# TP1 : Développement d'une API REST avec Spring Boot

Riham Kaddour Bakir

22 novembre 2025

## 1 Introduction

Ce TP a pour objectif de créer une API REST simple avec Spring Boot 21. L'application permet de gérer une liste d'étudiants, ainsi que de tester des fonctionnalités de base telles que l'affichage, l'ajout, la modification et la suppression.

Le travail est présenté étape par étape, avec des exemples et des captures d'écran illustrant le fonctionnement des différentes fonctionnalités.

## 2 Initialisation du projet Spring Boot

Pour commencer, nous avons créé un projet Spring Boot en utilisant la version 21 du JDK. L'application a été testée avec des endpoints simples pour vérifier le bon démarrage du sur le port 8080.

### 2.1 Test des endpoints de base

Deux endpoints ont été testés pour vérifier la configuration initiale :

— `/bonjour`

— `/bonsoir`

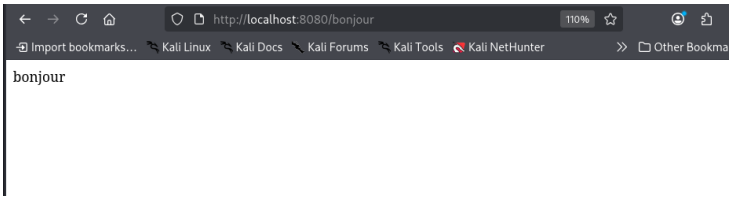


FIGURE 1 – Test du endpoint `/bonjour`

## 3 Création de la classe Étudiant

Une classe `Etudiant` a été définie pour représenter les étudiants avec les attributs suivants :

— identifiant

— nom

— moyenne

```
1
2  public class Etudiant {
3      private int identifiant;
4      private String nom;
5      private int moyenne;
6
7      public Etudiant() {}
8
9      public Etudiant(int identifiant, String nom, int moyenne) {
10         this.identifiant = identifiant;
11         this.nom = nom;
12         this.moyenne = moyenne;
13     }
14
15     // Getters et Setters
16 }
```

## 3.1 Récupération d'un étudiant

L'endpoint `/etudiant` permet de récupérer un étudiant prédéfini sans aucun paramètre. Cet endpoint est utile pour vérifier rapidement que la structure de l'objet `Etudiant` et la sérialisation JSON fonctionnent correctement dans l'application.

Lorsqu'un client envoie une requête HTTP de type `GET` vers :

`http://localhost:8080/etudiant`

le serveur retourne un objet `Etudiant` déjà instancié dans la méthode. Voici le code correspondant :

```
1 @GetMapping(value = "/etudiant")
2 public Etudiant getEtudiant() {
3     return new Etudiant(1, "riham", 19);
4 }
```

Cette méthode crée un nouvel étudiant avec un identifiant fixe, un nom et une moyenne, puis le renvoie automatiquement au format JSON grâce au mécanisme fourni par Spring Boot.

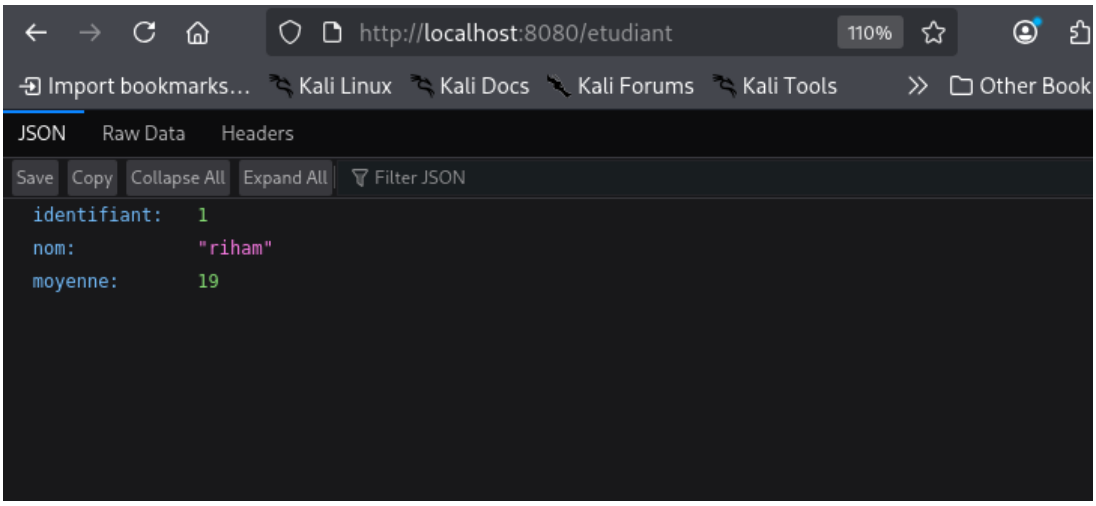


FIGURE 2 – ffichage d'un étudiant via /etudiant .

## 3.2 Calcul d'une somme

Pour tester le passage de paramètres dans l'URL, un endpoint `/somme` permet de calculer la somme de deux valeurs. Les paramètres `a` et `b` sont transmis en tant que variables de requête.

```
@GetMapping(value = "/somme")
public double somme(double a, double b) {
    return a + b;
}
```

La méthode récupère automatiquement les deux paramètres fournis dans l'URL et renvoie leur addition sous forme numérique.

L'appel suivant :

`http://localhost:8080/somme?a=1&b=2`

retourne la valeur `3`. Un test a également été réalisé via Postman pour vérifier le fonctionnement.

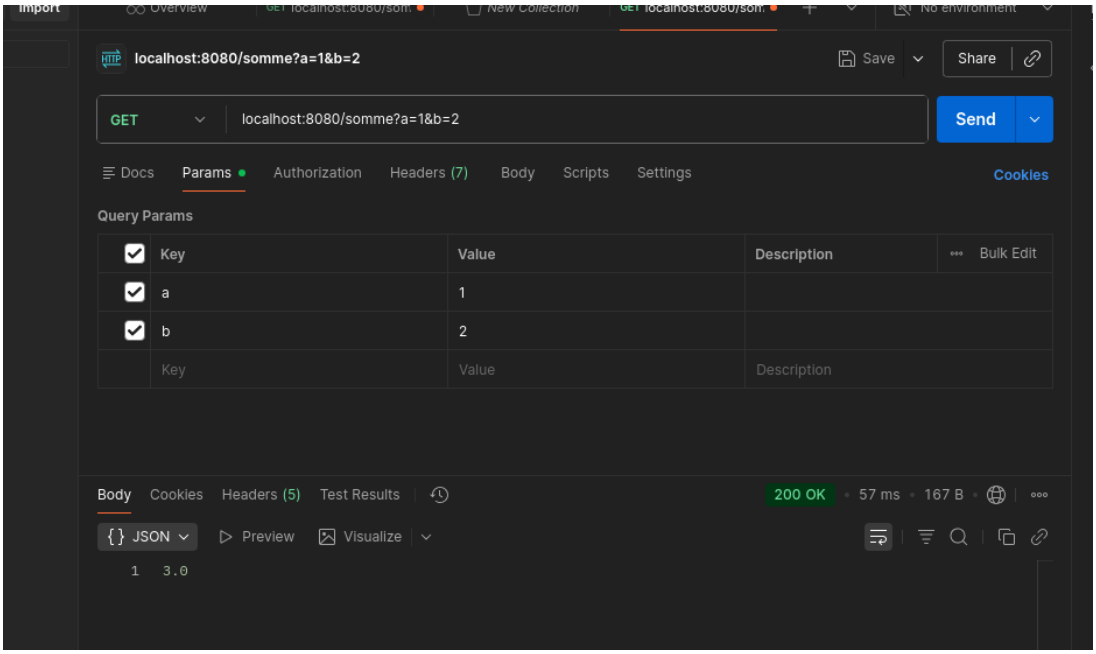


FIGURE 3 – Test de l'endpoint `/somme` avec Postman.

## 3.3 Liste des étudiants

L'endpoint `/liste` permet de récupérer l'ensemble des étudiants stockés dans la liste statique de l'application. Cet endpoint renvoie une collection d'objets *Etudiant* au format JSON.

```
@GetMapping(value = "/liste")
public Collection<Etudiant> getAllEtudiants() {
    return liste;
}
```

La méthode retourne directement la liste en mémoire. Grâce au mécanisme de sérialisation automatique, Spring convertit les objets en JSON.

L'appel suivant permet d'obtenir la liste complète :

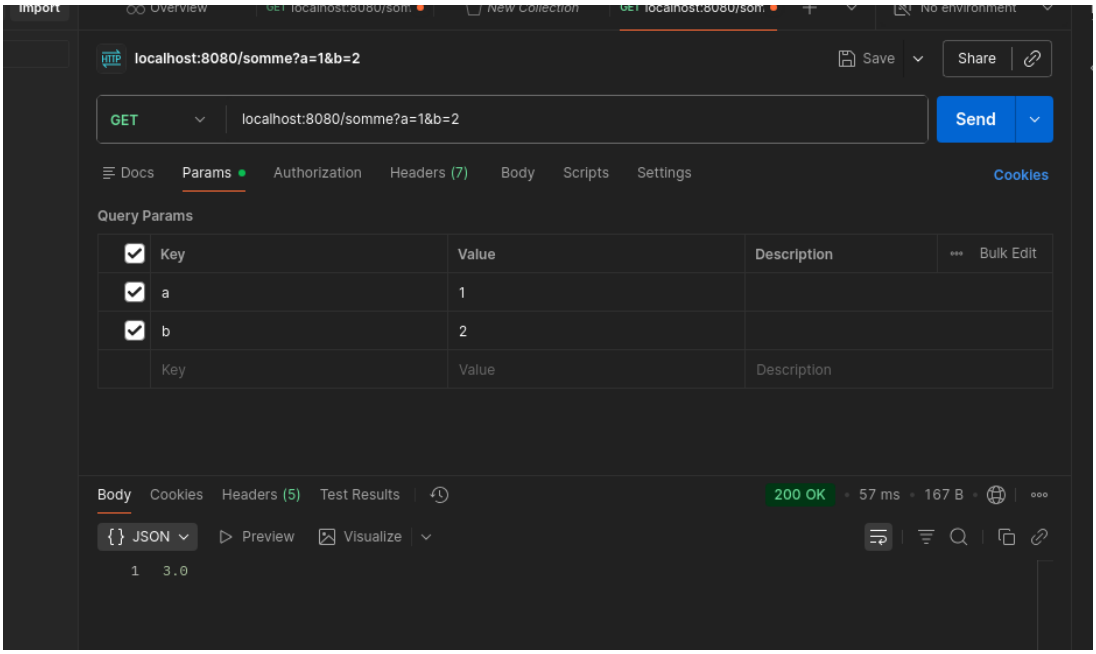


FIGURE 4 – Affichage de la liste complète des étudiants.

## 3.4 Ajout d'un étudiant

L'endpoint `/addEtudiant` permet d'ajouter un nouvel étudiant. La méthode suivante reçoit un objet `Etudiant` et l'ajoute à la liste en mémoire.

```
@PostMapping(value = "/addEtudiant")
public Etudiant addEtudiant(Etudiant etudiant){
    liste.add(etudiant);
    return etudiant;
}
```

L'appel suivant ajoute un étudiant avec l'identifiant `5`, le nom `test` et une moyenne de 20 :

`http://localhost:8080/addEtudiant?identifiant=5&nom=test&moyenne=20`

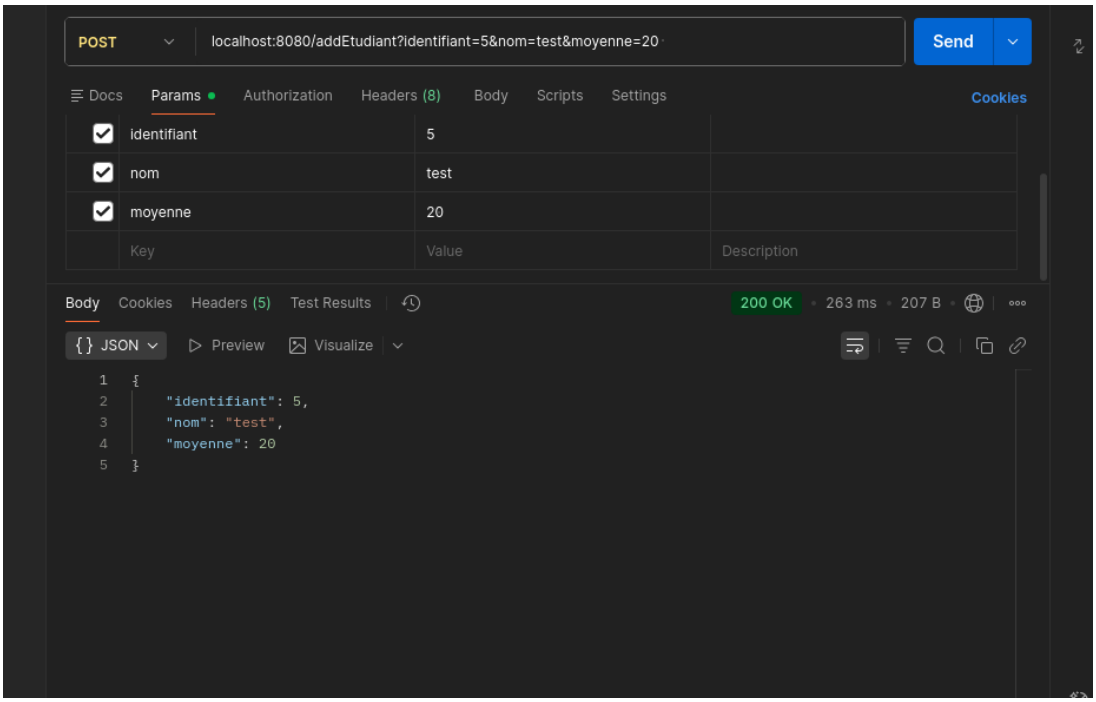


FIGURE 5 – Ajout d'un étudiant via l'endpoint /addEtudiant.

## 3.5 Suppression d'un étudiant

La suppression est réalisée via l'endpoint `/suppEtudiant`. La méthode retire de la liste l'étudiant correspondant à l'ID fourni.

```
@DeleteMapping(value = "/suppEtudiant")
public void supprimerEtudiant(int id){
    liste.remove(id);
}
```

Exemple d'appel :

`http://localhost:8080/suppEtudiant?id=3`

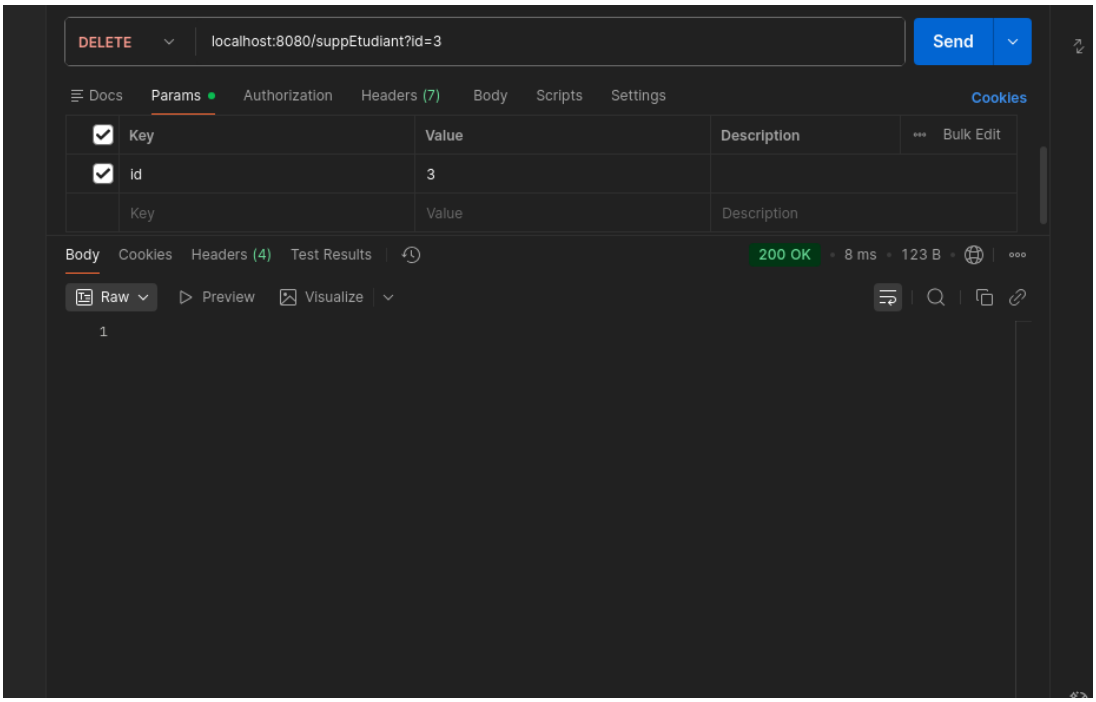


FIGURE 6 – Suppression de l'étudiant ayant l'identifiant 3.

## 3.6 Modification d'un étudiant

L'endpoint `/modEtudiant` permet de modifier le nom d'un étudiant identifié par son ID. La méthode suivante met à jour uniquement le champ `nom` :

```
@PutMapping(value = "/modEtudiant")
public void modifierEtudiant(int id, String nom){
    liste.get(id).setNom(nom);
}
```

Exemple d'appel :

`http://localhost:8080/modEtudiant?id=2&nom=rihamtest`

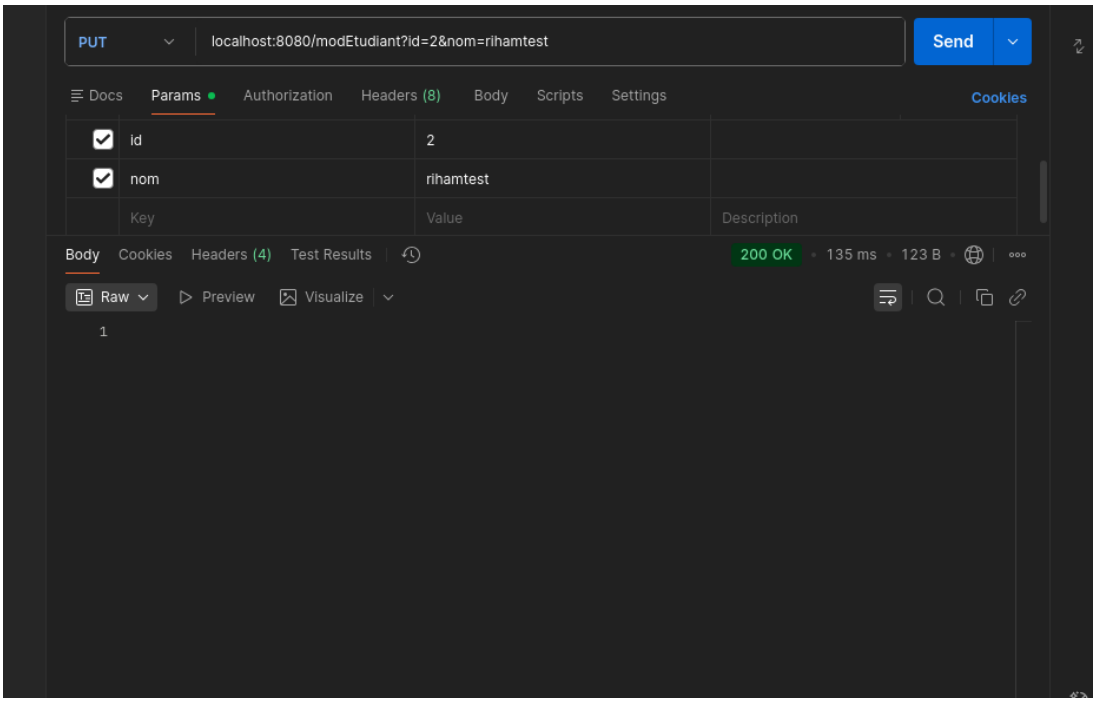


FIGURE 7 – Modification du nom de l'étudiant identifié par 2.

# 4 Conclusion

Ce TP permet de comprendre les principes de base d'une API REST avec Spring Boot :

— création de endpoints GET, POST, PUT et DELETE
— manipulation d'une liste d'objets en mémoire
— utilisation des paramètres pour personnaliser les requêtes

Les captures d'écran montrent que toutes les fonctionnalités ont été testées et fonctionnent correctement.