ECOLE NORMALE SUPÉRIEURE DE L'ENSEIGNEMENT TECHNIQUE DE MOHAMMEDIA
UNIVERSITÉ HASSAN II DE CASABLANCA

المدرسة العليا لأساتذة التعليم التقني المحمدية جامعة الحسن الثاني بالدار البيضاء

2022-2023

Département Mathématiques et Informatique
GLSID 3 – S5

Systèmes Distribués et Big Data Processing

Rapport de devoir 1
Mise en œuvre d'une architecture micro-services

Préparé par : Fatima Zahra HASBI
Encadré par : Mr Mohamed YOUSSFI

Sommaire

Introduction

Ce rapport porte sur le devoir 1 de Systèmes Distribués et Big Data Processing, il consiste à mettre en œuvre une architecture micro-services.

L'objectif alors est de mettre en œuvre une application distribuée basée sur deux micro-services en utilisant les bonnes pratiques :

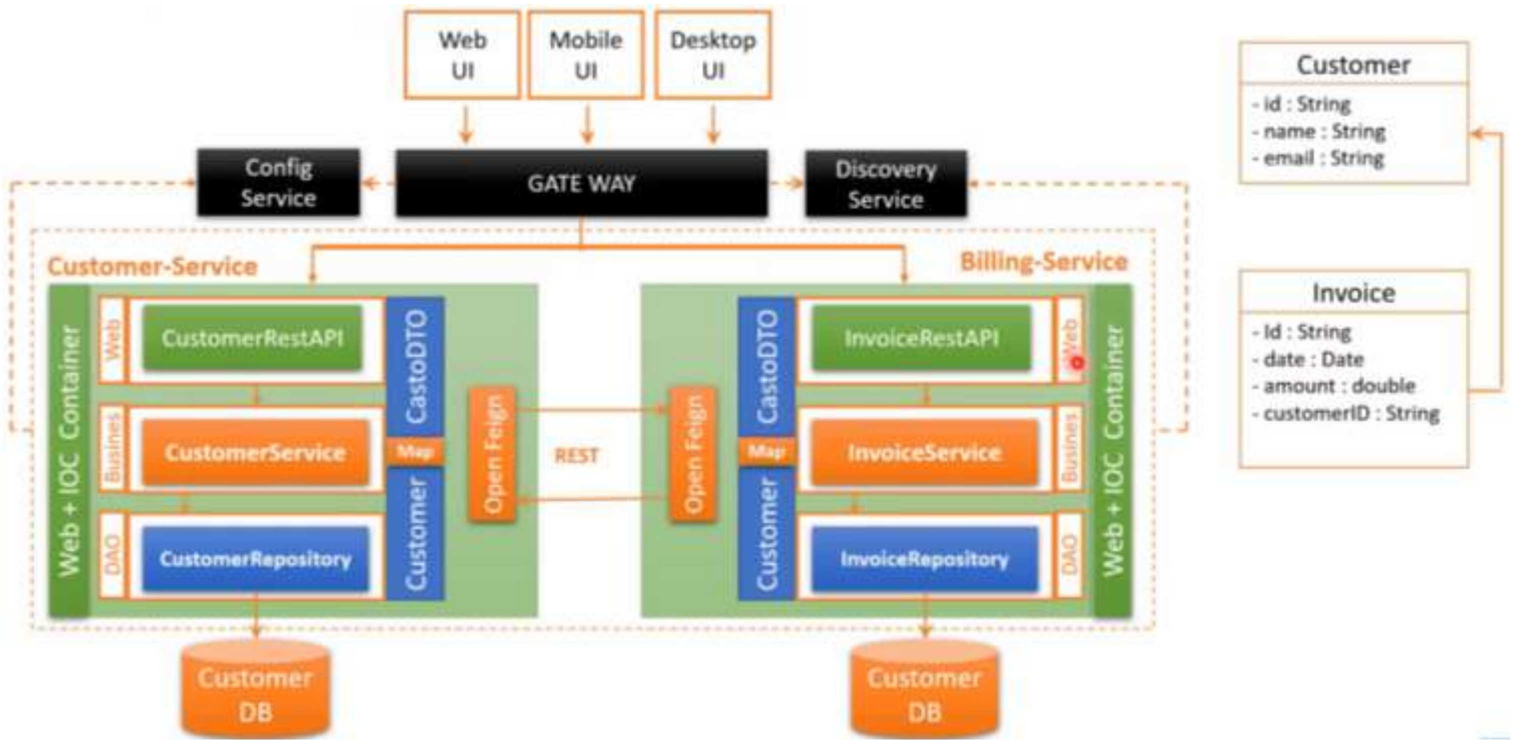
- Couches DAO, Service, Web, DTO
- Utilisation de MapStruct pour le mapping entre les objets Entities et DTO
- Génération des API-DOCS en utilisant SWAGGER3 (Open API)
- Communication entre micro-services en utilisant OpenFeign
- Spring Cloud Gateway
- Eureka Discovery Service

I- Structure de l'application

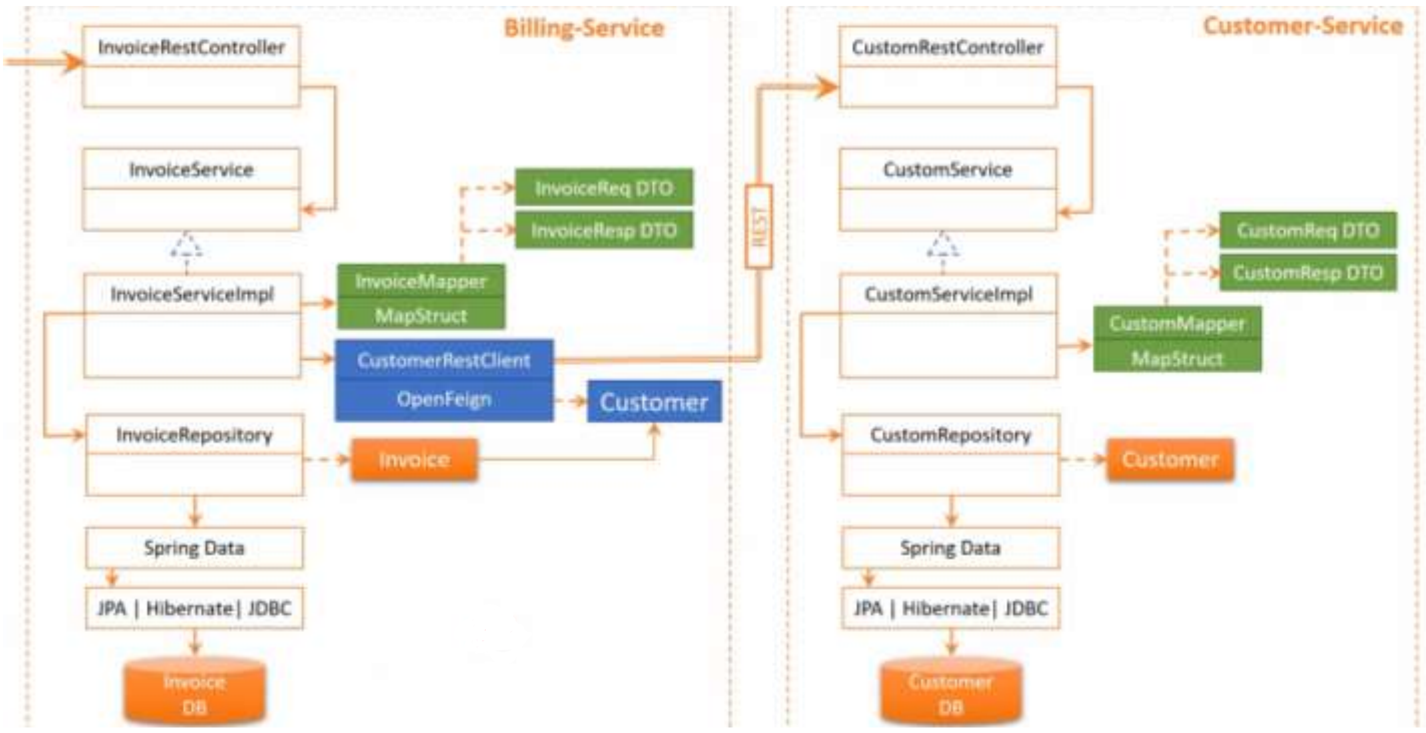
1- Modèle de l'application (Use case)

Dans cette application on va mettre en œuvre un application toutes les bonnes pratiques de l'architecture micro-services. Nous allons développer deux micro-services à savoir un pour la gestion du client et l'autre pour la facturation.

Le schéma ci-dessous représente le modèle que nous allons réaliser :



2- Architecture de l'application



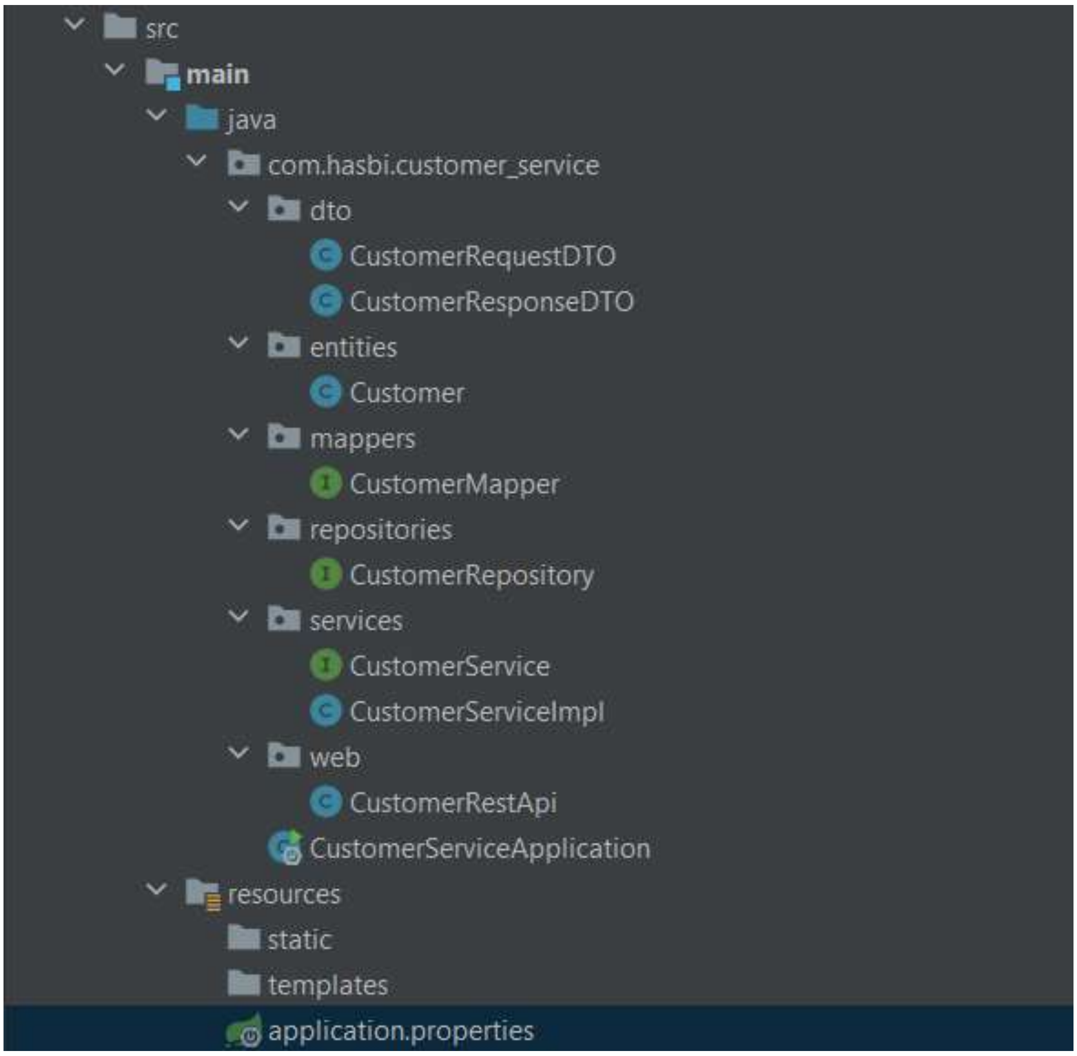
II- Premier micro-service Customer-service

1- Initialisation du projet sur Intellij (Les dépendances)

Lombok, Spring Web, Spring Data JPA, H2 Database, Eurila Discovery Client, Map struct, et Springdoc openapi (swagger)

```
dependencies {
    implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
    implementation 'org.springframework.boot:spring-boot-starter-web'
    implementation 'org.springframework.cloud:spring-cloud-starter-netflix-
eureka-client'
    compileOnly 'org.projectlombok:lombok'
    runtimeOnly 'com.h2database:h2'
    annotationProcessor 'org.projectlombok:lombok'
    testImplementation 'org.springframework.boot:spring-boot-starter-test'
    implementation 'org.springdoc:springdoc-openapi-ui:1.6.12'
    implementation 'org.mapstruct:mapstruct:1.5.3.Final'
    annotationProcessor 'org.mapstruct:mapstruct-processor:1.5.3.Final'
}
```

2- Les couches de l'application



```
src
  main
    java
      com.hasbi.customer_service
        dto
          CustomerRequestDTO
          CustomerResponseDTO
        entities
          Customer
        mappers
          CustomerMapper
        repositories
          CustomerRepository
        services
          CustomerService
          CustomerServiceImpl
        web
          CustomerRestApi
        CustomerServiceApplication
    resources
      static
      templates
      application.properties
```

3- Entities

```java
package com.hasbi.customer_service.entities;

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

import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
@Data @AllArgsConstructor @NoArgsConstructor
public class Customer {
    @Id
    private String id;
    private String name;
    private String email;
}
```

4- Repositories

```java
package com.hasbi.customer_service.repositories;

import com.hasbi.customer_service.entities.Customer;
import org.springframework.data.jpa.repository.JpaRepository;

public interface CustomerRepository extends JpaRepository<Customer, String>
{
}
```

5- DTO

```java
package com.hasbi.customer_service.dto;

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

@Data @AllArgsConstructor @NoArgsConstructor
public class CustomerRequestDTO {
    private String id;
    private String name;
    private String email;

}
```

```java
package com.hasbi.customer_service.dto;

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

@Data @AllArgsConstructor @NoArgsConstructor
public class CustomerResponseDTO {
    private String id;
    private String name;
    private String email;
}
```

6- Mappers

```
package com.hasbi.customer_service.mappers;

import com.hasbi.customer_service.dto.CustomerRequestDTO;
import com.hasbi.customer_service.dto.CustomerResponseDTO;
import com.hasbi.customer_service.entities.Customer;
import org.mapstruct.Mapper;

@Mapper(componentModel = "spring")
public interface CustomerMapper {
    CustomerResponseDTO customerToCustomerResponseDTO(Customer customer);
    Customer CustomerRequestDTOCustomer(CustomerRequestDTO
customerRequestDTO);
}
```

7- Services

```
package com.hasbi.customer_service.services;

import com.hasbi.customer_service.dto.CustomerRequestDTO;
import com.hasbi.customer_service.dto.CustomerResponseDTO;
import java.util.List;

public interface CustomerService {
    CustomerResponseDTO save(CustomerRequestDTO customerRequestDTO);
    CustomerResponseDTO getCustomer(String id);
    CustomerResponseDTO update(CustomerRequestDTO customerRequestDTO);
    List<CustomerResponseDTO> listCustomers();
    void deleteCustomer(String id);
}
```

```
package com.hasbi.customer_service.services;

import com.hasbi.customer_service.dto.CustomerRequestDTO;
import com.hasbi.customer_service.dto.CustomerResponseDTO;
import com.hasbi.customer_service.entities.Customer;
import com.hasbi.customer_service.mappers.CustomerMapper;
import com.hasbi.customer_service.repositories.CustomerRepository;
import lombok.AllArgsConstructor;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;
import java.util.stream.Collectors;

@Service
@Transactional @AllArgsConstructor
public class CustomerServiceImpl implements CustomerService {
    private CustomerRepository customerRepository;
    private CustomerMapper customerMapper;
    @Override
    public CustomerResponseDTO save(CustomerRequestDTO customerRequestDTO) {
        Customer customer =
customerMapper.CustomerRequestDTOCustomer(customerRequestDTO);
        Customer savedCustomer = customerRepository.save(customer);
        return customerMapper.customerToCustomerResponseDTO(savedCustomer);
    }

    @Override
    public CustomerResponseDTO getCustomer(String id) {
        Customer customer = customerRepository.findById(id).get();
        return customerMapper.customerToCustomerResponseDTO(customer);
    }

    @Override
    public CustomerResponseDTO update(CustomerRequestDTO customerRequestDTO)
{
        Customer customer =
customerMapper.CustomerRequestDTOCustomer(customerRequestDTO);
        Customer updatedCustomer = customerRepository.save(customer);
        return
customerMapper.customerToCustomerResponseDTO(updatedCustomer);
    }

    @Override
    public List<CustomerResponseDTO> listCustomers() {
        List<Customer> customers=customerRepository.findAll();
        List<CustomerResponseDTO> customerResponseDTOList =
                customers.stream()
                        .map(cust-
>customerMapper.customerToCustomerResponseDTO(cust))
                        .collect(Collectors.toList());
        return null;
    }

    @Override
    public void deleteCustomer(String id) {
        customerRepository.deleteById(id);
    }
}
```

8- Web

```
package com.hasbi.customer_service.web;

import com.hasbi.customer_service.dto.CustomerRequestDTO;
import com.hasbi.customer_service.dto.CustomerResponseDTO;
import com.hasbi.customer_service.services.CustomerService;
import org.springframework.web.bind.annotation.*;

import java.util.List;
import java.util.UUID;

@RestController
@RequestMapping(path ="/api")
public class CustomerRestApi {
    private CustomerService customerService;

    public CustomerRestApi(CustomerService customerService) {
        this.customerService = customerService;
    }

    @GetMapping(path = "/customers")
    public List<CustomerResponseDTO> allCustomers(){
        return customerService.listCustomers();
    }

    @PostMapping(path = "/customers")
    public CustomerResponseDTO save(@RequestBody CustomerRequestDTO
customerRequestDTO){
        customerRequestDTO.setId(UUID.randomUUID().toString());
        return customerService.save(customerRequestDTO);
    }

    @GetMapping(path = "/customers/{id}")
    public CustomerResponseDTO getCustomer(@PathVariable String id){
        return customerService.getCustomer(id);
    }

    @DeleteMapping(path = "/customers/{id}")
    public void deleteCustomer(@PathVariable String id){
        customerService.deleteCustomer(id);
    }

}
```

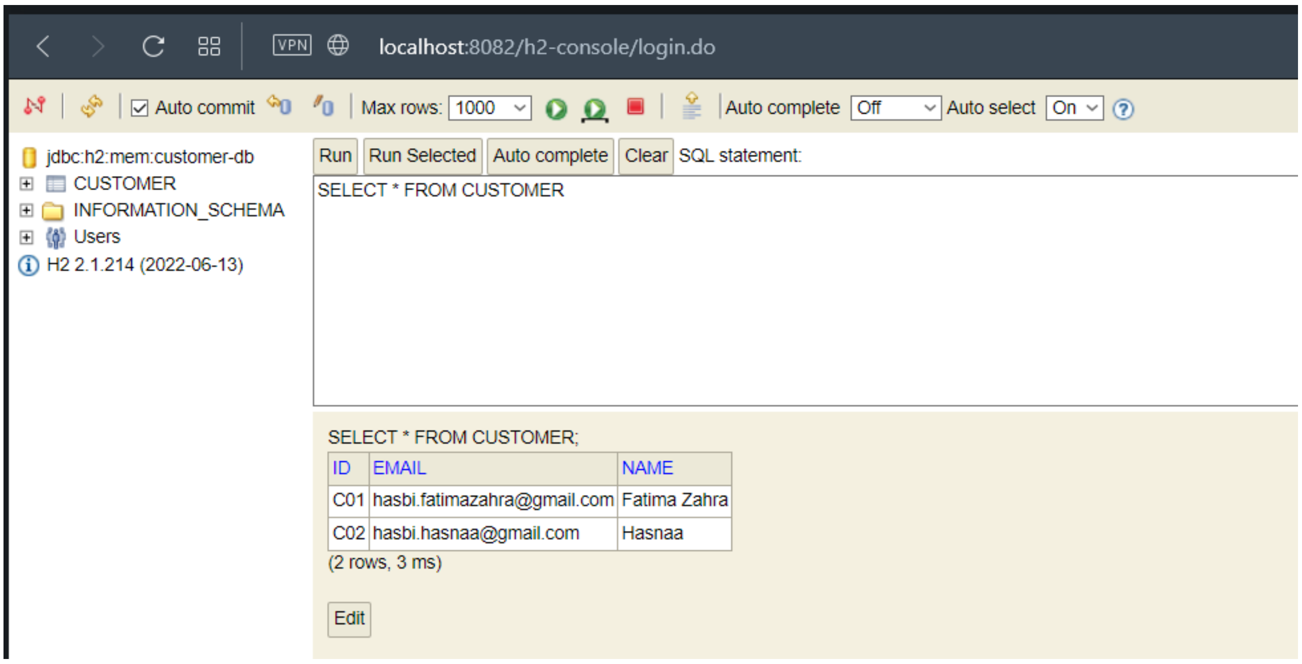
9- Application de test

```
package com.hasbi.customer_service;

import com.hasbi.customer_service.dto.CustomerRequestDTO;
import com.hasbi.customer_service.services.CustomerService;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
public class CustomerServiceApplication {

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

    @Bean
    CommandLineRunner start(CustomerService customerService){
        return args -> {
            customerService.save(new CustomerRequestDTO("C01", "Fatima
Zahra", "hasbi.fatimazahra@gmail.com"));
            customerService.save(new CustomerRequestDTO("C02", "Hasnaa",
"hasbi.hasnaa@gmail.com"));
        };
    }
}
```

10- application.properties

```
server.port=8082
spring.application.name=CUSTOMER_SERVICE
spring.h2.console.enabled=true
spring.cloud.discovery.enabled=false
spring.datasource.url=jdbc:h2:mem:customer-db
```

11- Test de ce micro-service



localhost:8082/h2-console/login.do

Auto commit | Max rows: 1000 | Auto complete: Off | Auto select: On

jdbc:h2:mem:customer-db
- CUSTOMER
- INFORMATION_SCHEMA
- Users

H2 2.1.214 (2022-06-13)

Run | Run Selected | Auto complete | Clear | SQL statement:

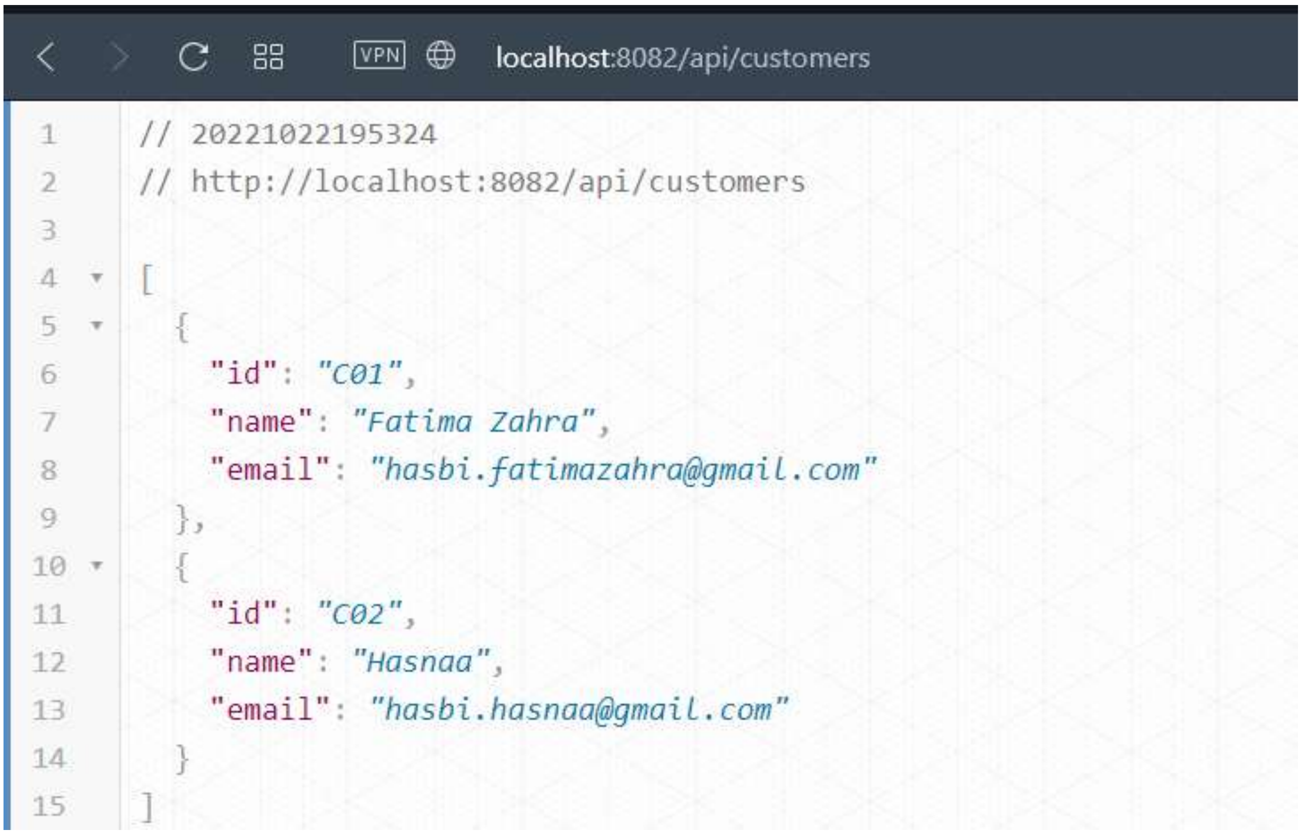
```
SELECT * FROM CUSTOMER
```

SELECT * FROM CUSTOMER;

ID	EMAIL	NAME
C01	hasbi.fatimazahra@gmail.com	Fatima Zahra
C02	hasbi.hasnaa@gmail.com	Hasnaa

(2 rows, 3 ms)

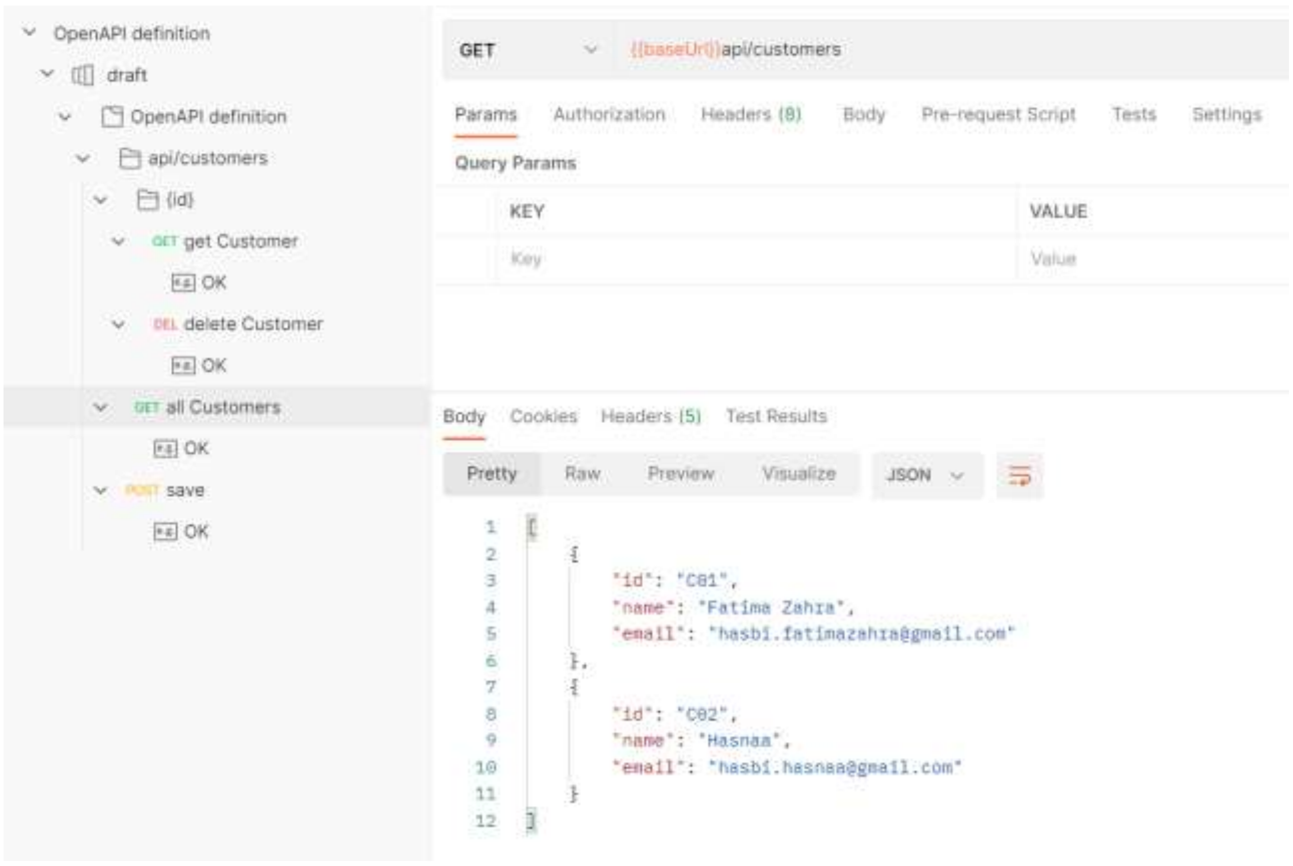
Edit



localhost:8082/api/customers

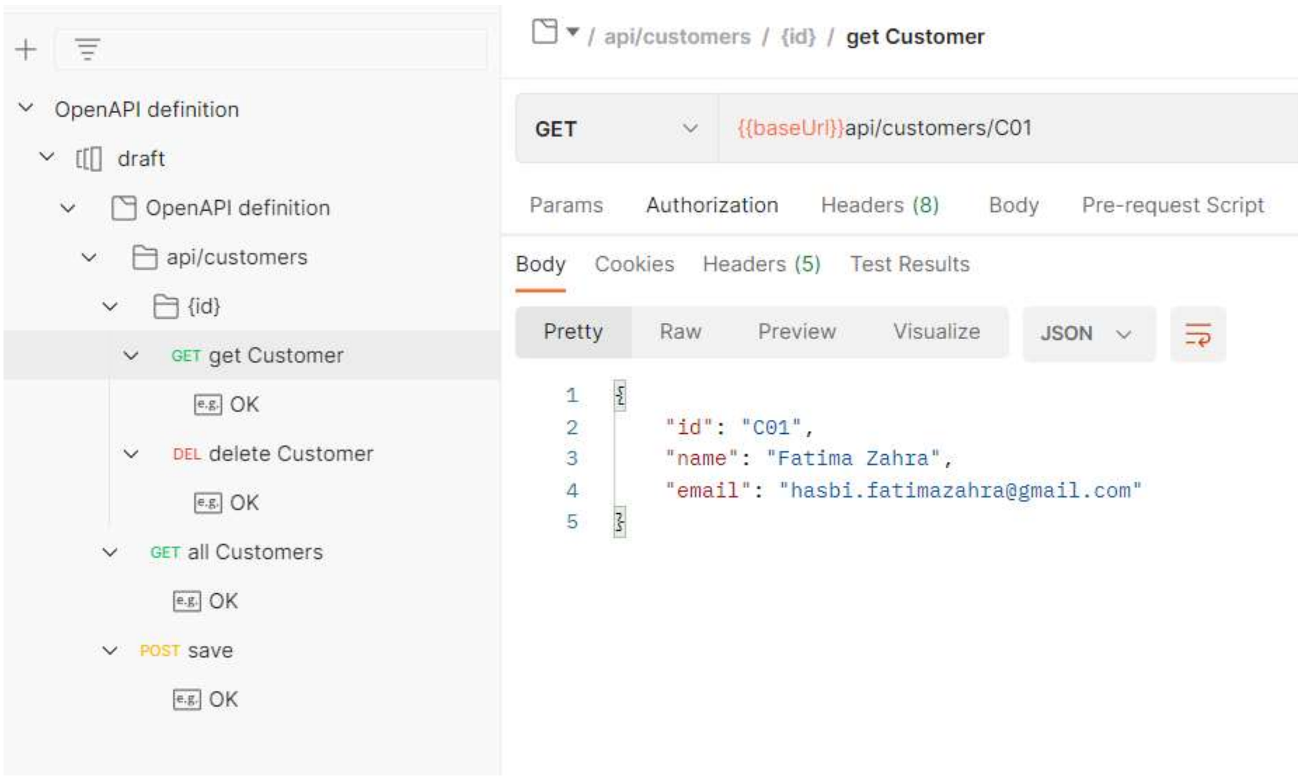
```
// 20221022195324
// http://localhost:8082/api/customers

[
  {
    "id": "C01",
    "name": "Fatima Zahra",
    "email": "hasbi.fatimazahra@gmail.com"
  },
  {
    "id": "C02",
    "name": "Hasnaa",
    "email": "hasbi.hasnaa@gmail.com"
  }
]
```

Sur Postman :



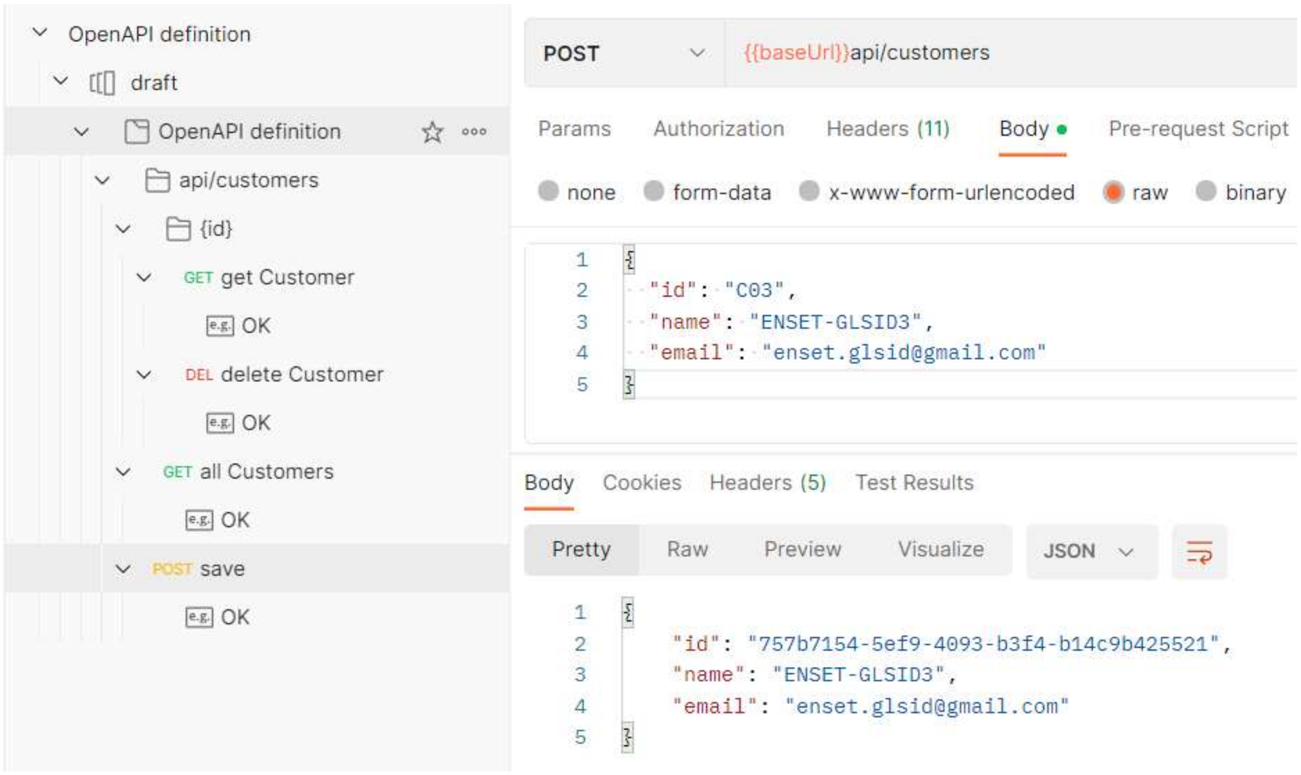
GET {{baseUrl}}api/customers

```
[
    {
        "id": "C01",
        "name": "Fatima Zahra",
        "email": "hasbi.fatimazahra@gmail.com"
    },
    {
        "id": "C02",
        "name": "Hasnaa",
        "email": "hasbi.hasnaa@gmail.com"
    }
]
```



GET {{baseUrl}}api/customers/C01

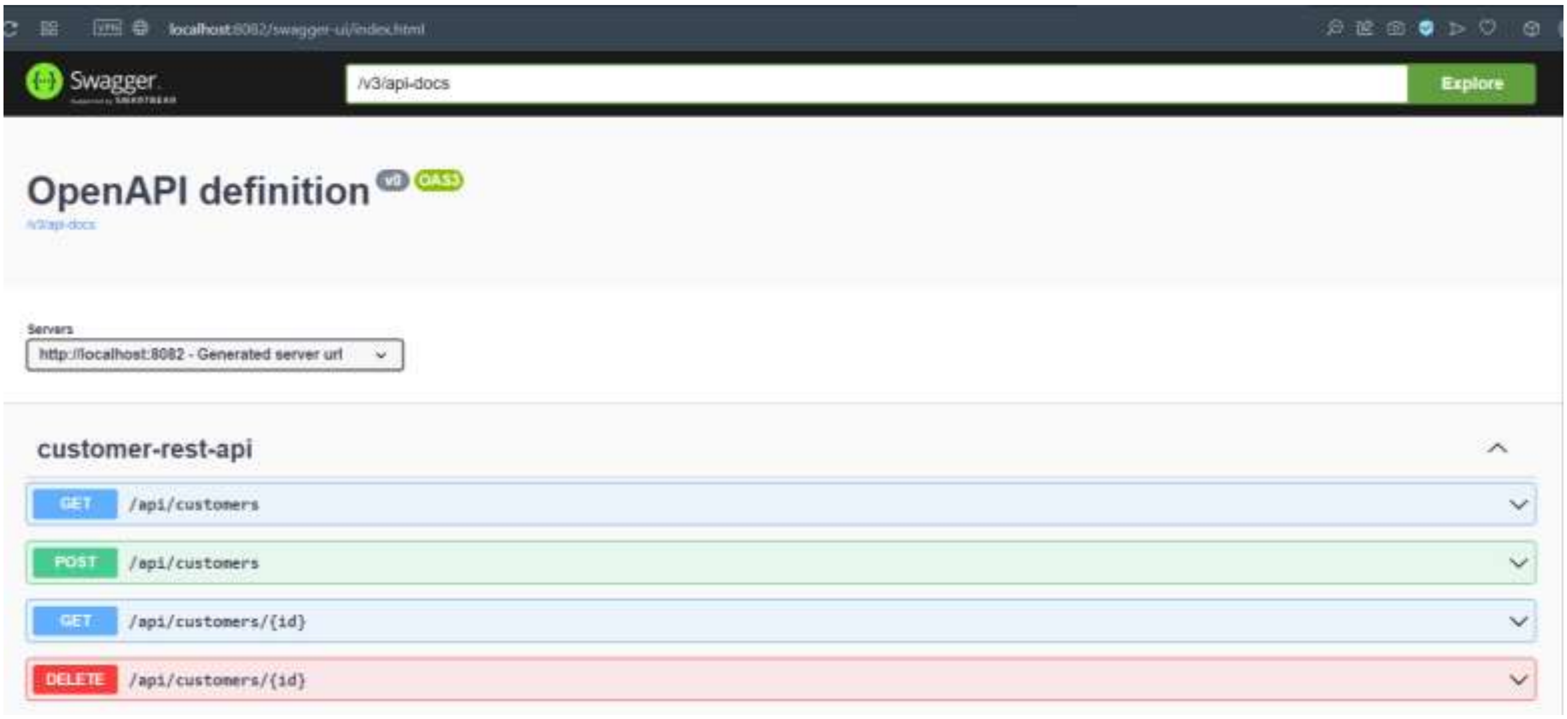
```
{
    "id": "C01",
    "name": "Fatima Zahra",
    "email": "hasbi.fatimazahra@gmail.com"
}
```



POST {{baseUrl}}api/customers

```
{
  "id": "C03",
  "name": "ENSET-GLSID3",
  "email": "enset.glsid@gmail.com"
}
```

```
{
    "id": "757b7154-5ef9-4093-b3f4-b14c9b425521",
    "name": "ENSET-GLSID3",
    "email": "enset.glsid@gmail.com"
}
```



A ce point, nous avons développé le premier micro-service (Customer_service) dans lequel nous avons respecté un ensemble de bonnes pratiques et séparant les entités et les DTOs, en utilisant les Mappers, open api…etc.

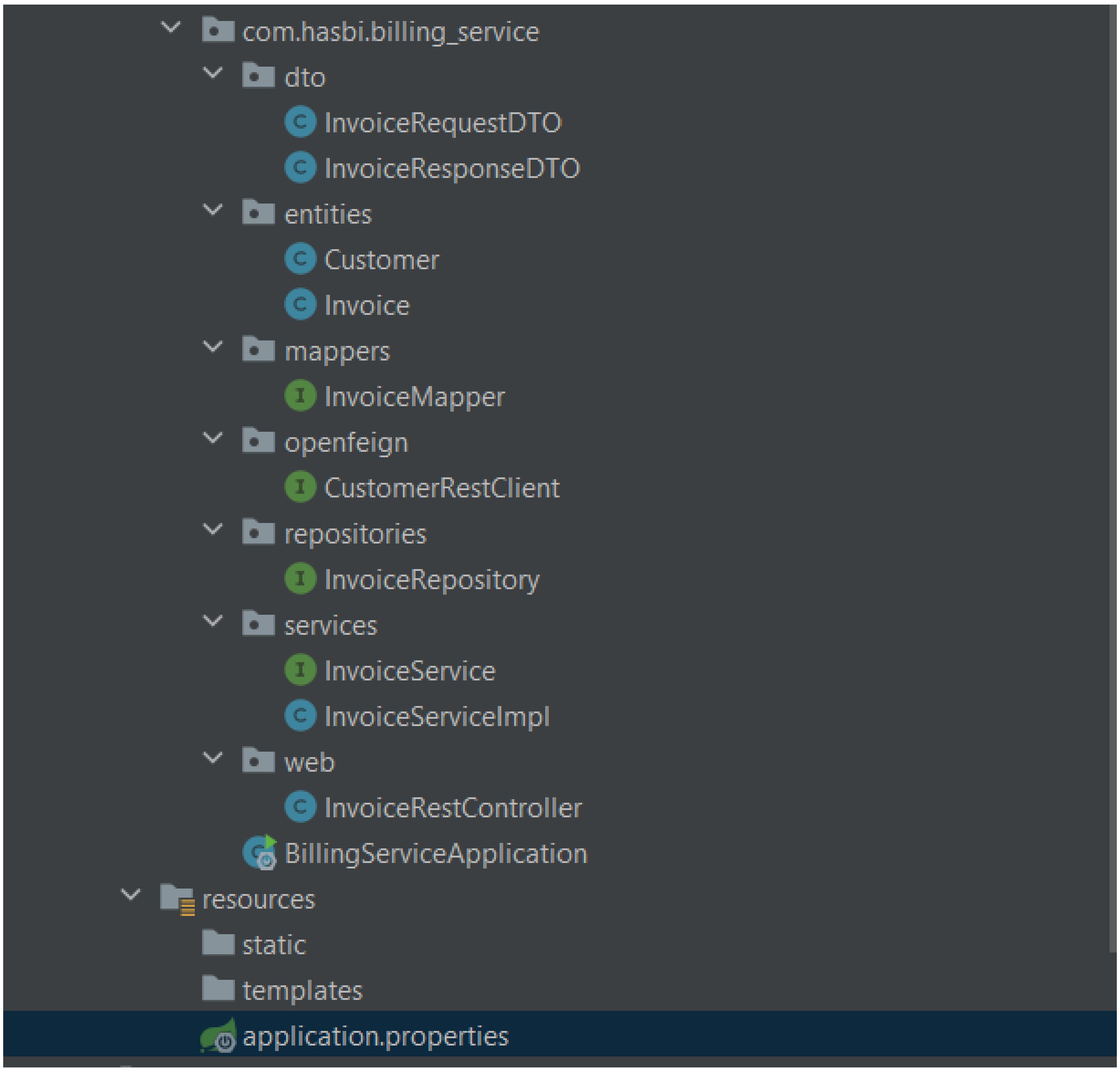
Nous allons continuer maintenant dans l’étape suivante qui vise à mettre en œuvre le micro service de service de facturation dans lequel nous allons utiliser OpenFeign pour pouvoir communiquer avec le micro-service customer_service.

III- Deuxième micro-service Billing-service

Nous allons utiliser les mêmes dépendances en ajoutant celui de OpenFeign :

```
dependencies {
    implementation 'org.springframework.cloud:spring-cloud-starter-
openfeign'

}
```

1- Les couches de l'application



```
com.hasbi.billing_service
  dto
    InvoiceRequestDTO
    InvoiceResponseDTO
  entities
    Customer
    Invoice
  mappers
    InvoiceMapper
  openfeign
    CustomerRestClient
  repositories
    InvoiceRepository
  services
    InvoiceService
    InvoiceServiceImpl
  web
    InvoiceRestController
  BillingServiceApplication
resources
  static
  templates
  application.properties
```

2- Entities

```java
package com.hasbi.billing_service.entities;

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

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Transient;
import java.math.BigDecimal;
import java.util.Date;

@Entity
@Data @NoArgsConstructor @AllArgsConstructor
public class Invoice {
    @Id
    private String id;
    private Date date;
    private BigDecimal amount;
    private String customerID;
    @Transient
    private Customer customer;
}
```

```java
package com.hasbi.billing_service.entities;

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

@Data @AllArgsConstructor @NoArgsConstructor
public class Customer {
    private String id;
    private String name;
    private String email;

}
```

3- Repositories

```java
package com.hasbi.billing_service.repositories;

import com.hasbi.billing_service.entities.Invoice;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface InvoiceRepository extends JpaRepository<Invoice, String> {
    List<Invoice> findByCustomerID(String customerId);
}
```

4- DTO

```java
package com.hasbi.billing_service.dto;

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

import java.math.BigDecimal;
import java.util.Date;

@Data @NoArgsConstructor @AllArgsConstructor
public class InvoiceRequestDTO {
    private BigDecimal amount;
    private String customerID;
}
```

```java
package com.hasbi.billing_service.dto;

import com.hasbi.billing_service.entities.Customer;
import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Transient;
import java.math.BigDecimal;
import java.util.Date;

@Data @NoArgsConstructor @AllArgsConstructor
public class InvoiceResponseDTO {
    private String id;
    private Date date;
    private BigDecimal amount;
    private Customer customer;
}
```

5- Mappers

```java
package com.hasbi.billing_service.mappers;

import com.hasbi.billing_service.dto.InvoiceRequestDTO;
import com.hasbi.billing_service.dto.InvoiceResponseDTO;
import com.hasbi.billing_service.entities.Invoice;
import org.mapstruct.Mapper;

@Mapper(componentModel = "spring")
public interface InvoiceMapper {
    Invoice fromInvoiceRequestDTO(InvoiceRequestDTO invoiceRequestDTO);
    InvoiceResponseDTO fromInvoice(Invoice invoice);
}
```

6- OpenFeign class

```java
package com.hasbi.billing_service.openfeign;

import com.hasbi.billing_service.entities.Customer;
import org.springframework.cloud.openfeign.FeignClient;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;

import java.util.List;

@FeignClient(name = "CUSTOMER_SERVICE")
public interface CustomerRestClient {
    @GetMapping(path = "/api/customers/{id}")
    Customer getCustomer(@PathVariable(name = "id") String id);

    @GetMapping(path = "/api/customers")
    List<Customer> allCustomers();

}
```

7- Services

```java
package com.hasbi.billing_service.services;

import com.hasbi.billing_service.dto.InvoiceRequestDTO;
import com.hasbi.billing_service.dto.InvoiceResponseDTO;

import java.util.List;

public interface InvoiceService {

    InvoiceResponseDTO save(InvoiceRequestDTO invoiceRequestDTO);
    InvoiceResponseDTO getInvoice(String invoiceId);
    List<InvoiceResponseDTO> invoicesByCustomerId(String customerId);
}
```

```java
package com.hasbi.billing_service.services;

import com.hasbi.billing_service.dto.InvoiceRequestDTO;
import com.hasbi.billing_service.dto.InvoiceResponseDTO;
import com.hasbi.billing_service.entities.Customer;
import com.hasbi.billing_service.entities.Invoice;
import com.hasbi.billing_service.mappers.InvoiceMapper;
import com.hasbi.billing_service.openfeign.CustomerRestClient;
import com.hasbi.billing_service.repositories.InvoiceRepository;
import lombok.AllArgsConstructor;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.util.List;
import java.util.UUID;
import java.util.stream.Collectors;

@Service
@Transactional
@AllArgsConstructor
public class InvoiceServiceImpl implements InvoiceService {
    private InvoiceRepository invoiceRepository;
    private InvoiceMapper mapper;
    private CustomerRestClient customerRestClient;

    @Override
    public InvoiceResponseDTO save(InvoiceRequestDTO invoiceRequestDTO) {
        Invoice invoice = mapper.fromInvoiceRequestDTO(invoiceRequestDTO);
        invoice.setId(UUID.randomUUID().toString());
        Invoice savedInvoice = invoiceRepository.save(invoice);
        return mapper.fromInvoice(savedInvoice);
    }

    @Override
    public InvoiceResponseDTO getInvoice(String invoiceId) {
        Invoice invoice = invoiceRepository.findById(invoiceId).get();
        Customer customer =
customerRestClient.getCustomer(invoice.getCustomerID());
        invoice.setCustomer(customer);
        return mapper.fromInvoice(invoice);
    }

    @Override
    public List<InvoiceResponseDTO> invoicesByCustomerId(String customerId)
{
        List<Invoice> invoices =
invoiceRepository.findByCustomerID(customerId);
        return invoices
                .stream()
                .map(invoice -> mapper.fromInvoice(invoice))
                .collect(Collectors.toList());
    }
}
```

8- Web

```
package com.hasbi.billing_service.web;

import com.hasbi.billing_service.dto.InvoiceRequestDTO;
import com.hasbi.billing_service.dto.InvoiceResponseDTO;
import com.hasbi.billing_service.services.InvoiceService;
import lombok.AllArgsConstructor;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping(path = "api")
@AllArgsConstructor
public class InvoiceRestController{
    private InvoiceService invoiceService;

    @GetMapping(path = "/invoices/{id}")
    public InvoiceResponseDTO getInvoice(@PathVariable(name = "id") String
invoiceId){
        return invoiceService.getInvoice(invoiceId);
    }

    @GetMapping(path = "/invoices/{customerId}")
    public List<InvoiceResponseDTO> getInvoicesByCustomer(@PathVariable
String customerId){
        return invoiceService.invoicesByCustomerId(customerId);
    }

    @PostMapping(path = "/invoices")
    public InvoiceResponseDTO save(@RequestBody InvoiceRequestDTO
invoiceRequestDTO){
        return invoiceService.save(invoiceRequestDTO);
    }
}
```

9- Application de test

```
package com.hasbi.billing_service;

import com.hasbi.billing_service.dto.InvoiceRequestDTO;
import com.hasbi.billing_service.services.InvoiceService;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.cloud.openfeign.EnableFeignClients;
import org.springframework.context.annotation.Bean;

import java.math.BigDecimal;

@SpringBootApplication
@EnableFeignClients
public class BillingServiceApplication {

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


    @Bean
    CommandLineRunner start(InvoiceService invoiceService){
        return args -> {
            invoiceService.save(new
InvoiceRequestDTO(BigDecimal.valueOf(10000), "C01"));
            invoiceService.save(new
InvoiceRequestDTO(BigDecimal.valueOf(12000), "C01"));
            invoiceService.save(new
InvoiceRequestDTO(BigDecimal.valueOf(20000), "C02"));
        };
    }
}
```

10- application.properties

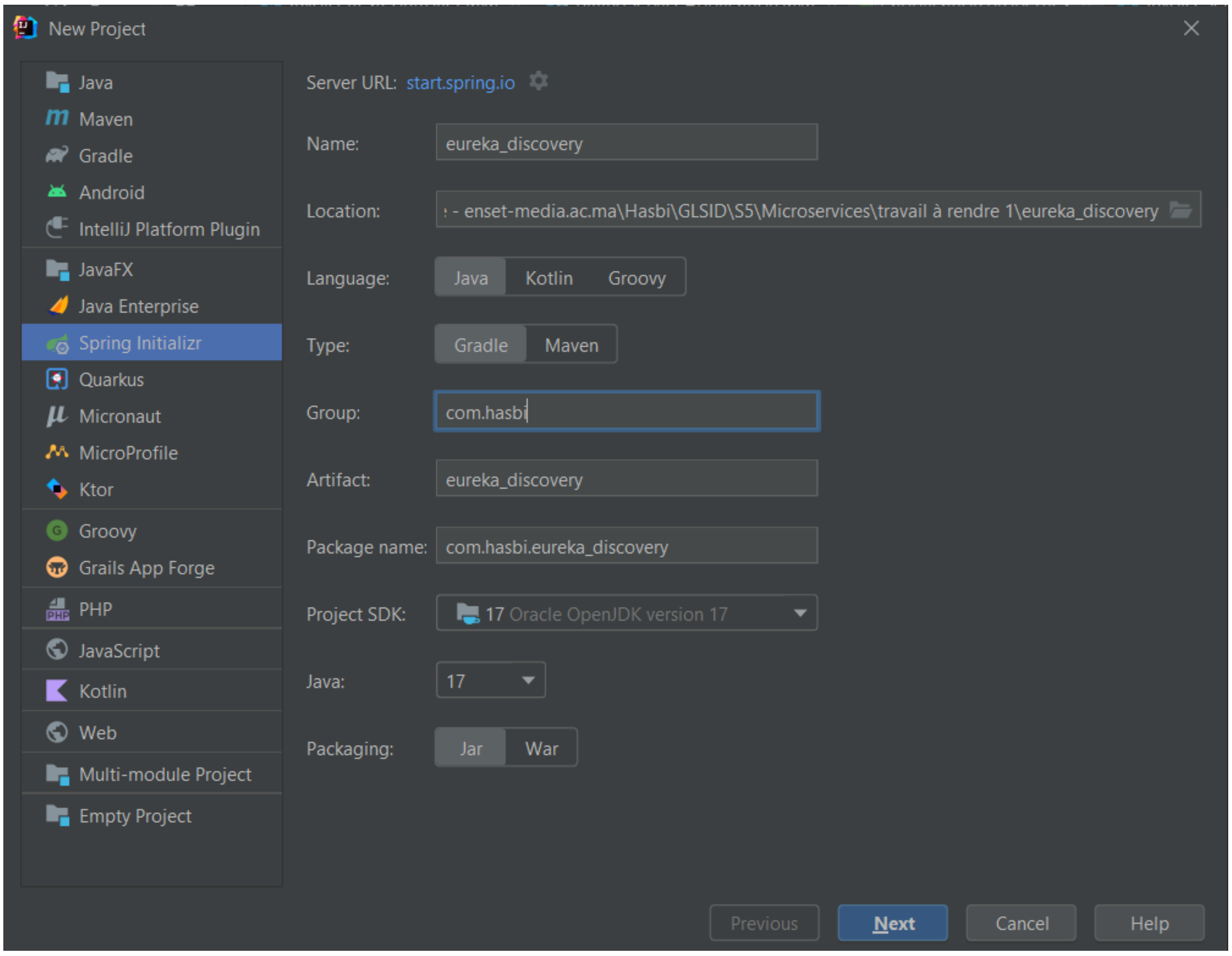
```
server.port=8083
spring.application.name=BILLING_SERVICE
spring.h2.console.enabled=true
spring.cloud.discovery.enabled=false
spring.datasource.url=jdbc:h2:mem:billing-db
```

10- Test de ce micro-service

Nous ne pouvons pas tester avant la création de service Gateway et Discovery car ce micro-service fait appel au micro-service Customer_service.

IV- Discovery service

Pour créer le service Discovery on aura besoin de créer un projet.



New Project

Server URL: start.spring.io

Name: eureka_discovery

Location: : - enset-media.ac.ma\Hasbi\GLSID\S5\Microservices\travail à rendre 1\eureka_discovery

Language: [✓ Java] Kotlin Groovy

Type: [✓ Gradle] Maven

Group: com.hasbi

Artifact: eureka_discovery

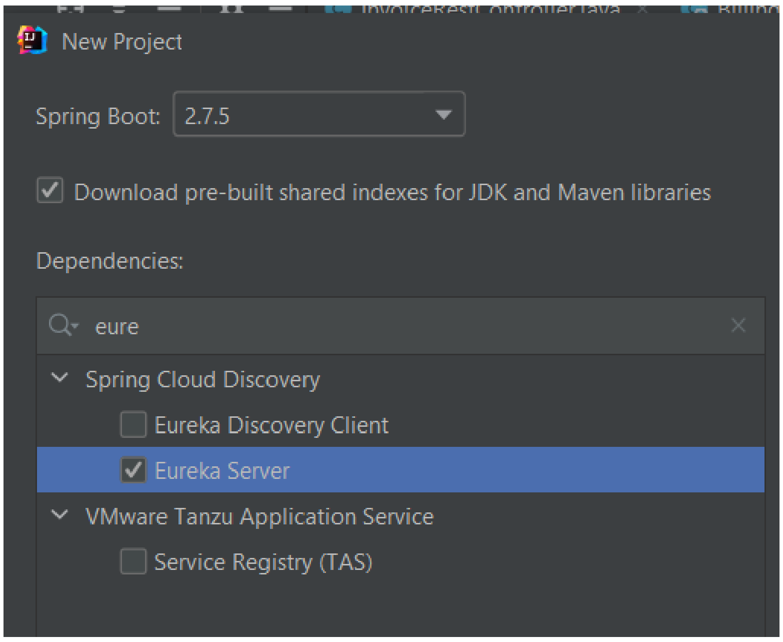
Package name: com.hasbi.eureka_discovery

Project SDK: 17 Oracle OpenJDK version 17

Java: 17

Packaging: [✓ Jar] War

Pour les dépendances on aura besoin juste de : eureka-server



New Project

Spring Boot: 2.7.5

☑ Download pre-built shared indexes for JDK and Maven libraries

Dependencies:

eure

Spring Cloud Discovery
- ☐ Eureka Discovery Client
- ☑ Eureka Server

VMware Tanzu Application Service
- ☐ Service Registry (TAS)

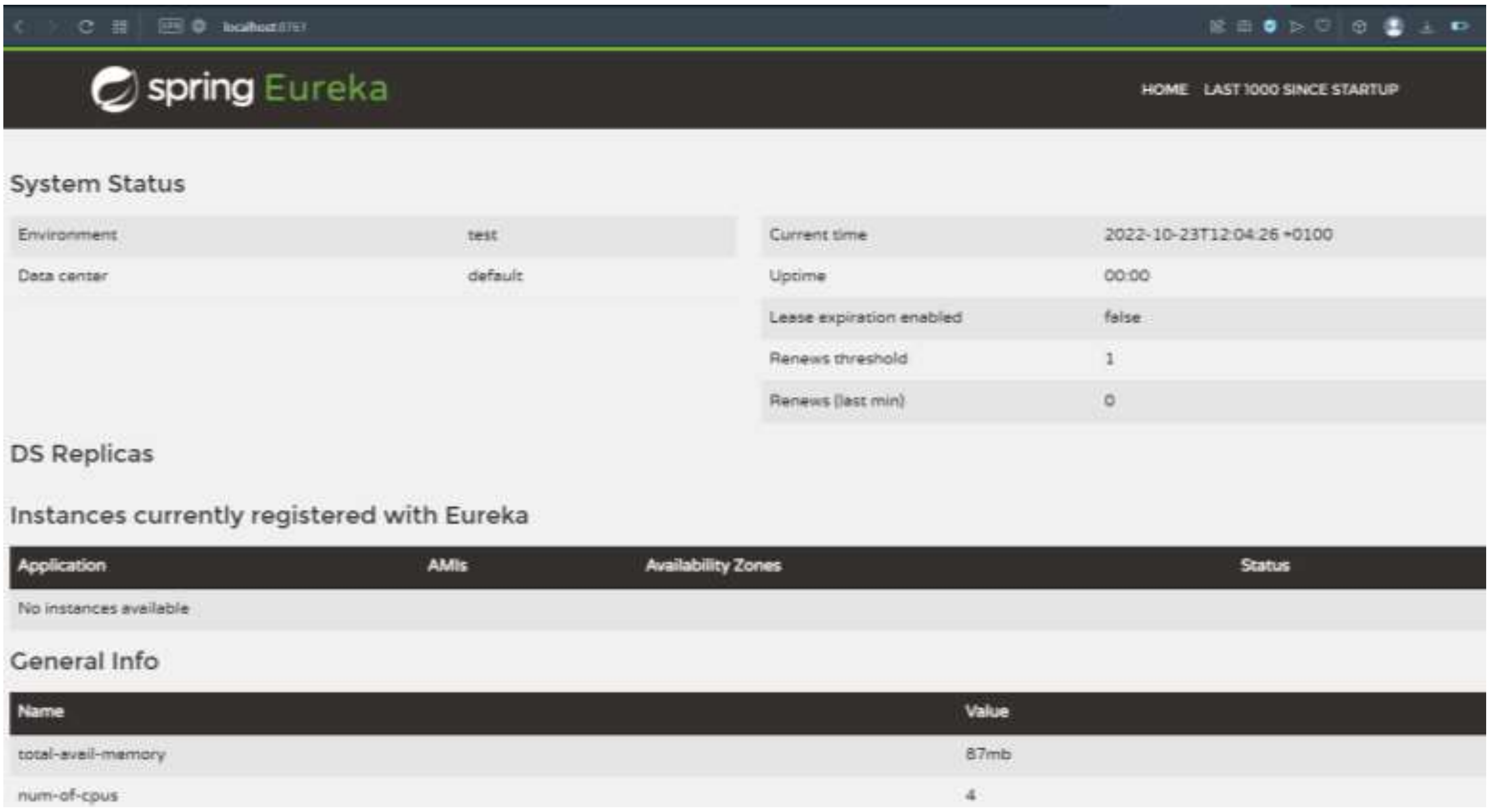
Maintenant on ajoute @EnableEurekaServer dans l'application main :

```java
package com.hasbi.eureka_discovery;

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);
    }

}
```

Dans application.properties

```
server.port=8761
# dont register server itself as a client
eureka.client.fetch-registry=false
#Does not register itself in the service registry
eureka.client.register-with-eureka=false
```

On démarre maintenant l'application et on accède à localhost:8761



System Status

Environment	test
Data center	default

Current time	2022-10-23T12:04:26 +0100
Uptime	00:00
Lease expiration enabled	false
Renews threshold	1
Renews (last min)	0

DS Replicas

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
No instances available			

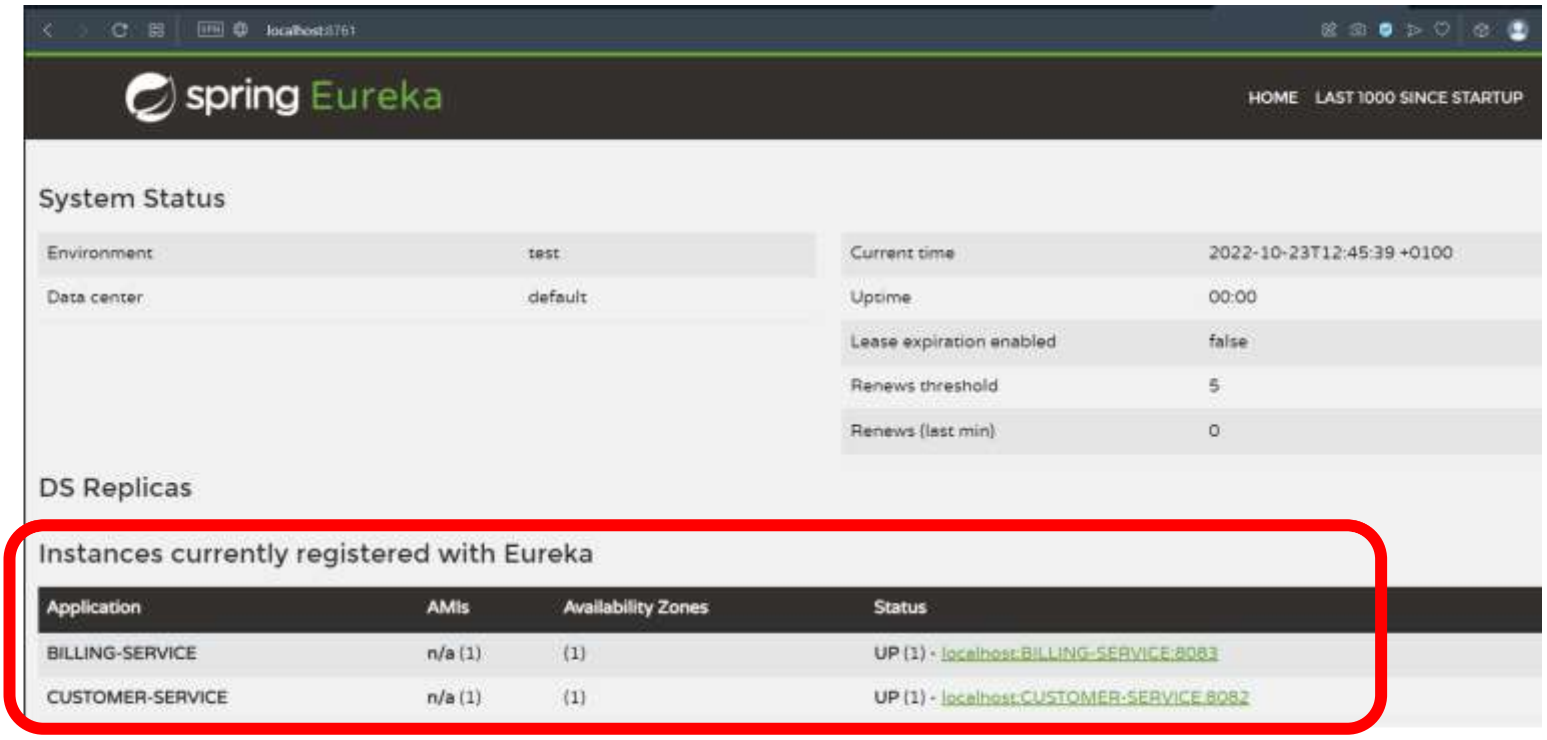
General Info

Name	Value
total-avail-memory	87mb
num-of-cpus	4

Nous allons modifier application.properties des micro-services qui sont client de ce serveur en activant la propriété spring.cloud.discovery :

```
spring.cloud.discovery.enabled=true
```

Lorsque nous démarrons nos deux micro-services maintenant ils vont être enregistrés dans eureka discovery :



spring Eureka

HOME LAST 1000 SINCE STARTUP

System Status

Environment	test
Data center	default

Current time	2022-10-23T12:45:39 +0100
Uptime	00:00
Lease expiration enabled	false
Renews threshold	5
Renews (last min)	0

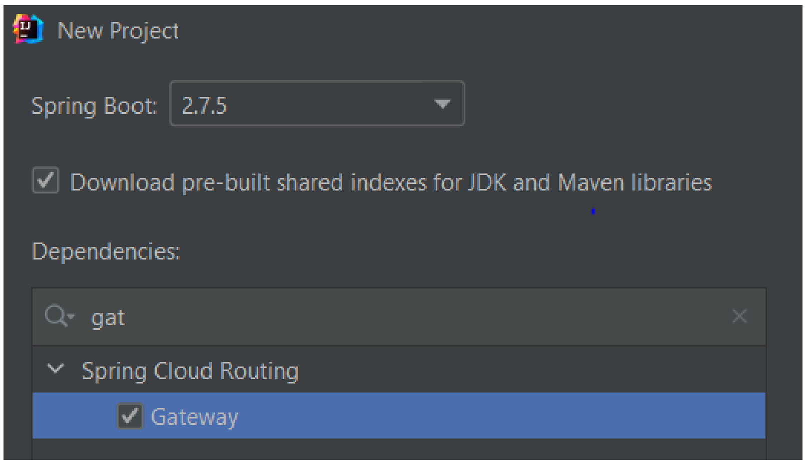
DS Replicas

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
BILLING-SERVICE	n/a (1)	(1)	UP (1) - localhost:BILLING-SERVICE:8083
CUSTOMER-SERVICE	n/a (1)	(1)	UP (1) - localhost:CUSTOMER-SERVICE:8082

V- Gateway service

Pour créer le service Gateway il suffit de créer un nouveau projet spring en ajoutant la dépendance Spring Cloud Gateway et celle de eureka client service pour s'enregistrer au discovery service. Et on va configurer les routes d'une manière dynamique.



New Project

Spring Boot: 2.7.5

✓ Download pre-built shared indexes for JDK and Maven libraries

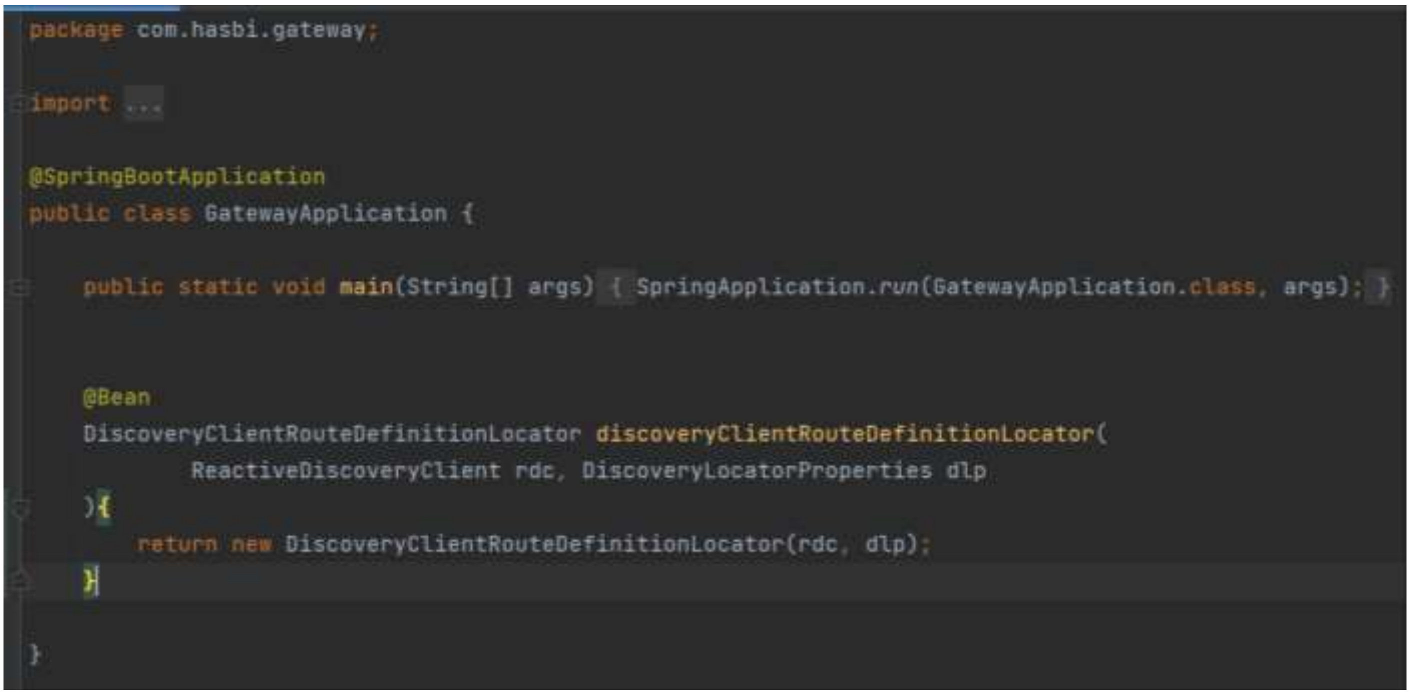
Dependencies:

gat

Spring Cloud Routing

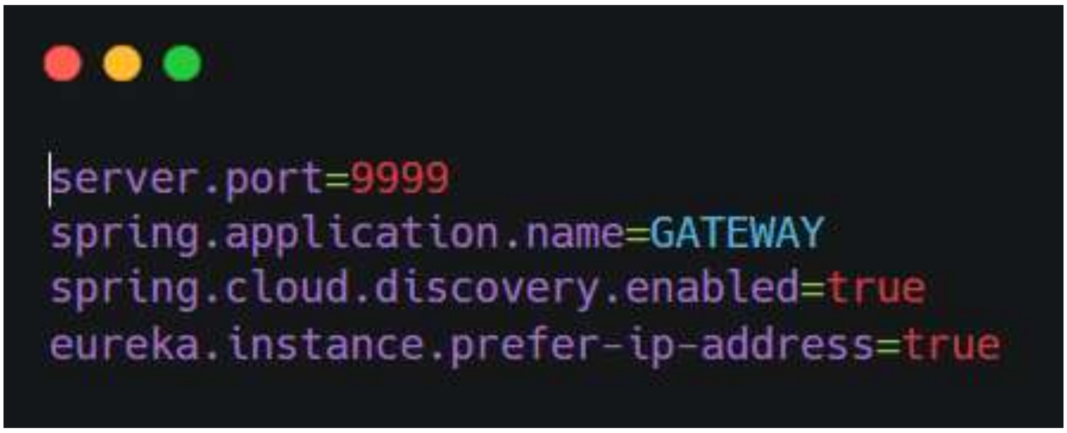
✓ Gateway

Ajoutons maintenant une méthode de configuration dynamique avec l'annotation @Bean :



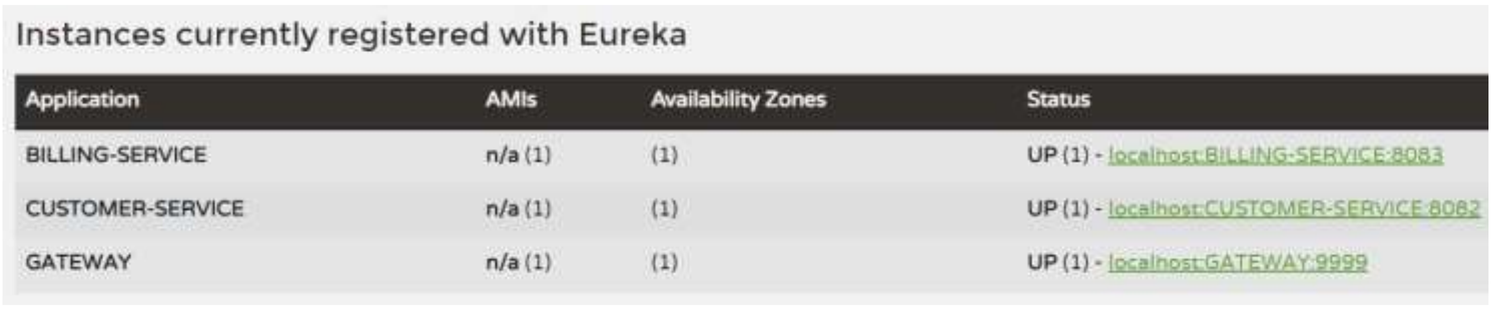
```
package com.hasbi.gateway;

import ...

@SpringBootApplication
public class GatewayApplication {

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


    @Bean
    DiscoveryClientRouteDefinitionLocator discoveryClientRouteDefinitionLocator(
            ReactiveDiscoveryClient rdc, DiscoveryLocatorProperties dlp
    ){
        return new DiscoveryClientRouteDefinitionLocator(rdc, dlp);
    }

}
```

Il nous reste que la configuration de application.properties



```
server.port=9999
spring.application.name=GATEWAY
spring.cloud.discovery.enabled=true
eureka.instance.prefer-ip-address=true
```

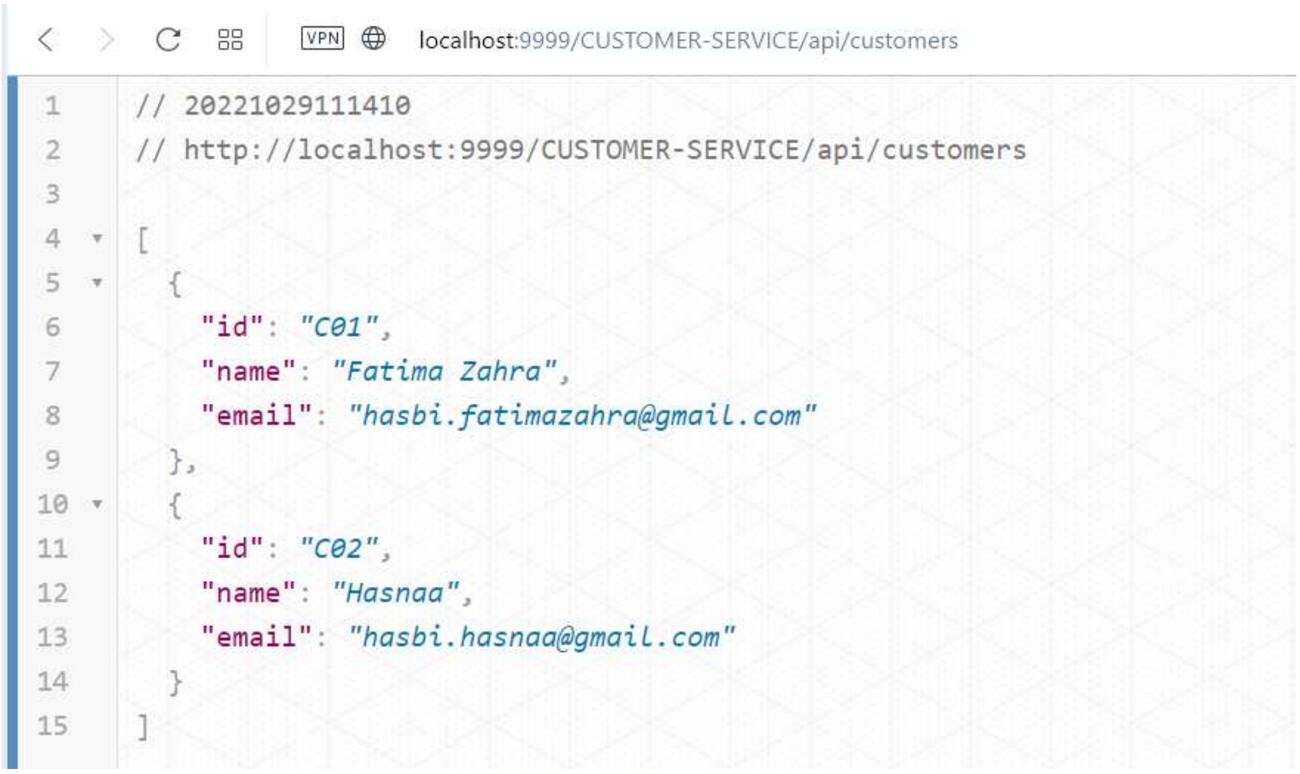
Maintenant si nous démarrons notre gateway, il sera également enregistré dans discovery service



Instances currently registered with Eureka

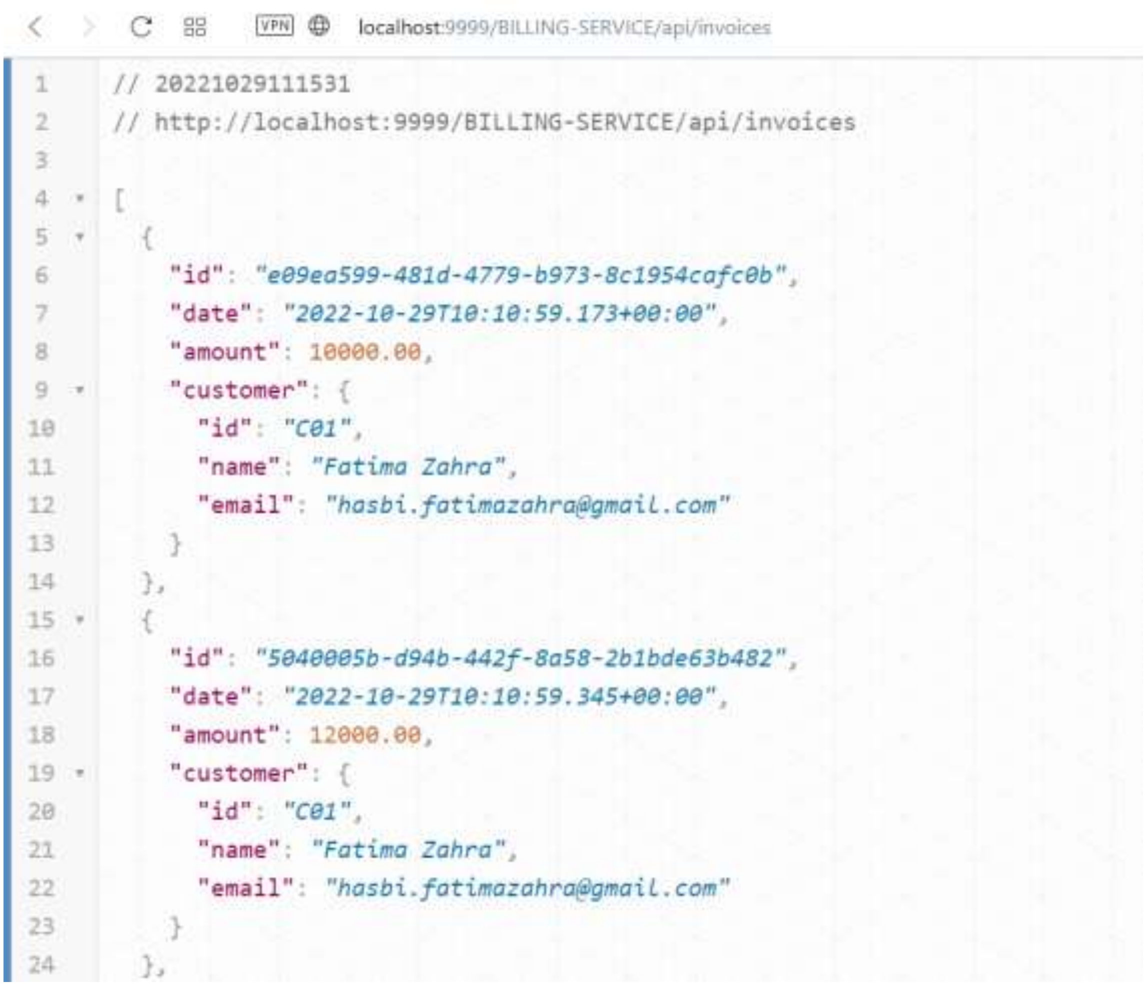
Application	AMIs	Availability Zones	Status
BILLING-SERVICE	n/a (1)	(1)	UP (1) - localhost:BILLING-SERVICE:8083
CUSTOMER-SERVICE	n/a (1)	(1)	UP (1) - localhost:CUSTOMER-SERVICE:8082
GATEWAY	n/a (1)	(1)	UP (1) - localhost:GATEWAY:9999

Maintenant nous allons accéder aux APIs via le gateway en utilisant le nom de micro-service dans le path: localhost:9999/CUSTOMER-SERVICE/api/customers



localhost:9999/CUSTOMER-SERVICE/api/customers

```
// 20221029111410
// http://localhost:9999/CUSTOMER-SERVICE/api/customers

[
  {
    "id": "C01",
    "name": "Fatima Zahra",
    "email": "hasbi.fatimazahra@gmail.com"
  },
  {
    "id": "C02",
    "name": "Hasnaa",
    "email": "hasbi.hasnaa@gmail.com"
  }
]
```



localhost:9999/BILLING-SERVICE/api/invoices

```
// 20221029111531
// http://localhost:9999/BILLING-SERVICE/api/invoices

[
  {
    "id": "e09ea599-481d-4779-b973-8c1954cafc0b",
    "date": "2022-10-29T10:10:59.173+00:00",
    "amount": 10000.00,
    "customer": {
      "id": "C01",
      "name": "Fatima Zahra",
      "email": "hasbi.fatimazahra@gmail.com"
    }
  },
  {
    "id": "5040005b-d94b-442f-8a58-2b1bde63b482",
    "date": "2022-10-29T10:10:59.345+00:00",
    "amount": 12000.00,
    "customer": {
      "id": "C01",
      "name": "Fatima Zahra",
      "email": "hasbi.fatimazahra@gmail.com"
    }
  },
```

Conclusion

Ce projet nous a donné une compréhension claire de l'architecture micro-services et comment créer une application basée sur cette architecture.

Nous avons pu alore mettre en place une application distribuée basée sur deux micro-services en utilisant les bonnes pratiques :

- Couches DAO, Service, Web, DTO
- Utilisation de MapStruct pour le mapping entre les objets Entities et DTO
- Génération des API-DOCS en utilisant SWAGGER3 (Open API)
- Communication entre micro-services en utilisant OpenFeign
- Spring Cloud Gateway
- Eureka Discovery Service

Le lien du projet sur GitHub :
https://github.com/FatimaZahraHASBI/micro-services