

---

# **Rapport sur le projet JEE (Etude 1 & 2)**

**5<sup>ème</sup> année**

**Ingénierie Informatique et Réseaux (MIAGE)**

---

**Sous le thème**

**Développement des Micro-services avec  
Spring Cloud**

**Réalisé par :**

**EL GHAZY Hamza**

**Oujeddi Mostapha**

**Encadré par :**

**Mr. ABDELILAH HSSAINI**

**Année universitaire 2023-2024**

➡ Énoncé :

Etude de cas (1) :

Ajouter un « microservice-commandes » qui permet de réaliser les opérations CRUD sur une « COMMANDE » avec 0 ligne SQL :

- a. La version (1) de la table « COMMANDE » est composée » des colonnes suivantes [id, description, quantité, date, montant]
- b. La configuration du « microservice-commandes » doit être gérée au niveau Spring Cloud et github
- c. La configuration du « microservice-commandes » contient une propriété personnalisée « mes-config-ms.commandes-last » qui permet d’afficher les dernières commandes reçues. Dans notre cas : « mes-config-ms.commandes-last = 10 » permet d’afficher les commandes reçues les 10 derniers jours.

En se basant sur le service Actuator de spring, modifier cette propriété à 20 et réaliser un chargement à chaud pour que le « microservice-commandes » affiche les commandes reçues les 20 derniers jours

- d. En se basant sur le service Actuator de spring, Implémenter la supervision la bonne santé du « microservice-commandes » : le statut à afficher « UP »
- e. Personnaliser la supervision de la bonne santé du « microservice-commandes » : dans notre cas, un « microservice-commandes » est en bonne santé lorsqu’il y’a des commandes dans la table « COMMANDE », dans ce cas, le statut est « UP » sinon le statut à afficher est « DOWN »

## I- Les Etapes de realisation de l'étude de cas 1 :

### A-

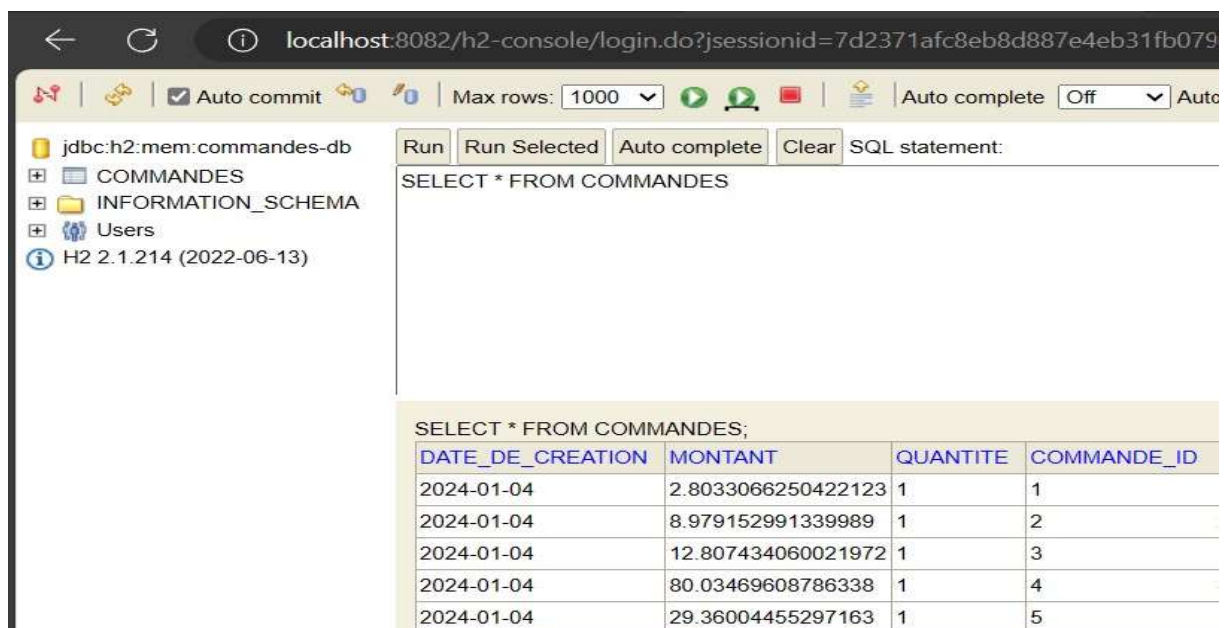
On a commencé par la création du micro-service commande qui permet la gestion des commandes en réalisant des opérations d'ajout de modification, suppression et l'affichage.

La classe principale de ce micro-service est « Commandes » :

```
1 package com.example.servicecommandes.model;
2 import jakarta.persistence.*;
3 import lombok.*;
4
5 import java.time.LocalDate;
6
7 @Entity
8 @Getter @Setter @ToString @NoArgsConstructor @AllArgsConstructor @Builder
9 public class Commandes {
10     @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
11     private Long commandeId;
12     private String description;
13     private Integer quantite;
14     private LocalDate dateDeCreation;
15     private Double montant;
16 }
17
```

Cette classe contient cinq attributs dont l'un est géré automatiquement par Spring Data JPA en utilisant l'annotation `@GeneratedValue` avec la stratégie de type identité.

Cette classe sera représentée dans la base de données sous forme d'une table via l'annotation `@Entity` et en définissant la clé primaire de cette table via `@Id`.



The screenshot shows the H2 console interface. The left sidebar displays the database structure: jdbc:h2:mem:commandes-db, with a table named COMMANDES. The main area shows the SQL statement `SELECT * FROM COMMANDES` and its results. The results are displayed in a table with four columns: DATE\_DE\_CREATION, MONTANT, QUANTITE, and COMMANDE\_ID. There are five rows of data.

DATE_DE_CREATION	MONTANT	QUANTITE	COMMANDE_ID
2024-01-04	2.8033066250422123	1	1
2024-01-04	8.979152991339989	1	2
2024-01-04	12.807434060021972	1	3
2024-01-04	80.03469608786338	1	4
2024-01-04	29.36004455297163	1	5

Cette figure représente la base de données de notre classe principale « Commandes ».

On se basant sur cette classe en va réaliser des opérations de CRUD. Commençant par l'affichage :

```
31
32
33
34
35 @GetMapping("/commandes")
36 public List<Commandes> commandesList(){
37
38     List<Commandes> commandes = commandeRepository.findAll();
39     commandes.forEach(c->{
40         c.setProduits(produitsRestClient.findProduitsById(c.getProduitId()));
41     });
42
43     List<Commandes> listeLimite = commandes.subList(0,applicationPropertiesConfiguration.getLimitDeCommands());
44     System.out.println(listeLimite);
45     return listeLimite;
46 }
47
```

Après l'exécution au niveau du Postman pour le test des « api rest ».

The screenshot shows the Postman interface for a REST client. The top bar displays the URL `http://localhost:8082/commadeapp/commandes` and the method `GET`. The 'Params' tab is active, showing a table with two columns: 'Key' and 'Value'. The 'Body' tab is also visible, showing a JSON response in 'Pretty' format. The response is a list containing one object with the following fields: `commandeId`, `description`, `quantite`, `dateDeCreation`, and `montant`.

Key	Value
Key	

```
[
  {
    "commandeId": 1,
    "description": "La commande Numero: 1",
    "quantite": 1,
    "dateDeCreation": "2024-01-04",
    "montant": 2.8033066250422123,
  }
]
```

Passant à l'opération d'affichage par commande spécifique.

```
49
50 @GetMapping("/commandes/{id}")
51 public Commandes commandesById(@PathVariable Long id){
52     Commandes commande = commandeRepository.findById(id).get();
53     Produits produit = produitsRestClient.findProduitsById(commande.getProduitId());
54     System.out.println(produit.toString());
55     commande.setProduits(produit);
56     return commande;
57 }
58
```

Résultats :

The screenshot shows a REST client interface with a GET request to `http://localhost:8082/commadeapp/commandes/1`. The response is a JSON object with the following fields:

```
1 {
2   "commandeId": 1,
3   "description": "La commande Numero: 1",
4   "quantite": 1,
5   "dateDeCreation": "2024-01-04",
6   "montant": 2.8033066250422123,
```

L'ajout d'une commande :

```
61 @PostMapping("/commandes")
62 public ResponseEntity<Commandes> addCommande(@RequestBody Commandes commande) {
63     Produits produit = produitsRestClient.findProduitsById(commande.getProduitId());
64
65     if (produit != null) {
66         commande.setProduits(produit);
67         Commandes savedCommande = commandeRepository.save(commande);
68         return ResponseEntity.status(HttpStatus.CREATED).body(savedCommande);
69     } else {
70         return ResponseEntity.status(HttpStatus.BAD_REQUEST).build();
71     }
72 }
73
```

Résultats :

GET http://localhost:8081/acti... POST http://localhost:8082/cc...

HTTP http://localhost:8082/commadeapp/commandes

POST http://localhost:8082/commadeapp/commandes

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

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

```
1 {
2   "description": "La commande Numero: 6",
3   "quantite": 15,
4   "dateDeCreation": "2024-01-05",
5   "montant": 9.8033066250422123
6 }
```

Body Cookies Headers (5) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "commandeId": 6,
3   "description": "La commande Numero: 6",
4   "quantite": 15,
5   "dateDeCreation": "2024-01-05",
6   "montant": 9.803306625042213,
7 }
```

L'opération de modification :

```
77 @PutMapping("/commandes/{id}")
78 public ResponseEntity<Commandes> updateCommande(@PathVariable Long id, @RequestBody Commandes updatedCommande) {
79     Commandes existingCommande = commandeRepository.findById(id).orElse(null);
80     if (existingCommande != null) {
81         Produits produit = produitsRestClient.findProduitsById(updatedCommande.getProduitId());
82         if (produit != null) {
83             existingCommande.setDateDeCreation(updatedCommande.getDateDeCreation());
84             existingCommande.setDescription(updatedCommande.getDescription());
85             existingCommande.setMontant(updatedCommande.getMontant());
86             existingCommande.setQuantite(updatedCommande.getQuantite());
87             existingCommande.setProduits(produit);
88             commandeRepository.save(existingCommande);
89             return ResponseEntity.ok(existingCommande);
90         } else {
91             return ResponseEntity.status(HttpStatus.BAD_REQUEST).build();
92         }
93     } else {
94         return ResponseEntity.notFound().build();
95     }
96 }
```



Résultats :

The screenshot shows a REST client interface with a PUT request to `http://localhost:8082/commandeapp/commandes/1`. The request body is a JSON object with the following fields:

```
{
  "description": "La commande Numero: 7",
  "quantite": 5,
  "dateDeCreation": "2024-01-07",
  "montant": 7.803306625042213
}
```

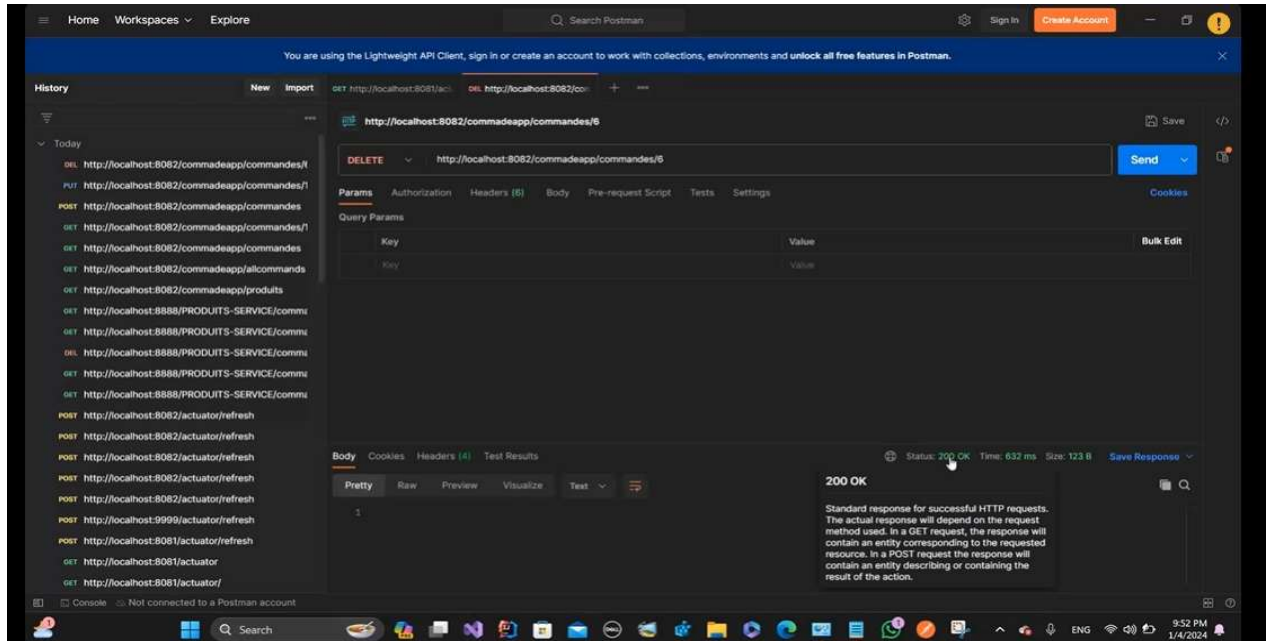
The response is displayed in the bottom panel, showing the same JSON object with the `commandeId` field set to 1:

```
{
  "commandeId": 1,
  "description": "La commande Numero: 7",
  "quantite": 5,
  "dateDeCreation": "2024-01-07",
  "montant": 7.803306625042213
}
```

L'opération de suppression :

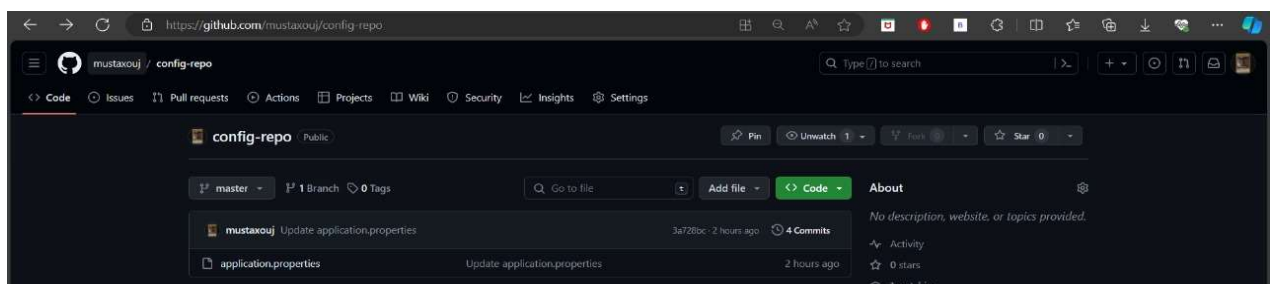
```
99
100     @DeleteMapping("/{commandes/{id}}")
101     public ResponseEntity<?> deleteCommande(@PathVariable Long id) {
102         if (commandeRepository.existsById(id)) {
103             commandeRepository.deleteById(id);
104             return ResponseEntity.ok().build();
105         } else {
106             return ResponseEntity.notFound().build();
107         }
108     }
109 }
```

## Résultats :

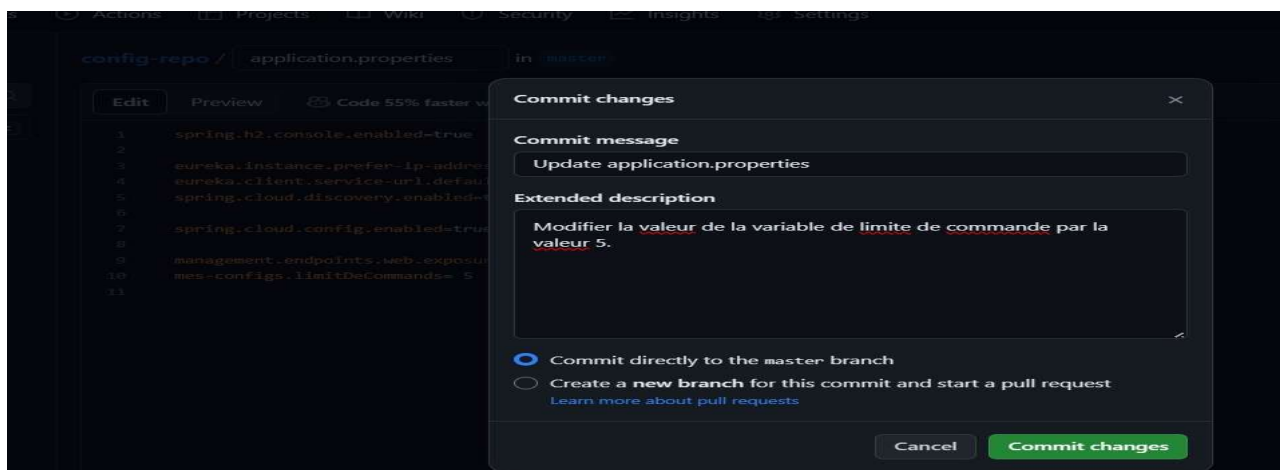


## B-

Après la réalisation de cette partie on est passé a la création du dépôt local qu'on va le publier sur GitHub afin de garantir la centralisation de la configuration en utilisant le spring cloud.

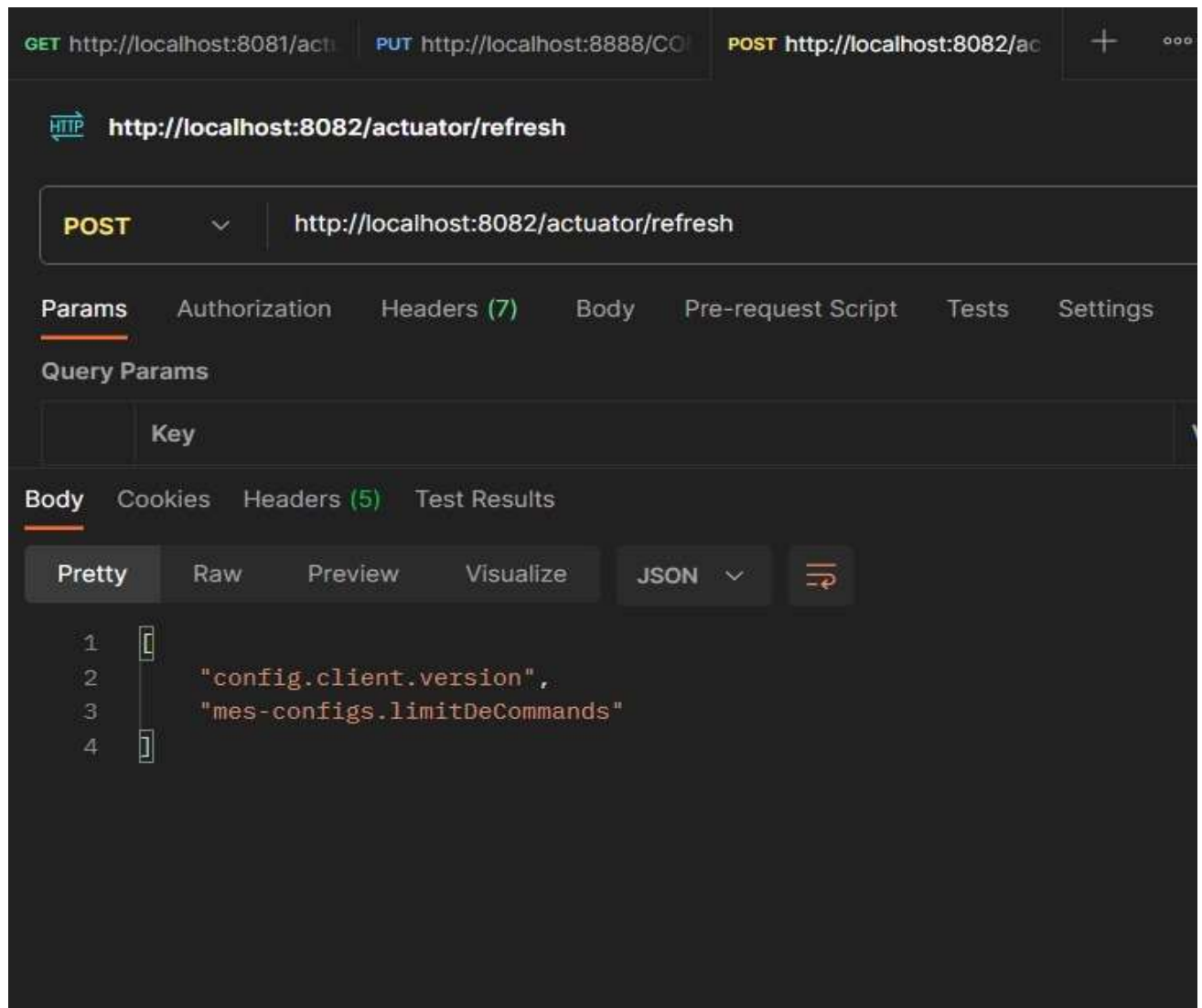


## C- Le changement de la configuration à chaud :



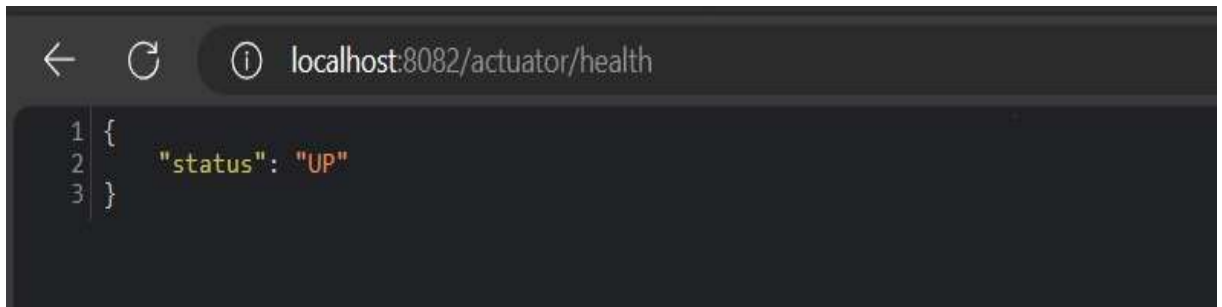


La réalisation du refresh via la requête post en utilisant postman l'outil de test des apis.  
Ce refresh permet d'avoir une synchronisation entre le répertoire local et celui a distant  
« GitHub »



#### D-

Puis on a lancé le micro-service commande est on a visualiser sont statuts via le spring actuator en utilisant le end point `/actuator/health` qui est activé via :  
`management.endpoints.web.exposure.include=*`



```
localhost:8082/actuator/health
1 {
2   "status": "UP"
3 }
```

E-

Puis la personnalisation du Heath via la méthode Heath implémenter auprès de l'interface HealthIndicator :



```
106
107
108 @Override
109 public Health health() {
110     System.out.println("***** Actuator : ProductController health() ");
111     List<Commandes> products = commandeRepository.findAll();
112     if (products.isEmpty()) {
113         return Health.down().build();
114     }
115     return Health.up().build();
116 }
117
```

## Etude de cas (2):

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

- a. Les microservice-commandes et microservice-produit doivent être enregistrés auprès d'Eureka
- b. Implémenter une API Gateway comme point d'accès unique à l'application
- c. Implémenter les fonctionnalités CRUD du « microservice-commandes »
- d. Simuler un Timeout d'un des deux microservices, et implémenter un mécanisme de contournement pour protéger le microservice appelant avec Hystrix.

## II- Les Etapes de realisation de l'étude de cas 2 :

### A-

Au niveau de la deuxième commande la classe commande va être comme suite :

```
1 package com.example.servicecommandes.model;
2
3 import com.example.servicecommandes.model.Produits;
4 import com.fasterxml.jackson.annotation.JsonProperty;
5 import jakarta.persistence.*;
6 import lombok.*;
7 import java.time.LocalDate;
8
9 @Entity
10 @Getter @Setter @ToString @NoArgsConstructor @AllArgsConstructor @Builder
11 public class Commandes {
12     @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
13     private Long commandeId;
14     private String description;
15     private Integer quantite;
16     private LocalDate dateDeCreation;
17     private Double montant;
18
19     @Transient //cela signifie ignorer ce attribut
20     private Produits produits;
21     private Long produitId;
22 }
```

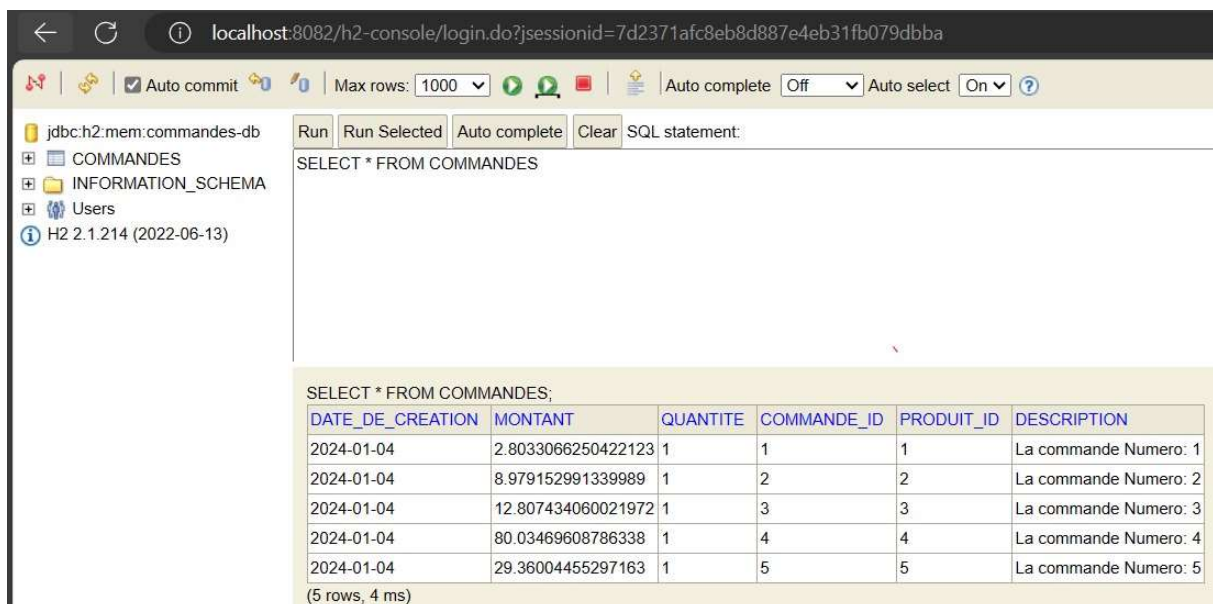
La classe commande contient l'attribut clé primaire de la classe commande.

L'ajout de l'annotation @Transient permet d'éliminé cette colonne au niveau de la table de la base de donnée.

La classe nouvelle classe qu'on va ajouter est la classe Produit qui représente la classe principale au niveau du micro-service produit et qu'on va réaliser son CRUD par la suite.

```
1 package com.example.serviceproduits.model;
2
3 import jakarta.persistence.Entity;
4 import jakarta.persistence.GeneratedValue;
5 import jakarta.persistence.GenerationType;
6 import jakarta.persistence.Id;
7 import lombok.*;
8
9 @Entity
10 @Getter @Setter @ToString @NoArgsConstructor @AllArgsConstructor @Builder
11 public class Produits {
12     @Id @GeneratedValue(strategy = GenerationType.IDENTITY)
13
14     private Long id;
15     private String name;
16     private String description;
17 }
18
```

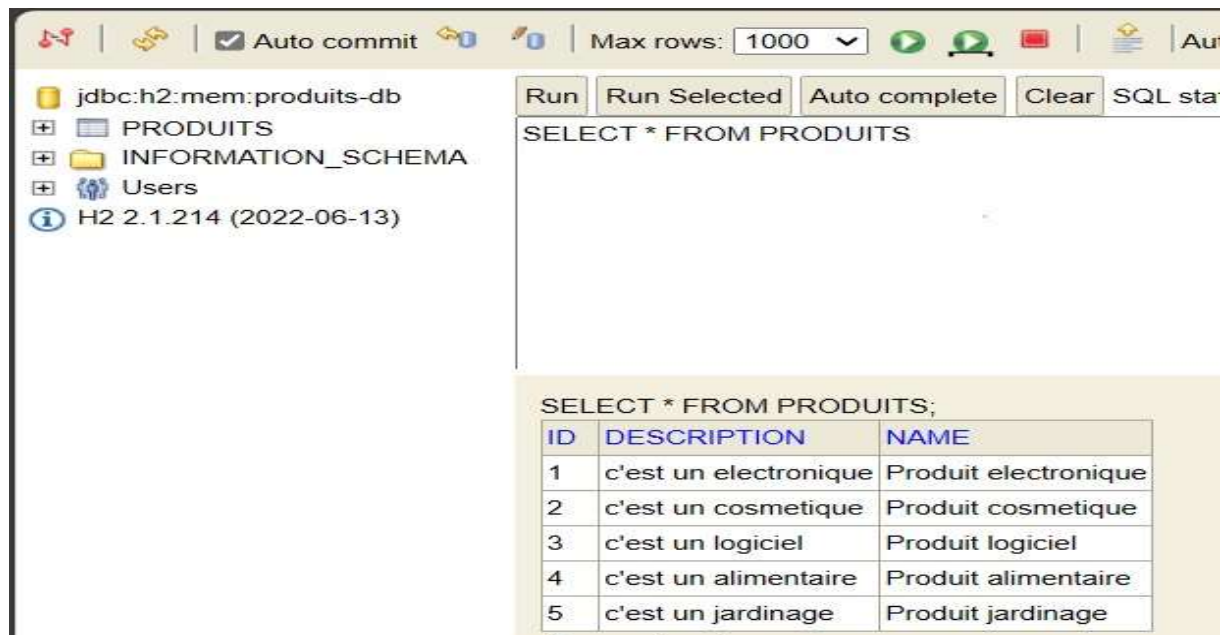
Voici leurs représentations au niveau de la base de données H2 :



The screenshot shows the H2 console interface. The URL bar indicates the connection to localhost:8082/h2-console. The left sidebar shows the database structure with 'jdbc:h2:mem:commandes-db' selected. The main area displays the SQL statement 'SELECT \* FROM COMMANDES;' and its results in a table format. The table has 6 columns: DATE\_DE\_CREATION, MONTANT, QUANTITE, COMMANDE\_ID, PRODUIT\_ID, and DESCRIPTION. There are 5 rows of data, each representing a command with a unique ID and a description 'La commande Numero: X'.

DATE_DE_CREATION	MONTANT	QUANTITE	COMMANDE_ID	PRODUIT_ID	DESCRIPTION
2024-01-04	2.8033066250422123	1	1	1	La commande Numero: 1
2024-01-04	8.979152991339989	1	2	2	La commande Numero: 2
2024-01-04	12.807434060021972	1	3	3	La commande Numero: 3
2024-01-04	80.03469608786338	1	4	4	La commande Numero: 4
2024-01-04	29.36004455297163	1	5	5	La commande Numero: 5

(5 rows, 4 ms)



The screenshot shows a database client interface with a sidebar on the left displaying the database structure: jdbc:h2:mem:produits-db, PRODUITS, INFORMATION\_SCHEMA, Users, and H2 2.1.214 (2022-06-13). The main area contains a SQL query editor with the text "SELECT \* FROM PRODUITS;" and buttons for "Run", "Run Selected", "Auto complete", "Clear", and "SQL sta". Below the editor, a table displays the query results.

ID	DESCRIPTION	NAME
1	c'est un electronique	Produit electronique
2	c'est un cosmetique	Produit cosmetique
3	c'est un logiciel	Produit logiciel
4	c'est un alimentaire	Produit alimentaire
5	c'est un jardinage	Produit jardinage

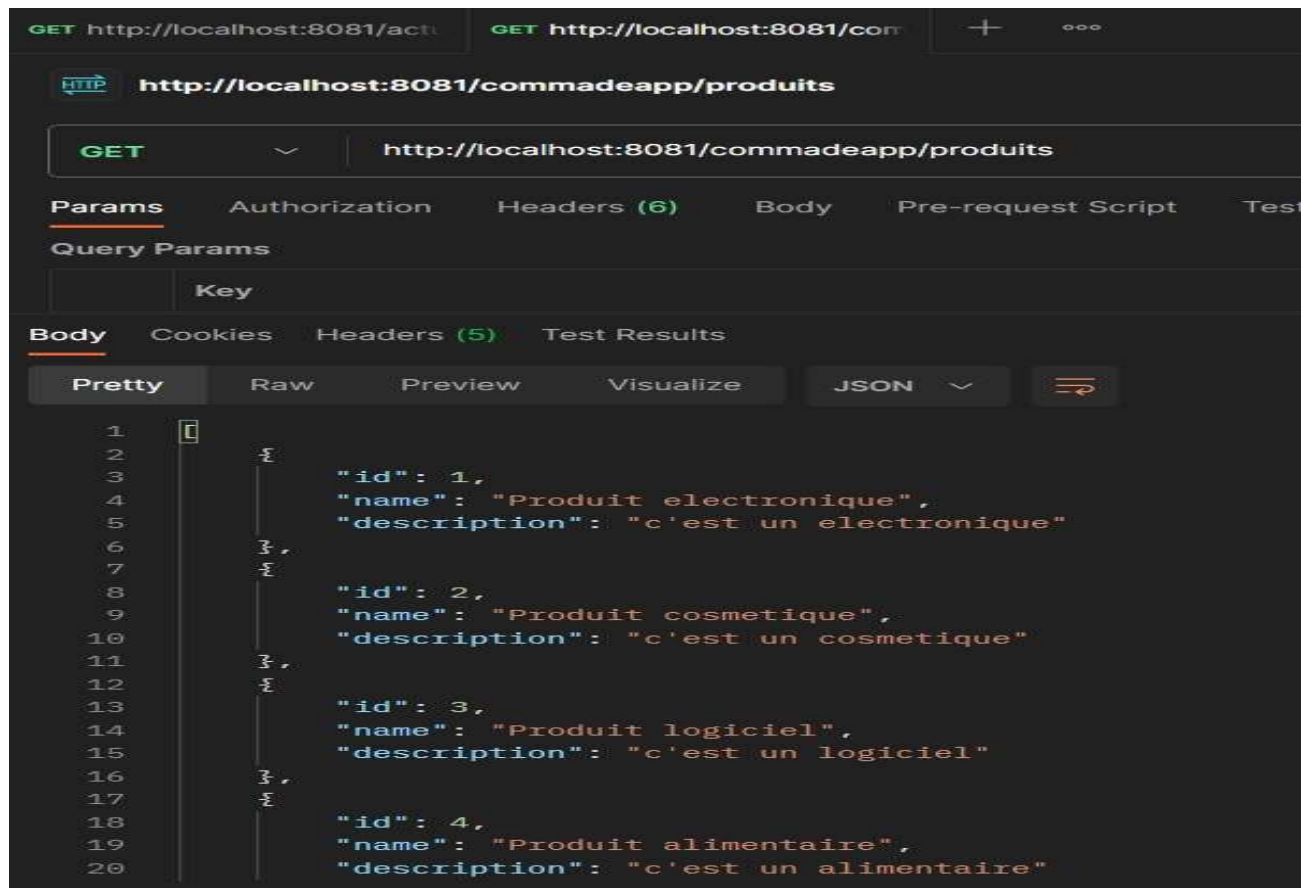
Puis on a réalisé les opérations du « CRUD » au niveau de cette classe Produit.

L'affichage :



The screenshot shows a code editor with a Java REST controller method. The method is annotated with @GetMapping("/produits") and returns a List of Produits objects by calling the findAll() method of the produitRepository.

```
23  
24 @GetMapping("/produits")  
25 public List<Produits> produitsList(){  
26  
27     return produitRepository.findAll();  
28 }
```



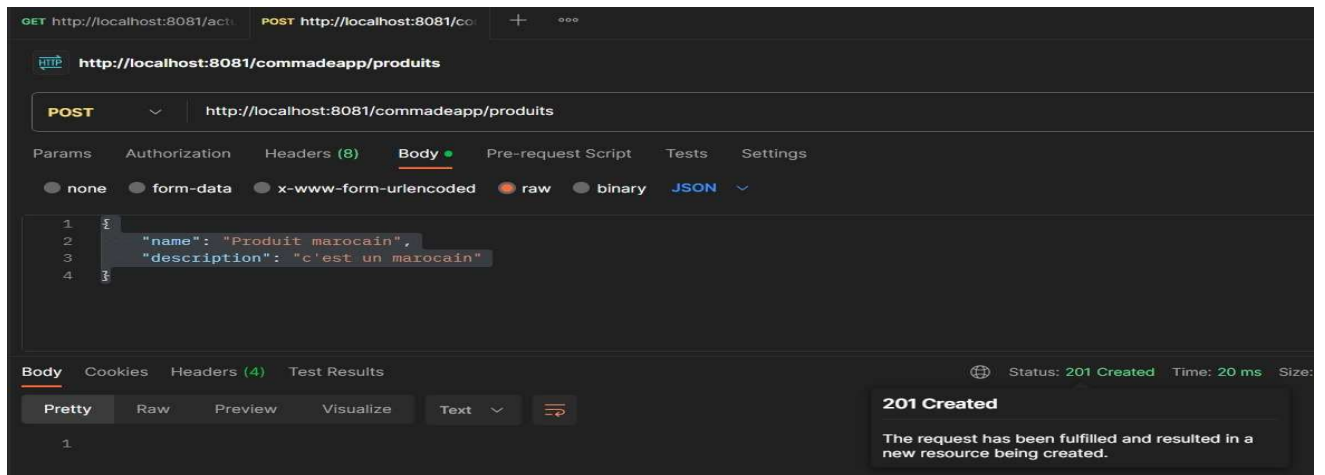
L'opération de l'ajout au niveau du CRUD :

« L'ajout »

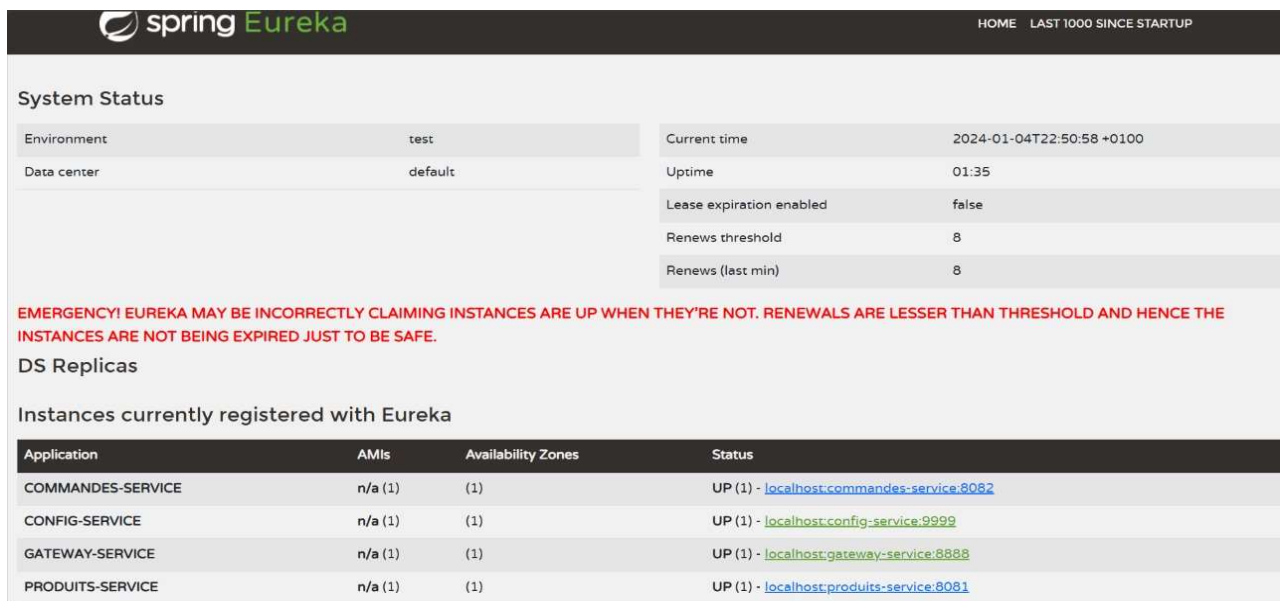
```
39
40     @PostMapping("/produits")
41     public ResponseEntity<Produits> addCommande(@RequestBody Produits produits){
42         produitRepository.save(produits);
43         return new ResponseEntity<>(HttpStatus.CREATED);
44     }
45
```



Résultats :



L'enregistrement des micro-services « commande et produit » au niveau de Eureka :



Cette figure représente l'interface d'Eureka qu'on l'accède via localhost/8761 qui permet de montrer les micro-services enregistrer et leurs statuts « UP ou Down ».

## B-

L'implémentation d'API Gateway :

La représentation de la classe principale du API Gateway :

```
1 package com.example.servicegateway;
2
3 import org.springframework.boot.SpringApplication;
4 import org.springframework.boot.autoconfigure.SpringBootApplication;
5 import org.springframework.cloud.client.discovery.ReactiveDiscoveryClient;
6 import org.springframework.cloud.gateway.discovery.DiscoveryClientRouteDefinitionLocator;
7 import org.springframework.cloud.gateway.discovery.DiscoveryLocatorProperties;
8 import org.springframework.context.annotation.Bean;
9
10 @SpringBootApplication
11 public class ServiceGatewayApplication {
12
13     public static void main(String[] args) {
14         SpringApplication.run(ServiceGatewayApplication.class, args);
15     }
16     //configuration dynamic des routes
17     @Bean
18     DiscoveryClientRouteDefinitionLocator locator(ReactiveDiscoveryClient rdc, DiscoveryLocatorProperties dlp){
19         return new DiscoveryClientRouteDefinitionLocator(rdc, dlp);
20     }
21 }
```

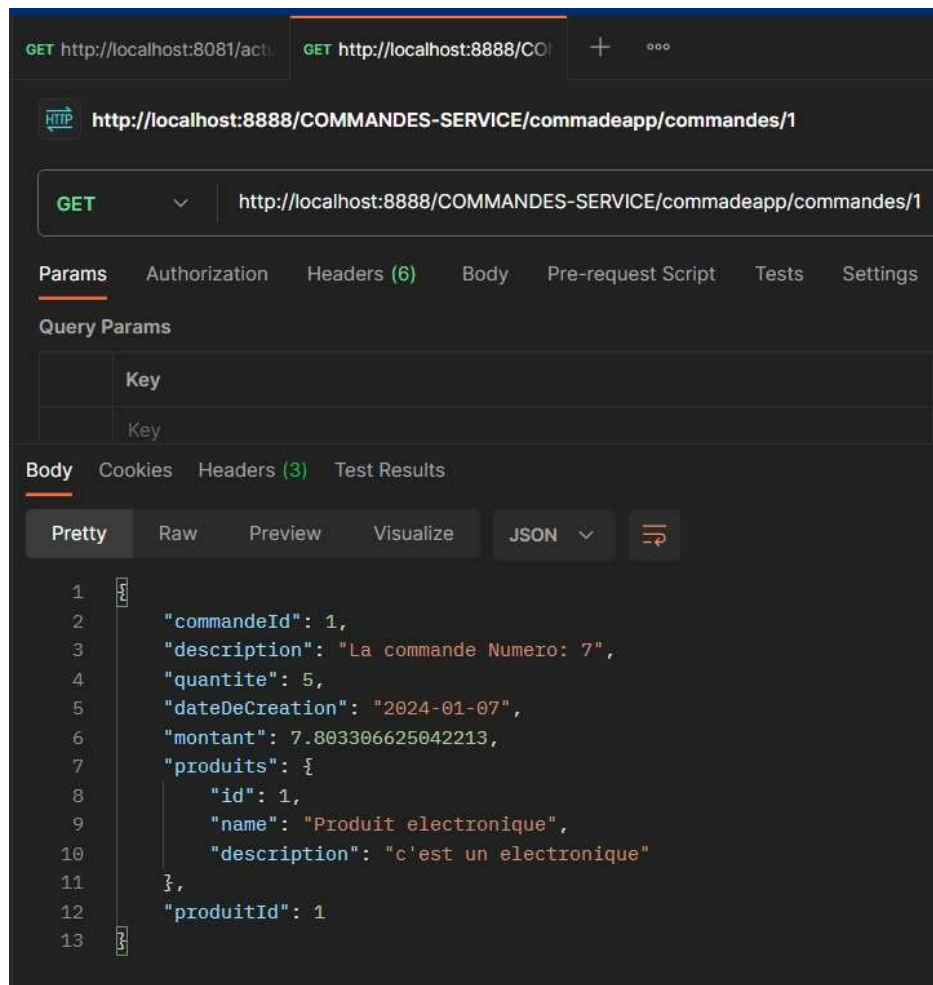
Sa configuration au niveau du fichier application. Properties :

```
1 spring.application.name=gateway-service
2 server.port=8888
3 eureka.instance.prefer-ip-address=true
4 eureka.client.service-url.defaultZone=http://localhost:8761/eureka
5 management.endpoints.web.exposure.include=*
```

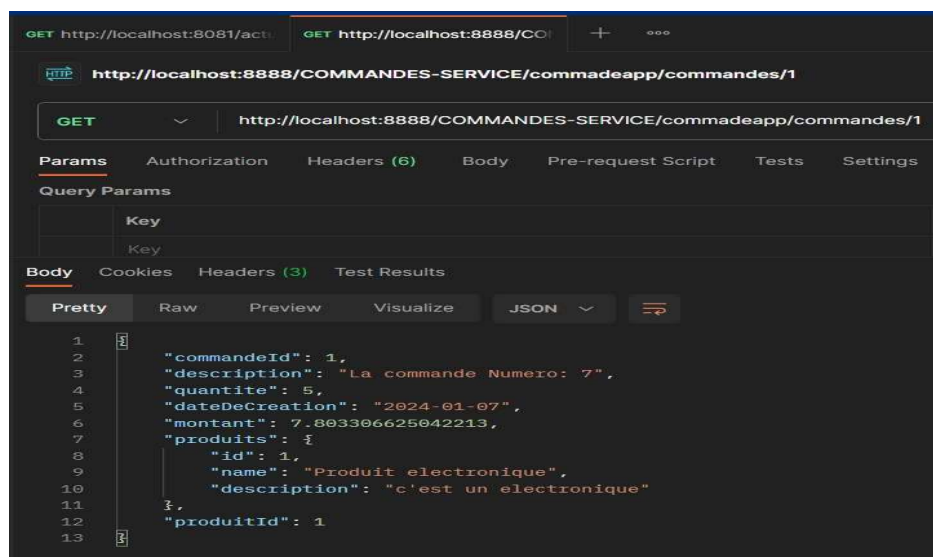
## C-

Voici les résultats de l'affichage en se basant sur cette api afin de traiter l'ensembles de fonctionnalités de la commande et de produits.

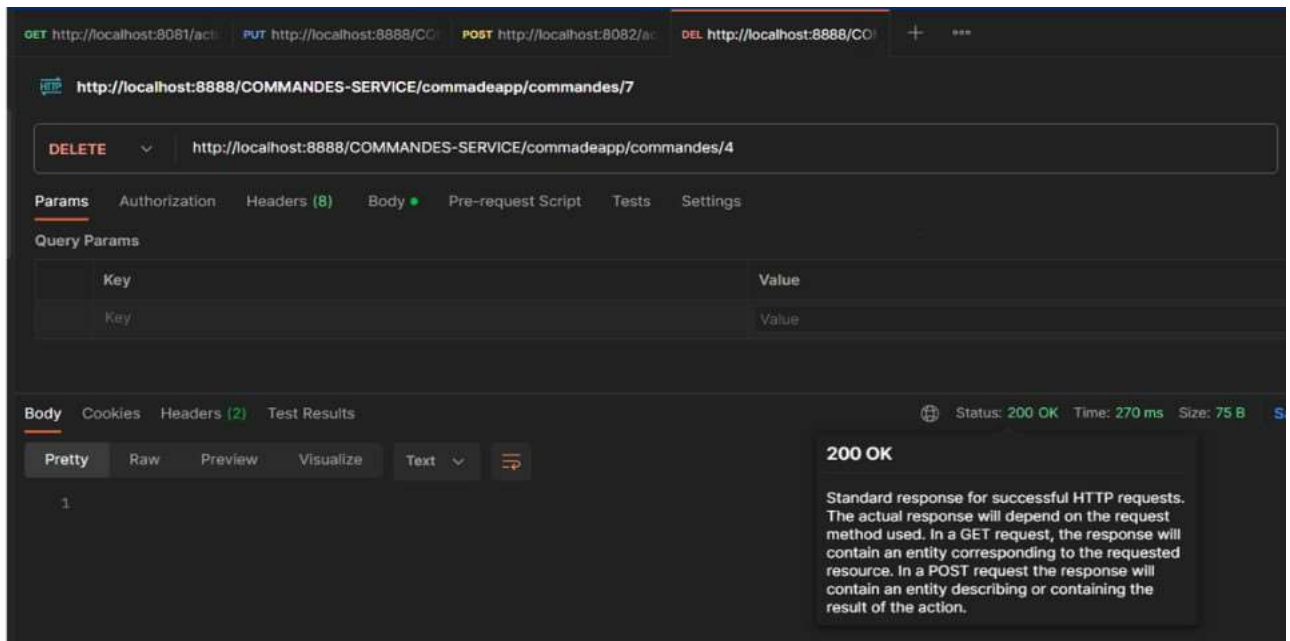
L'affichage de toutes les commandes et leurs produits :



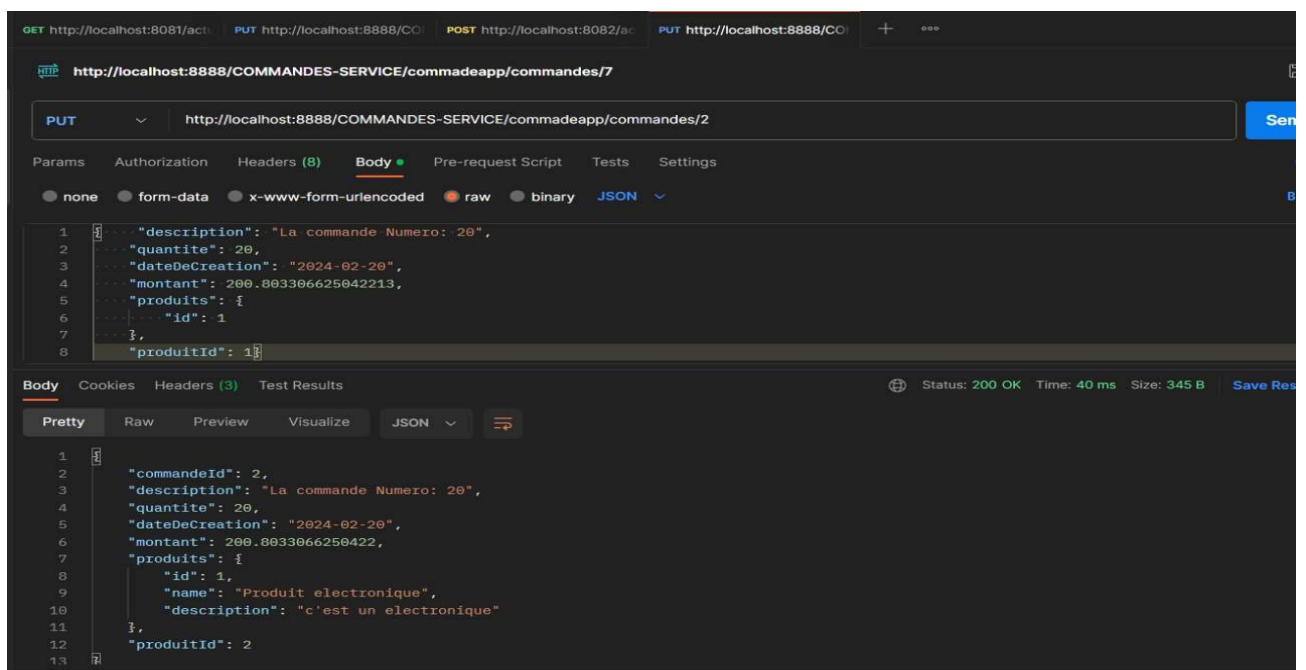
L'affichage d'une commande X avec les produits :



Suppression d'une commande X :



Modification d'une commande X en utilisant la Gateway :



Voici le micro-service commande après qu'on arrête le micro-service produit (statuts est down).



localhost:8888/COMMANDES-SERVICE/commadeapp/commande

```
1  [
2    {
3      "commandeId": 1,
4      "description": "La commande Numero: 1",
5      "quantite": 1,
6      "dateDeCreation": "2024-01-05",
7      "montant": 72.17763550671245,
8      "produits": {
9        "id": 1,
10       "name": "Non disponible",
11       "description": "Non disponible"
12     },
13     "produitId": 1
14   },
15   {
16     "commandeId": 2,
17     "description": "La commande Numero: 2",
18     "quantite": 1,
19     "dateDeCreation": "2024-01-05",
20     "montant": 64.59339268954692,
21     "produits": {
22       "id": 2,
23       "name": "Non disponible",
24       "description": "Non disponible"
25     },
26     "produitId": 2
27   },
28   {
29     "commandeId": 3,
30     "description": "La commande Numero: 3",
31     "quantite": 1,
32     "dateDeCreation": "2024-01-05",
33     "montant": 29.5065287559764,
34     "produits": {
35       "id": 3,
36       "name": "Non disponible",
37       "description": "Non disponible"
38     },
39     "produitId": 3
40   },
41   {
42     "commandeId": 4,
43     "description": "La commande Numero: 4",
44     "quantite": 1,
45     "dateDeCreation": "2024-01-05",
46     "montant": 79.78847078926349,
47     "produits": {
48       "id": 4,
49       "name": "Non disponible",
50       "description": "Non disponible"
51     },
52     "produitId": 4
53   },
54   {
55     "commandeId": 5,
56     "description": "La commande Numero: 5",
57     "quantite": 1,
58     "dateDeCreation": "2024-01-05",
59     "montant": 82.36471217450972,
60     "produits": {
61       "id": 5,
62       "name": "Non disponible",
63       "description": "Non disponible"
64     },
65     "produitId": 5
66   }
67 ]
```

D-

```
7 import org.springframework.web.bind.annotation.PathVariable;
8 import java.util.List;
9
10 5 usages
11 @FeignClient(name = "PRODUITS-SERVICE")
12 public interface ProduitsRestClient {
13     4 usages
14     @CircuitBreaker(name = "produitService", fallbackMethod = "getDefaultProduits")
15     @GetMapping("/commadeapp/produits/{id}")
16     Produits findProduitsById(@PathVariable Long id);
17     1 usage
18     @CircuitBreaker(name = "produitService", fallbackMethod = "getAllProduits")
19     @GetMapping("/commadeapp/produits")
20     List<Produits> allProduits();
21
22     default Produits getDefaultProduits(Long id, Exception exception) {
23         Produits produits = new Produits();
24         produits.setId(id);
25         produits.setName("Non disponible");
26         produits.setDescription("Non disponible");
27         return produits;
28     }
29     default List<Produits> getAllProduits(Exception exception) {
30         return List.of();
31     }
32 }
```