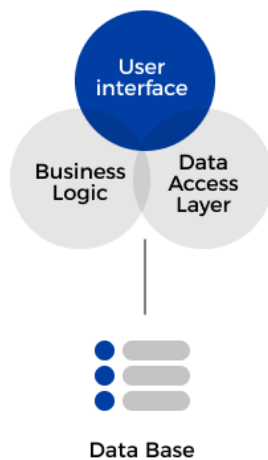




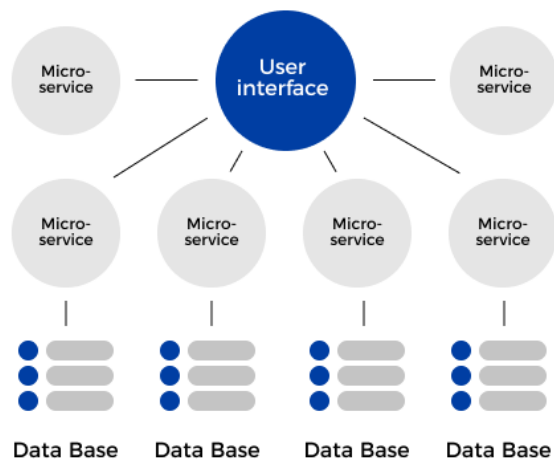
## COMPTE-RENDU DE MISE EN OEUVRE D'UNE ARCHITECTURE MICRO-SERVICES

Filière : « Ingénierie Informatique : Big Data et Cloud  
Computing » II-BDCC

### MONOLITHIC ARCHITECTURE



### MICROSERVICE ARCHITECTURE



Réalisé par :

Khadija BENJILALI

Encadré par :

Pr. Mohamed YOUSSEFI

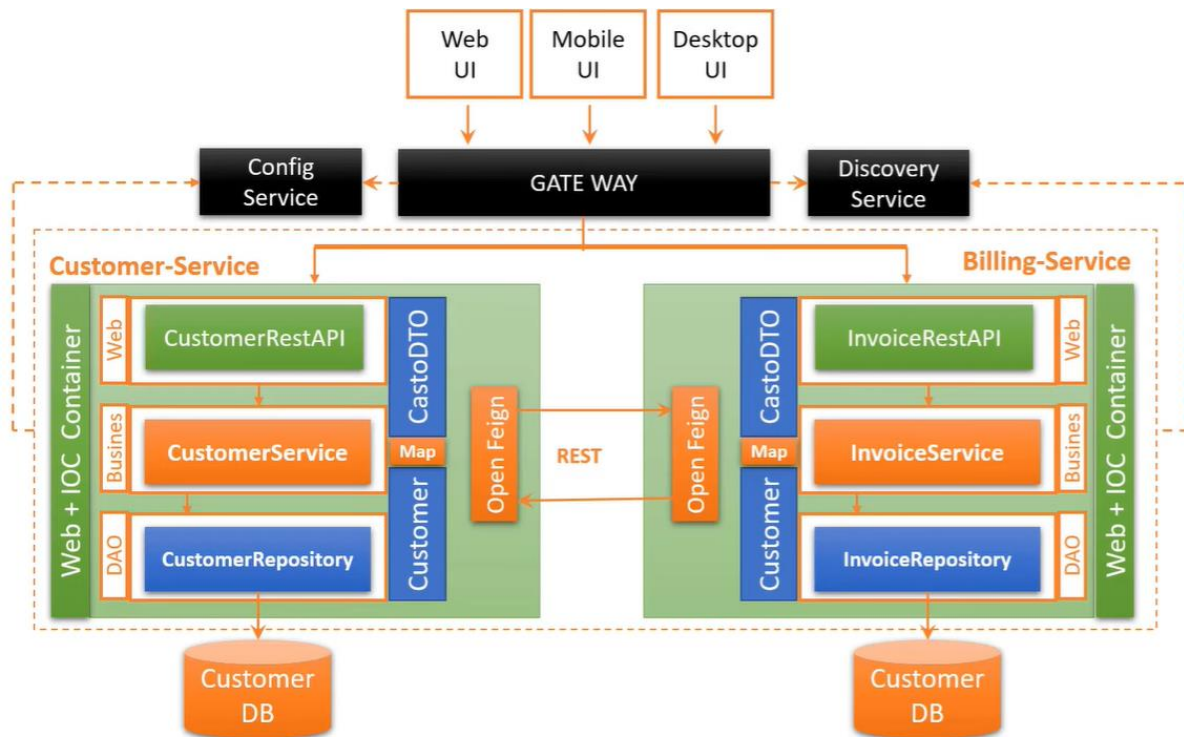
Année Universitaire : 2022-2023

# Sommaire

Travail à faire.....	1
<b>PARTIE 1 : Création de micro-service Customer .....</b>	<b>2</b>
1. Structure du projet : .....	2
2. Entité Customer : .....	2
3. Mapping entre Entité et DTO en utilisant MapStruct : .....	2
4. Service de Customer : .....	3
5. Rest API de Customer: .....	3
6. API-DOCS de Customer en utilisant Open API: .....	3
<b>PARTIE 2 : Création de micro-service Billing.....</b>	<b>4</b>
1. Configuration de service : .....	4
2. Entités de BILLING-SERVICE: .....	4
4. Service de Invoice – Exemple de l’ajout: .....	5
5. Test à ajouter des factures : .....	5
6. Résultat de test: .....	5
7. Retourner les factures d’un client : .....	6
<b>PARTIE 3 : Création de Eureka Discovery Service .....</b>	<b>6</b>
1. Configuration de Eureka Discovery: .....	6
2. Mettez en place le service Eureka : .....	7
3. Lancer Eureka : .....	7
<b>PARTIE 4 : Création de Spring Cloud Gateway : .....</b>	<b>7</b>
1. Inclure Spring Cloud Gateway dans l’application: .....	7
2. Propriétés de Gateway: .....	7
3. Gateway dans Eureka Discovery: .....	8
4. Les routes de Gateway: .....	8
<b>PARTIE 5 : Test global de l’application: .....</b>	<b>8</b>
1. Service de Customer .....	8
2. Service de Billing: .....	9
<b>PARTIE 6 : Dockerisation des services: .....</b>	<b>10</b>
<b>Résumé .....</b>	<b>12</b>

# Travail à faire

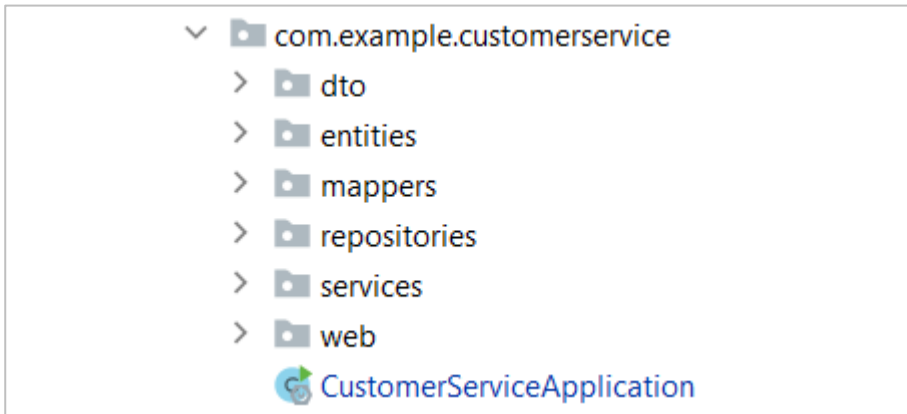
Mise en œuvre d'une application distribuée basée sur deux micro-services en utilisant les bonnes pratiques :



- 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

## PARTIE 1 : Création de micro-service Customer

### 1. Structure du projet :



### 2. Entité Customer :

```
10 @Entity
11 @Data @NoArgsConstructor @AllArgsConstructor
12 public class Customer {
13     @Id
14     private String id;
15     private String name;
16     private String email;
17 }
```

### 3. Mapping entre Entité et DTO en utilisant MapStruct :

```
public interface CustomerMapper
{
    CustomerResponseDTO customerToCustomerResponseDTO(Customer customer);
    Customer customerRequestDTOToCustomer(CustomerRequestDTO customerRequestDTO);
}
```

MapStruct génère cette implémentation dont elle est charge de faire le mapping entre l'objet de l'entité Customer et de CustomerResponseDTO(par exemple).

```
14 @Component
15 public class CustomerMapperImpl implements CustomerMapper {
16
17     @Override
18     public CustomerResponseDTO customerToCustomerResponseDTO(Customer customer) {
19         if ( customer == null ) {
20             return null;
21         }
22
23         CustomerResponseDTO customerResponseDTO = new CustomerResponseDTO();
24
25         customerResponseDTO.setId( customer.getId() );
26         customerResponseDTO.setName( customer.getName() );
27         customerResponseDTO.setEmail( customer.getEmail() );
28
29         return customerResponseDTO;
30     }
```

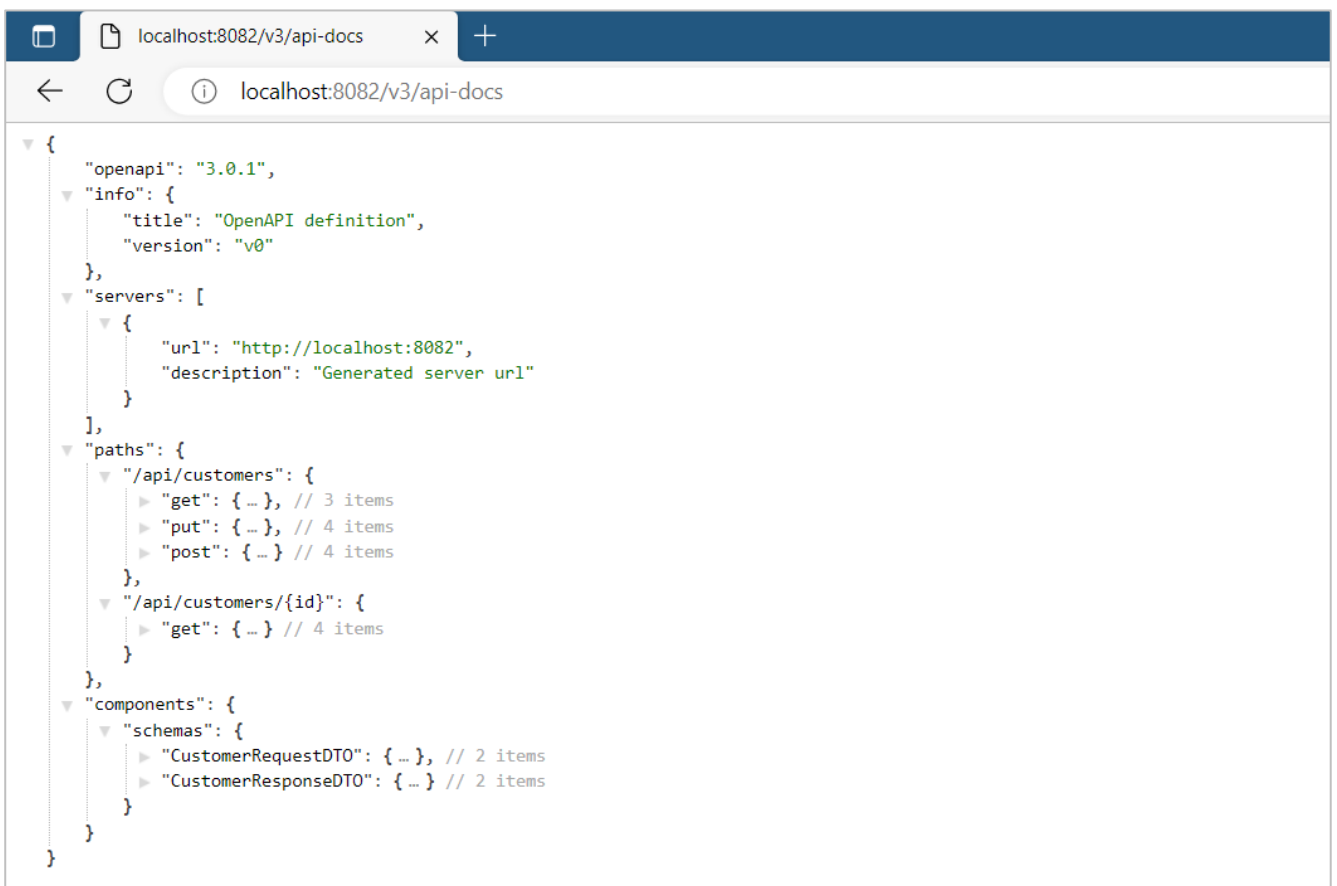
#### 4. Service de Customer :

```
8 public interface CustomerService {
9     CustomerResponseDTO addCustomer(CustomerRequestDTO customerRequestDTO);
10    CustomerResponseDTO getCustomer(String id);
11    CustomerResponseDTO update(CustomerRequestDTO customerRequestDTO);
12    List<CustomerResponseDTO> listCustomers();
13 }
```

#### 5. Rest API de Customer:

```
11 @RestController
12 @RequestMapping(path = "/api")
13 public class CustomerRestAPI
14 {
15     private CustomerService customerService;
16
17     public CustomerRestAPI(CustomerService customerService) { this.customerService = customerService; }
18
19     @GetMapping(path = "/customers")
20     public List<CustomerResponseDTO> allCustomers() { return customerService.listCustomers(); }
21
22     @GetMapping(path = "/customers/{id}")
23     public CustomerResponseDTO getCustomer(@PathVariable String id) { return customerService.getCustomer(id); }
24 }
```

#### 6. API-DOCS de Customer en utilisant Open API:



```
{
  "openapi": "3.0.1",
  "info": {
    "title": "OpenAPI definition",
    "version": "v0"
  },
  "servers": [
    {
      "url": "http://localhost:8082",
      "description": "Generated server url"
    }
  ],
  "paths": {
    "/api/customers": {
      "get": { /* 3 items */ },
      "put": { /* 4 items */ },
      "post": { /* 4 items */ }
    },
    "/api/customers/{id}": {
      "get": { /* 4 items */ }
    }
  },
  "components": {
    "schemas": {
      "CustomerRequestDTO": { /* 2 items */ },
      "CustomerResponseDTO": { /* 2 items */ }
    }
  }
}
```

## PARTIE 2 : Création de micro-service Billing

### 1. Configuration de service :

```
1  server.port=8083
2  spring.application.name=BILLING-SERVICE
3  spring.h2.console.enabled=true
4  spring.cloud.discovery.enabled=true
5  #eureka.instance.prefer-ip-address=true
6  spring.datasource.url=jdbc:h2:mem:billing-db
```

### 2. Entités de BILLING-SERVICE:

➤ Entité JPA persistante **Invoice** :

```
13  @Entity
14  @Data @AllArgsConstructor @NoArgsConstructor
15  public class Invoice {
16      @Id
17      private String id;
18      private Date date;
19      private BigDecimal amount;
20      private String customerId;
21      @Transient // cette attribut n'est pas persistant
22      private Customer customer;
23  }
```

➤ Entité **Customer** : pour stocker l'objet Customer récupéré par OpenFeign à partir le service CUSTOMER-SERVICE et l'attribué à une facture(Invoice)

### 3. Communication entre micro-services en utilisant OpenFeign :

OpenFeign est un outil qui permet la simplification de la communication entre micro services, en générant automatiquement les requêtes HTTP à partir des données fournies dans des classes.

Une fois la réponse du micro service distant reçue on obtient en retour directement des objets Java prêts à l'emploi.

```
11  @FeignClient(name = "CUSTOMER-SERVICE")
12  public interface CustomerRestClient
13  {
14      @GetMapping(path = "/api/customers/{id}")
15      Customer getCustomer(@PathVariable(name = "id") String idCustomer);
16
17      @GetMapping(path = "/api/customers")
18      List<Customer> getAllCustomers();
19  }
```

#### 4. Service de Invoice – Exemple de l'ajout:

Cette méthode ajoute à la base h2 une nouvelle facture, mais avant d'effectuer cette opération une instance de l'interface CustomerRestClient qui sert à communiquer entre les micro service à l'aide de l'outil OpenFeign, récupère l'objet Customer à partir son id depuis le micro-service « CUSTOMER-SERVICE » et si ce client existe dans la base de ce micro-service cette fonction ajoute la facture de ce client à sa base de données.

```
35      @Override
36      public InvoiceResponseDTO addInvoice(InvoiceRequestDTO invoiceRequestDTO)
37      {
38          /*
39              Vérification de l'intégrité référencielle Invoice / Customer
40          */
41          Customer customer = null;
42          try {
43              customer = customerRestClient.getCustomer(invoiceRequestDTO.getCustomerId());
44          }
45          catch (Exception e)
46          {
47              throw new CustomerNotFoundException("Customer not found");
48          }
49          Invoice invoice = invoiceMapper.fromInvoiceRequestDTO(invoiceRequestDTO);
50          invoice.setId(UUID.randomUUID().toString());
51          invoice.setDate(new Date());
52          Invoice saveInvoice = invoiceRepository.save(invoice); // save est de jpa
53          saveInvoice.setCustomer(customer);
54          InvoiceResponseDTO invoiceResponseDTO = invoiceMapper.fromInvoice(saveInvoice);
55          return invoiceResponseDTO;
56      }
```

#### 5. Test à ajouter des factures :

```
13  @SpringBootApplication
14  @EnableFeignClients //pour activer le service openFeign
15  public class BillingServiceApplication {
16
17      public static void main(String[] args) { SpringApplication.run(BillingServiceApplication.class, args); }
18
19
20
21      @Bean
22      CommandLineRunner strat(InvoiceService invoiceService)
23      {
24          return args -> {
25              invoiceService.addInvoice(new InvoiceRequestDTO(BigDecimal.valueOf(90000), customerId: "C01"));
26              invoiceService.addInvoice(new InvoiceRequestDTO(BigDecimal.valueOf(40000), customerId: "C01"));
27              invoiceService.addInvoice(new InvoiceRequestDTO(BigDecimal.valueOf(10000), customerId: "C02"));
28          };
29      }
30
31  }
```

#### 6. Résultat de test:

SELECT \* FROM INVOICE;

ID	AMOUNT	CUSTOMER_ID	DATE
d7f5bea5b-8be8-4294-92c4-3f868c303e28	90000.00	C01	2022-10-31 01:09:09.54
b8f5afe9-e59e-4a15-bb06-277e3ba60371	40000.00	C01	2022-10-31 01:09:09.737
ba560062-7ea2-43a4-8f83-420200d142a6	10000.00	C02	2022-10-31 01:09:09.747

(3 rows, 5 ms)

## 7. Retourner les factures d'un client :

The screenshot shows a REST client interface with the following sections:

- GET** `/api/invoicesByCustomer/{customerId}`
- Parameters**: A table with columns 'Name' and 'Description'. It contains one parameter: `customerId` (string, path) with the value `C02`.
- Execute** and **Clear** buttons.
- Responses**:
  - Curl**: `curl -X 'GET' \ 'http://localhost:8083/api/invoicesByCustomer/C02' \ -H 'accept: */*'`
  - Request URL**: `http://localhost:8083/api/invoicesByCustomer/C02`
  - Server response**:
    - Code**: 200
    - Details**:

```
{
  "id": "ba560062-7ea2-43a4-8f83-420200d142a6",
  "date": "2022-10-31T08:09:09.747+00:00",
  "amount": 10000,
  "customer": {
    "id": "C02",
    "name": "OpenLab",
    "email": "open@openlab.com"
  }
}
```

## PARTIE 3 : Création de Eureka Discovery Service

Quand l'application répond à une montée en charge et on a plusieurs instances de chaque microservice, il est vital de pouvoir garder un **registre de toutes les instances disponibles** afin de distribuer la charge entre celles-ci.

**Eureka** de Netflix remplit précisément cette fonction. Une fois en place, les instances des microservices viennent s'enregistrer dans le registre d'Eureka. Pour appeler un microservice, il suffira de piocher dans cette liste d'instances qu'Eureka expose via une API REST.

### 1. Configuration de Eureka Discovery:

#### ➤ Pom.xml:

```
22 <dependency>
23   <groupId>org.springframework.cloud</groupId>
24   <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
25 </dependency>
```

#### ➤ application.properties :

```
application.properties
1 server.port=8761
2 #don't register server itself as a client
3 eureka.client.fetch-registry=false
4 # Doesn't register itself in the service registry
5 eureka.client.register-with-eureka=false
```

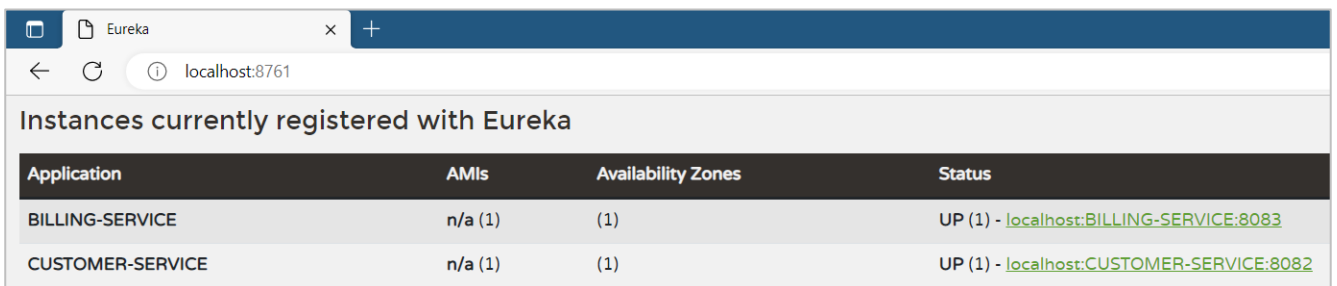


## 2. Mettez en place le service Eureka :

```
7 @SpringBootApplication
8 @EnableEurekaServer
9 public class EurekaDiscoveryServiceApplication {
10
11     public static void main(String[] args) {
12         SpringApplication.run(EurekaDiscoveryServiceApplication.class, args);
13     }
14 }
```

## 3. Lancer Eureka :

Cette page présente un certain nombre d'informations sur notre serveur Eureka. Les plus importantes sont celles-ci :



The screenshot shows a web browser window with the title 'Eureka' and the address bar 'localhost:8761'. The main content area is titled 'Instances currently registered with Eureka' and contains a table with the following data:

Application	AMIs	Availability Zones	Status
BILLING-SERVICE	n/a (1)	(1)	UP (1) - <a href="#">localhost:BILLING-SERVICE:8083</a>
CUSTOMER-SERVICE	n/a (1)	(1)	UP (1) - <a href="#">localhost:CUSTOMER-SERVICE:8082</a>

## PARTIE 4 : Création de Spring Cloud Gateway :

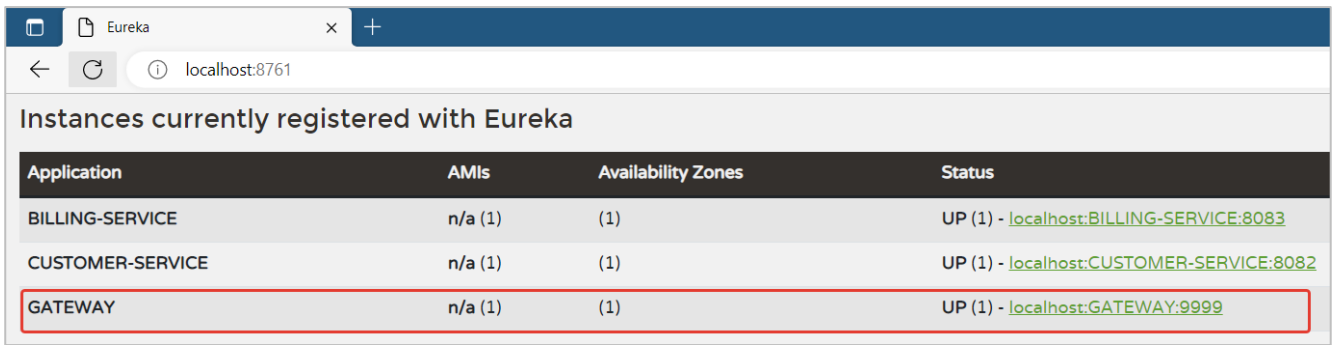
### 1. Inclure Spring Cloud Gateway dans l'application:

```
22 <dependency>
23     <groupId>org.springframework.cloud</groupId>
24     <artifactId>spring-cloud-starter-gateway</artifactId>
25 </dependency>
26
27 <dependency>
28     <groupId>org.springframework.cloud</groupId>
29     <artifactId>spring-cloud-starter-netflix-eureka-client</artifactId>
30 </dependency>
```

### 2. Propriétés de Gateway:

```
1 server.port=9999
2 spring.application.name=GATEWAY
3 spring.cloud.discovery.enabled=true
4 eureka.instance.ip-address=true
```

### 3. Gateway dans Eureka Discovery:



The screenshot shows the Eureka Discovery UI in a web browser. The title bar says 'Eureka'. The address bar shows 'localhost:8761'. The main heading is 'Instances currently registered with Eureka'. Below it is a table with four columns: 'Application', 'AMIs', 'Availability Zones', and 'Status'. The table lists three services: BILLING-SERVICE, CUSTOMER-SERVICE, and GATEWAY. The GATEWAY row is highlighted with a red border. The status for GATEWAY is 'UP (1) - localhost:GATEWAY:9999'.

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

### 4. Les routes de Gateway:



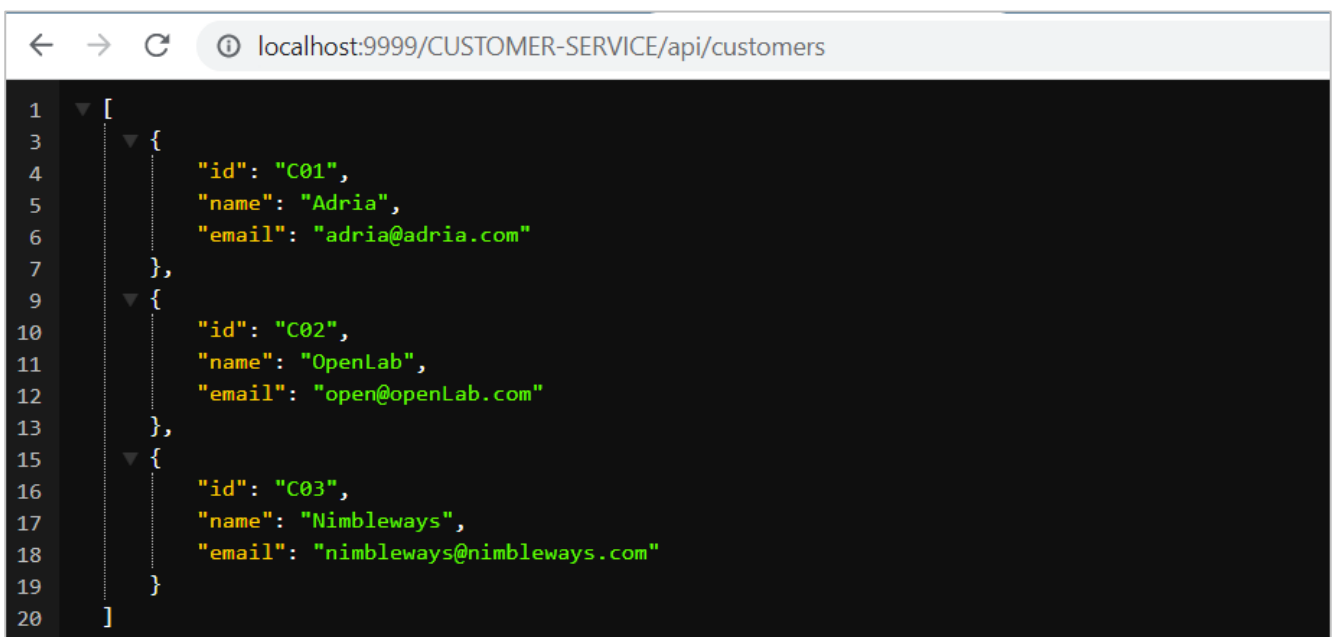
The screenshot shows a Java code editor with the following code:

```
10 @SpringBootApplication
11 public class GatewayApplication {
12
13     public static void main(String[] args) { SpringApplication.run(GatewayApplication.class, args); }
14
15
16
17     @Bean
18     DiscoveryClientRouteDefinitionLocator discoveryClientRouteDefinitionLocator(
19         ReactiveDiscoveryClient rdc, DiscoveryLocatorProperties dlp
20     )
21     {
22         return new DiscoveryClientRouteDefinitionLocator(rdc, dlp);
23     }
24
25 }
```

## PARTIE 5 : Test global de l'application:

### 1. Service de Customer

➤ **Retourner la liste des Customers:**



The screenshot shows a web browser with the address bar 'localhost:9999/CUSTOMER-SERVICE/api/customers'. The main content area displays a JSON array of three customer objects. The first object has id 'C01', name 'Adria', and email 'adria@adria.com'. The second object has id 'C02', name 'OpenLab', and email 'open@openLab.com'. The third object has id 'C03', name 'Nimbleways', and email 'nimbleways@nimbleways.com'.

```
1  [
2    {
3      "id": "C01",
4      "name": "Adria",
5      "email": "adria@adria.com"
6    },
7    {
8      "id": "C02",
9      "name": "OpenLab",
10     "email": "open@openLab.com"
11   },
12   {
13     "id": "C03",
14     "name": "Nimbleways",
15     "email": "nimbleways@nimbleways.com"
16   }
17 ]
```

➤ Retourner un Customer à partir son id:

```
localhost:9999/CUSTOMER-SERVICE/api/customers/C01

1 {
2   "id": "C01",
3   "name": "Adria",
4   "email": "adria@adria.com"
5 }
```

➤ Ajouter un Customer :

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

Params Authorization Headers (10) **Body** Pre-request Script Tests Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON**

```
1 {
2   "name": "IBM",
3   "email": "ibm@gmail.com"
4 }
```

Body Cookies Headers (3) Test Results

Pretty Raw Preview Visualize **JSON**

```
1 {
2   "id": "c28d6ae9-ec15-49cd-a121-6b43449b0f35",
3   "name": "IBM",
4   "email": "ibm@gmail.com"
5 }
```

## 2. Service de Billing:

➤ Retourner la liste des Customers:

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

Params Authorization Headers (8) Body Pre-request Script Tests Settings

Body Cookies Headers (3) Test Results

Pretty Raw Preview Visualize **JSON**

```
1 [
2   {
3     "id": "d7fba5b-8be8-4294-92c4-3f868c303e28",
4     "date": "2022-10-31T00:09:09.540+00:00",
5     "amount": 90000.00,
6     "customer": {
7       "id": "C01",
8       "name": "Adria",
9       "email": "adria@adria.com"
10    }
11  },
12  {
13    "id": "b8f5afe9-e59e-4a15-bb06-277e3ba60371",
14    "date": "2022-10-31T00:09:09.737+00:00",
15    "amount": 40000.00,
16    "customer": {
17      "id": "C01",
18      "name": "Adria",
19      "email": "adria@adria.com"
20    }
21  },
22 ]
```

### ➤ Retourner une facture à partir son id:

GET `http://localhost:9999/BILLING-SERVICE/api/invoices/d7fba5b-8be8-4294-92c4-3f868c303e28`

Params Authorization Headers (8) Body Pre-request Script Tests Settings

Body Cookies Headers (3) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": "d7fba5b-8be8-4294-92c4-3f868c303e28",
3   "date": "2022-10-31T00:09:09.540+00:00",
4   "amount": 90000.00,
5   "customer": {
6     "id": "C01",
7     "name": "Adria",
8     "email": "adria@adria.com"
9   }
10 }
```

### ➤ Ajouter une facture:

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

Params Authorization Headers (10) Body Pre-request Script Tests Settings

none form-data x-www-form-urlencoded raw binary GraphQL JSON

```
1 {
2   "amount": 300000,
3   "customerId": "C02"
4 }
```

Body Cookies Headers (3) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "id": "97d58071-793b-472e-8919-c5f1bee291c9",
3   "date": "2022-10-31T01:23:14.525+00:00",
4   "amount": 300000,
5   "customer": {
6     "id": "C02",
7     "name": "OpenLab",
8     "email": "open@openLab.com"
9   }
10 }
```

## PARTIE 6 : Dockerisation des services:

### ➤ Les conteneurs des services:

	NAME	IMAGE	STATUS	PORT...	STARTED	ACTIONS
	<b>docker-compose</b> 4 containers	-	Running (4/4)	-		
	<b>eureka-container</b> 81e16b233e37	<a href="#">eureka-discovery-service.jar:latest</a>	Running	8761	1 minute ago	
	<b>billing-container</b> 5c71bb927196	<a href="#">billing-service.jar:latest</a>	Running	8083	1 minute ago	
	<b>customer-container</b> dd03adc271d2	<a href="#">customer-service.jar:latest</a>	Running	8082	1 minute ago	
	<b>gateway-container</b> 419237319f06	<a href="#">gateway.jar:latest</a>	Running	8040	1 minute ago	

## ➤ Les images des services:

NAME ↑		TAG	IMAGE ID	CREATED	SIZE
billing-service.jar	IN USE	latest	ada688ec32d9	about 5 hours ago	537.87 MB
customer-service.jar	IN USE	latest	6dec4b3a484c	about 5 hours ago	537.36 MB
eureka-discovery-service.j...	IN USE	latest	ac76c43568a0	about 5 hours ago	517.9 MB
gateway.jar	IN USE	latest	f0e411737af5	about 5 hours ago	513.88 MB
hello-world		latest	feb5d9fea6a5	about 1 year ago	13.26 KB

## ➤ Le fichier docker-compose:

```
1  version: "3"
2  >> services:
3  > billing-service:
4      image: billing-service.jar
5      container_name: billing-container
6      ports:
7          - "8083:8080"
8      depends_on:
9          - customer-service
10         - eureka-discovery
11  > customer-service:
12      image: customer-service.jar
13      container_name: customer-container
14      ports:
15          - "8082:8080"
16      depends_on:
17          - eureka-discovery
18  > eureka-discovery:
19      image: eureka-discovery-service.jar
20      container_name: eureka-container
21      ports:
22          - "8761:8080"
23  > gateway:
24      image: gateway.jar
25      container_name: gateway-container|
26      ports:
27          - "8040:8080"
28      depends_on:
29          - customer-service
30          - billing-service
31          - eureka-discovery
```

## ➤ Démarrer les conteneurs des images :

```
Terminal: Local + -
PS C:\Users\del\Documents\S5\SD_micro-service\travaux-pratiques\tp1-architecture-microServices\docker\docker-compose> docker-compose up
Starting eureka-container ... done
Creating customer-container ... done
Creating billing-container ... done
Creating gateway-container ... done
Attaching to eureka-container, customer-container, billing-container, gateway-container
eureka-container |
eureka-container | .  ____
eureka-container | / \  ____'  ____'  ____'  ____'
eureka-container | ( ( \____'  ____'  ____'  ____'
eureka-container | \ \ ____'  ____'  ____'  ____'
eureka-container |  '  ____'  ____'  ____'  ____'
eureka-container | =====|=====|=====|=====|
eureka-container | :: Spring Boot :: (v2.7.5)
eureka-container |
eureka-container | 2022-10-31 01:40:17.612 INFO 1 --- [main] c.e.e.EurekaDiscoveryServiceApplication : Starting EurekaDiscoveryServiceApplication v0.0.1-SNAPSHOT using
eureka-container | Java 17.0.2 on 81e16b233e37 with PID 1 (/eureka-discovery-service.jar started by root in /)
eureka-container | 2022-10-31 01:40:17.626 INFO 1 --- [main] c.e.e.EurekaDiscoveryServiceApplication : No active profile set, falling back to 1 default profile: "default
t"
customer-container |
customer-container | .  ____
customer-container | / \  ____'  ____'  ____'  ____'
customer-container | ( ( \____'  ____'  ____'  ____'
customer-container | \ \ ____'  ____'  ____'  ____'
customer-container |  '  ____'  ____'  ____'  ____'
customer-container | =====|=====|=====|=====|
customer-container | :: Spring Boot :: (v2.7.5)
customer-container |
```

## Résumé

