://localhost:8083/bills/1"
          },
          bill: {
            href: "http://localhost:8083/bills/1"
          },
          productItems: {
            href: "http://localhost:8083/bills/1/productItems"
          }
        }
      }
    ]
  },
  _links: {
    self: {
      href: "http://localhost:8083/bills"
    },
    profile: {
      href: "http://localhost:8083/profile/bills"
    }
  },
  page: {
    size: 20,
    totalElements: 1,
    totalPages: 1,
    number: 0
  }
}

```

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

jdbc:h2:mem:billing-db
 BILL
 PRODUCT_ITEM
 INFORMATION_SCHEMA
 Sequences
 Users
 H2 1.4.200 (2019-10-14)

Run Run Selected Auto complete Clear SQL statement:

SELECT * FROM PRODUCT_ITEM

SELECT * FROM PRODUCT_ITEM;

ID	PRICE	PRODUCTID	QUANTITY	BILL_ID
1	7000.0	1	11.0	1
2	584.0	2	74.0	1
3	40000.0	3	76.0	1

(3 rows, 2 ms)

Securité

Code du Micro Service d'authentification : Couche DAO

```
@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class AppRole {
    @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String roleName;
}
```

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

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

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

```
server.port=8085
spring.datasource.url=jdbc:h2:mem:users-db
```

Couche Service

```
package org.sid.sevservice.sec.service;

import org.sid.sevservice.sec.entities.AppRole;
import org.sid.sevservice.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();
}
```

```
@Service
@Transactional
public class AccountServiceImpl implements AccountService {

    private AppUserRepository appUserRepository;
    private AppRoleRepository appRoleRepository;
    private PasswordEncoder passwordEncoder;

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

    @Override
    public AppUser addNewUser(AppUser appUser) {
        String pw=appUser.getPassword();
        appUser.setPassword(passwordEncoder.encode(pw));
        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(); }
}
```

```

@SpringBootApplication
public class SevServiceApplication {

    public static void main(String[] args) { SpringApplication.run(SevServiceApplication.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: "Custommer_Manager"));
            accountService.addNewRole(new AppRole( id: null, roleName: "Product_Manager"));
            accountService.addNewRole(new AppRole( id: null, roleName: "Billing_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: "Admin");
            accountService.addRoleToUser( username: "admin", roleName: "User");
            accountService.addRoleToUser( username: "user2", roleName: "Custommer_Manager");
            accountService.addRoleToUser( username: "user3", roleName: "User");
            accountService.addRoleToUser( username: "user3", roleName: "Product_Manager");
            accountService.addRoleToUser( username: "user4", roleName: "User");
        }
    }
}

```

SQL statement:

Run Run Selected Auto complete Clear SQL statement:

SELECT * FROM APP_ROLE

SELECT * FROM APP_ROLE;

ID	ROLE_NAME
1	User
2	Admin
3	Custommer_Manager
4	Product_Manager
5	Billing_Manager

(5 rows, 2 ms)

[Edit](#)

Couche Web : Rest API pour la gestion des utilisateurs et les groupes

```
@RestController
public class AccountRestController {
    private AccountService accountService;

    public AccountRestController(AccountService accountService) { this.accountService = accountService; }

    @GetMapping(path = "/users")
    @PostAuthorize("hasAuthority('User')")
    public List<AppUser> appUsers() { return accountService.listUsers(); }

    @PostMapping(path = "/users")
    @PostAuthorize("hasAuthority('Admin')")
    public AppUser saveUser(@RequestBody AppUser appUser) { return accountService.addNewUser(appUser); }

    @PostMapping(path = "/roles")
    @PostAuthorize("hasAuthority('Admin')")
    public AppRole saveRole(@RequestBody AppRole appRole) { return accountService.addNewRole(appRole); }

    @PostMapping(path = "/addRoleToUser")
    @PostAuthorize("hasAuthority('Admin')")
    public void addRoleToUser(@RequestBody RoleUserForm roleUserForm){
        accountService.addRoleToUser(roleUserForm.getUsername(),roleUserForm.getRoleName());
    }
}
```

Spring Security Configuration : JWT Authentication Filter

```
getAuthenticationService(userDetailsService);
}

@Override
protected void configure(HttpSecurity http) throws Exception {
    http.csrf().disable();
    http.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);
    http.headers().frameOptions().disable();
    http.authorizeRequests().antMatchers( ...antMatchers: "/h2-console/**","/refreshToken/**","/login/**").permitAll();
    //http.formLogin();
    /*part11
    http.authorizeRequests().antMatchers(HttpMethod.POST,"/users/**").hasAuthority("Admin");
    http.authorizeRequests().antMatchers(HttpMethod.GET,"/users/**").hasAuthority("User");
    */
    http.authorizeRequests().anyRequest().authenticated();
    http.addFilter(new JwtAuthenticationFilter(authenticationManagerBean()));
    //part 11
    http.addFilterBefore(new JwtAuthorizationFilter(), UsernamePasswordAuthenticationFilter.class);
}

@Bean
@Override
public AuthenticationManager authenticationManagerBean() throws Exception {
    return super.authenticationManagerBean();
}
}
```

```

public class JwtAuthenticationFilter extends UsernamePasswordAuthenticationFilter {
    private AuthenticationManager authenticationManager;

    public JwtAuthenticationFilter(AuthenticationManager authenticationManager) {
        this.authenticationManager = authenticationManager;
    }

    @Override
    public Authentication attemptAuthentication(HttpServletRequest request, HttpServletResponse response) throws AuthenticationException {
        System.out.println("attemptAuthentication");
        String username=request.getParameter( name: "username");
        String password=request.getParameter( name: "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 successfulAuthentication) throws IOException {
    System.out.println("successfulAuthentication");
    User user=(User) authResult.getPrincipal();
    Algorithm algo1=Algorithm.HMAC256(JWTUtil.SECRET);
    String jwtAccessToken = JWT.create()
        .withSubject(user.getUsername())
        .withExpiresAt(new Date(System.currentTimeMillis()+JWTUtil.EXPIRE_ACCESS_TOKEN))
        .withIssuer(request.getRequestURL().toString())
        .withClaim( name: "roles",user.getAuthorities().stream().map(ga->ga.getAuthority()).collect(Collectors.toList()))
        .sign(algo1);
    String jwtRefreshToken = JWT.create()
        .withSubject(user.getUsername())
        .withExpiresAt(new Date(System.currentTimeMillis()+JWTUtil.EXPIRE_REFRESH_TOKEN))
        .withIssuer(request.getRequestURL().toString())
        .sign(algo1);

    Map<String,String> idToken=new HashMap<>();
    idToken.put("access-token", jwtAccessToken);
    idToken.put("refresh-token",jwtRefreshToken);
    response.setContentType("application/json");
    new ObjectMapper().writeValue(response.getOutputStream(),idToken);
    //response.setHeader("Authorization",jwtAccessToken);
    System.out.println(jwtAccessToken);
}

```

Spring Security Configuration : JWT Authorization Filter

```
public class JwtAuthorizationFilter extends OncePerRequestFilter {
    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response, FilterChain filterChain)
    {
        if(request.getServletPath().equals("/refreshToken") && request.getServletPath().equals("/login")){
            filterChain.doFilter(request,response);
        }
        else
        {
            String authorizationToken=request.getHeader(JWTUtil.AUTH_HEADER);
            if(authorizationToken!=null && authorizationToken.startsWith(JWTUtil.PREFIX)){
                try{
                    String jwt=authorizationToken.substring(JWTUtil.PREFIX.length());
                    System.out.println(jwt);
                    Algorithm algorithm=Algorithm.HMAC256(JWTUtil.SECRET);
                    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( name: "erreur-message",e.getMessage());
                    response.sendError(HttpServletResponse.SC_FORBIDDEN);
                }
            }
        }
    }
}
```

```
        filterChain.doFilter(request,response);
    }catch(Exception e){
        response.setHeader( name: "erreur-message",e.getMessage());
        response.sendError(HttpServletResponse.SC_FORBIDDEN);
    }
}
else{
    filterChain.doFilter(request,response);
}
}
}
```


Tests : Authentification

The screenshot shows the ARC REST client interface. On the left, a sidebar lists the request history with the current request highlighted. The main panel displays the details of a POST request to `http://localhost:8085/login`. The request body is in the 'Form data (www-url-form-encoded)' format, containing `username=admin` and `password=1234`. The response status is `200 OK` with a response time of `576.94 ms`. The response body contains JSON tokens: `refresh-token` and `access-token`.

Request details:

- Method: POST
- Request URL: `http://localhost:8085/login`
- Body content type: `application/x-www-form-urlencoded`
- Form data: `username=admin`, `password=1234`
- Status: 200 OK
- Response time: 576.94 ms
- Response body:

```
{  "refresh-token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1pb1IsImZyI6Imh0bG9sb2NhbGhvc3Q6ODAA4NS9sb2dpbiIsImV4cCI6MTYwODk5ODU0NDk5LmV4c2FkcyJ9eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1pb1IsImZyI6Imh0bG9sb2NhbGhvc3Q6ODAA4NS9sb2dpbiIsImV4c2FkcyJ9eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1pb1IsImZyI6Imh0bG9sb2NhbGhvc3Q6ODAA4NS9sb2dpbiIsImV4c2FkcyJ9",  "access-token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1pb1IsImZyI6Imh0bG9sb2NhbGhvc3Q6ODAA4NS9sb2dpbiIsImV4c2FkcyJ9"}

```

Tests : Consulter les utilisateurs sans JWT

The screenshot shows the ARC REST client interface. On the left, a sidebar lists the request history with the current request highlighted. The main panel displays the details of a GET request to `http://localhost:8085/users`. The request headers include `content-type: application/json` and `Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1pb1IsImZyI6Imh0bG9sb2NhbGhvc3Q6ODAA4NS9sb2dpbiIsImV4c2FkcyJ9`. The response status is `403 Forbidden` with a response time of `65.89 ms`. The response body contains JSON error details: `timestamp`, `status`, `error`, `message`, and `path`.

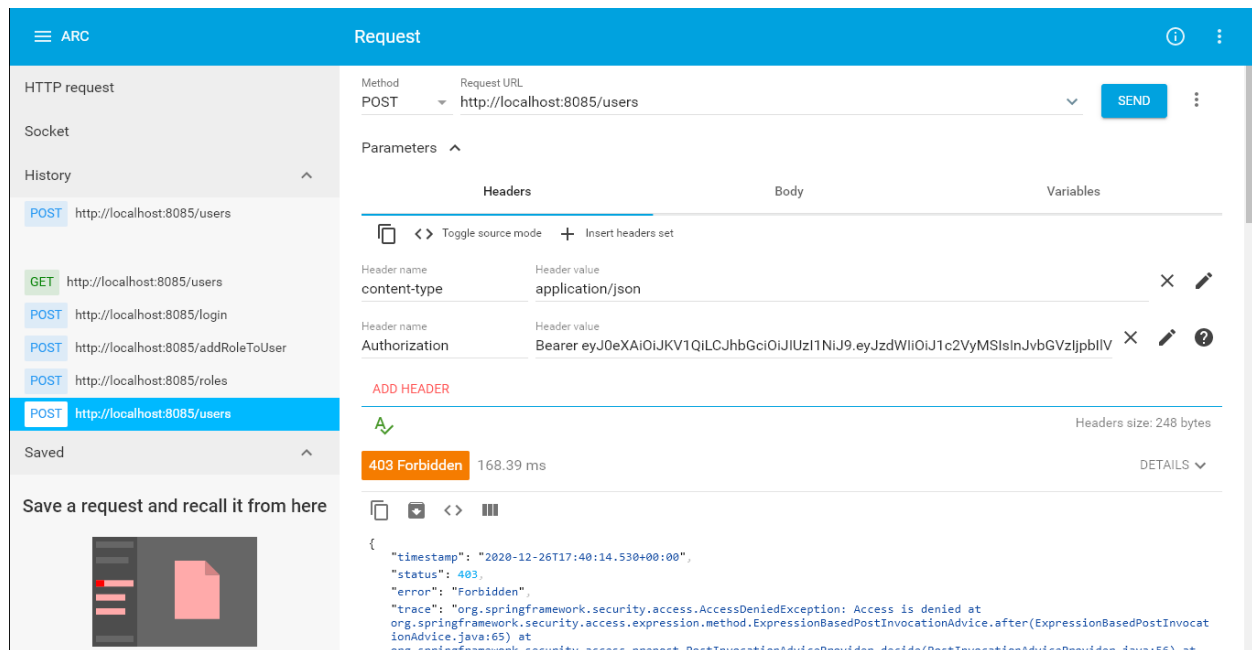
Request details:

- Method: GET
- Request URL: `http://localhost:8085/users`
- Headers: `content-type: application/json`, `Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1pb1IsImZyI6Imh0bG9sb2NhbGhvc3Q6ODAA4NS9sb2dpbiIsImV4c2FkcyJ9`
- Status: 403 Forbidden
- Response time: 65.89 ms
- Response body:

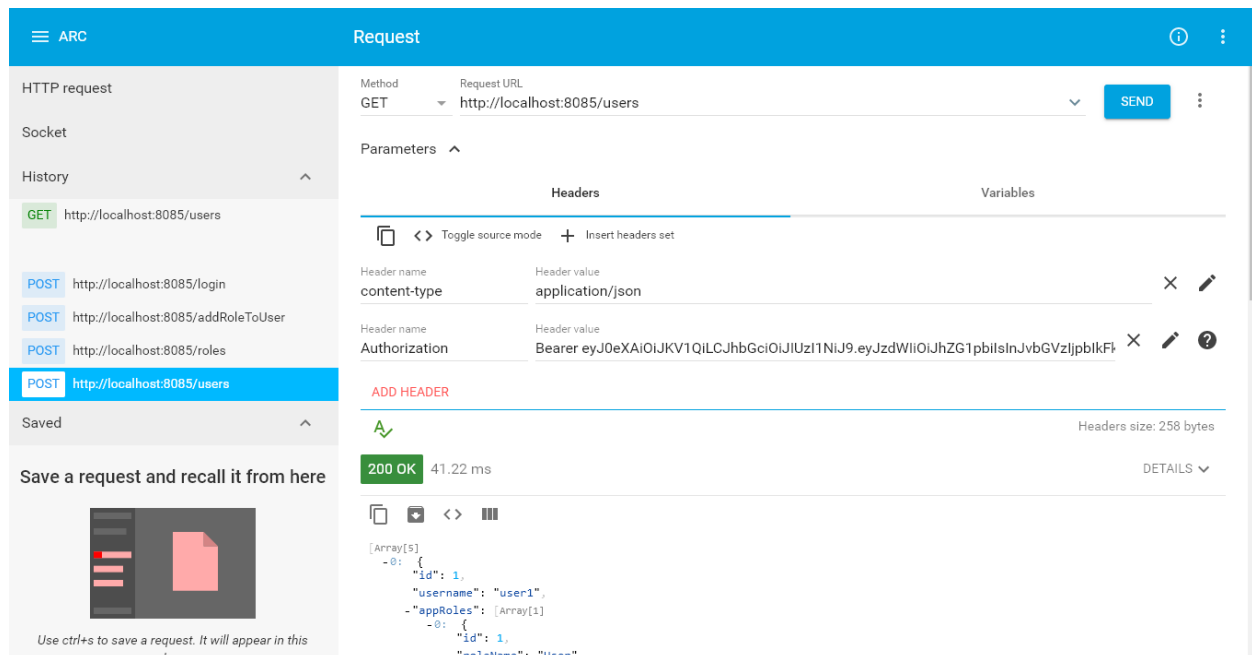
```
{  "timestamp": "2020-12-26T17:21:07.962+00:00",  "status": 403,  "error": "Forbidden",  "message": "Access Denied",  "path": "/users"}

```

Tests : Consulter les utilisateurs avec JWT qui a expiré



Tests : Consulter les utilisateurs avec un JWT Valide



Tests : Ajouter un un utilisateur avec le Rôle USER

The screenshot shows the ARC REST client interface. On the left, a list of requests is shown, with the selected request being a POST to `http://localhost:8085/users`. The main panel displays the details of this request. The Method is POST and the Request URL is `http://localhost:8085/users`. The Headers tab is active, showing two headers: `content-type` with value `application/json` and `authorization` with value `Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWwiOiJ1c2VybMSlInJvbGVzIjpbIlVzZXliIiwiaXNjaW50eSI6Zm9udC91bnRlcnQzLnR5bGVzZXliIiwiaWF0IjoiMjAyMC12-26T17:32:04.949+00:00"`. The Body tab is also visible, showing a JSON body. The response status is `403 Forbidden` with a response time of 23.49 ms. The response body is a JSON object:

```
{  "timestamp": "2020-12-26T17:32:04.949+00:00",  "status": 403,  "error": "Forbidden",  "message": "Forbidden",  "path": "/users"}
```

Tests : Ajouter un un utilisateur avec le Rôle ADMIN

The screenshot shows the ARC REST client interface. On the left, a list of requests is shown, with the selected request being a POST to `http://localhost:8085/users`. The main panel displays the details of this request. The Method is POST and the Request URL is `http://localhost:8085/users`. The Headers tab is active, showing two headers: `content-type` with value `application/json` and `authorization` with value `Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWwiOiJ1c2VybMSlInJvbGVzIjpbIlVzZXliIiwiaXNjaW50eSI6Zm9udC91bnRlcnQzLnR5bGVzZXliIiwiaWF0IjoiMjAyMC12-26T17:32:04.949+00:00"`. The Body tab is also visible, showing a JSON body. The response status is `200 OK` with a response time of 147.85 ms. The response body is a JSON object:

```
{  "username": "az",  "password": "1234"}
```

Refresh token

The screenshot displays the ARC REST client interface. On the left, a sidebar shows a list of HTTP requests, with the selected request being a GET request to `http://localhost:8085/refreshToken`. The main panel shows the details of this request, including the method (GET), the request URL (`http://localhost:8085/refreshToken`), and the headers. The headers are `content-type: application/json` and `authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1pbil:` (truncated). The response is a 200 OK status with a response time of 29.53 ms. The response body contains JSON tokens.

Method: GET
Request URL: `http://localhost:8085/refreshToken`

Parameters: (None)

Headers:

Header name	Header value
content-type	application/json
authorization	Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1pbil:

Response: 200 OK, 29.53 ms

Response Body:

```
{
  "refresh-token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1pbilIsInJvbnGVzIjpbIkFkbWluIiwiaXN1ciJdLCJpc3MiOiJodHRwOi8vbG9jYXRob3N0OjgwODUvbnB9bnaw4iLCJleHAiOiJlZ2MDkwMDY1NDN9.bBs8rQPmwiX1iewY1kwZp0D2V2brIslos5J3-hQH0Yc",
  "access-token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJzdWIiOiJhZG1pbilIsInJvbnGVzIjpbIkFkbWluIiwiaXN1ciJdLCJpc3MiOiJodHRwOi8vbG9jYXRob3N0OjgwODUvbnB9bnaw4iLCJleHAiOiJlZ2MDkwMDY1NDN9.bBs8rQPmwiX1iewY1kwZp0D2V2brIslos5J3-hQH0Yc"
}
```

2.Partie Frontend:

Page produits:

```
import { Component, OnInit } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Component({
  selector: 'app-products',
  templateUrl: './products.component.html',
  styleUrls: ['./products.component.css']
})
export class ProductsComponent 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);
    })
  }
}
```

```
<div class="grow"></div>
<div class="df">
  <div class="container">
    <div class="col-md-7">
      <ul class="list-group">
        <table class="table">
          <tr>
            <th>name</th>
            <th>price</th>
            <th>quantity</th>
          </tr>

          <tr *ngFor="let v of products._embedded.products">
            <td>{{v.name}} </td>
            <td>{{v.price}} </td>
            <td>{{v.quantity}} </td>

            <td><a onclick="return confirm('Etes vous sure de vouloir supprimer?')"
              class="btn btn-danger">Delete</a>
            </td>
            <td><a class="btn btn-success">edit</a></td>
          </tr>
        </table>
      </ul>
    </div>
  </div>
</div>
```

products works!

name	price	quantity		
Ord	7000	100	Delete	edit
Imp	584	48	Delete	edit
Phone	40000	15	Delete	edit

Page Customers :

```
<div class="grow"></div>
<div class="df">
  <div class="container">
    <div class="col-md-7">
      <ul class="list-group">
        <table class="table">
          <tr>
            <th>name</th>
            <th>email</th>
          </tr>

          <tr *ngFor="let v of customers._embedded.customers">
            <td>{{v.name}} </td>
            <td>{{v.email}} </td>

            <td><a onclick="return confirm('Etes vous sure de vouloir supprimer?')"
              class="btn btn-danger">Delete</a>
            </td>
            <td><a class="btn btn-success">edit</a></td>

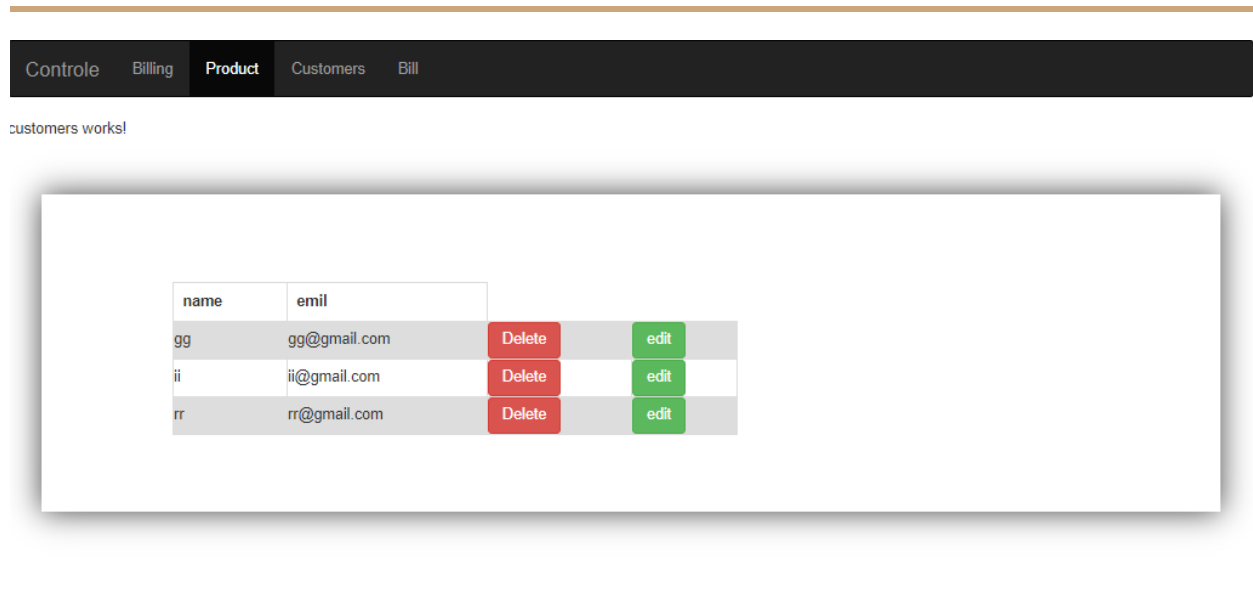
          </tr>
        </table>
      </ul>
    </div>
  </div>
</div>
```

```
import { Component, OnInit } from '@angular/core';
import { HttpClient } from '@angular/common/http';

@Component({
  selector: 'app-customers',
  templateUrl: './customers.component.html',
  styleUrls: ['./customers.component.css']
})
export class CustomersComponent implements OnInit {

  public customers;
  constructor(private http:HttpClient) { }

  ngOnInit(): void {
    this.http.get( url: "http://localhost:8888/CUSTOMER-SERVICE/customers").subscribe( next: data=>{
      this.customers=data;
      console.log(data)
    }, error: err=>{
      console.log(err);
    })
  }
}
```

Page Affichage Bill :

```
<div id="sheet" *ngFor="let b of bill._embedded.bills" >
  <div id="header" class="flex">
    <div id="societyInfos" class="flex col">
      <input type="text" name="societyName" id="societyName" placeholder="Nom de la société">
      <input type="text" name="societyTel" id="societyTel" placeholder="Numéro de Téléphone">
      <input type="text" name="societyAddress" id="societyAddress" placeholder="Numéro, Adresse">
      <div class="flex">
        <input type="text" name="societyZipcode" id="societyZipcode" placeholder="Code Postal">
        <div class="inputSeparator"></div>
        <input class="grow" type="text" name="societyCity" id="societyCity" placeholder="Ville">
      </div>
      <input type="text" name="societyCountry" id="societyCountry" placeholder="Pays">
      <input type="text" name="societySiret" id="societySiret" placeholder="Numéro de SIRET">
    </div>

    <div class="grow"></div>

    <div id="clientInfos" class="flex col">
      <div id="clientInfosLabel">
        COORDONNÉES CLIENT
      </div>
      <div class="flex">
        <select name="clientGender" id="clientGender">
          <option value="M.">M.</option>
          <option value="Mme">Mme</option>
          <option value="Mlle">Mlle</option>
        </select>
        <div class="inputSeparator"></div>
      </div>
    </div>
  </div>
</div>
```

```

<div id="billNumberBanner">
  FACTURE N° <input type="number" name="billNumber" id="billNumber" min="0" value="{{b.id}}" / établie en EUROS
</div>

<div id="billInfos">
  DATE de FACTURE: <input type="text" name="billDate" id="billDate" value="{{b.billingDate}}" / A Régler avant le : <input type="text" name="dueDate" id="dueDate" value="" /
</div>

<div id="billDetail">
  <table>
    <thead>
      <tr>
        <th>Référence</th>
        <th>Description</th>
        <th>Quantité</th>
        <th>Prix Unitaire</th>
        <th>Montant</th>
      </tr>
    </thead>
    <tbody>
      <tr *ngFor="let p1 of products.embedded.products">
        <td><input type="text" value="REF001"></td>
        <td width="40%">{{p1.name}}</td>
        <td>{{p1.quantity}}</td>
        <td>{{p1.price}}</td>
        <td>{{p1.quantity*p1.price}}</td>
      </tr>
    </tbody>
  </table>

```

```

export class BillComponent implements OnInit {
  public customers;
  public products;
  public bill;
  constructor(private http:HttpClient) { }

  ngOnInit(): void {
    this.http.get( url: "http://localhost:8888/CUSTOMER-SERVICE/customers").subscribe( next: data=>{
      this.customers=data;
      console.log(data)
    }, error: err=>{
      console.log(err);
    })
    this.http.get( url: "http://localhost:8888/PRODUCT-SERVICE/products").subscribe( next: data=>{
      this.products=data;
      console.log(data)
    }, error: err=>{
      console.log(err);
    })
    this.http.get( url: "http://localhost:8888/BILLING-SERVICE/bills").subscribe( next: data=>{
      this.bill=data;
      console.log(data)
    }, error: err=>{
      console.log(err);
    })
  }
}

```

Nom de la société

Numéro de Téléphone

Numéro, Adresse

Code Postal

Ville

Pays

Numéro de SIRET

COORDONNÉES CLIENT

M. Nom et Prénom

Numéro, Adresse

Code Postal

Ville

FRANCE

anglais

Toujours trad.

Google Translate

FACTURE N° / établie en EUROS

DATE de FACTURE: 2021-01-07T23:07:40.459+ / A Régler avant le : / Paiement : CB / Chèque

Référence	Description	Quantité	Prix Unitaire	Montant
REF001	Ord	100	7000	700000
REF001	Imp	48	584	28032
REF001	Phone	15	40000	600000
Total à Payer				0

Générer la facture

Page Ajout Bill :

```
<div id="sheet">
  <div id="header" class="flex">

    <div class="grow"></div>

    <div id="clientInfos" class="flex col">
      <div id="clientInfosLabel">
        COORDONNÉES CLIENT
      </div>
      <div class="flex">
        <select name="clientGender" id="customers" >
          <option value="M." *ngFor="let v of customers._embedded.customers">{{v.name}}</option>

        </select>
        <div class="inputSeparator"></div>
        <input class="grow" type="text" name="clientAdresse" id="clientAdresse" placeholder="Adresse">
      </div>

      <input type="text" name="clientCountry" id="clientCountry" placeholder="Pays" value="MAROC">
    </div>
  </div>

  <div id="billNumberBanner">
    FACTURE N° <input type="number" name="billNumber" id="billNumber" min="0"> / établie en EUROS
  </div>
</div>
```

```

</div>

<div id="billInfos">
  DATE de FACTURE: <input type="text" name="billDate" id="billDate"> / A Régler avant le : <input type="text" name="billDeadline" id="billDeadline">
</div>

<div id="billDetail">
  <table (change)="somme()">
    <thead>
      <tr>
        <th>Référence</th>
        <th>Description</th>
        <th>Quantité</th>
        <th>Prix Unitaire</th>
        <th>Montant</th>
      </tr>
    </thead>
    <tbody>
      <tr>
        <td><input type="text" value="REF001"></td>
        <td width="40%"><select name="clientGender" id="products1" (change)="getPrice1($event.target.value)" >
          <option *ngFor="let p1 of products._embedded.products" value="{{p1.price}}"> {{p1.name}}</option>
        </select></td>
        <td><input type="number" value="0" min="0" class="quantity" id="l1" (change)="yy1($event.target.value)"></td>
        <td>{{currentProduct1}}</td>
        <td><div class="amount">{{test1}}</div></td>
      </tr>
      <tr>
        <td><input type="text" value="REF002"></td>
        <td width="40%"><select name="clientGender" id="products2" (change)="getPrice2($event.target.value)" >
          <option value="M."*ngFor="let p2 of products._embedded.products" value="{{p2.price}}">{{p2.name}}</option>
        </select></td>
        <td><input type="number" value="0" min="0" class="quantity" id="teeeest" (change)="yy2($event.target.value)"></td>
        <td>{{currentProduct2}}</td>
        <td><div class="amount">{{test2}}</div></td>
      </tr>
      <tr>
        <td><input type="text" value="REF003"></td>
        <td width="40%"><select name="clientGender" id="products3" (change)="getPrice3($event.target.value)" >
          <option *ngFor="let p3 of products._embedded.products" value="{{p3.price}}">{{p3.name}}</option>
        </select></td>
        <td><input type="number" value="0" min="0" class="quantity" (change)="yy3($event.target.value)"></td>
        <td>{{currentProduct3}}</td>
        <td><div class="amount">{{test3}}</div></td>
      </tr>
      <tr>
        <td><input type="text" value="REF004"></td>
        <td width="40%"><select name="clientGender" id="products4" (change)="getPrice4($event.target.value)" >
          <option value="{{p4.price}}"*ngFor="let p4 of products._embedded.products">{{p4.name}}</option>
        </select></td>
        <td><input type="number" value="0" min="0" class="quantity" (change)="yy4($event.target.value)"></td>
        <td>{{currentProduct4}}</td>
        <td><div class="amount">{{test4}}</div></td>
      </tr>
    </tbody>
  </table>
</div>

```

```

public sum=0;
constructor(private http:HttpClient) { }

ngOnInit(): void {
    this.http.get(_url: "http://localhost:8888/CUSTOMER-SERVICE/customers").subscribe( next: data=>{
        this.customers=data;
        console.log(data)
    }, error: err=>{
        console.log(err);
    })
    this.http.get( _url: "http://localhost:8888/PRODUCT-SERVICE/products").subscribe( next: data=>{
        this.products=data;
        console.log(data)
    }, error: err=>{
        console.log(err);
    })
}

getPrice1(v: any):void {
    this.currentProduct1=v;
    console.log(this.currentProduct1)
}

getPrice2(v: any):void {
    this.currentProduct2=v;

    console.log(this.currentProduct2)
}

getPrice3(v: any):void {

```

```

yy3(v: any):void {
    this.test3=this.currentProduct3*v;

    console.log(this.currentProduct5)
}
yy4(v: any):void {
    this.test4=this.currentProduct4*v;

    console.log(this.currentProduct5)
}
yy5(v: any):void {
    this.test5=this.currentProduct5*v;

    console.log(this.currentProduct5)
}
somme():void {
    this.sum=this.test1+this.test2+this.test3+this.test4+this.test5;
}

```

COORDONNÉES CLIENT

gg Adresse

MAROC

FACTURE N° / établie en EUROS

DATE de FACTURE: 12/30/2020 / A Régler avant le : 12/30/2020 / Paiement : CB / Chèque

Référence	Description	Quantité	Prix Unitaire	Montant
REF001	Imp	1	584	584
REF002	Ord	1	7000	7000
REF003	Phone	2	40000	80000
REF004	Imp	2	584	1168
REF005	Phone	2	40000	80000
Total à Payer				168752

Générer la facture

Conclusion

Au cours de la réalisation de toutes les parties du projet nous avons pu apprendre de nouvelles connaissances aussi bien théorique que pratique tel que la maîtrise des outils de développement de conception et de réalisation d'un spring starter project micro services avec deux types d'authentification Stateful et Stateless .