



**ECOLE MAROCAINE DES
SCIENCES DE L'INGENIEUR**

Membre de

HONORIS UNITED UNIVERSITIES

Rapport de Projet – JEE

Distributed HPC – Micro Services

Réaliser Par

Yassine Agourram

Encadrer Par

Pr. MOHAMMED YOUSSEFI

Sommaire

I. Introduction	4
CHAPITRE I : CONTEXTE GÉNÉRALE	5
I. Cahier de Charge	6
CHAPITRE II : ANALYSE ET CONCEPTION	8
I. Conception	9
CHAPITRE III : ENVIRONNEMENT ET OUTILS	11
I. Outils	12
CHAPITRE IV : RÉALISATION & DÉMONSTRATION	16
I. Back End :	17

Table des Figures

Figure 1 : Architecture Inventory Service	18
Figure 2 : Products Items - Inventory Service	18
Figure 3 : Produit 1 - Inventoy Service	19
Figure 4 : Architecture Customer Service	19
Figure 5 : Liste Customers - Customer Service	20
Figure 6 : Customer 1 - Customer Service	20
Figure 7 : Architecture Billing Service	21
Figure 8: Liste Billing Service	21
Figure 9 : Service Executer dans Eureka	22
Figure 10 : Architecture Security Service	22

I. Introduction

Dans le cadre de notre troisième année du cycle ingénieurs en Informatique & Réseaux Option MIAGE, notre professeur de JEE,

Pr. Mohammed Youssfi nous est proposé un projet nous permettant de mettre en pratique nos connaissances et nos compétences professionnelles à travers un cahier de charge ayant comme finalité de

realiser la conception et développer une application avec Micro-Service

Ainsi, nous articulons notre rapport autour de quatre gros titres :

- Le premier consiste en une présentation du projet (cahier de charge).
- Le deuxième aura l'analyse et conception.
- Le troisième aura comme but de définir les outils de développement utilisés.
- Le quatrième titre porte sur la réalisation et la configuration de l'application.
- Finalement, nous expliquons dans le dernier titre le fonctionnement de l'application web MVC tout en mettant l'accent sur le test de l'application.

CHAPITRE I : CONTEXTE GÉNÉRALE

I. Cahier de Charge

1. Contexte

Ce projet s'intègre dans le cadre d'appliquer les modalités suivant :

⇒ Créer le micro service Customer-service.

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

⇒ Créer le micro service Inventory-service.

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

⇒ 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.

- ⇒ Créer l'annuaire Registry Service basé sur Netflix Eureka Server.
- ⇒ Tester le proxy en utilisant une configuration dynamique de Gestion des routes vers les micros services enregistrés dans l'annuaire Eureka Server.
- ⇒ Créer Le service Billing-Service en utilisant Open Feign pour communiquer avec les services Customer-service et Inventory-service.
- ⇒ Créer un client Angular qui permet d'afficher une facture ou une autre interface frontend à proposer.

CHAPITRE II : ANALYSE ET CONCEPTION

I. Conception



La **conception de logiciel** met en œuvre un ensemble d'activités qui à partir d'une demande d'informatisation d'un processus (demande qui peut aller de la simple question orale jusqu'au cahier des charges complet) permettent la conception, l'écriture et la mise au point d'un logiciel (et donc de programmes informatiques) jusqu'à sa livraison au demandeur.

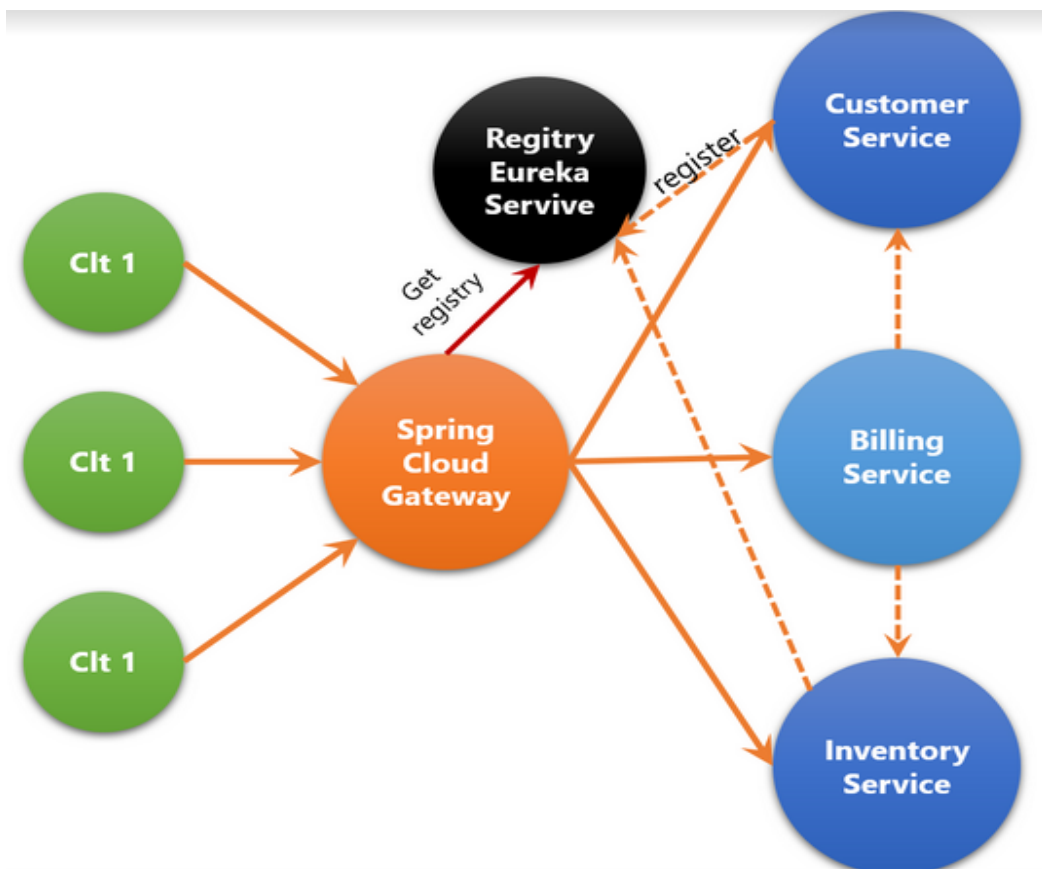
En règle générale, la fabrication d'un logiciel va suivre trois grandes phases :

- ⇒ Phase d'analyse (fonctionnelle) ou de conception
 - Durant cette phase, on effectue simultanément l'étude des données et l'étude des traitements à effectuer. C'est en général dans cette phase que s'appliquent les techniques de modélisation. Il en découle la description des bases de données éventuelles à créer et les programmes à écrire et la manière dont tout cela va être intégré.
 - Spécification
 - Conception
 - Définition de l'architecture
- ⇒ Phase de réalisation ou de programmation (écriture et tests des programmes)
 - Algorithmique
 - Programmation
 - Gestion des versions

- Factorisation
- Tests unitaires
- Optimisation du code
- ⇒ Phase de livraison
 - Intégration
 - Validation
 - Documentation du logiciel
 - Packaging

1. Architecture technique:

L'architecture technique est une vue tournée vers les différents éléments matériels et l'infrastructure dans laquelle le système informatique s'inscrit, les liaisons physiques et logiques entre ces éléments et les informations qui y circulent.



CHAPITRE III : ENVIRONNEMENT ET OUTILS

I. Outils :

1. Java & Jee



JEE est une plate-forme fortement orientée serveur pour le développement et l'exécution d'applications distribuées. Elle est composée de deux parties essentielles :

- ⇒ Un ensemble de spécifications pour une infrastructure dans laquelle s'exécutent les composants écrits en Java : un tel environnement se nomme serveur d'applications.
- ⇒ Un ensemble d'API qui peuvent être obtenues et utilisées séparément. Pour être utilisées, certaines nécessitent une implémentation de la part d'un fournisseur tiers.

Sun propose une implémentation minimale des spécifications de J2EE : le J2EE SDK. Cette implémentation permet de développer des applications respectant les spécifications mais n'est pas prévue pour être utilisée dans un environnement de production. Ces spécifications doivent être respectées par les outils développés par des éditeurs tiers.

L'utilisation de J2EE pour développer et exécuter une application offre plusieurs avantages :

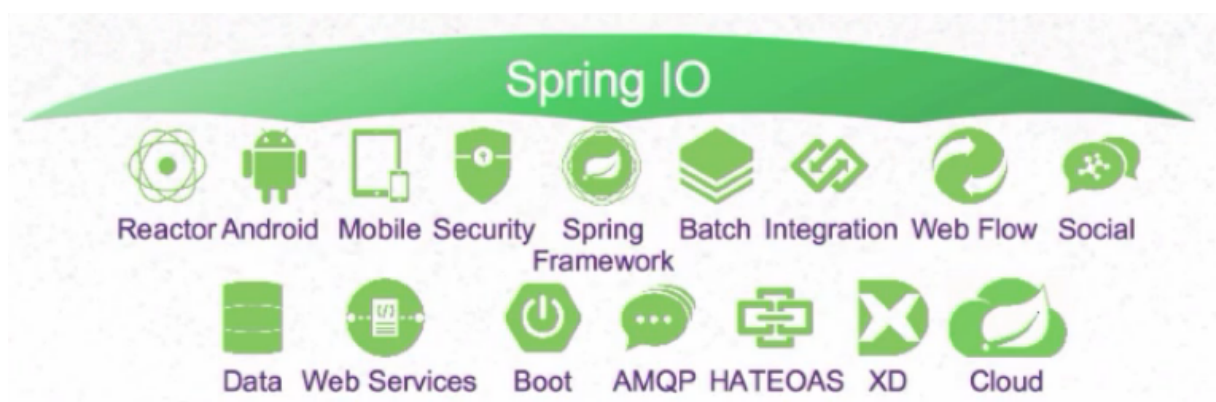
- ⇒ Une architecture d'applications basée sur les composants qui permet un découpage de l'application et donc une séparation des rôles lors du développement.
- ⇒ La possibilité de s'interfacer avec le système d'information existant grâce à de nombreuses API : JDBC, JNDI, JMS, JCA ...
- ⇒ La possibilité de choisir les outils de développement et le ou les serveurs d'applications utilisés qu'ils soient commerciaux ou libres.

J2EE permet une grande flexibilité dans le choix de l'architecture de l'application en combinant les différents composants. Ce choix dépend des besoins auxquels doit répondre l'application mais aussi des compétences dans les différentes API de J2EE. L'architecture d'une application se découpe idéalement en au moins trois tiers :

- ⇒ La partie cliente : c'est la partie qui permet le dialogue avec l'utilisateur. Elle peut être composée d'une application standalone, d'une application web ou d'applets.
- ⇒ La partie métier : c'est la partie qui encapsule les traitements (dans des EJB ou des JavaBeans).
- ⇒ La partie donnée : c'est la partie qui stocke les données.

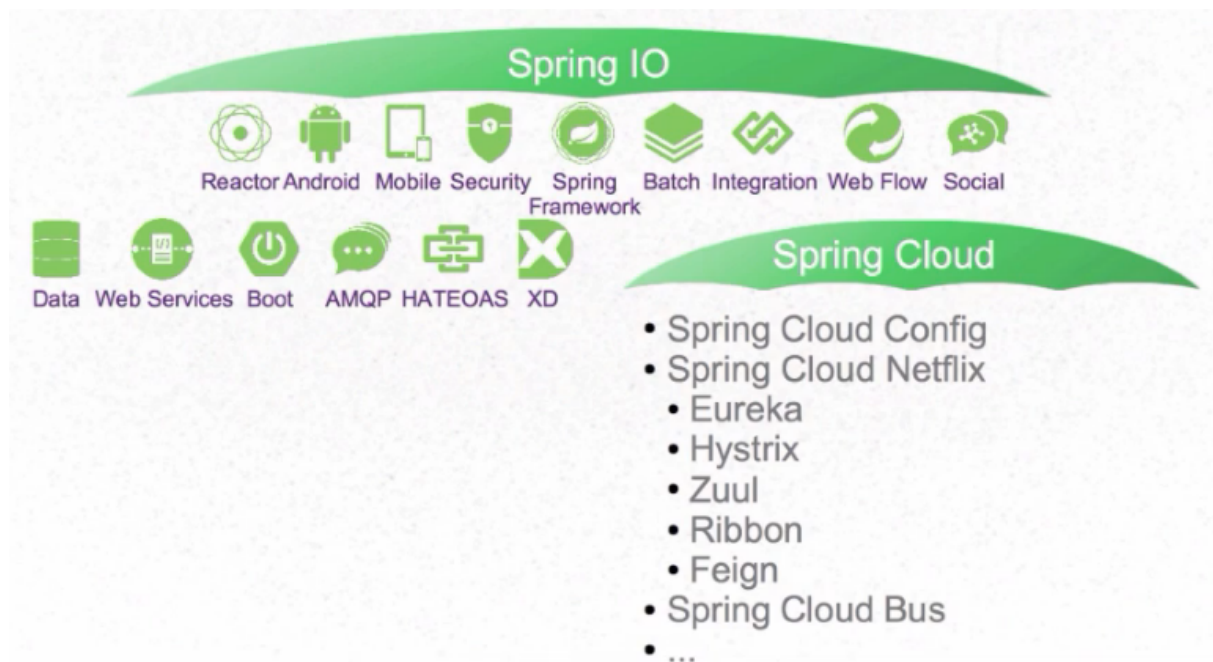
2. Spring Cloud

Spring est une plateforme construite pour le développement d'applications web en langage **Java**. Il a été introduit en 2004. En 2006, des sous-projets (sub-projet) sont apparus. Chaque sous-projet se concentre sur un domaine différent. Jusqu'à présent, vous pouvez voir les sous-projets listés comme l'illustration suivante



Spring IO (Spring Integration Objects) (Des objets de Spring Intégration) est le nom utilisé pour la famille des sous-projets du **Spring**. Il est considéré comme un parapluie et les sous-projets sont situés en dessous d'un tel parapluie.

Spring Cloud est un sous-projet (sub-project) situé dans le parapluie **Spring IO** (Spring IO Umbrella), et il est lui-même un parapluie, un sous-parapluie (Sub-Umbrella)



Ci-dessous est la liste des sous-projets et des modèles (pattern) dans **Spring Cloud** :

MAIN PROJECTS

- Spring Cloud Config
- Spring Cloud Netflix
- Spring Cloud Bus
- Spring Cloud for Cloud Foundry
- Spring Cloud Cluster
- Spring Cloud Consul
- Spring Cloud Security
- Spring Cloud Sleuth
- Spring Cloud Data Flow
- Spring Cloud Stream
- Spring Cloud Stream Modules
- Spring Cloud Task
- Spring Cloud Zookeeper
- Spring Cloud for Amazon Web Services
- Spring Cloud Connectors
- Spring Cloud Starters
- Spring Cloud CLI

PATTERNS

- Distributed/Versioned Configuration Management
- Service Registration & Discovery
- Routing & Load Balancing
- Fault Tolerance (Circuit Breakers)
- Monitoring
- Concurrent API Aggregation & Transformation
- Distributed messaging

3. Angular



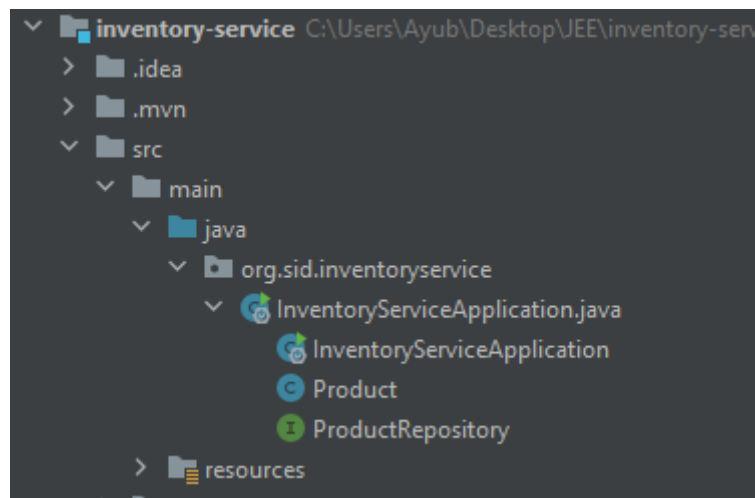
Angular (communément appelé "**Angular 2+**" ou "**Angular v2 et plus**") est un cadriciel (framework) côté client, open source, basé sur TypeScript, et co-dirigé par l'équipe du projet « Angular » à Google et par une communauté de particuliers et de sociétés. Angular est une réécriture complète d'AngularJS, cadriciel construit par la même équipe. Il permet la création d'applications Web et plus particulièrement de ce qu'on appelle des « *Single Page Applications* » : des applications web accessibles via une page web

unique qui permet de fluidifier l'expérience utilisateur et d'éviter les chargements de pages à chaque nouvelle action. Le Framework est basé sur une architecture du type MVC et permet donc de séparer les données, le visuel et les actions pour une meilleure gestion des responsabilités. Un type d'architecture qui a largement fait ses preuves et qui permet une forte maintenabilité et une amélioration du travail collaboratif.

CHAPITRE IV : RÉALISATION & DÉMONSTRATION

I. Back End :

1. Inventory Service



```
InventoryServiceApplication.java x
1 package org.sid.inventoryservice;
2
3 import ...
4
19
20 @SpringBootApplication
21 public class InventoryServiceApplication {
22
23     public static void main(String[] args) {
24
25         SpringApplication.run(InventoryServiceApplication.class, args);
26     }
27
28     @Bean
29     CommandLineRunner start(ProductRepository productRepository, RepositoryRestConfiguration restConfiguration)
30     {
31         restConfiguration.exposeIdsFor(Product.class);
32         return args ->{
33             productRepository.save(new Product( id: null, name: "Ordinateur", price: 1828, quantity: 42));
34             productRepository.save(new Product( id: null, name: "Imprimante", price: 6878, quantity: 72));
35             productRepository.save(new Product( id: null, name: "Smartphone", price: 8888, quantity: 92));
36             productRepository.findAll().forEach(p->{
37                 System.out.println(p.getName());
38             });
39         };
40     }
41 }
```

Figure 1 : Architecture Inventory Service



Figure 2 : Products Items - Inventory Service

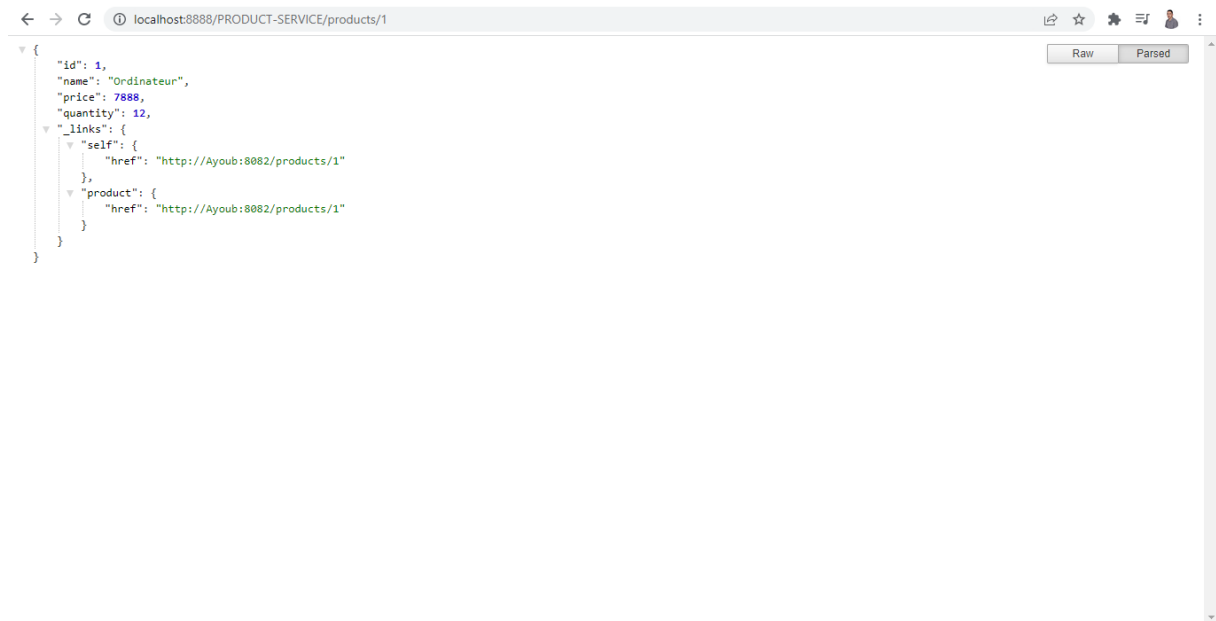


Figure 3 : Produit 1 - Inventory Service

2. Customer Service

```
CustomerServiceApplication.java x
1 package org.sid.customerservice;
2
3 import ...
10
11 @SpringBootApplication
12 public class CustomerServiceApplication {
13
14     public static void main(String[] args) {
15
16         SpringApplication.run(CustomerServiceApplication.class, args);
17     }
18
19     @Bean
20     CommandLineRunner start(CustomerRepository customerRepository, RepositoryRestConfiguration restConfigu
21     restConfiguration.exposeIdsFor(Customer.class);
22     return arg ->{
23         customerRepository.save(new Customer( id: null, name: "yassine", email: "yassine.ag@gmail.com"));
24         customerRepository.save(new Customer( id: null, name: "rida", email: "rida@gmail.com"));
25         customerRepository.findAll().forEach(c-> {
26             System.out.println(c.toString());
27         });
28     };
29 }
30
31
```

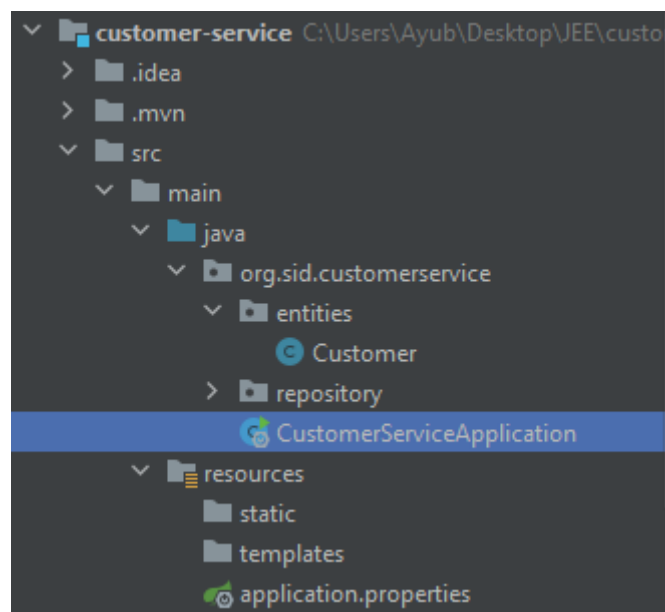



Figure 4 : Architecture Customer Service



```
{
  "_embedded": {
    "customers": [
      {
        "id": 1,
        "name": "Ayoub",
        "email": "Ayoub.kabir@gmail.com",
        "_links": {
          "self": {
            "href": "http://Ayoub:8081/customers/1"
          },
          "customer": {
            "href": "http://Ayoub:8081/customers/1"
          }
        }
      },
      {
        "id": 2,
        "name": "Kamal",
        "email": "Kamal.Dahbi@gmail.com",
        "_links": {
          "self": {
            "href": "http://Ayoub:8081/customers/2"
          },
          "customer": {
            "href": "http://Ayoub:8081/customers/2"
          }
        }
      },
      {
        "id": 3,
        "name": "Nouhaila",
        "email": "Nouhaila.Montasir@gmail.com",
        "_links": {
          "self": {
            "href": "http://Ayoub:8081/customers/3"
          }
        }
      }
    ]
  }
}
```

Figure 5 : Liste Customers - Customer Service



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

Figure 6 : Customer 1 - Customer Service

3. Billing Service

```
BillingServiceApplication.java
22 @EnableFeignClients
23 public class BillingServiceApplication {
24
25     public static void main(String[] args) {
26
27         SpringApplication.run(BillingServiceApplication.class, args);
28     }
29     @Bean
30     CommandLineRunner start(BillRepository billRepository,
31                             ProductItemRepository productItemRepository,
32                             CustomerRestClient customerRestClient,
33                             ProductItemRestClient productItemRestClient){
34         return args -> {
35             Customer customer=customerRestClient.getCustomerById(1L);
36             Bill bill1=billRepository.save(new Bill(null,new Date(),null,customer.getId(),null));
37             PagedModel<Product> productPagedModel=productItemRestClient.pageProducts();
38             productPagedModel.forEach(p-> {
39                 ProductItem productItem=new ProductItem();
40                 productItem.setPrice(p.getPrice());
41                 productItem.setQuantity(1+new Random().nextInt(100));
42                 productItem.setBill(bill1);
43                 productItem.setProductID(p.getId());
44                 productItemRepository.save(productItem);
45             });
46         }
47     }
48 }
```

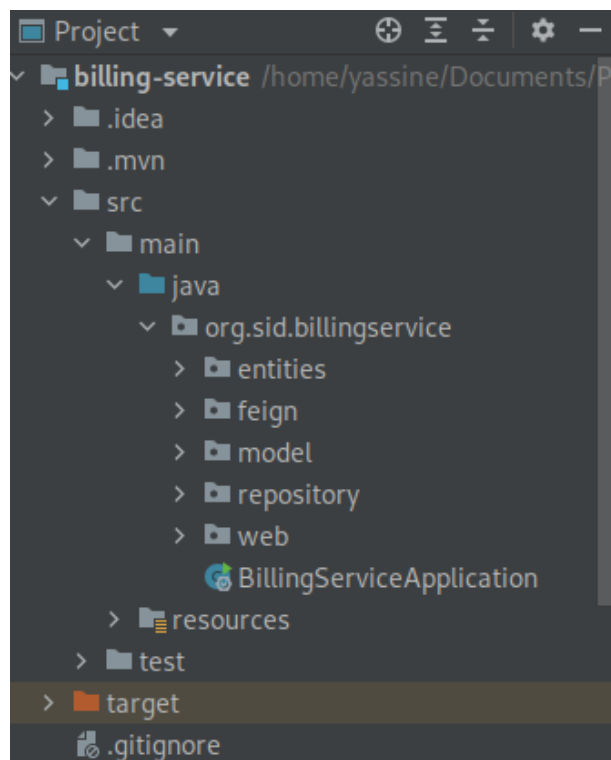


figure 7 : Architecture Billing Service

```
localhost:8888/BILLING-SERVICE/fullBill/1
{
  "id": 1,
  "billingDate": "2022-02-20T12:51:32.885+00:00",
  "productItems": [
    {
      "id": 1,
      "quantity": 63,
      "price": 7888,
      "productID": 1,
      "product": {
        "id": 1,
        "price": 7888,
        "name": "Ordinateur",
        "quantity": 12
      }
    },
    {
      "id": 2,
      "quantity": 95,
      "price": 78,
      "productID": 2,
      "product": {
        "id": 2,
        "price": 78,
        "name": "Imprimante",
        "quantity": 172
      }
    },
    {
      "id": 3,
      "quantity": 75,
      "price": 788,
      "productID": 3,
      "product": {
        "id": 3,
        "price": 788,
        "name": "Smartphone"
      }
    }
  ]
}
```

Figure 8: Liste Billing Service

4. Eureka

localhost:8761

Renews (last min) 14

DS Replicas

localhost

Instances currently registered with Eureka

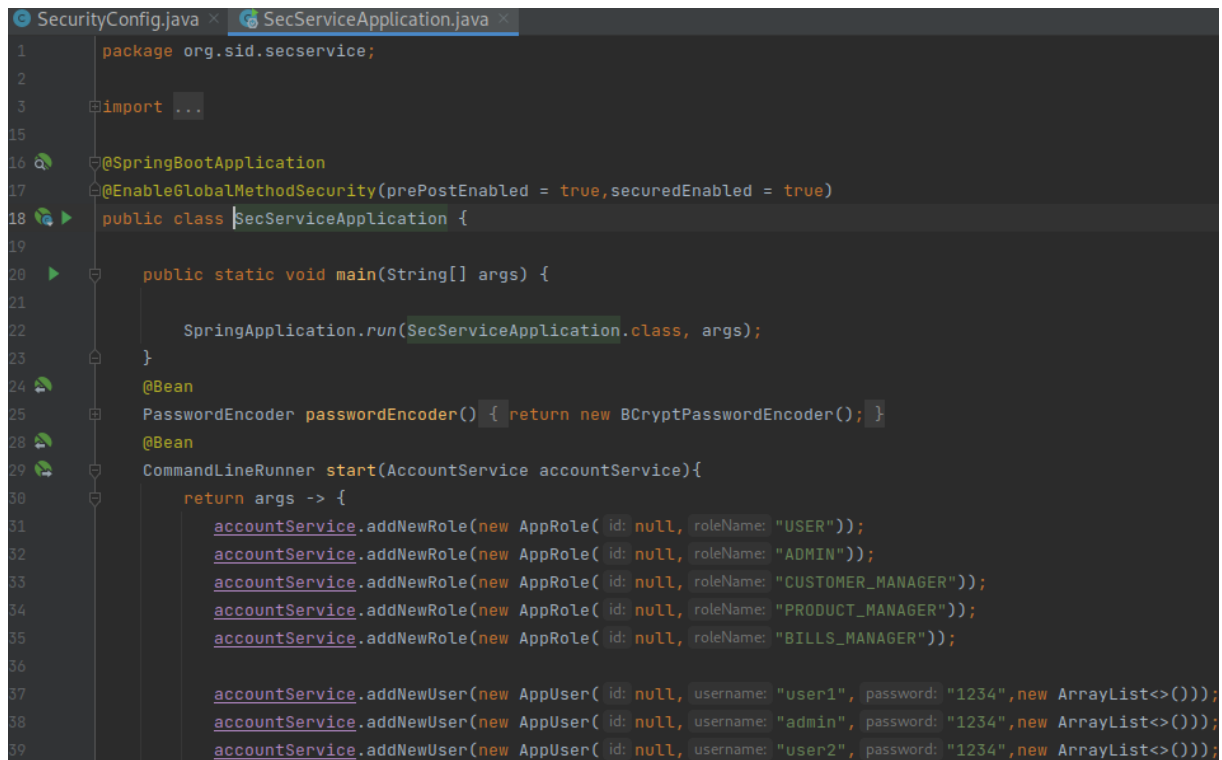
Application	AMIs	Availability Zones	Status
BILLING-SERVICE	n/a (1)	(1)	UP (1) - Ayoub.billing-service:8083
CUSTOMER-SERVICE	n/a (1)	(1)	UP (1) - Ayoub.customer-service:8081
GATEWAY-SERVICE	n/a (1)	(1)	UP (1) - Ayoub.gateway-service:8888
PRODUCT-SERVICE	n/a (1)	(1)	UP (1) - Ayoub.product-service:8082

General Info

Name	Value
total-avail-memory	220mb
num-of-cpus	4
current-memory-usage	34mb (15%)
server-uptime	00:17
registered-replicas	http://localhost:8761/eureka/

Figure 9 : Service Exécutée dans Eureka

5. Security Service:



```
1 package org.sid.secservice;
2
3 import ...
4
5
6
7
8
9
10
11
12
13
14
15
16 @SpringBootApplication
17 @EnableGlobalMethodSecurity(prePostEnabled = true,securedEnabled = true)
18 public class SecServiceApplication {
19
20     public static void main(String[] args) {
21
22         SpringApplication.run(SecServiceApplication.class, args);
23     }
24
25     @Bean
26     PasswordEncoder passwordEncoder() { return new BCryptPasswordEncoder(); }
27
28     @Bean
29     CommandLineRunner start(AccountService accountService){
30         return args -> {
31             accountService.addNewRole(new AppRole( id: null, roleName: "USER"));
32             accountService.addNewRole(new AppRole( id: null, roleName: "ADMIN"));
33             accountService.addNewRole(new AppRole( id: null, roleName: "CUSTOMER_MANAGER"));
34             accountService.addNewRole(new AppRole( id: null, roleName: "PRODUCT_MANAGER"));
35             accountService.addNewRole(new AppRole( id: null, roleName: "BILLS_MANAGER"));
36
37             accountService.addNewUser(new AppUser( id: null, username: "user1", password: "1234",new ArrayList<>()));
38             accountService.addNewUser(new AppUser( id: null, username: "admin", password: "1234",new ArrayList<>()));
39             accountService.addNewUser(new AppUser( id: null, username: "user2", password: "1234",new ArrayList<>()));
40         }
41     }
42 }
```

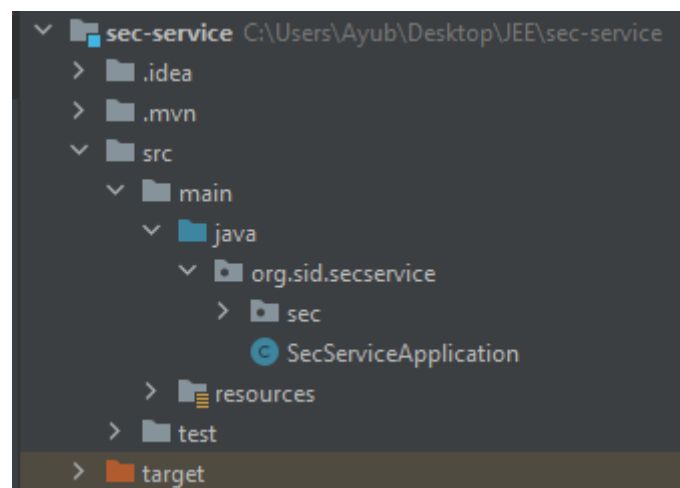


Figure 10 : Architecture Security Service

