Dr. E_H_B TOURE DMI/FST/UCAD Sokhna Diarra DIOP

M1_GLSI

Développement d'applications Java

Lab_2 : Premiers pas avec Spring-boot

Dans ce lab, nous allons créer une petite application avec Spring Boot. Nous allons mettre en place un petit service web avec Spring Boot, en créant une API de type REST. Nous aurons également besoin d'un outil comme Postman afin de tester l'API ainsi créée.

Exercice 1

Créer une nouvelle application Spring Boot nommée sb-coffee-1.

`Création d' un simple domaine :`

Créer une classe `Coffee` dans le fichier principal de l'application comme suit :

La classe `Coffee` dispose de trois constructeurs :

- Un constructeur sans arguments (dont le corps est vide)

```
public class SbCoffee1Application {

    // Définition de la classe Coffee à l'intérieur du fichier principal
    static class Coffee {
        private String id;
        private String name;

        // Constructeur sans arguments
        public Coffee() {}
```

- Un constructeur avec tous les champs renseignés

```
        // Constructeur avec tous les champs
        public Coffee(String id, String name) {
            this.id = id;
            this.name = name;
        }
```

- Un constructeur qui ne renseigne que le nom du café, l'ID sera généré par un UUID.

```
        // Constructeur qui génère un ID aléatoire
        public Coffee(String name) {
            this.id = UUID.randomUUID().toString();
            this.name = name;
        }
```

Ensuite, cette classe est complétée avec les getters et setters nécessaires.

```java
    // Getters et Setters
    public String getId() {
        return id;
    }

    public void setId(String id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
```

Création du contrôleur REST :

Créer, toujours dans le même fichier, une classe nommée `RestCoffeeController` qui va permettre de définir les différentes routes associées aux verbes HTTP : GET, POST, PUT, DELETE.Ces méthodes permettent de mettre en œuvre les opérations CRUD :

```java
@RestController
@RequestMapping("/coffees")
class RestCoffeeController {
    private final List<Coffee> coffeeList = new ArrayList<>();

    // Ajouter quelques cafés par défaut
    public RestCoffeeController() {
        coffeeList.add(new Coffee(name:"Espresso"));
        coffeeList.add(new Coffee(name:"Cappuccino"));
        coffeeList.add(new Coffee(name:"Latte"));
    }
```

- **GET** : Recherche (Retrieve)

```java
// Récupérer **tous** les cafés (GET /coffees)
@GetMapping
public List<Coffee> getAllCoffees() {
    return coffeeList;
}

// Récupérer un café par **ID** (GET /coffees/{id})
@GetMapping("/{id}")
public Coffee getCoffeeById(@PathVariable String id) {
    Optional<Coffee> coffee = coffeeList.stream()
            .filter(c -> c.getId().equals(id))
            .findFirst();
    return coffee.orElse(null);
}
```

- **POST** : Création (Create)

```java
// Ajouter un **nouveau** café (POST /coffees)
@PostMapping
public Coffee addCoffee(@RequestBody Coffee newCoffee) {
    coffeeList.add(newCoffee);
    return newCoffee;
}
```

- **PUT** : Mise à jour (Update)

```java
// Modifier un café existant (PUT /coffees/{id})
@PutMapping("/{id}")
public Coffee updateCoffee(@PathVariable String id, @RequestBody Coffee updatedCoffe
    for (int i = 0; i < coffeeList.size(); i++) {
        if (coffeeList.get(i).getId().equals(id)) {
            coffeeList.set(i, updatedCoffee);
            return updatedCoffee;
        }
    }
    return null;
}
```

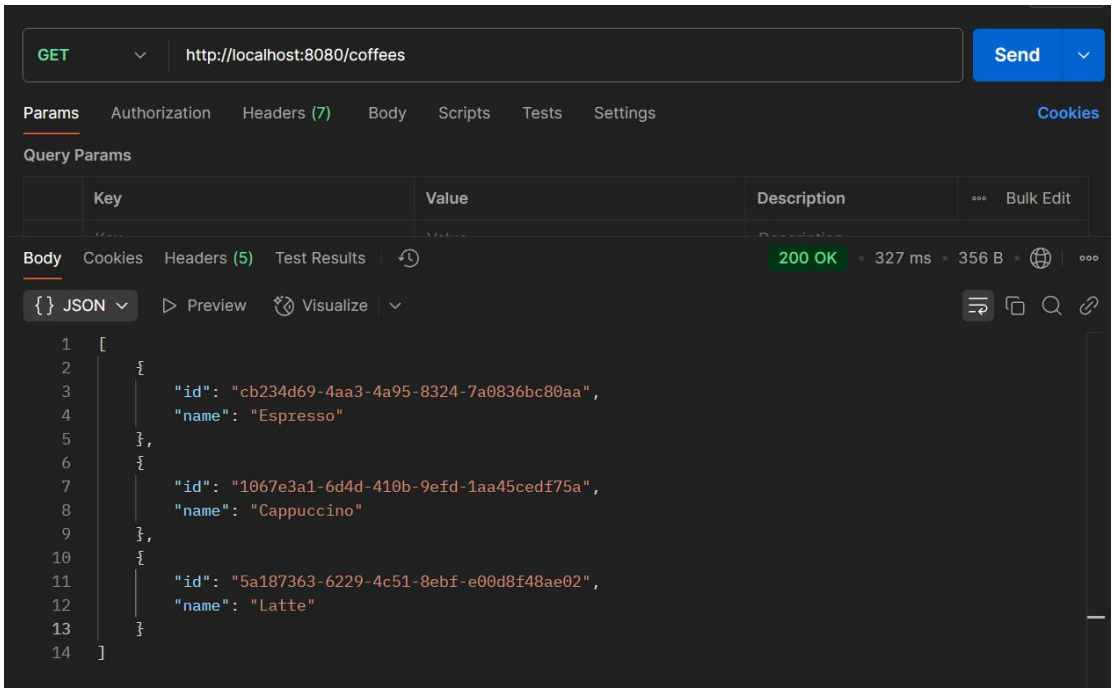
- **DELETE** : Suppression (Delete)

```java
// Supprimer un café (DELETE /coffees/{id})
@DeleteMapping("/{id}")
public void deleteCoffee(@PathVariable String id) {
    coffeeList.removeIf(c -> c.getId().equals(id));
}
```

Dans cette implémentation, nous stockons un ensemble de cafés en utilisant une `ArrayList`. Dans le constructeur, nous ajoutons un ensemble de cafés pour obtenir des résultats dès la première requête de recherche.

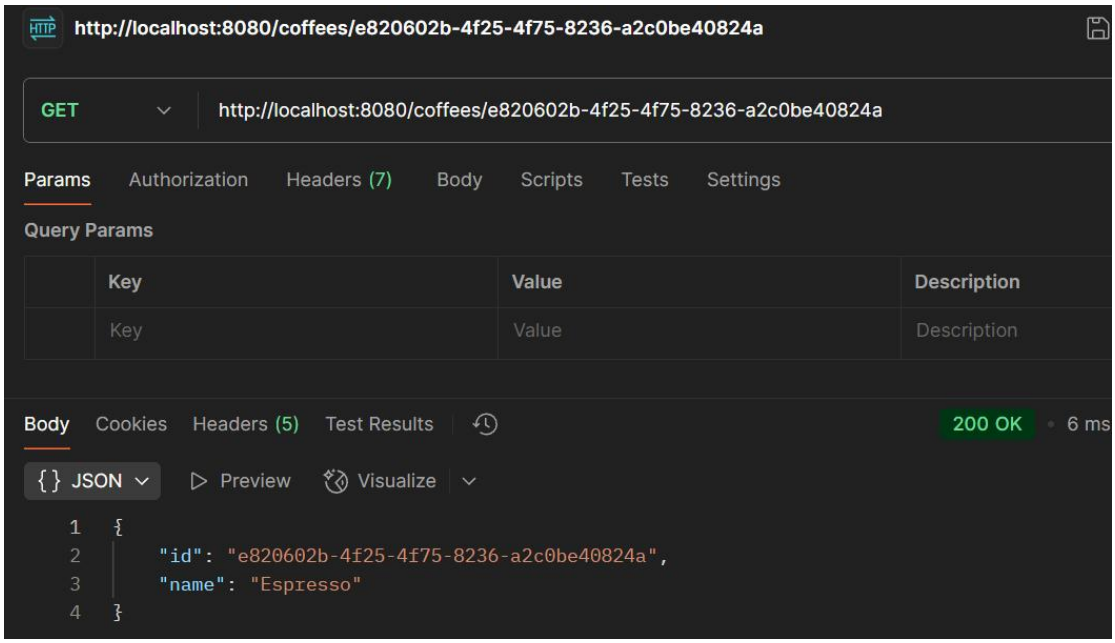
1) Recherche : tous les cafés

Définir une route sur `/coffees` qui retourne l'ensemble des cafés disponibles.



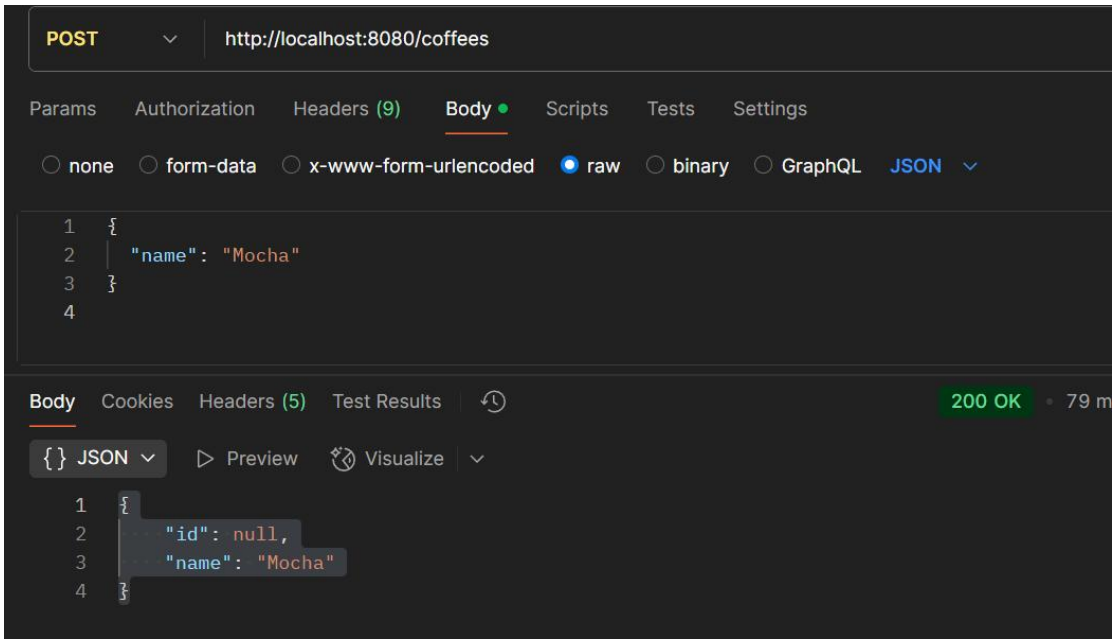
2) Recherche : un café par ID

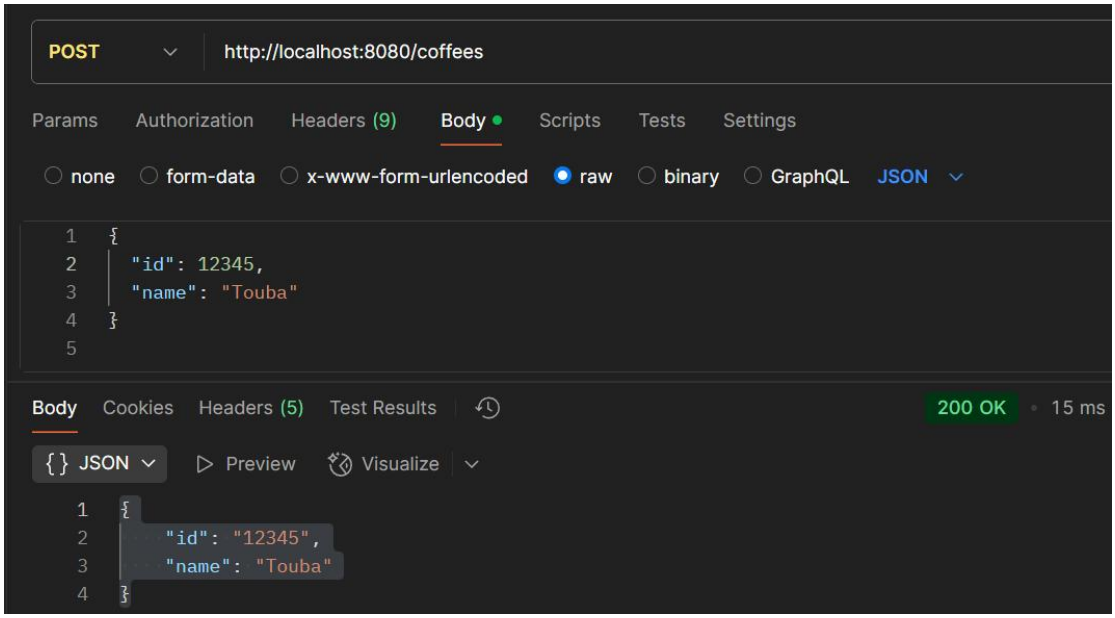
Modifier le contrôleur en ajoutant une nouvelle route permettant de récupérer un café spécifique en fonction de son ID. Cette méthode utilise `Optional` de Java 8 pour éviter les erreurs liées aux valeurs `null`.

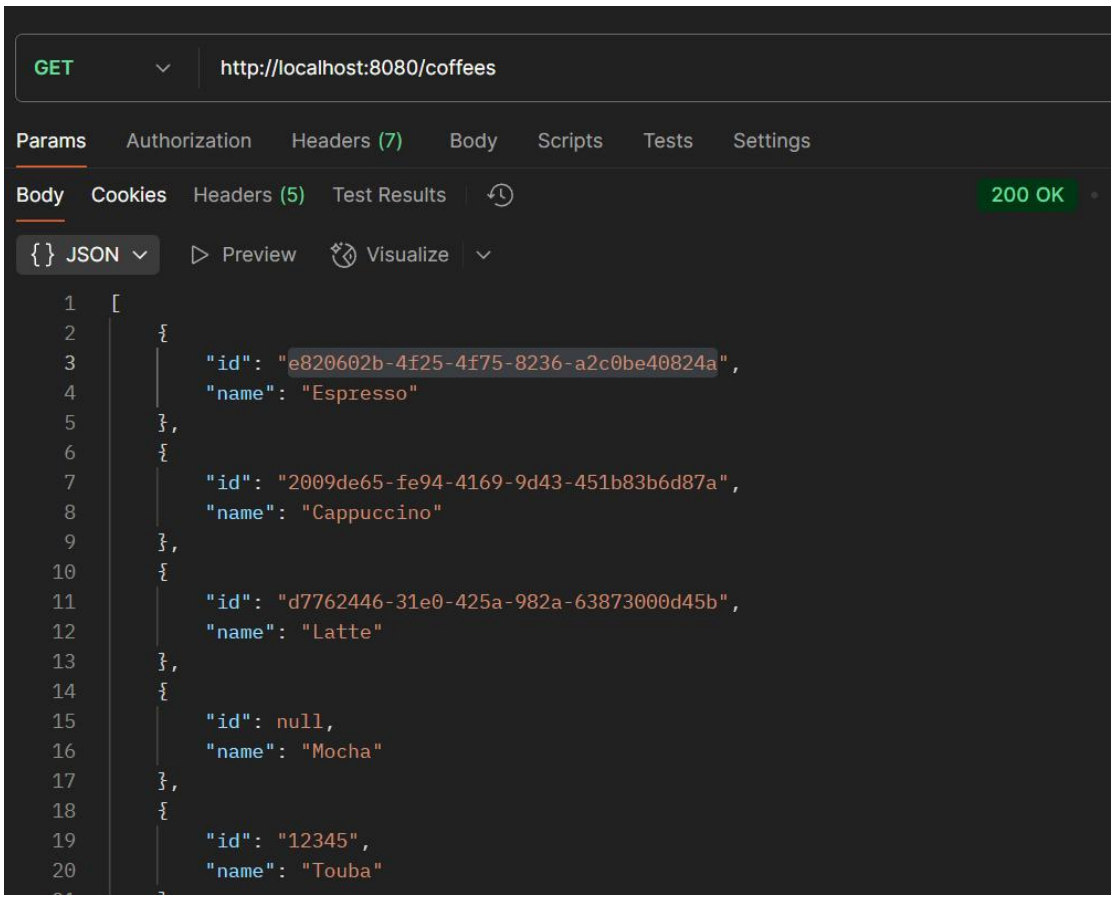


3) Création d'un café

Utiliser la méthode `POST` pour créer un nouveau café. Cette requête nécessite un corps contenant les informations de la ressource à ajouter.

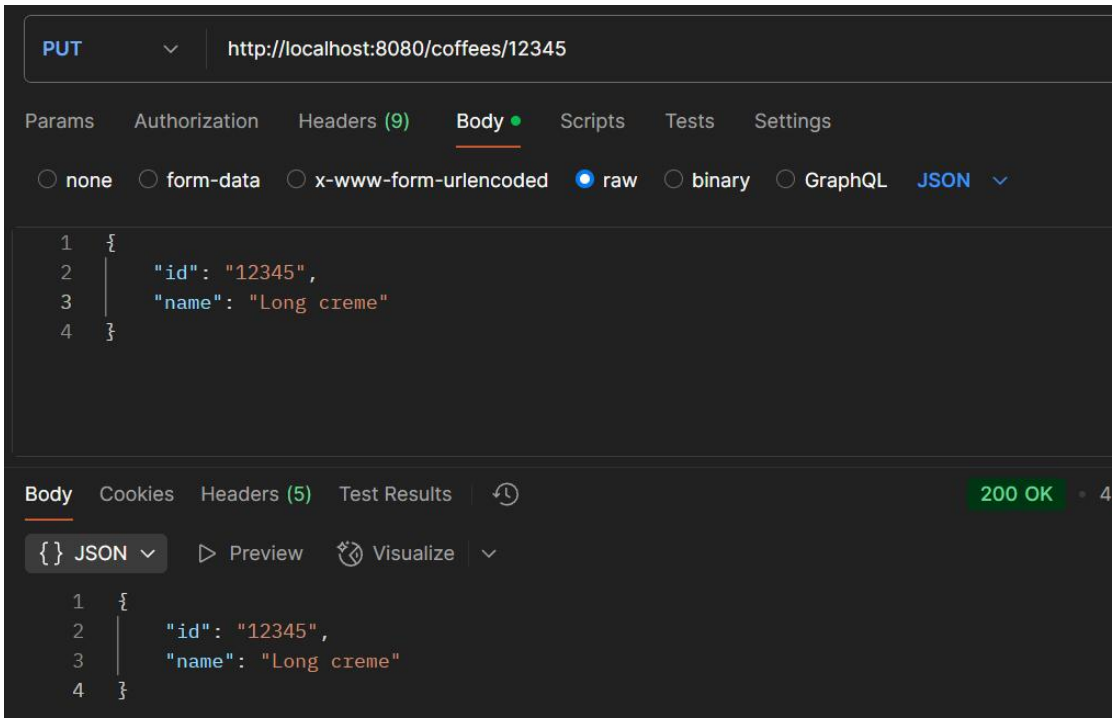

POST
http://localhost:8080/coffees
Params
Authorization
Headers (9)
Body
Scripts
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
{
"name": "Mocha"
}
Body
Cookies
Headers (5)
Test Results
200 OK
79 m
{} JSON
Preview
Visualize
{
"id": null,
"name": "Mocha"
}

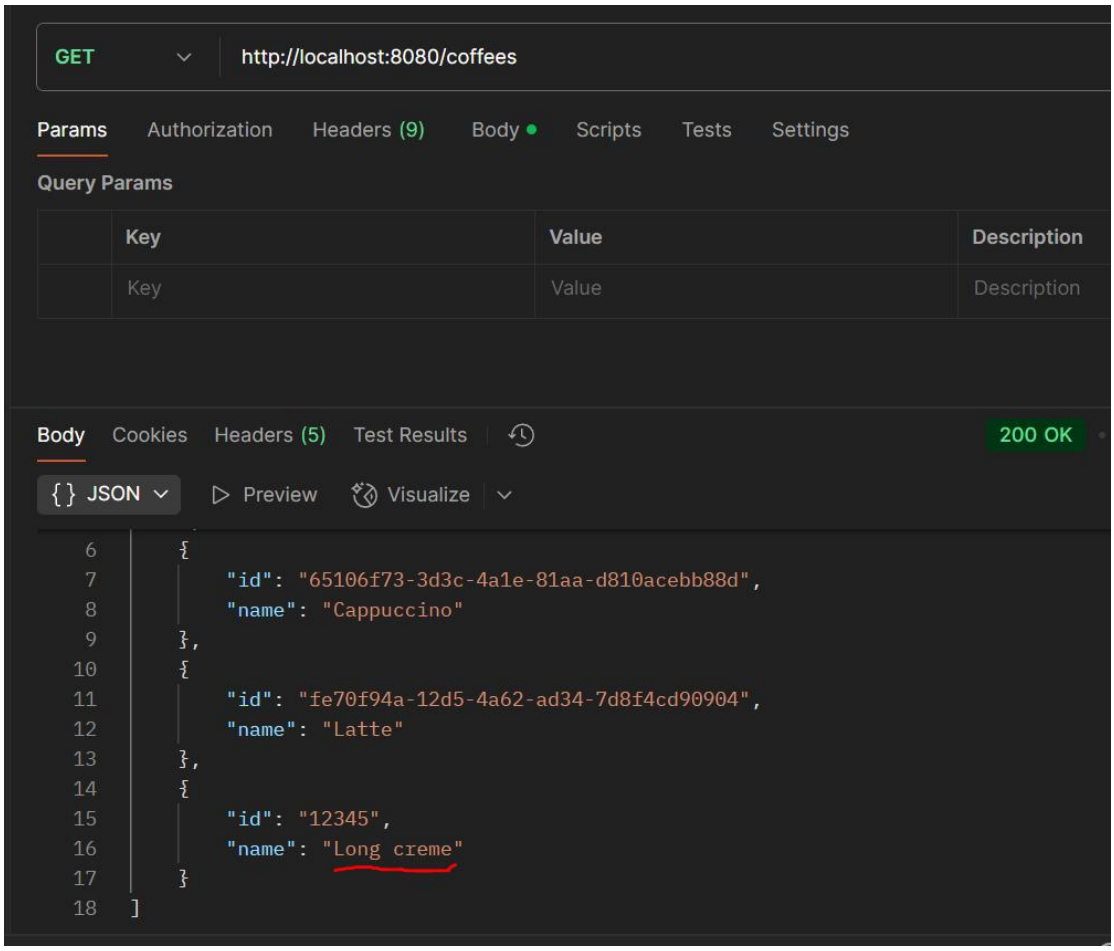

POST
http://localhost:8080/coffees
Params
Authorization
Headers (9)
Body
Scripts
Tests
Settings
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
JSON
{
"id": 12345,
"name": "Touba"
}
Body
Cookies
Headers (5)
Test Results
200 OK
15 ms
{} JSON
Preview
Visualize
{
"id": "12345",
"name": "Touba"
}



4) Modification d’ un café

Utiliser la méthode `PUT` pour mettre à jour un café existant. Si l’ID fourni ne correspond à aucun café existant, un nouveau café est créé.

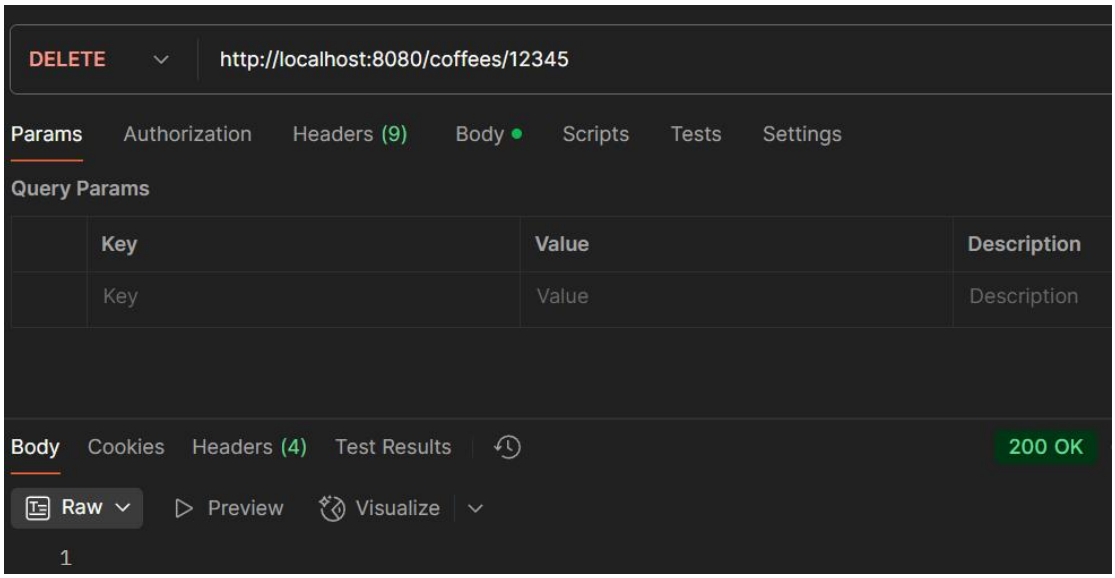




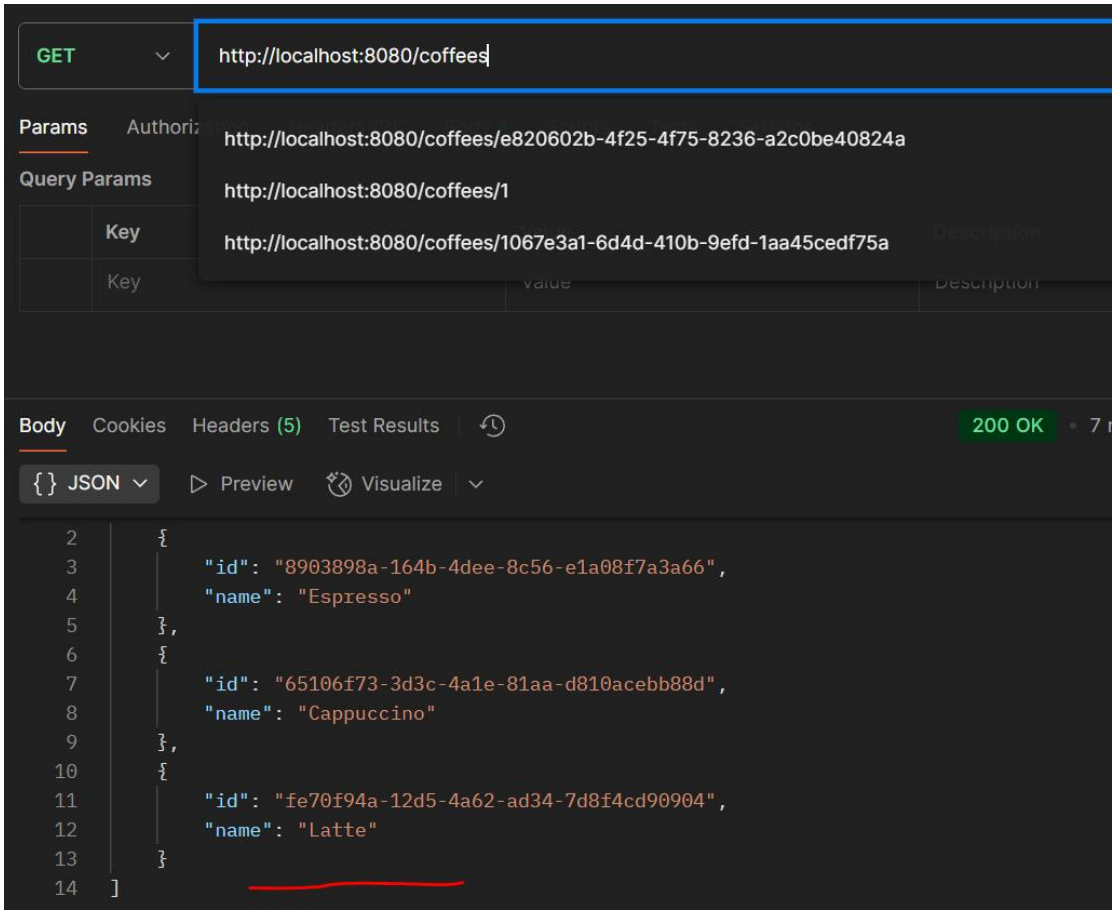
Touba devient Long creme.

5) Suppression d’ un café

Utiliser la méthode `DELETE` pour supprimer un café en fonction de son ID.



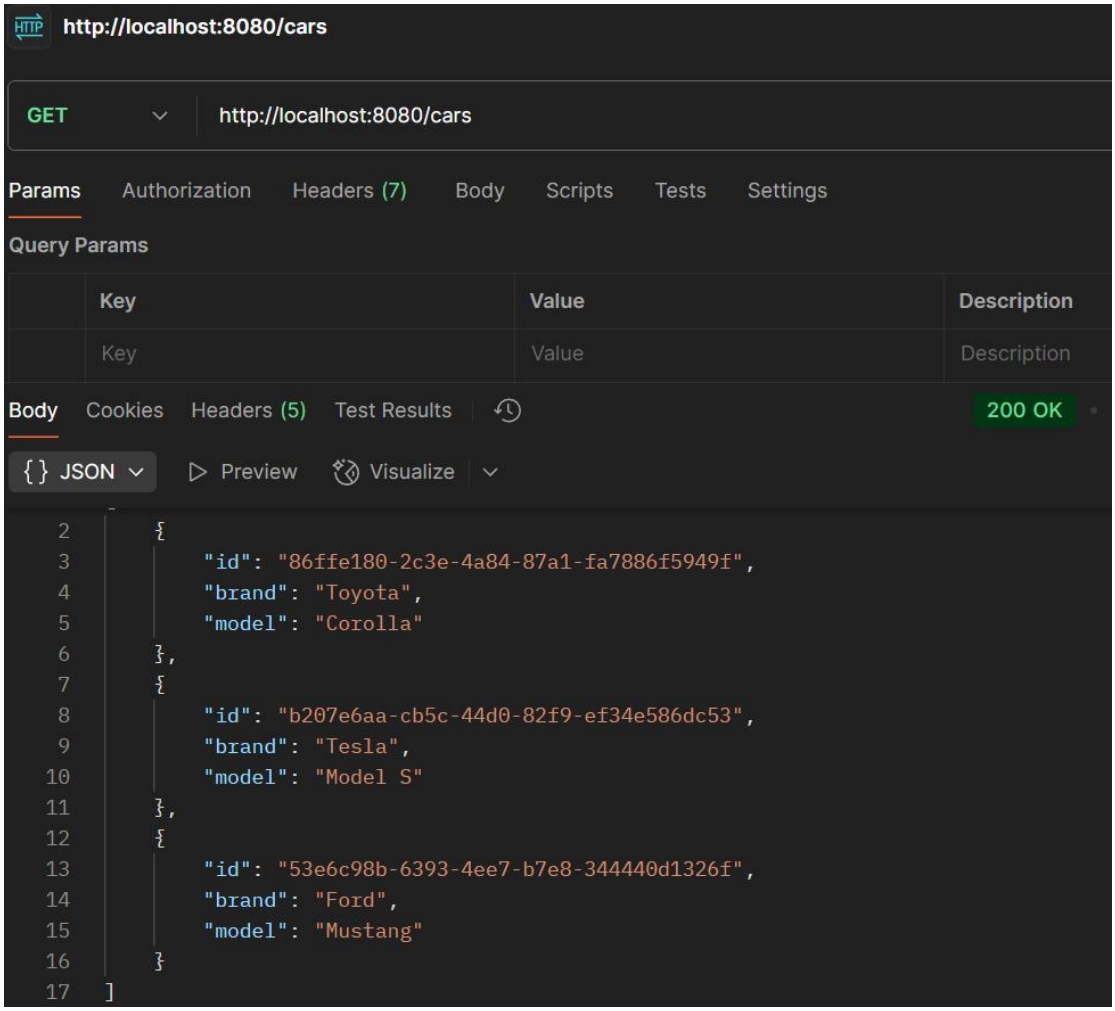
Note : Les données ne sont pas persistées, ce qui signifie qu’elles seront réinitialisées à chaque redémarrage du serveur. Par la suite, une base de données sera utilisée pour assurer la persistance des données.



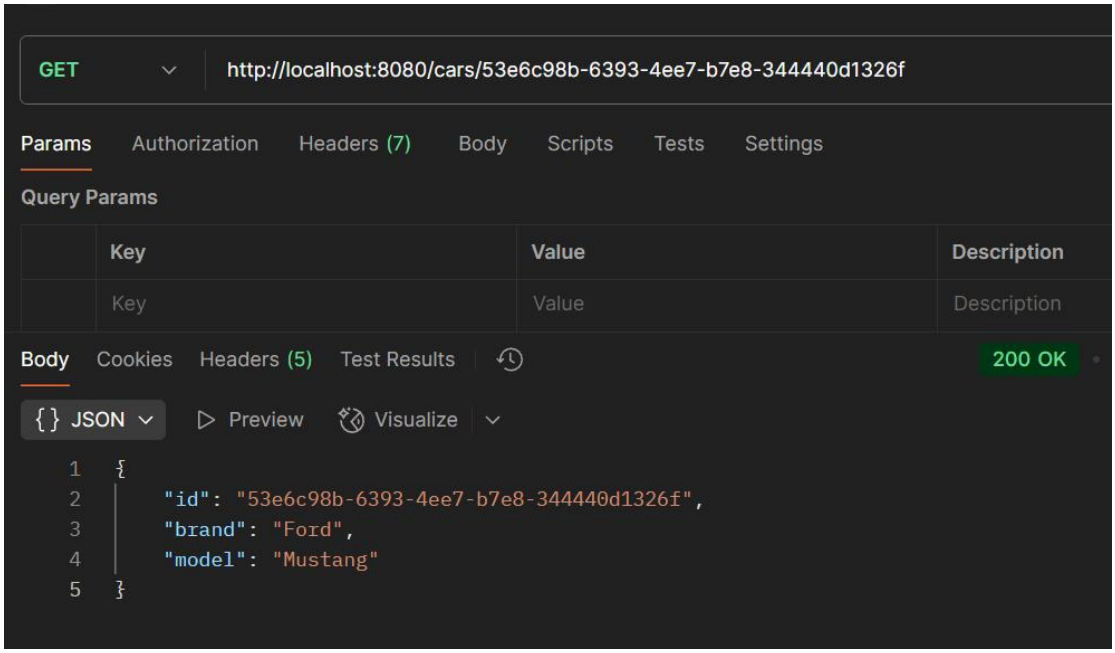
Exercice 2

Créer une application Spring Boot nommée sb-car-1 qui implémente une API REST pour gérer une collection de voitures. Cette API doit prendre en charge les opérations CRUD, avec un modèle similaire à celui utilisé pour les cafés. Ajouter un attribut `id` comme dans l'Exercice 1.

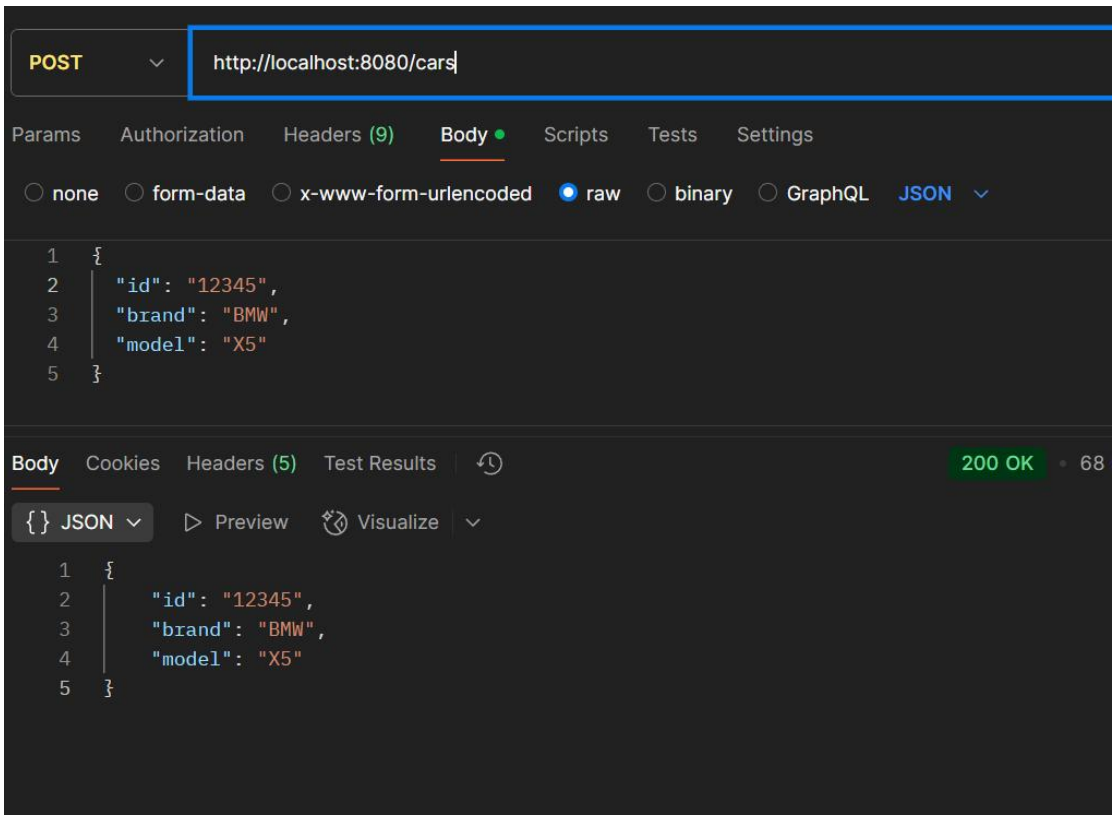
1) Recherche : tous les cars

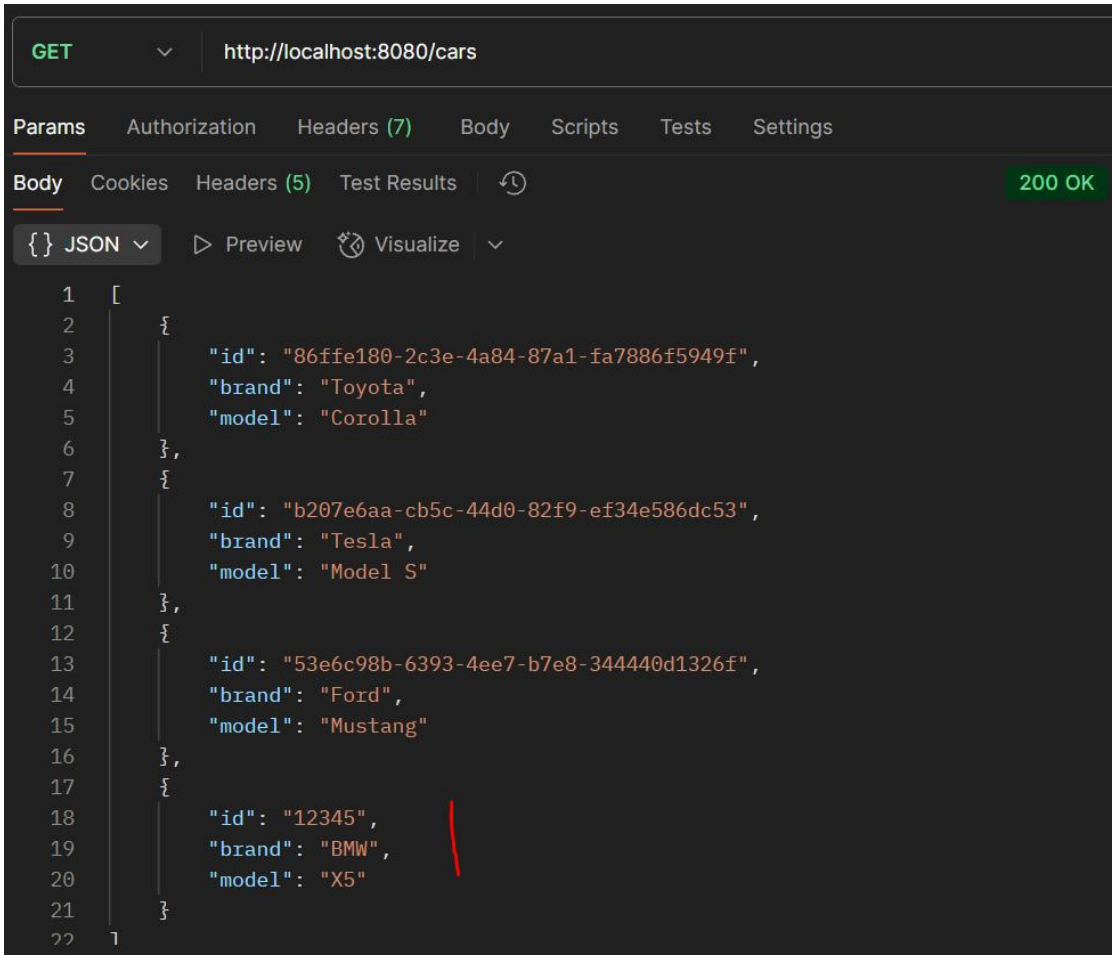


2) Recherche : un car par ID

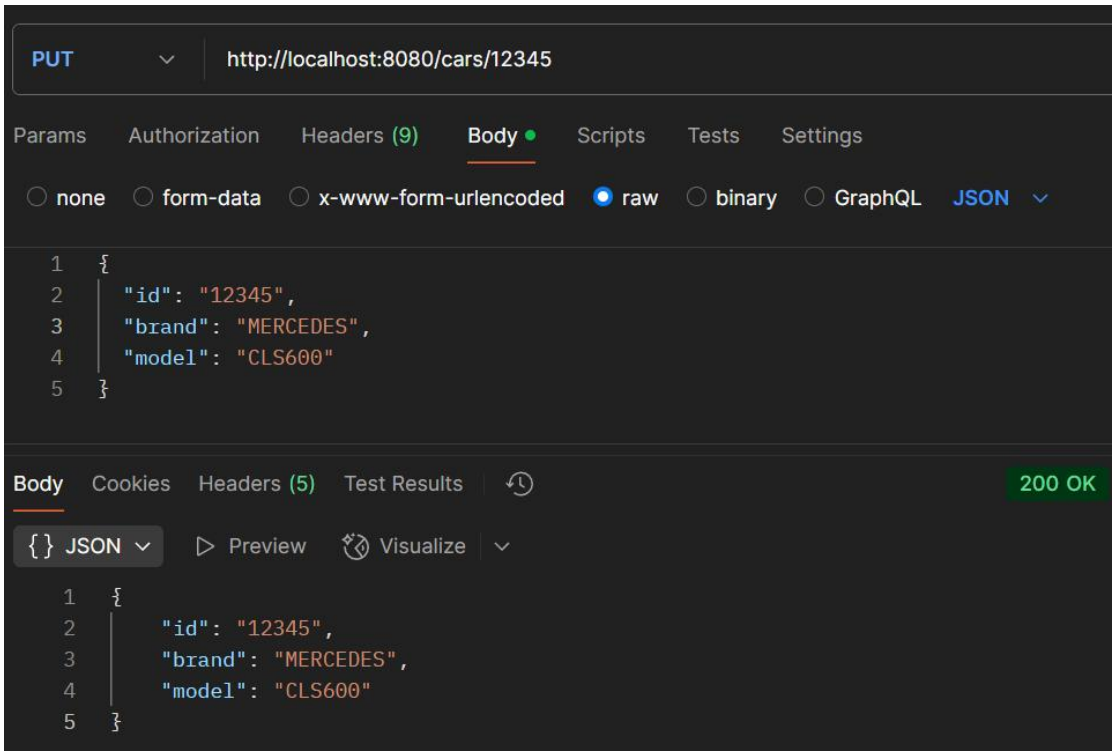


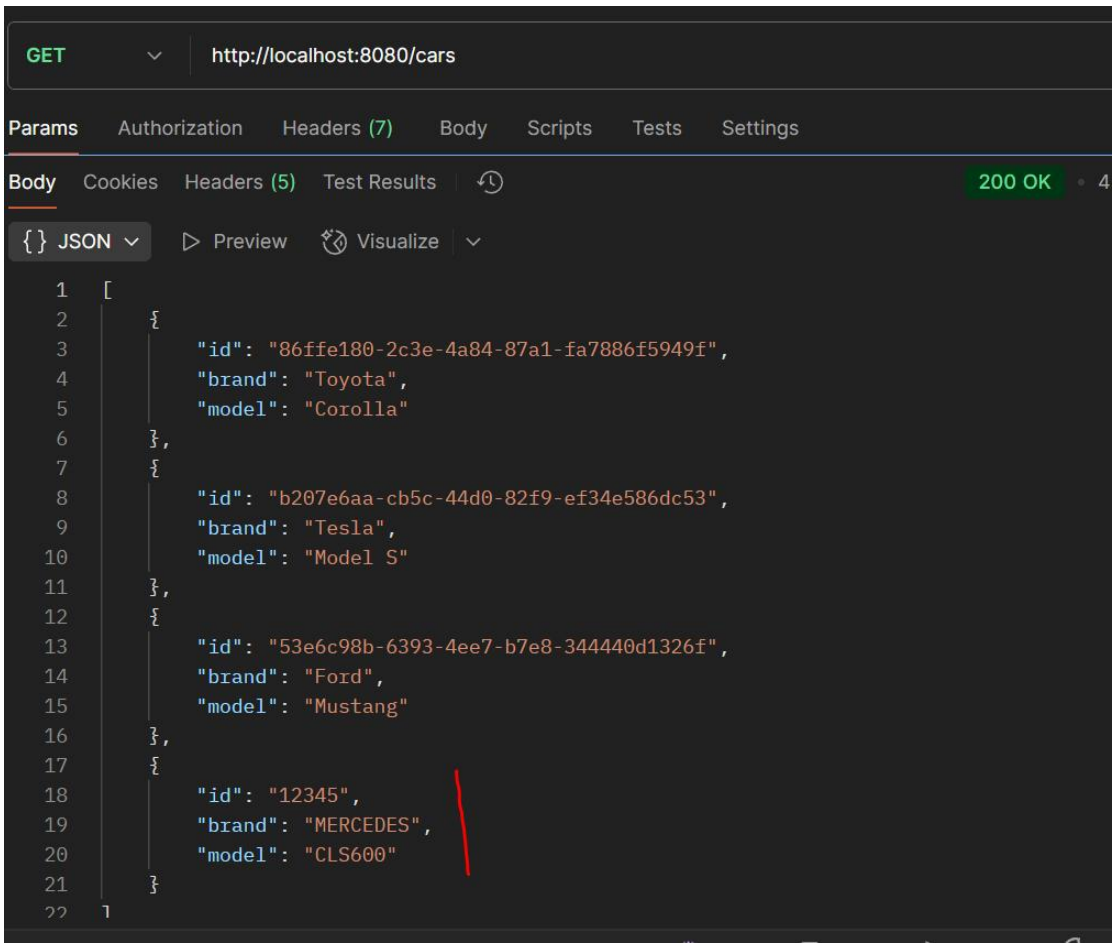
3) Création d’ un car



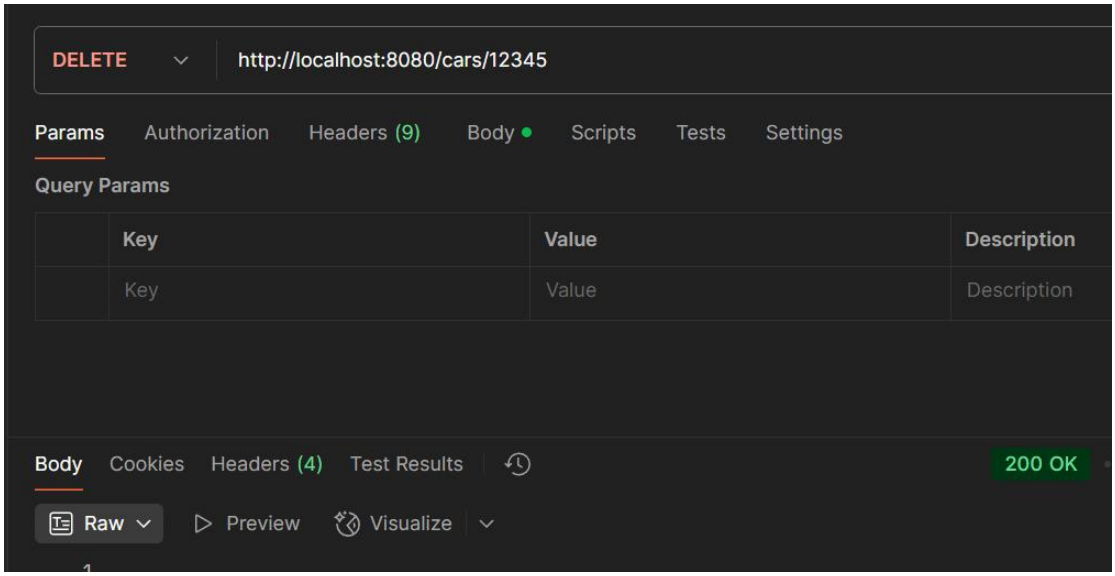


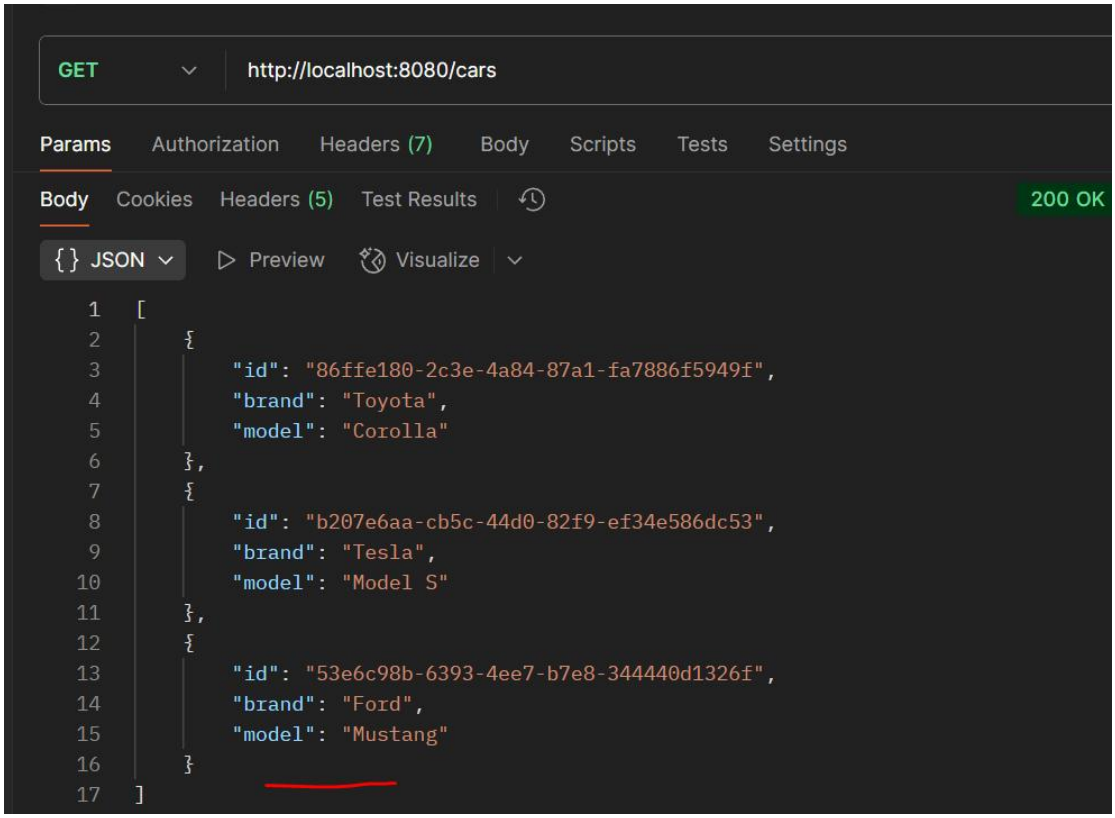
4) Modification d’ un café





5) Suppression d’ un car


DELETE
http://localhost:8080/cars/12345
Params
Authorization
Headers (9)
Body
Scripts
Tests
Settings
Query Params
Key
Value
Description
Key
Value
Description
Body
Cookies
Headers (4)
Test Results
200 OK
Raw
Preview
Visualize


GET
http://localhost:8080/cars
Params
Authorization
Headers (7)
Body
Scripts
Tests
Settings
Body
Cookies
Headers (5)
Test Results
200 OK
JSON
Preview
Visualize
[
{
"id": "86ffe180-2c3e-4a84-87a1-fa7886f5949f",
"brand": "Toyota",
"model": "Corolla"
},
{
"id": "b207e6aa-cb5c-44d0-82f9-ef34e586dc53",
"brand": "Tesla",
"model": "Model S"
},
{
"id": "53e6c98b-6393-4ee7-b7e8-344440d1326f",
"brand": "Ford",
"model": "Mustang"
}
]