

# **DEVOIR N°1 Module JEE : Développement Microservices avec Spring Cloud**

*Modèle explicatif*

**Réalisé par :**

**ANASS EL HOUSNI**

**&**

**REDA AMIMI**

## Etude de cas (1) :

### Développement du « microservice-commandes »

1. Table COMMANDE avec les champs :

ID	DATE	DESCRIPTION	MONTANT	QUANTITE

2. Réalisation des opérations CRUD sur une « COMMANDE » avec 0 ligne SQL en utilisation l'outil POSTMAN en envoyant des requêtes HTTP et en visualisant les réponses :

#### ✓ Create (POST METHOD)

Postman interface showing a POST request to `http://localhost:9001/Commandes`. The request body is a JSON object:

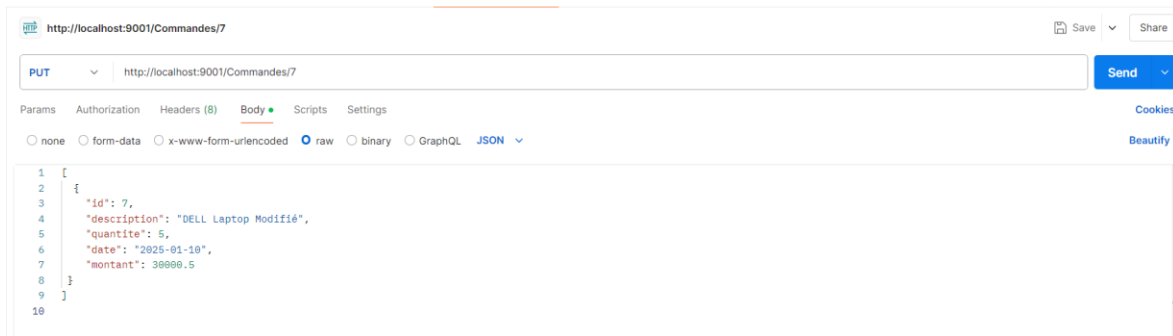
```
{  "id": 7,  "description": "DELL Laptop",  "date": "2025-01-10",  "montant": 30000.5,  "quantite": 5}
```

The response is a 200 OK status with a 24 ms response time and 253 B of data.

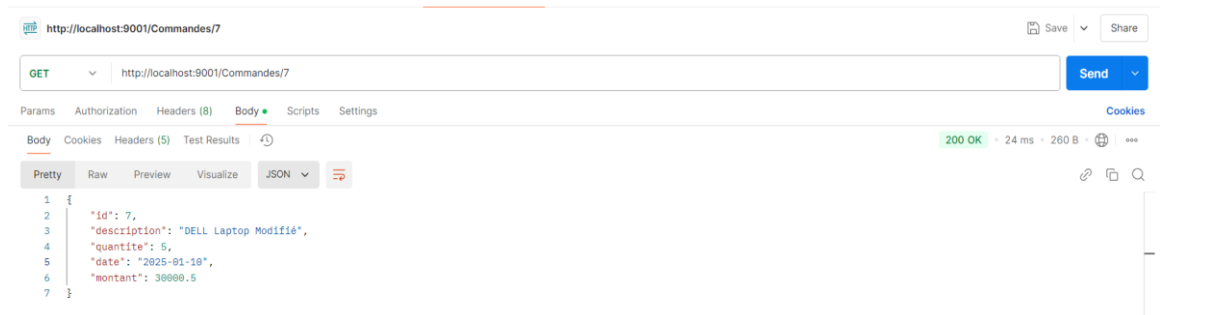
#### ✓ Read (GET METHOD)

Postman interface showing a GET request to `http://localhost:9001/Commandes/7`. The response is a 200 OK status with a 37 ms response time and 251 B of data.

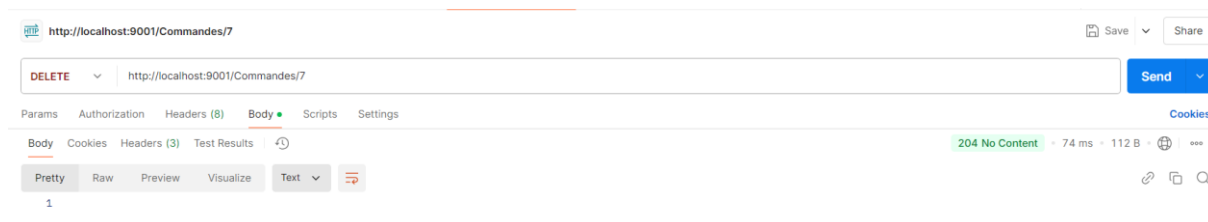
#### ✓ Update (PUT METHOD)



→ Vérification du MAJ par GET METHOD /{ID=7}

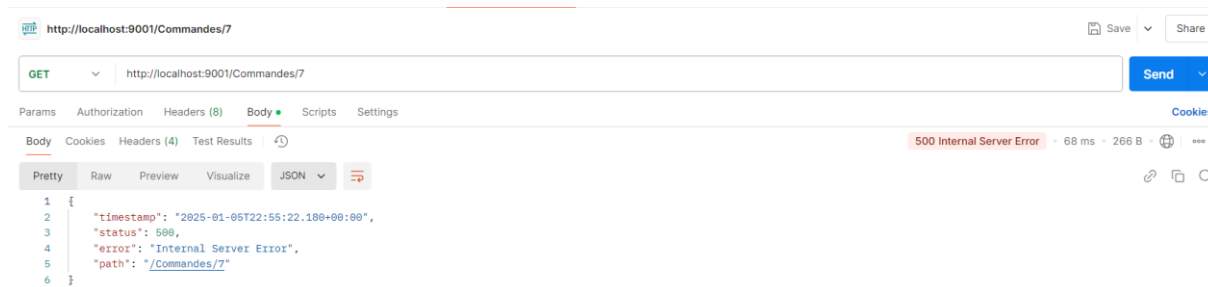


✓ Delete



→ Renvoie un statut 204 No Content ça veut dire que la suppression est réussie\*

Vérification :



3. La gestion de la configuration est au niveau Spring Cloud et github (configuration distante)

→ Les dépendances nécessaires

```

1 <spring-cloud.version>2021.0.8</spring-cloud.version>
2 <dependency>
3     <groupId>org.springframework.cloud</groupId>
4     <artifactId>spring-cloud-starter-config</artifactId>
5 </dependency>
6 <dependency>
7     <groupId>org.springframework.cloud</groupId>
8     <artifactId>spring-cloud-starter-bootstrap</artifactId>
9 </dependency>

```

→ la configuration git hub

The screenshot shows the GitHub interface for a repository named 'config-server-commandes'. The repository is private and has 1 branch (master) and 0 tags. A commit by AnassElhousni is highlighted, titled 'Update and rename zuul-server.properties to api-gateway.properties'. Below the commit, a table lists the files changed in this commit:

File	Commit Message	Time Ago
api-gateway.properties	Update and rename zuul-server.properties to api-gateway.pr...	4 minutes ago
eureka-server.properties	first commit	2 weeks ago
microservice-commandes.properties	Update microservice-commandes.properties	5 hours ago
microservice-produits.properties	Update microservice-produits.properties	2 hours ago

- ❖ Personnalisation d'une propriété **mes-config-ms.commandes-last** pour afficher juste les dernières commandes reçues . Dans notre cas : « mes-config-ms.commandes-last = 5 » permet d'afficher les commandes reçues les 5 derniers jours.

→ Personnalisation sur github :

```

16 #Les configurations personnalisés
17 mes-config-ms.commandes-last = 5

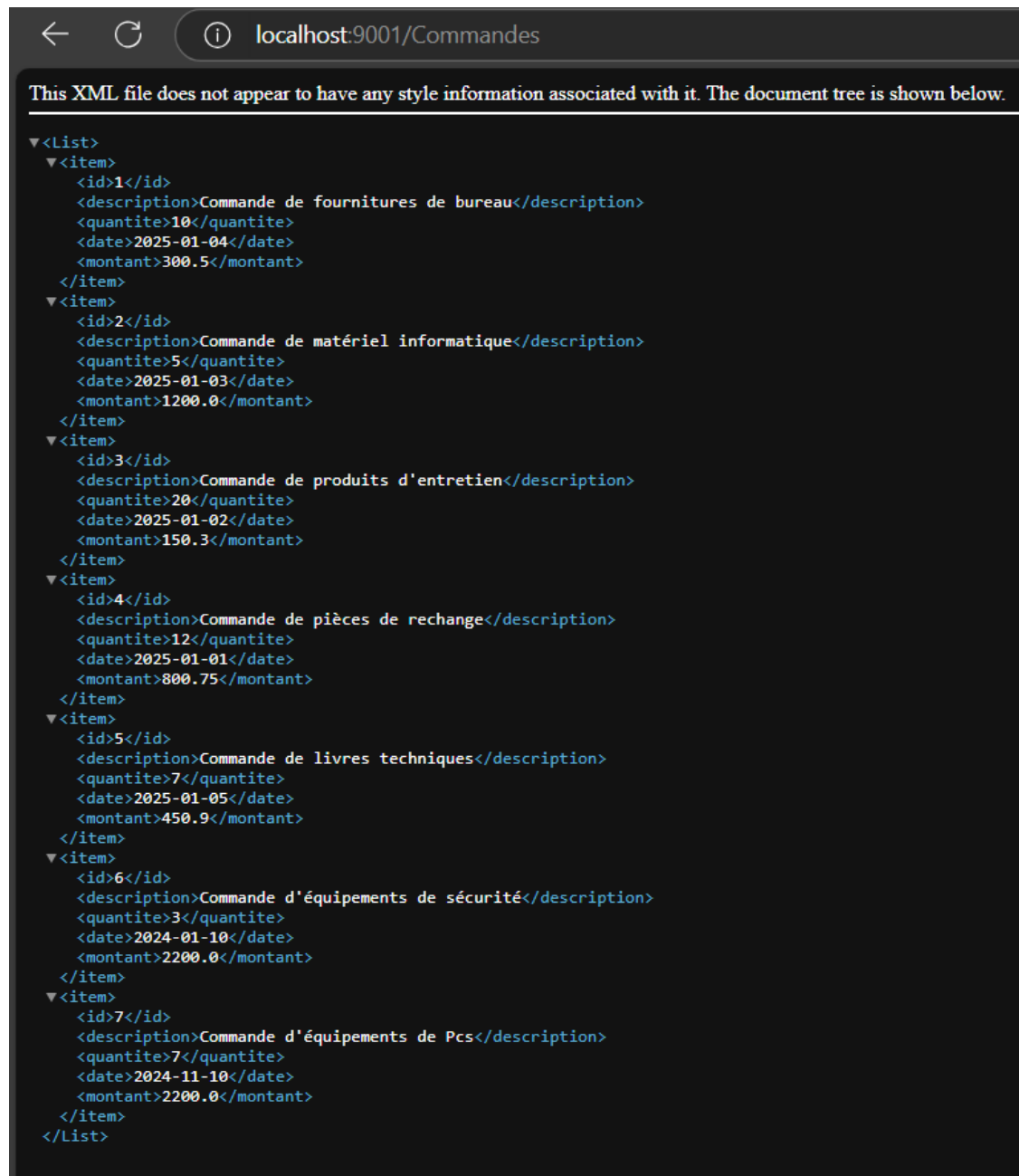
```

→ Configuration de la propriété dans le MS :

```
1  @Component
2  @ConfigurationProperties("mes-config-ms")
3  @RefreshScope
4  public class ApplicationPropertiesConfiguration {
5      // Correspond à la propriété « mes-config-ms.commandes-last » dans le fichier de configuration du MS
6      private int commandesLast;
7
8      public int getCommandesLast() {
9          return commandesLast;
10     }
11
12     public void setCommandesLast(int commandesLast) {
13         this.commandesLast = commandesLast;
14     }
15 }
16
```

➔ Avant l'application de la propriété qui limite l'affichage des commandes, Insertion de 7 commandes :

```
16      #Les configurations personnalisés
17      #mes-config-ms.commandes-last = 5
```



← ↻ ⓘ localhost:9001/Commandes

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<List>
  <item>
    <id>1</id>
    <description>Commande de fournitures de bureau</description>
    <quantite>10</quantite>
    <date>2025-01-04</date>
    <montant>300.5</montant>
  </item>
  <item>
    <id>2</id>
    <description>Commande de matériel informatique</description>
    <quantite>5</quantite>
    <date>2025-01-03</date>
    <montant>1200.0</montant>
  </item>
  <item>
    <id>3</id>
    <description>Commande de produits d'entretien</description>
    <quantite>20</quantite>
    <date>2025-01-02</date>
    <montant>150.3</montant>
  </item>
  <item>
    <id>4</id>
    <description>Commande de pièces de rechange</description>
    <quantite>12</quantite>
    <date>2025-01-01</date>
    <montant>800.75</montant>
  </item>
  <item>
    <id>5</id>
    <description>Commande de livres techniques</description>
    <quantite>7</quantite>
    <date>2025-01-05</date>
    <montant>450.9</montant>
  </item>
  <item>
    <id>6</id>
    <description>Commande d'équipements de sécurité</description>
    <quantite>3</quantite>
    <date>2024-01-10</date>
    <montant>2200.0</montant>
  </item>
  <item>
    <id>7</id>
    <description>Commande d'équipements de Pcs</description>
    <quantite>7</quantite>
    <date>2024-11-10</date>
    <montant>2200.0</montant>
  </item>
</List>
```

➔ Après l'application du propriété `mes-config-ms.commandes-last = 5` qui affiche les commandes reçues ces 5 dernières jours

```
16 #Les configurations personnalisés
17 mes-config-ms.commandes-last = 5
```

➔ Réalisation d'un chargement à chaud en basant sur le service Actuator de spring

➤ Dépendance nécessaire

```
1 <dependency>
2     <groupId>org.springframework.boot</groupId>
3     <artifactId>spring-boot-starter-actuator</artifactId>
4 </dependency>
```

➤ Configuration de l'endpoint **Refresh** sur ms.properties sur Github

```
13 #Actuator : management.endpoints.web.exposure.include=*
14 management.endpoints.web.expose= info, health, refresh
```

➤ Avec l'ajout l'annotation **@RefreshScope** dans la configuration de propriété

```
1 @ConfigurationProperties("mes-config-ms")
2 @RefreshScope
```

➤ Envoi de la requête par la méthode POST

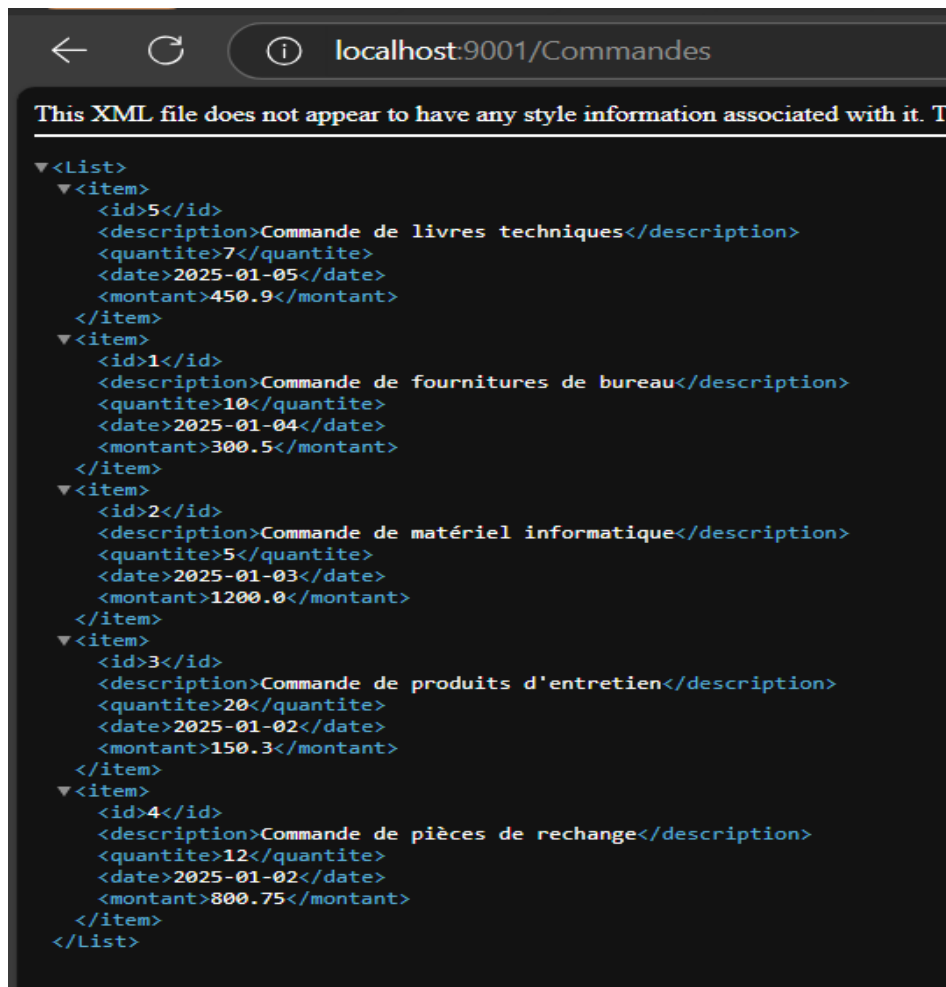
 localhost:9001/actuator/refresh

POST



localhost:9001/actuator/refresh

➤ Résultat :



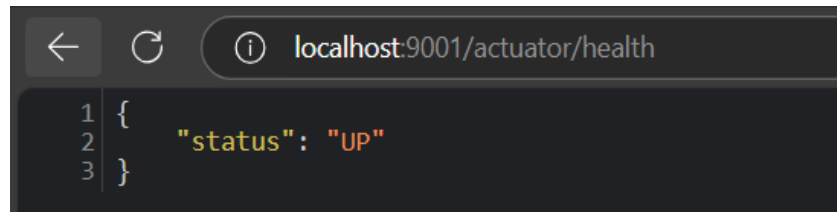
➔ Voici le code pour implémenter cette propriété :

```
1 @GetMapping(value = "/Commandes")
2 public List<Commande> listeDesCommandes() throws CommandeNotFoundException {
3     System.out.println("***** CommandeController listeDesCommandes() ");
4
5     List<Commande> commandes = commandeDao.findAll();
6     System.out.println("Nombre de commandes récupérées : " + commandes.size());
7
8     if (commandes.isEmpty()) {
9         throw new CommandeNotFoundException("Aucune commande n'a été trouvée.");
10    }
11
12    // Récupérer la limite des commandes par la propriété mes config last
13    int limit = appProperties.getCommandesLast();
14    System.out.println("Limite définie dans la configuration : " + limit);
15
16    // la date d'aujourd'hui
17    LocalDate today = LocalDate.now();
18
19    // Filtrages des commandes for the last 'limit' days
20    List<Commande> commandesFiltrees = commandes.stream()
21        .filter(commande -> commande.getDate().isAfter(today.minusDays(limit)))
22        .sorted((c1, c2) -> {
23            int dateComparison = c2.getDate().compareTo(c1.getDate());
24            if (dateComparison == 0) {
25                return c1.getId().compareTo(c2.getId());
26            }
27            return dateComparison;
28        })
29        .collect(Collectors.toList());
30
31    System.out.println("Nombre de commandes filtrées et triées : " + commandesFiltrees.size());
32
33    if (commandesFiltrees.isEmpty()) {
34        throw new CommandeNotFoundException("Aucune commande n'a été trouvée dans les derniers jours.");
35    }
36    return commandesFiltrees;
```



- Implémentation de la supervision la bonne santé du « microservice-commandes » : le statut à afficher « UP » lorsqu'il y'a des commandes dans la table «COMMANDE», sinon le statut à afficher est «DOWN », En se basant sur le service Actuator de spring.

🚦 Cas 1 : il y a des commandes dans la table :



```
1 {
2   "status": "UP"
3 }
```

🚦 Cas 2 : la table est vide :



```
1 {
2   "status": "DOWN"
3 }
```

- Voici le code pour implémenter cette endpoint using healthindicator :

```
1 import org.springframework.boot.actuate.health.Health;
2 import org.springframework.boot.actuate.health.HealthIndicator;
3
4 @RestController
5 public class CommandeController implements HealthIndicator {
6     @Autowired
7     CommandeDao commandeDao;
8     @Override
9     public Health health() {
10         System.out.println("***** Actuator : CommandeController health() ");
11         List<Commande> commandes = commandeDao.findAll();
12
13         if (commandes.isEmpty()) {
14             return Health.down().build();
15         }
16         return Health.up().build();
17     }
18 }
19
```

## Etude de cas (2):

### 1. Développement du ms produit

La version (2) de la table « COMMANDE » est composée » des colonnes suivantes [id, description, quantité, date, montant, id\_produit]

- Table COMMANDE avec les champs

ID	DATE	DESCRIPTION	ID_PRODUIT	MONTANT	QUANTITE
----	------	-------------	------------	---------	----------

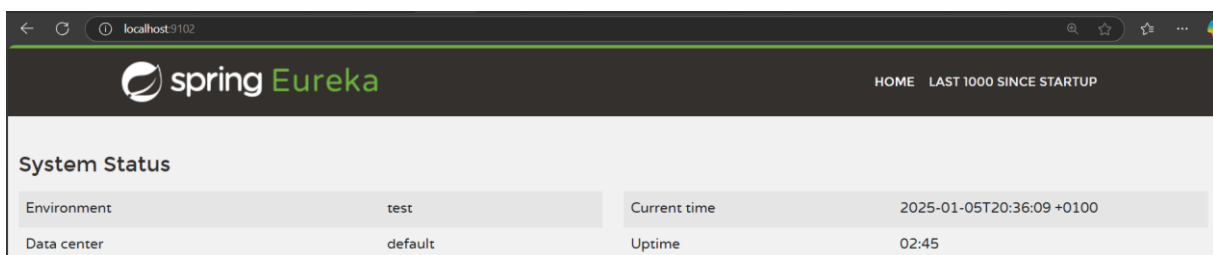
- Réalisation des opérations CRUD

(Même que de la première version de la table COMMANDE)

2. L'enregistrement des MS : microservice-commandes et microservice-produit auprès Eureka server :

- La configuration de Eureka server, il s'écoute le port 9102

```
1 server.port= 9102
2 spring.application.name=eureka-server
3 # Single eureka mode, and note Cluster mode
4 eureka.client.registerWithEureka=false
5 eureka.client.fetchRegistry=false
6 spring.cloud.config.import-check.enabled=false
7 #To avoid eureka server Connect to localhost:8761 timed out
8 eureka.server.maxThreadsForPeerReplication=0
```



- Indication sur la configuration des deux microservices l'url d'Eureka à laquelle il faut s'enregistrer

```
11 #Eureka :indique l'URL d'Eureka à laquelle il faut s'enregistrer
12 eureka.client.serviceUrl.defaultZone=http://localhost:9102/eureka/
```

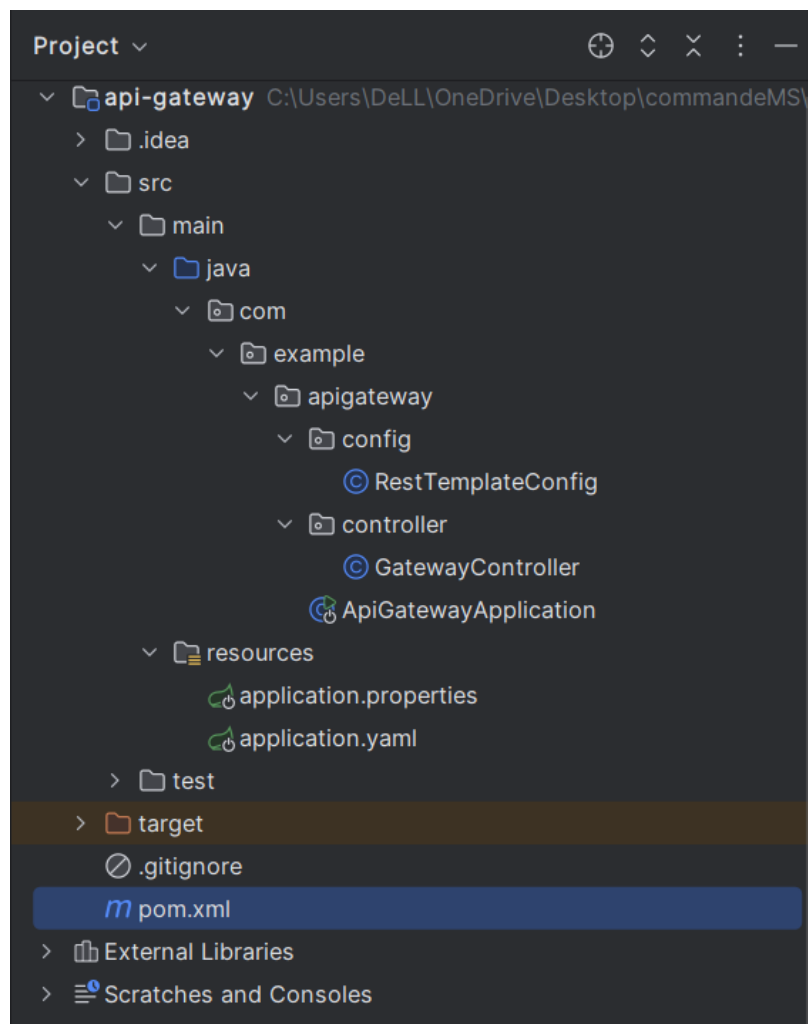
- Démarrage des ms : microservice-commandes et microservice-produit
  - \* microservice-commandes s'écoute sur le port : 9001
  - \* microservice-produit s'écoute sur le port : 9002

Localhost://9102

Instances currently registered with Eureka			
Application	AMIs	Availability Zones	Status
MICROSERVICE-COMMANDES	n/a (1)	(1)	UP (1) - <a href="#">DESKTOP-ENRLEIG:microservice-commandes:9001</a>
MICROSERVICE-PRODUITS	n/a (1)	(1)	UP (1) - <a href="#">DESKTOP-ENRLEIG:microservice-produits:9002</a>

### 3. Implémentation d'un API Gateway comme point d'accès unique à l'application :

- Utilisation de l'API Gateway avec Spring Boot MVC
- La hiérarchie de API Gateway



- La classe Gateway controller

```

1  @RestController
2  public class GatewayController {
3
4      private final RestTemplate restTemplate = new RestTemplate();
5      @GetMapping("/api/commandes")
6      public ResponseEntity<?> routeCommandes() {
7          String url = "http://localhost:9001/commandes";
8          return restTemplate.getForEntity(url, String.class);
9      }
10
11     @GetMapping("/api/commandes/{id}")
12     public ResponseEntity<?> routeCommandeById(@PathVariable Long id) {
13         String url = "http://localhost:9001/commandes/" + id;
14         return restTemplate.getForEntity(url, String.class);
15     }
16
17     @PostMapping("/api/commandes")
18     public ResponseEntity<?> addCommande(@RequestBody Object commande) {
19         String url = "http://localhost:9001/commandes";
20         return restTemplate.postForEntity(url, commande, String.class);
21     }
22 }

```

- Application.yaml ou se fait la définition des paths de l'entrée unique à l'application

```

1  spring:
2    cloud:
3      gateway:
4        routes:
5          - id: commandes-service
6            uri: http://localhost:9001
7            predicates:
8              - Path=/api/Commandes/**
9            filters:
10              - RewritePath=/api/Commandes/(?<segment>.*), /Commandes/${segment}
11

```

IMPORTANT : l'API Gateway doit être enregistré auprès le serveur Eureka :

## Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
API-GATEWAY	n/a (1)	(1)	UP (1) - <a href="#">DESKTOP-ENRLEIG:api-gateway:9004</a>
MICROSERVICE-COMMANDES	n/a (1)	(1)	UP (1) - <a href="#">DESKTOP-ENRLEIG:microservice-commandes:9001</a>
MICROSERVICE-PRODUITS	n/a (1)	(1)	UP (1) - <a href="#">DESKTOP-ENRLEIG:microservice-produits:9002</a>

- ❖ Accès direct au microservice « Commande » sans passer par la Gateway :

[localhost:9001/Commandes/](http://localhost:9001/Commandes/)

The screenshot shows a web browser window with the address bar displaying 'localhost:9001/Commandes'. The page content shows an XML response. At the top, a message states: 'This XML file does not appear to have any style information associated with it. The document root is <List>'. Below this, the XML structure is displayed with collapsible nodes. The root element is <List>, which contains five <item> elements. Each item contains sub-elements for <id>, <description>, <quantite>, <date>, and <montant>.

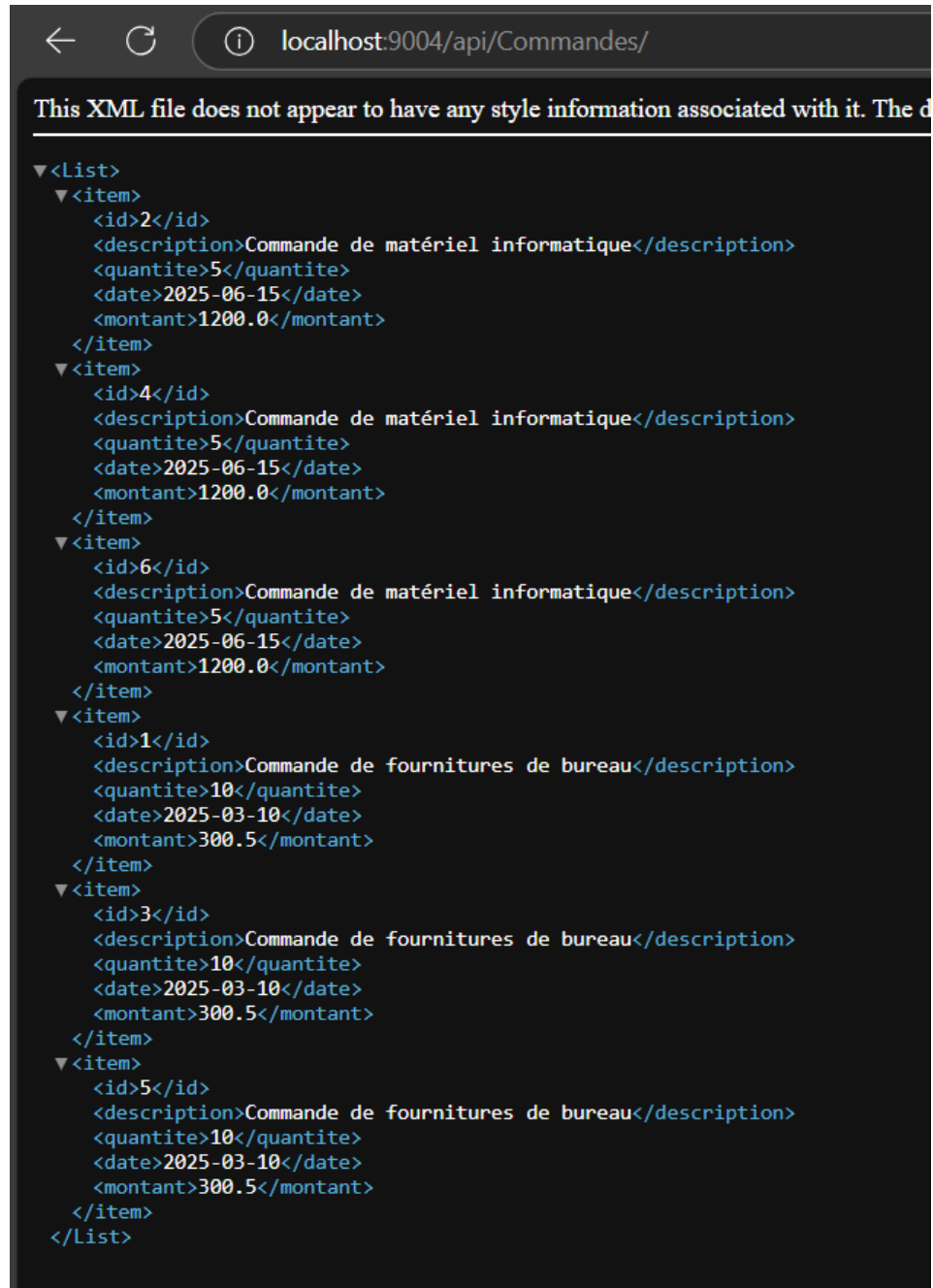
```

<?xml version="1.0" encoding="UTF-8" standalone="yes">
<List>
  <item>
    <id>2</id>
    <description>Commande de matériel informatique</description>
    <quantite>5</quantite>
    <date>2025-06-15</date>
    <montant>1200.0</montant>
  </item>
  <item>
    <id>4</id>
    <description>Commande de matériel informatique</description>
    <quantite>5</quantite>
    <date>2025-06-15</date>
    <montant>1200.0</montant>
  </item>
  <item>
    <id>6</id>
    <description>Commande de matériel informatique</description>
    <quantite>5</quantite>
    <date>2025-06-15</date>
    <montant>1200.0</montant>
  </item>
  <item>
    <id>1</id>
    <description>Commande de fournitures de bureau</description>
    <quantite>10</quantite>
    <date>2025-03-10</date>
    <montant>300.5</montant>
  </item>
  <item>
    <id>3</id>
    <description>Commande de fournitures de bureau</description>
    <quantite>10</quantite>
    <date>2025-03-10</date>
    <montant>300.5</montant>
  </item>
  <item>
    <id>5</id>
    <description>Commande de fournitures de bureau</description>
    <quantite>10</quantite>
    <date>2025-03-10</date>
    <montant>300.5</montant>
  </item>
</List>

```

- ❖ Accès direct au microservice « Commande » en passant par la Gateway :

[localhost:9004/api/Commandes/](http://localhost:9004/api/Commandes/)



4. Simulation d'un Timeout d'un des deux microservices, et implémentation un mécanisme de de contournement pour protéger le microservice appelant avec Resilience4j :
  - Configuration de resilience4j circuit breaker

```

18 # Resilience4j Circuit Breaker Configuration
19 resilience4j.circuitbreaker.instances.commandesCircuitBreaker.registerHealthIndicator=true
20 resilience4j.circuitbreaker.instances.commandesCircuitBreaker.failureRateThreshold=50
21 resilience4j.circuitbreaker.instances.commandesCircuitBreaker.waitDurationInOpenState=10000
22 resilience4j.circuitbreaker.instances.commandesCircuitBreaker.slidingWindowSize=100
23 resilience4j.circuitbreaker.instances.commandesCircuitBreaker.permittedNumberOfCallsInHalfOpenState=10

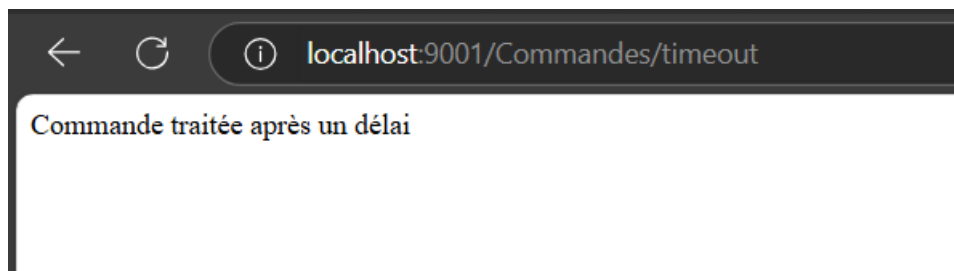
```

- Simulation d'un timeout de 5s :

```

1 @CircuitBreaker(name = "commandesCircuitBreaker", fallbackMethod = "handleTimeout")
2 public List<Commande> listeDesCommandes() throws CommandeNotFoundException {
3     System.out.println("***** CommandeController listeDesCommandes() ");
4
5     // Simulating a delay to trigger the timeout
6     try {
7         Thread.sleep(5000); // 5 seconds delay
8     } catch (InterruptedException e) {
9         e.printStackTrace();
10    }

```



- Fallback methode to handle timeout

```

1 // Fallback method to handle timeout
2 public List<Commande> handleTimeout(Exception ex) {
3     System.out.println("***** Fallback activated: The service is temporarily unavailable.");
4     return List.of(); // Return empty list as fallback
5 }

```

- Visualisation du console lors l'accès au commandes et la liste est vide



