Rapport TP 3 – Base de données avancées

Nom : Lyes Ahfir
Date : 30 avril 2025

Partie 1 :

L'objectif de ce TP est d'utiliser Spring Boot avec personnellement Visual Studio Code comme environnement de développement intégré (IDE).

Initialisation du projet

Le projet a été généré grâce à l'outil intégré `Spring Initializr` de VS Code, avec les paramètres suivants :

— Type de projet : Maven
— Langage : Java
— Version de Spring Boot : 3.4.5
— Version de Java : 17
— Dépendance ajoutée : Spring Web

Le projet généré possède une structure standard, comprenant les répertoires :

— `src/main/java` : code source de l'application
— `src/main/resources` : ressources (fichiers de configuration)
— `pom.xml` : fichier de gestion de projet Maven

Création des classes principales

Deux classes principales ont été créées :

— **HellospringApplication.java** : point d'entrée de l'application Spring Boot.
— **HelloController.java** : contrôleur REST exposant deux routes HTTP.

La classe principale permet de démarrer l'application via la méthode `main` :

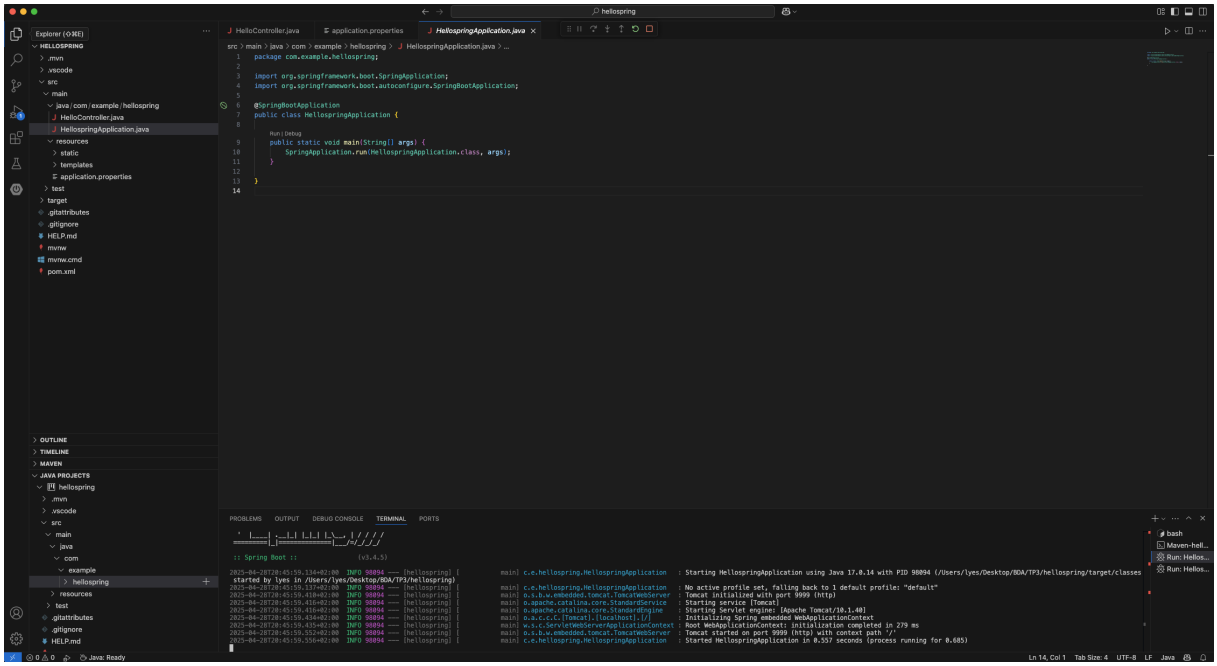

Figure 1 – Classe principale HellospringApplication.java.

Le contrôleur REST expose deux endpoints :
— `/hello` : retourne "Hello !"
— `/bonjour` : retourne "Bonjour !"

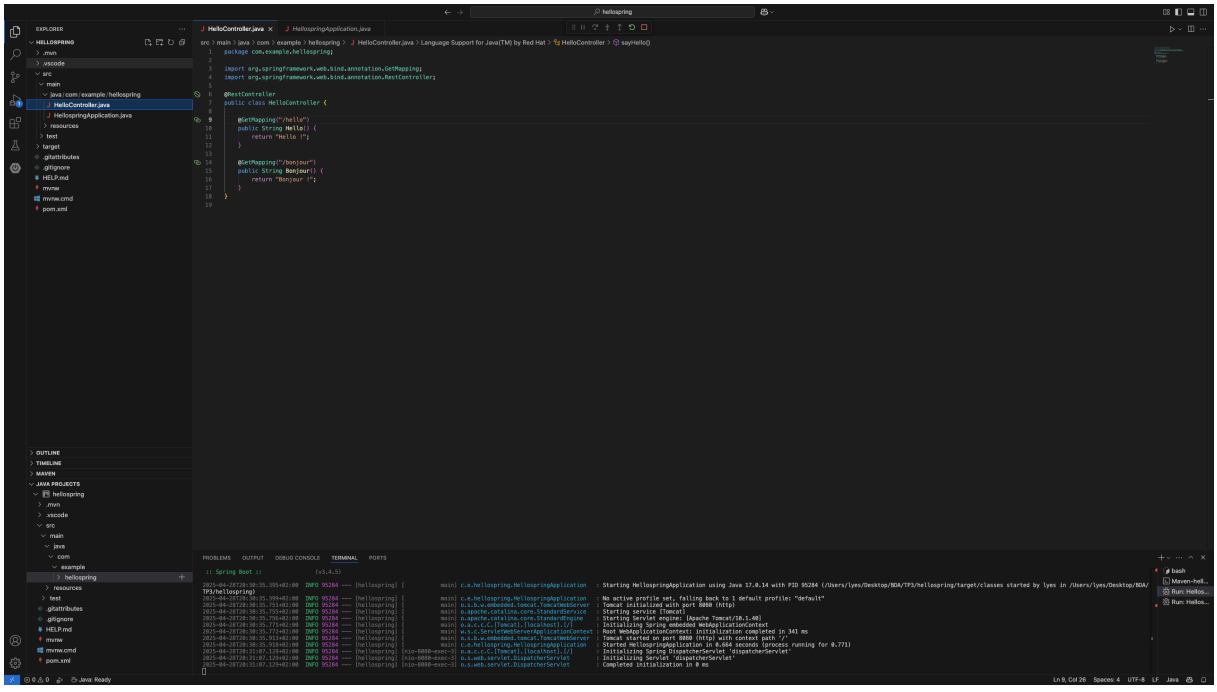

Figure 2 – Contrôleur REST HelloController.java exposant les routes /hello et /bonjour.

Après avoir lancé l'application depuis VS Code, les routes ont été testées via un navigateur web.

Test de la route /hello

L'accès à l'URL suivante :

`http://localhost:8080/hello`

retourne :

Figure 3 – Résultat du test de la route /hello.

Test de la route /bonjour

L'accès à l'URL suivante :

`http://localhost:8080/bonjour`

retourne :

Figure 4 – Résultat du test de la route /bonjour.

Les tests montrent que l'application fonctionne correctement et que les routes exposées répondent comme prévu.

Configuration du port de l'application

Par défaut, une application Spring Boot utilise le port `8080`. Dans ce projet, nous avons choisi de modifier ce port pour utiliser `9999`.

La configuration du port a été réalisée dans le fichier `application.properties` situé dans `src/main/resources`, comme montré ci-dessous :

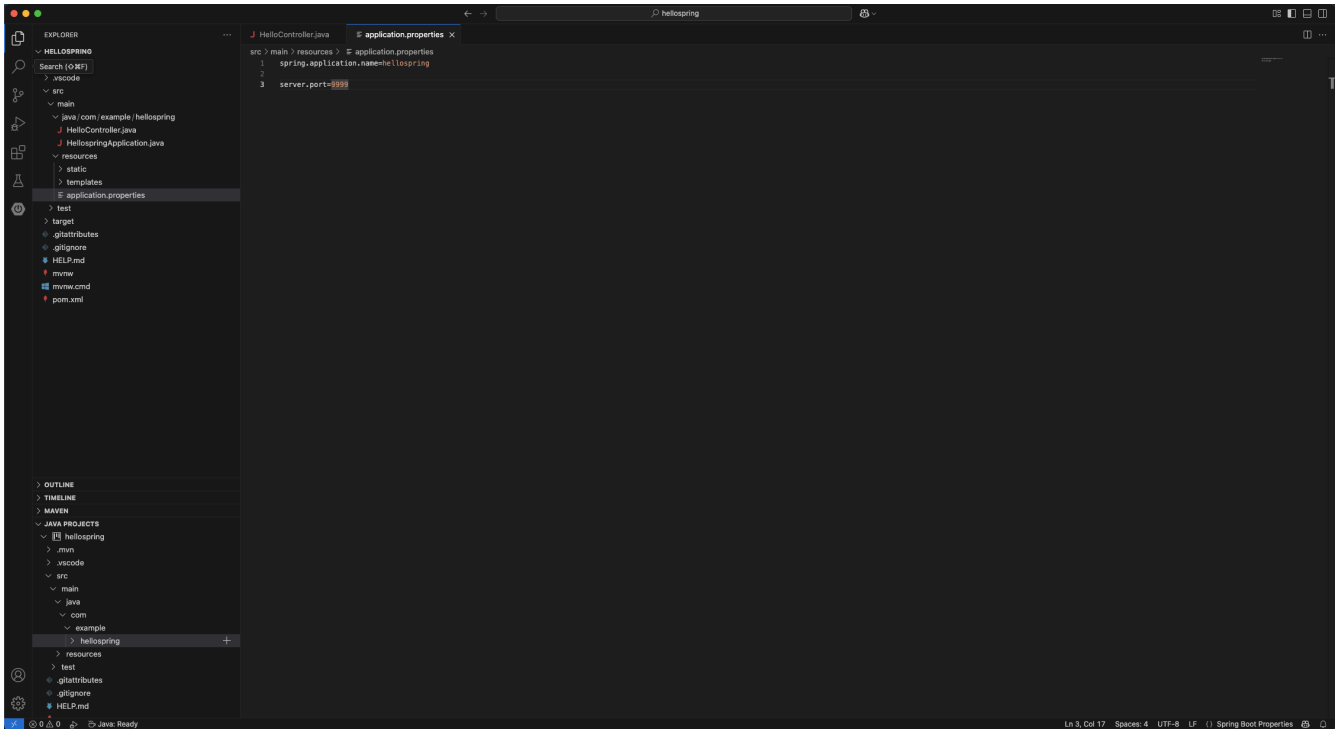

Figure 5 – Modification du port d'écoute dans `application.properties`.

La propriété `server.port` a été définie à `9999`, ce qui permet d'accéder à l'application via :

`http://localhost:9999`

Vérification de la route /hello sur le nouveau port

Après le redémarrage de l'application, la route `/hello` a été testée via le navigateur en utilisant le nouveau port `9999`. Le test confirme que l'application écoute correctement sur le port spécifié.

Figure 6 – Résultat du test de la route /hello sur le port 9999.

Ajout d'une nouvelle route : /etudiant

Une nouvelle route REST a été ajoutée dans le contrôleur pour exposer un objet étudiant. Cette route est accessible via `/etudiant` et retourne un objet JSON contenant trois champs :

- `identifiant`
- `nom`
- `moyenne`

Le code correspondant à cette nouvelle route est ajouté dans la classe `HelloController` :

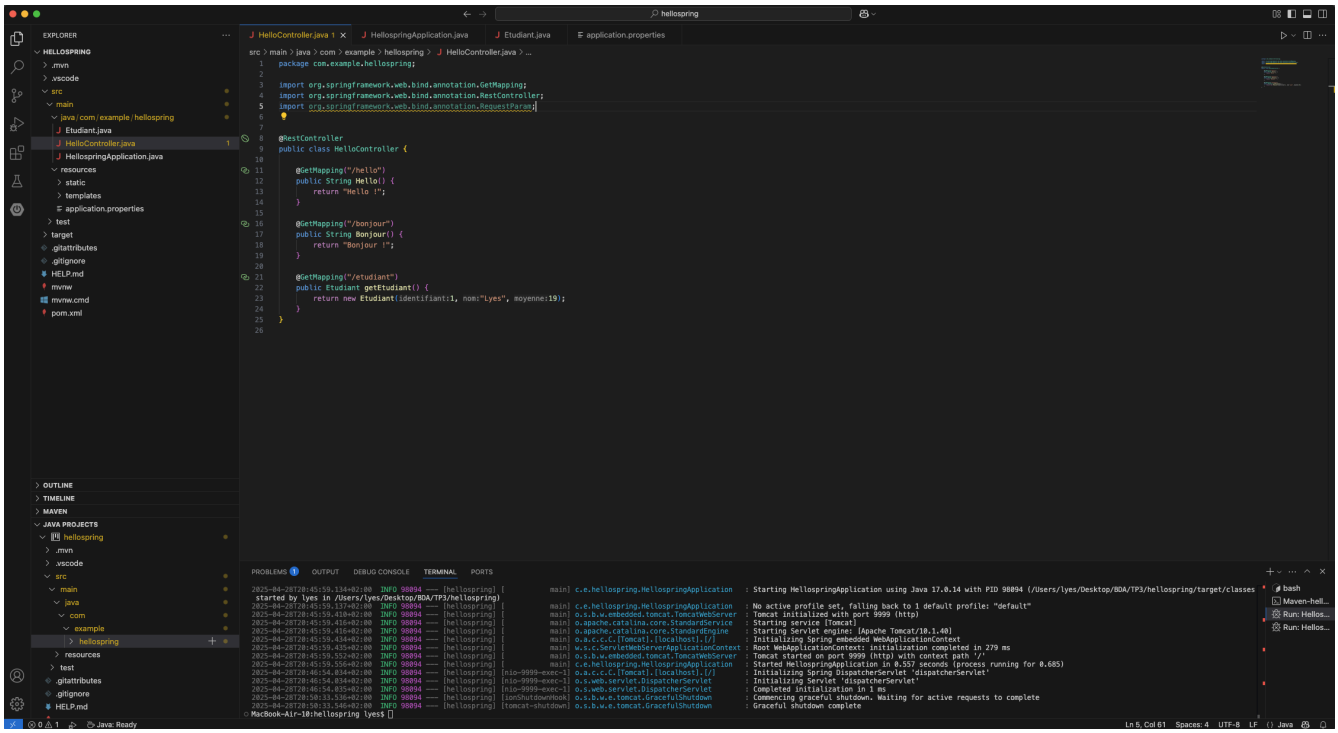

Figure 7 – Ajout de la route `/etudiant` dans `HelloController`.

Test de la route /etudiant

Après redémarrage de l'application, la nouvelle route a été testée via un navigateur web. La réponse est renvoyée comme illustré ci-dessous :

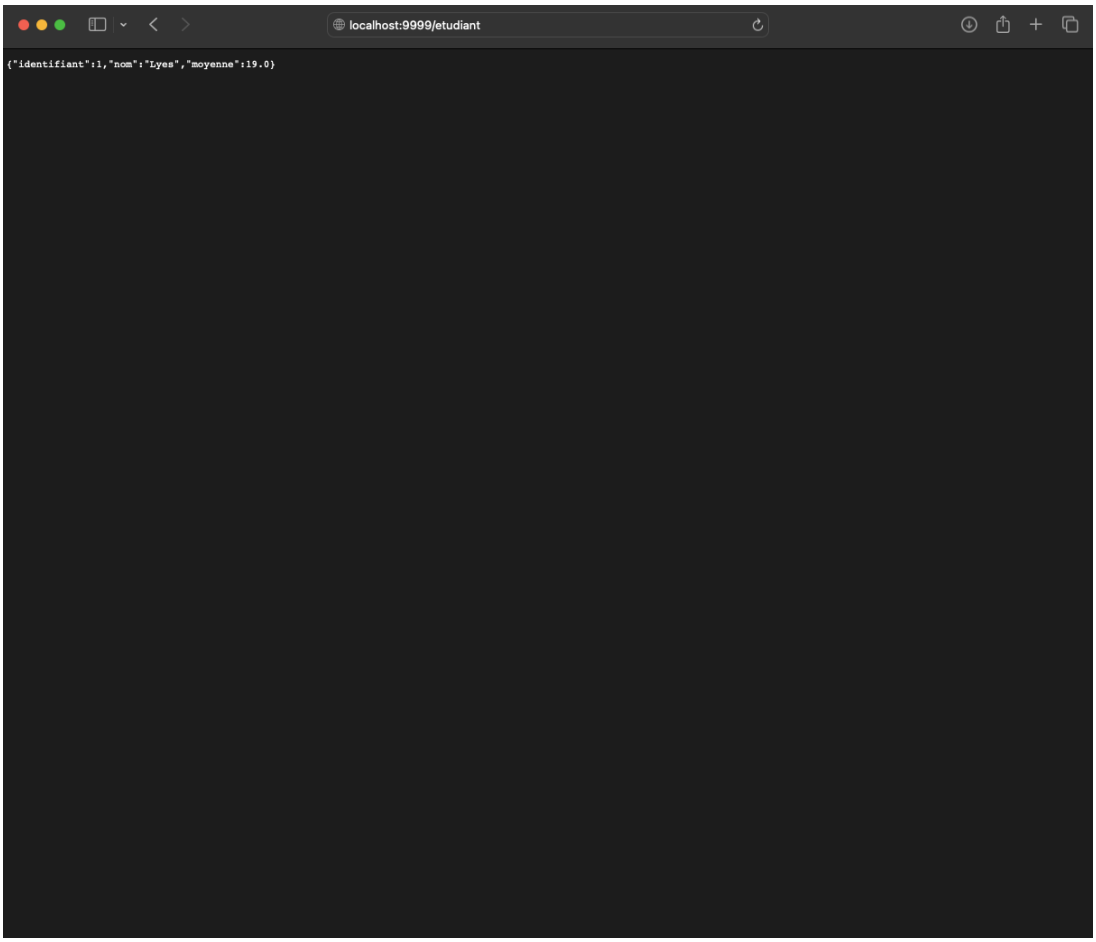

Figure 8 – Résultat du test de la route `/etudiant`.

L'affichage confirme que le service fonctionne correctement et que les données de l'étudiant sont renvoyées au format attendu.

Partie 2 :

Dans cette seconde partie, nous avons enrichi notre application Spring Boot en créant une nouvelle route permettant d'effectuer une opération arithmétique simple.

Ajout de la route /somme

La méthode suivante a été ajoutée au contrôleur `HelloController` :
— `/somme` : accepte deux paramètres `a` et `b` et retourne leur somme.
Afin de tester la nouvelle route, nous avons utilisé l'outil **Postman**.

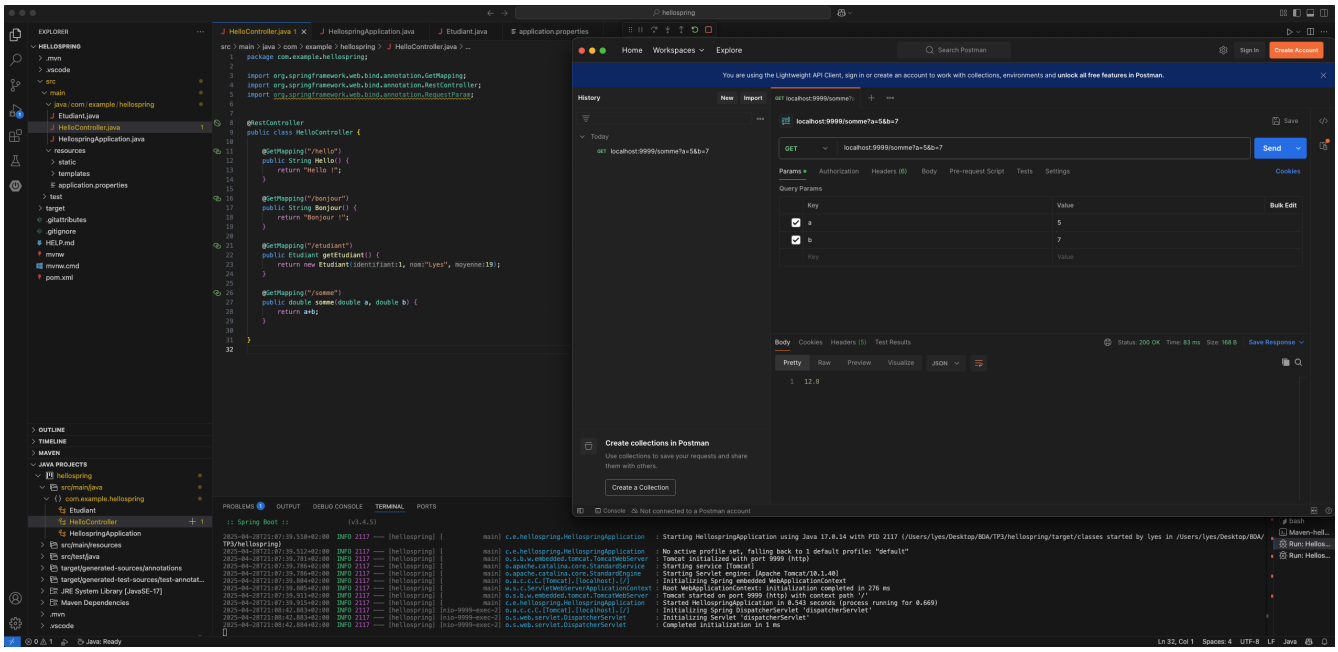

Figure 9 – Ajout de la méthode `/somme` dans `HelloController`.

La route fonctionne correctement et renvoie bien le résultat attendu.

Partie 3 :

Dans cette partie, nous avons enrichi l'application afin de manipuler une collection d'étudiants et de la retourner sous forme de liste JSON.

Ajout de la liste des étudiants

Une collection statique `liste` a été ajoutée dans la classe `HelloController`. Elle est initialisée avec plusieurs objets de type `Etudiant`, chacun ayant un identifiant, un nom et une moyenne.

Une nouvelle route REST a été créée pour permettre de récupérer l'ensemble de la collection :

— `/liste` : retourne tous les étudiants sous forme de liste JSON.

La route `/liste` a été testée à l'aide de Postman. La réponse obtenue est une liste JSON contenant tous les étudiants :

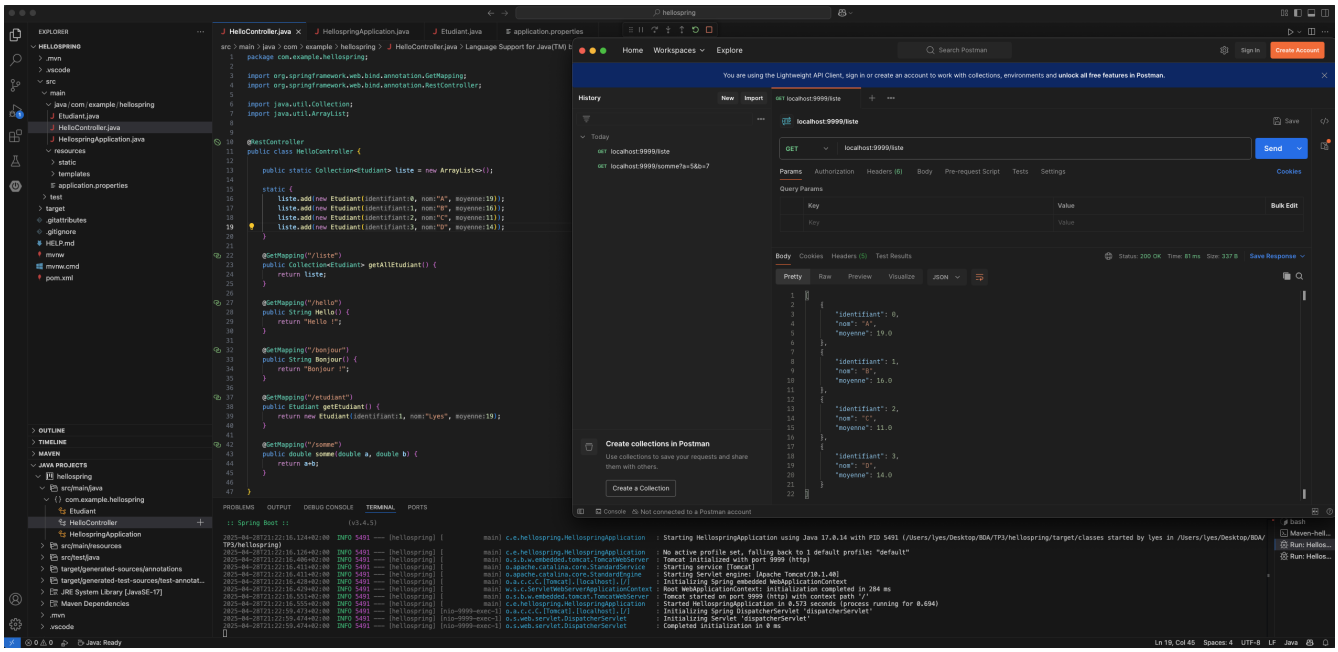

Figure 10 – Résultat du test de la route `/liste` via Postman.

La réponse est correcte : chaque étudiant est affiché sous forme d'un objet JSON avec ses trois attributs.

Partie 4 :

Dans cette dernière partie, nous avons enrichi notre application avec la possibilité :
— De récupérer un étudiant spécifique via son identifiant.
— D'ajouter dynamiquement un nouvel étudiant à la collection existante.

Route /getEtudiant

La route `/getEtudiant` permet de récupérer un étudiant de la liste en fonction de son identifiant.

Testé via Postman, nous avons envoyé la requête suivante :
— Méthode : `GET`
— URL : `http://localhost:9999/getEtudiant?identifiant=2`
La réponse reçue :
— `{"identifiant":2, "nom":"C", "moyenne":11.0}`

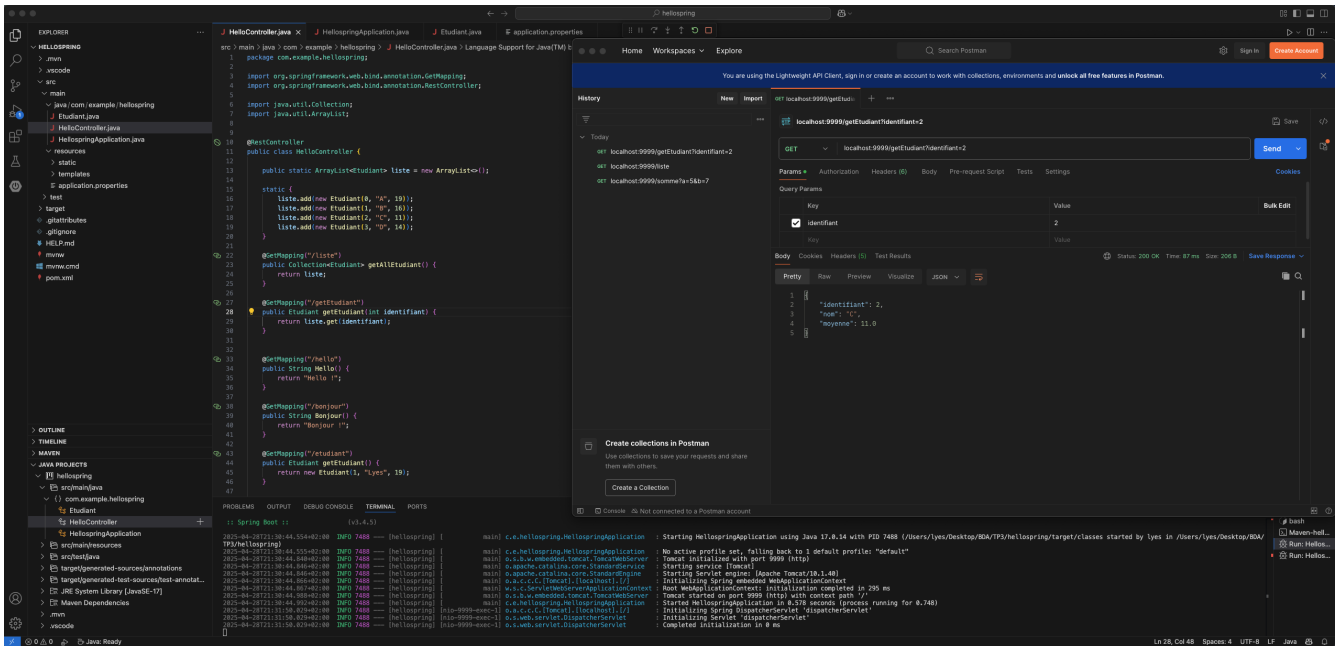

Figure 11 – Test de récupération d'un étudiant spécifique.

Route /addEtudiant

La route `/addEtudiant` permet d'ajouter un nouvel étudiant à la collection en passant ses attributs.

Le test avec Postman est réalisé avec la requête suivante :

— Méthode : `POST`
— URL : `http://localhost:9999/addEtudiant`
— Paramètres :
  — `identifiant = 4`
  — `nom = E`
  — `moyenne = 12`

La réponse reçue :
— `{"identifiant":4, "nom":"E", "moyenne":12.0}`

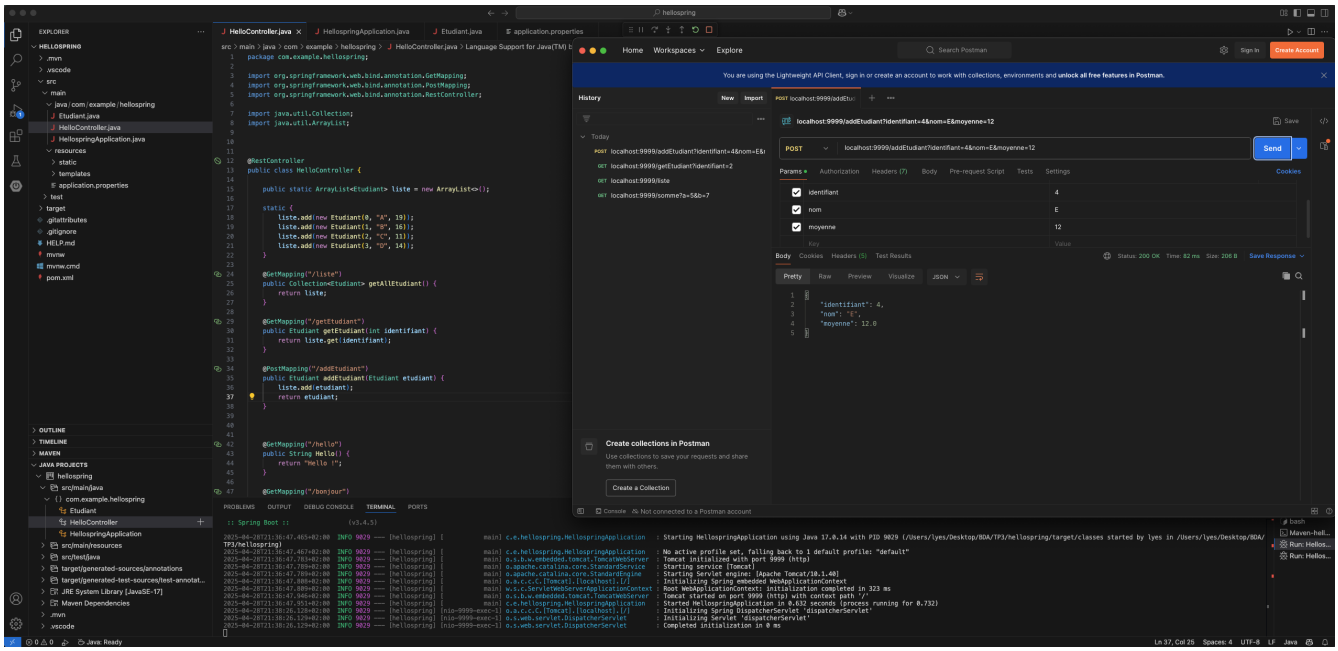

Figure 12 – Test d'ajout dynamique d'un étudiant via Postman.

Ainsi, notre application Spring Boot devient totalement interactive.

Partie 5 :

Dans cette partie, deux nouvelles fonctionnalités sont ajoutées :
— **Suppression** d'un étudiant par son identifiant via une requête HTTP `DELETE`.
— **Modification** du nom d'un étudiant existant via une requête HTTP `PUT`.

Suppression d'un étudiant

La suppression se fait avec l'annotation `@DeleteMapping`, en passant l'identifiant de l'étudiant à supprimer.

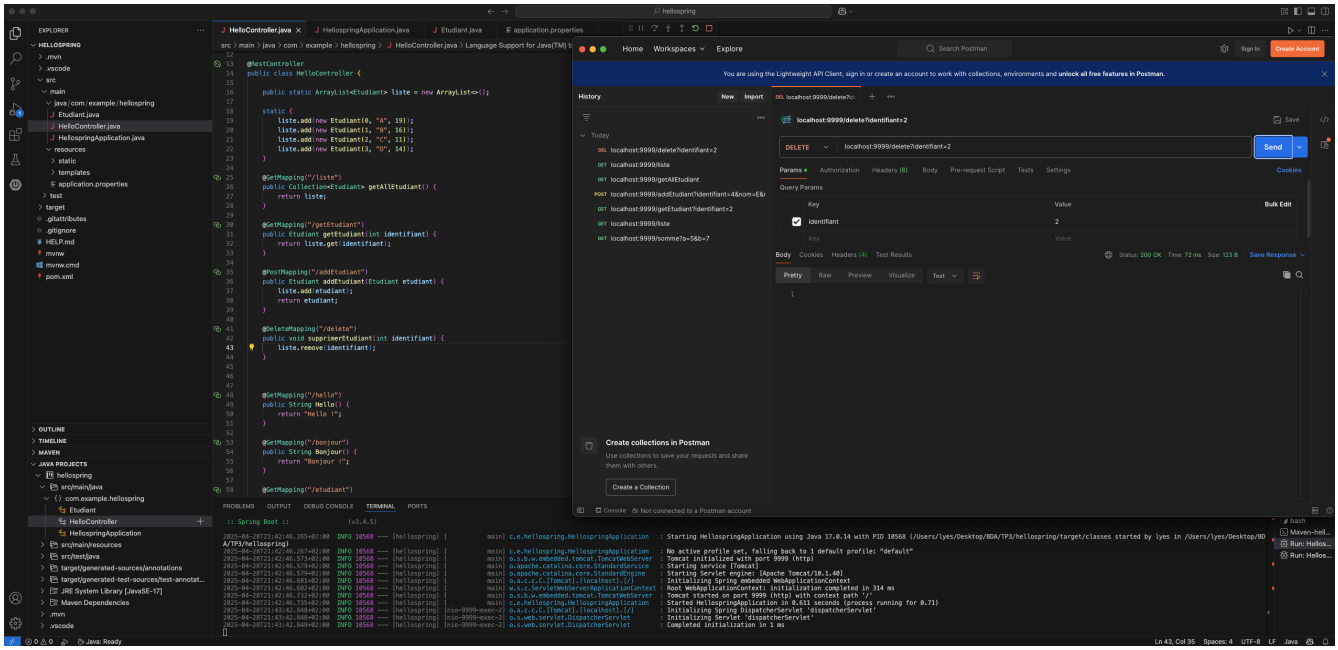

Figure 13 – Suppression d'un étudiant à l'aide de Postman (identifiant = 2)

Modification d'un étudiant

La modification utilise l'annotation `@PutMapping`. On modifie ici uniquement le `nom` de l'étudiant spécifié par son identifiant.

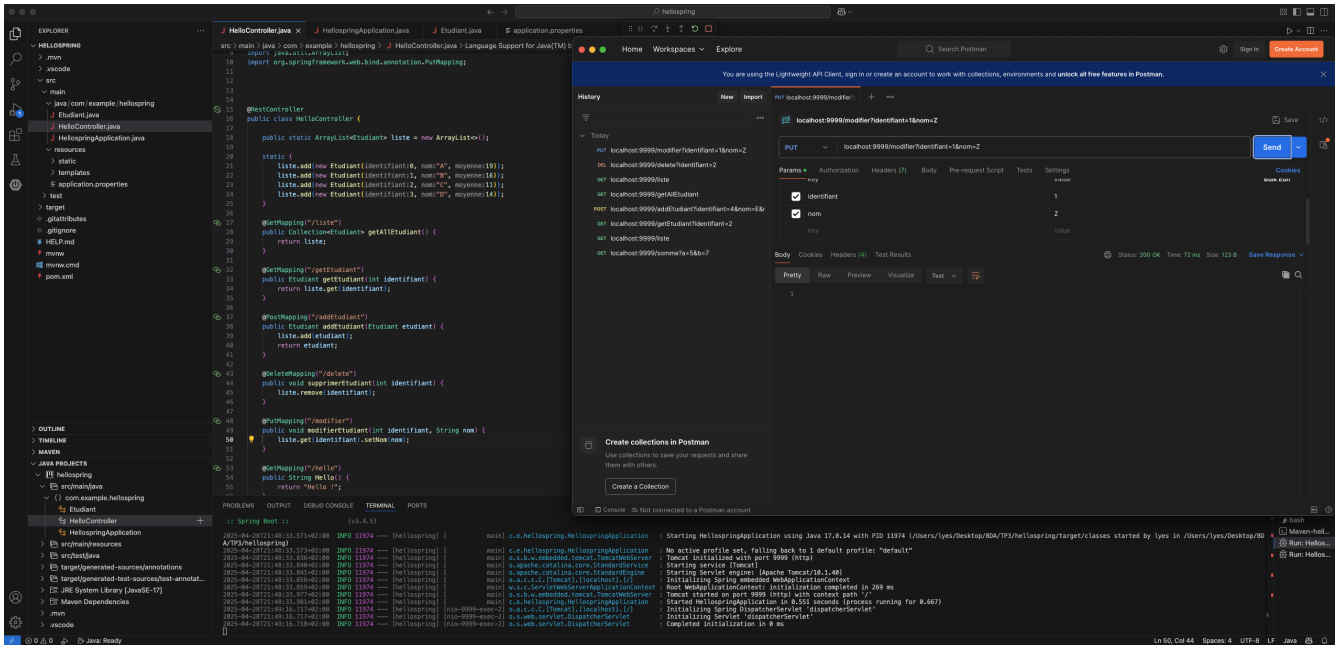

Figure 14 – Modification du nom d'un étudiant à l'aide de Postman (identifiant = 1)

Partie 6 :

Dans cette dernière partie, nous avons mis en place une application Spring Boot connectée à une base de données H2.

Nous avons d'abord créé une classe `Adherent` annotée avec `@Entity`, représentant un adhérent avec plusieurs attributs : un identifiant, un nom, une ville et un âge. Cette classe est accompagnée des getters et setters nécessaires à la manipulation des données.

Ensuite, nous avons défini une interface `AdherentRepository` qui hérite de `JpaRepository`. Cela permet d'exploiter toutes les méthodes de base pour gérer les entités sans écrire de code SQL.

La configuration de la base de données est effectuée dans le fichier `application.properties`, où :

— Le nom de l'application est défini par `spring.application.name`.
— Le port du serveur est modifié en `9191`.
— La console H2 est activée grâce à la propriété `spring.h2.console.enabled=true`.

Dans la classe principale `DemoH2Application`, nous avons ajouté un `CommandLineRunner` pour insérer automatiquement quelques adhérents à l'initialisation de l'application. Chaque adhérent est sauvegardé dans la base en utilisant la méthode `save` du repository.

Après démarrage de l'application, nous avons pu accéder à la console H2 et vérifier la connexion à la base de données :

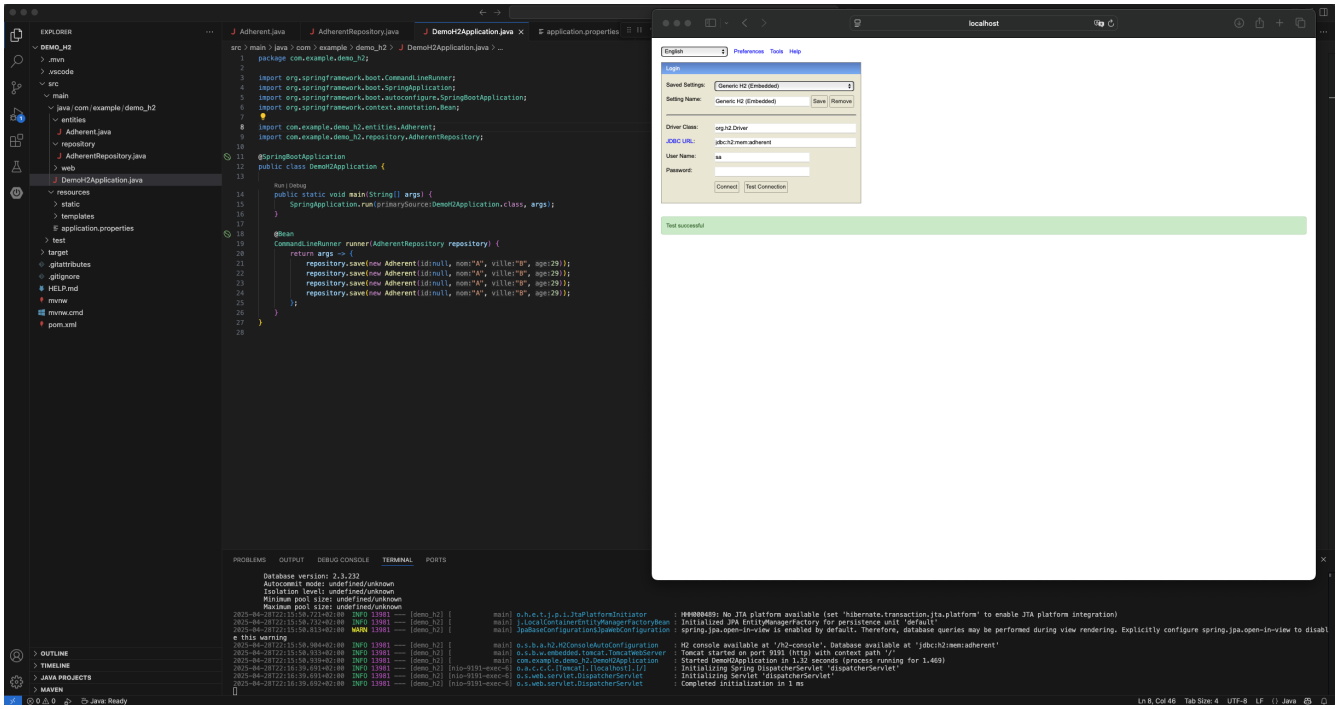

Figure 15 – Connexion réussie à la console H2

Enfin, nous avons exécuté une requête SQL `SELECT * FROM ADHERENT` pour afficher le contenu de la table :

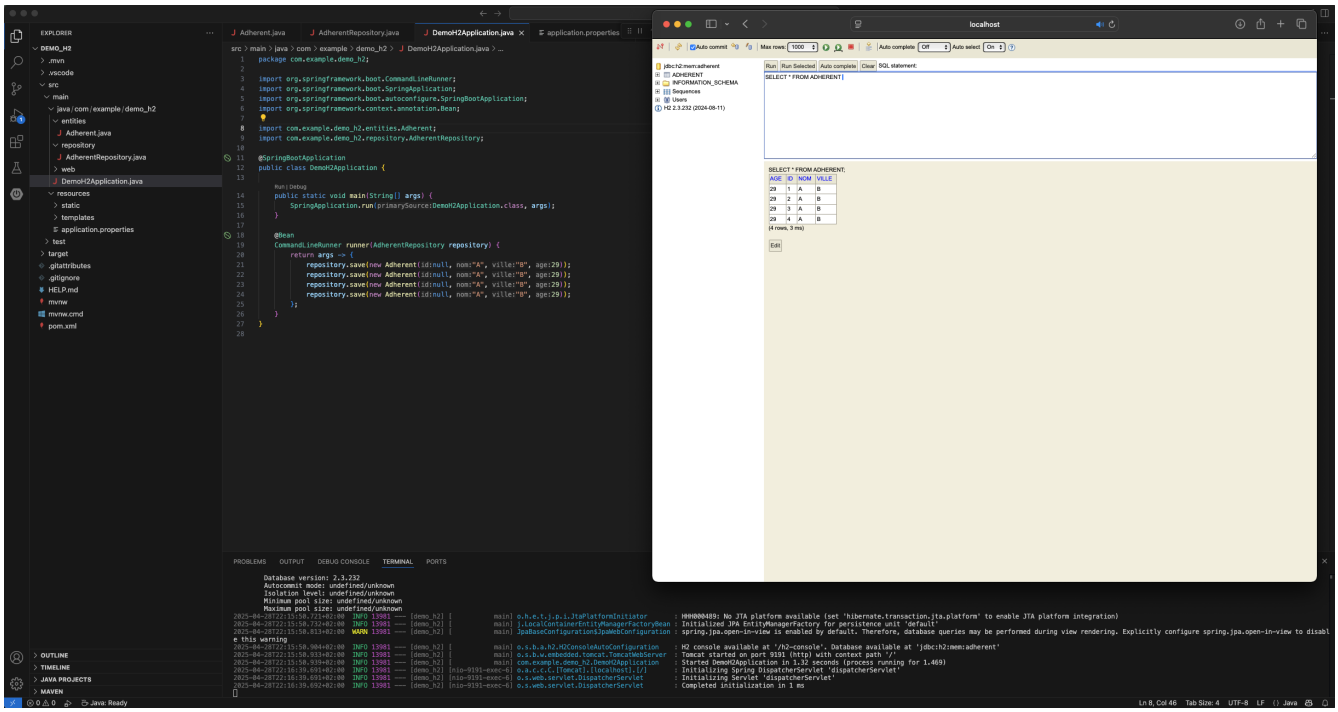

Figure 16 – Affichage des données insérées dans la table ADHERENT