

TP n° 3 : Micro-service, Hibernate, Base de données in memory (H2)

Nombre de pages : 17

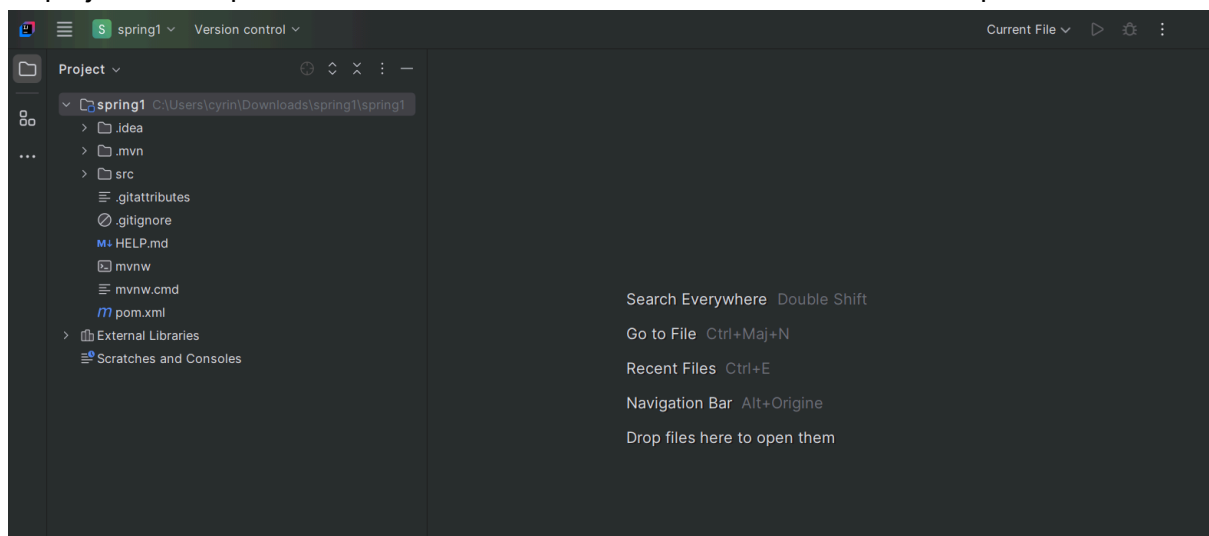
Partie 1 : Création du projet

Le projet a été généré via Spring Initializr avec les configurations suivantes :

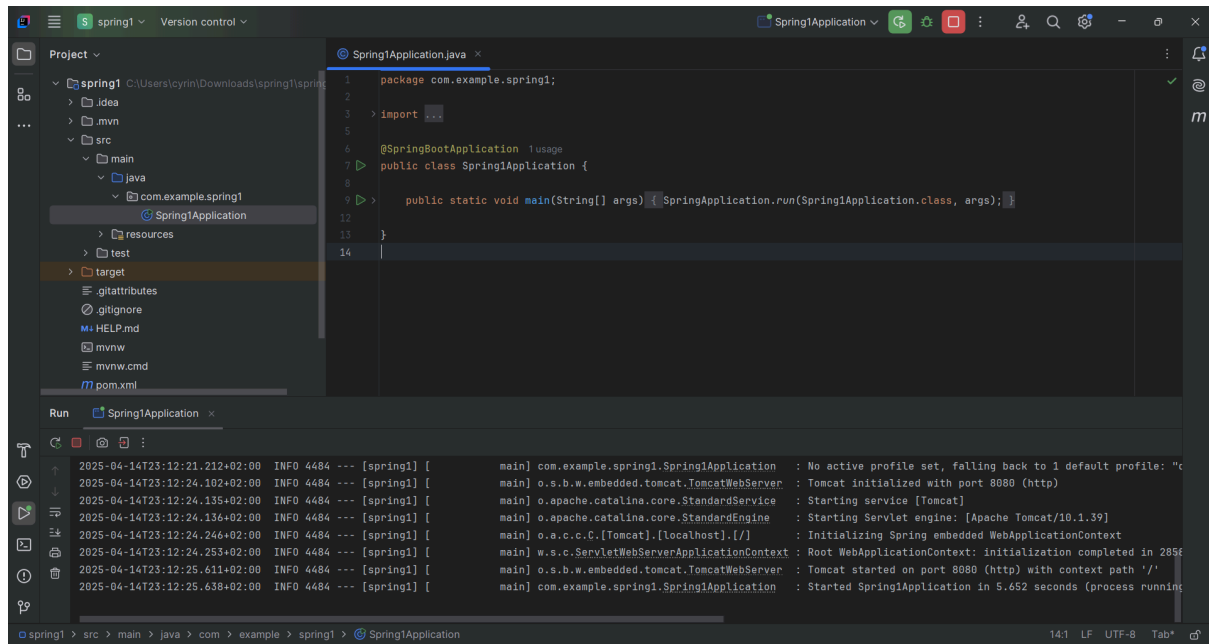
- Language : Java
- Type de projet : Maven
- Dépendances : Spring Web (pour la création d'API REST)

The screenshot shows the Spring Initializr web interface. The 'Project' section has 'Gradle - Groovy' and 'Gradle - Kotlin' selected. The 'Language' section has 'Java' selected. The 'Spring Boot' section has '3.5.0 (M3)' selected. The 'Project Metadata' section has 'Group' as 'com.example', 'Artifact' as 'spring1', 'Name' as 'spring1', and 'Description' as 'Spring 1'. The 'Dependencies' section has 'Spring Web' selected. The 'Package name' is 'com.example.spring1'. The 'Packaging' is 'Jar'. The 'Java' version is '17'. The 'GENERATE' button is highlighted.

Le projet a été importé avec succès dans IntelliJ IDEA en utilisant le fichier pom.xml.



⇒ Pour exécuter : Ouvrez Spring1Application.java > Cliquez sur le bouton Run



The screenshot shows an IDE with the Spring1Application.java file open. The code is as follows:

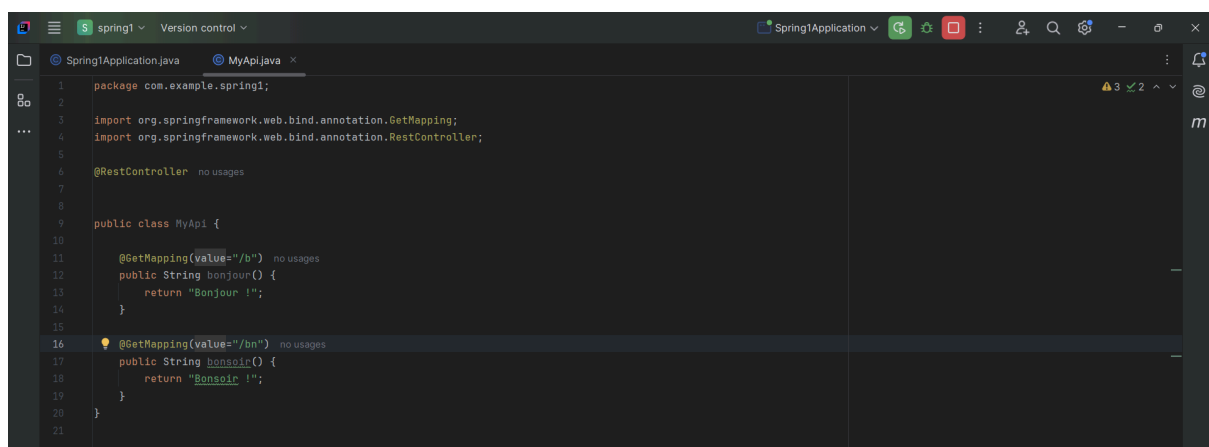
```
1 package com.example.spring1;
2
3 import org.springframework.boot.SpringApplication;
4
5 @SpringBootApplication
6 public class Spring1Application {
7
8     public static void main(String[] args) {
9         SpringApplication.run(Spring1Application.class, args);
10    }
11 }
```

The Run console shows the following output:

```
2025-04-14T23:12:21.212+02:00 INFO 4484 --- [spring1] [main] com.example.spring1.Spring1Application : No active profile set, falling back to 1 default profile: "c
2025-04-14T23:12:24.102+02:00 INFO 4484 --- [spring1] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8080 (http)
2025-04-14T23:12:24.135+02:00 INFO 4484 --- [spring1] [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2025-04-14T23:12:24.136+02:00 INFO 4484 --- [spring1] [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.39]
2025-04-14T23:12:24.246+02:00 INFO 4484 --- [spring1] [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2025-04-14T23:12:24.253+02:00 INFO 4484 --- [spring1] [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 285f
2025-04-14T23:12:25.411+02:00 INFO 4484 --- [spring1] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path '/'
2025-04-14T23:12:25.638+02:00 INFO 4484 --- [spring1] [main] com.example.spring1.Spring1Application : Started Spring1Application in 5.652 seconds (process running)
```

Deux endpoints ont été implémentés :

- Méthode : GET/b
- Méthode : GET/bn

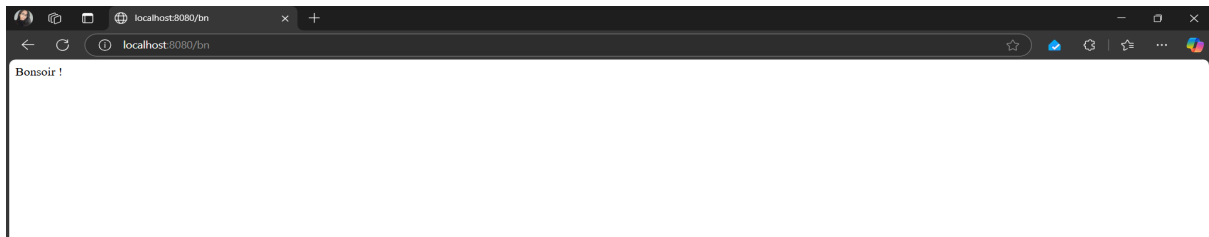


The screenshot shows the MyApi.java file in the IDE. The code is as follows:

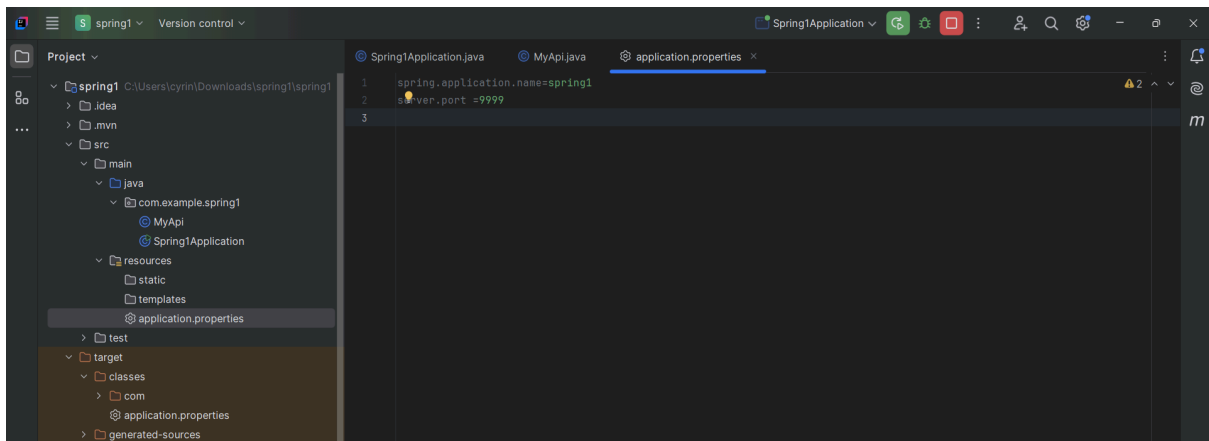
```
1 package com.example.spring1;
2
3 import org.springframework.web.bind.annotation.GetMapping;
4 import org.springframework.web.bind.annotation.RestController;
5
6 @RestController
7
8 public class MyApi {
9
10     @GetMapping(value="/b")
11     public String bonjour() {
12         return "Bonjour !";
13     }
14
15     @GetMapping(value="/bn")
16     public String bonsoir() {
17         return "Bonsoir !";
18     }
19 }
20
21
```

On teste l'implémentation de ces deux endpoints avec le navigateur après l'exécution du projet :

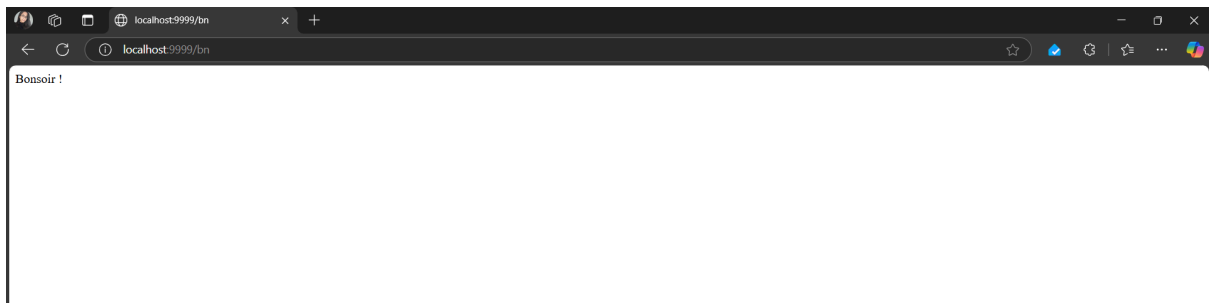




Pour changer le port d'écoute du serveur Spring Boot (par défaut 8080), on modifie le fichier `application.properties` :



Après redémarrage du projet Spring Boot, le port 9999 a bien été pris en compte :



Implémentation de la classe Etudiant avec :

- Attributs : identifiant, nom, moyenne.
- Constructeurs : Un constructeur par défaut et un constructeur paramétré.
- Getters/Setters : Pour accéder et modifier les champs.

```

1 package com.example.spring1;
2
3 public class Etudiant { no usages
4     private int identifiant; 3 usages
5     private String nom; 3 usages
6     private double moyenne; 3 usages
7
8     public Etudiant() { no usages
9     }
10
11     public Etudiant(int identifiant, String nom, double moyenne) { no usages
12         this.identifiant = identifiant;
13         this.nom = nom;
14         this.moyenne = moyenne;
15     }
16
17     public int getIdentifiant() { no usages
18         return identifiant;
19     }
20
21     public void setIdentifiant(int identifiant) { no usages
22         this.identifiant = identifiant;
23     }
24
25     public String getNom() { no usages
26         return nom;
27     }
28
29     public void setNom(String nom) { no usages
30         this.nom = nom;
31     }
32 }

```

Endpoint pour retourner un nouvel étudiant avec les arguments fournis :

```

1 package com.example.spring1;
2
3 import org.springframework.web.bind.annotation.GetMapping;
4 import org.springframework.web.bind.annotation.RestController;
5
6 @RestController no usages
7
8 public class MyApi {
9
10     @GetMapping(value="/h") no usages
11     public String bonjour() {
12         return "Bonjour !";
13     }
14
15     @GetMapping(value="/bn") no usages
16     public String bonsoir() {
17         return "Bonsoir !";
18     }
19
20     @GetMapping(value="/etudiant") no usages
21     public Etudiant getEtudiant() {
22         return new Etudiant(1, "A", 19);
23     }
24 }

```

Test de cet endpoint :

```

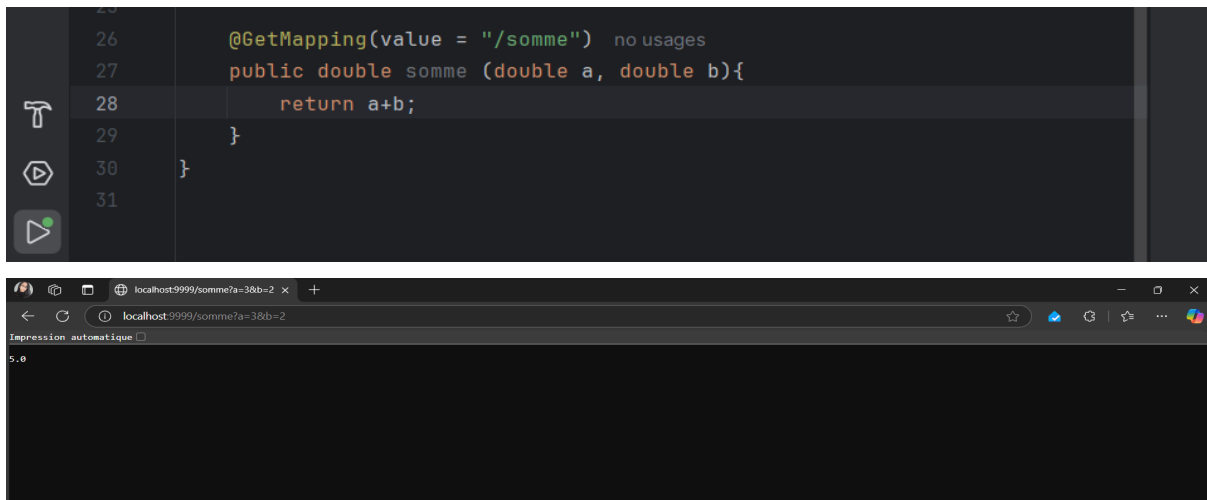
{
  "identifiant": 1,
  "nom": "A",
  "moyenne": 19
}

```

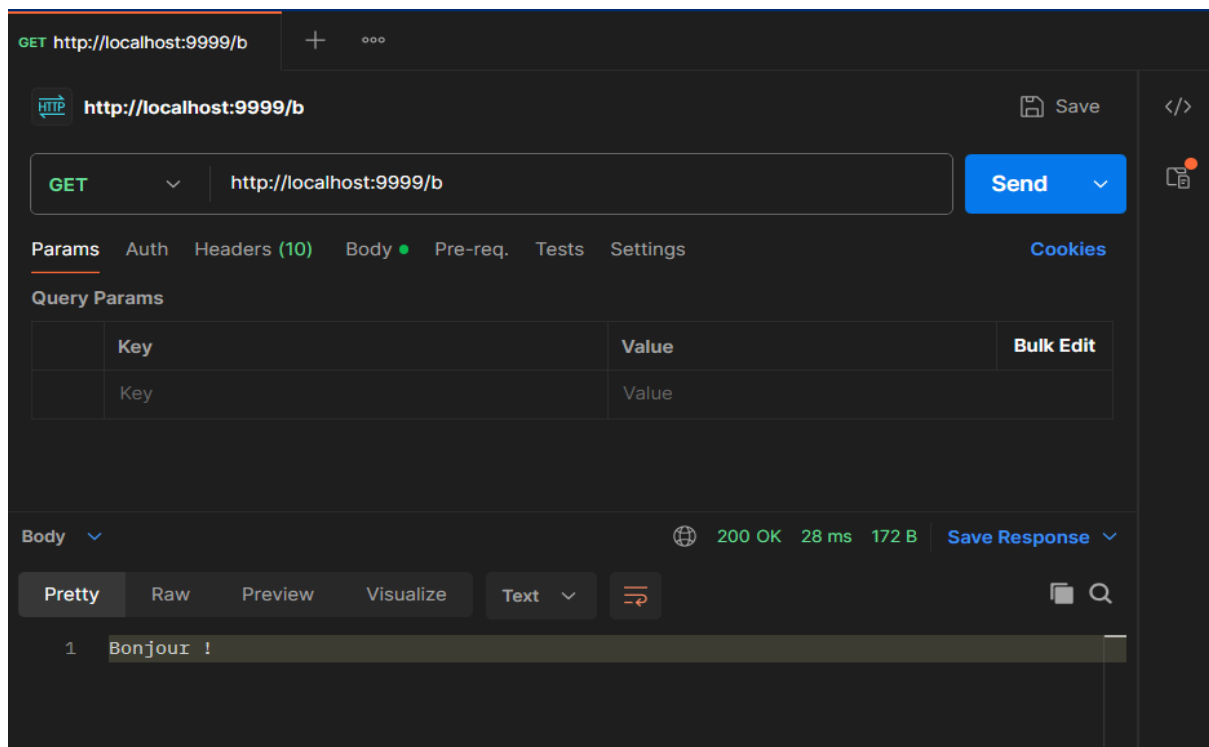
Partie 2 :

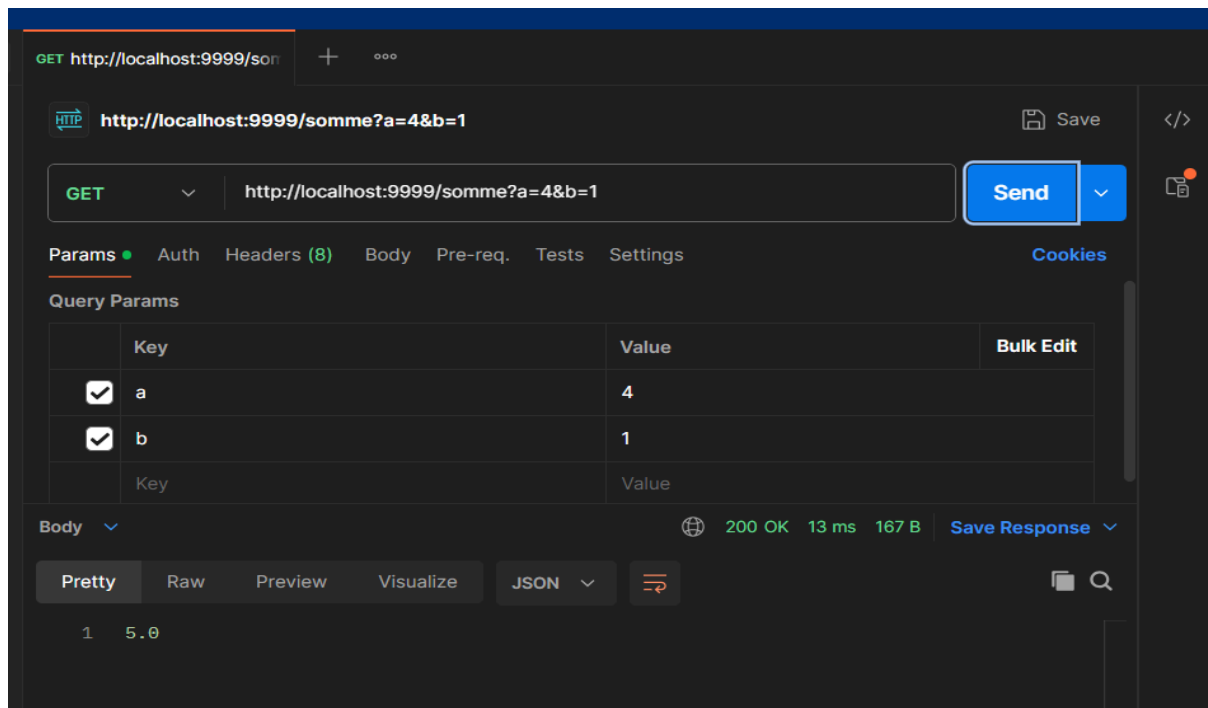
Endpoint de calcul :

- Méthode : GET /somme
- Fonctionnalité : Effectue la somme de deux nombres (a et b).



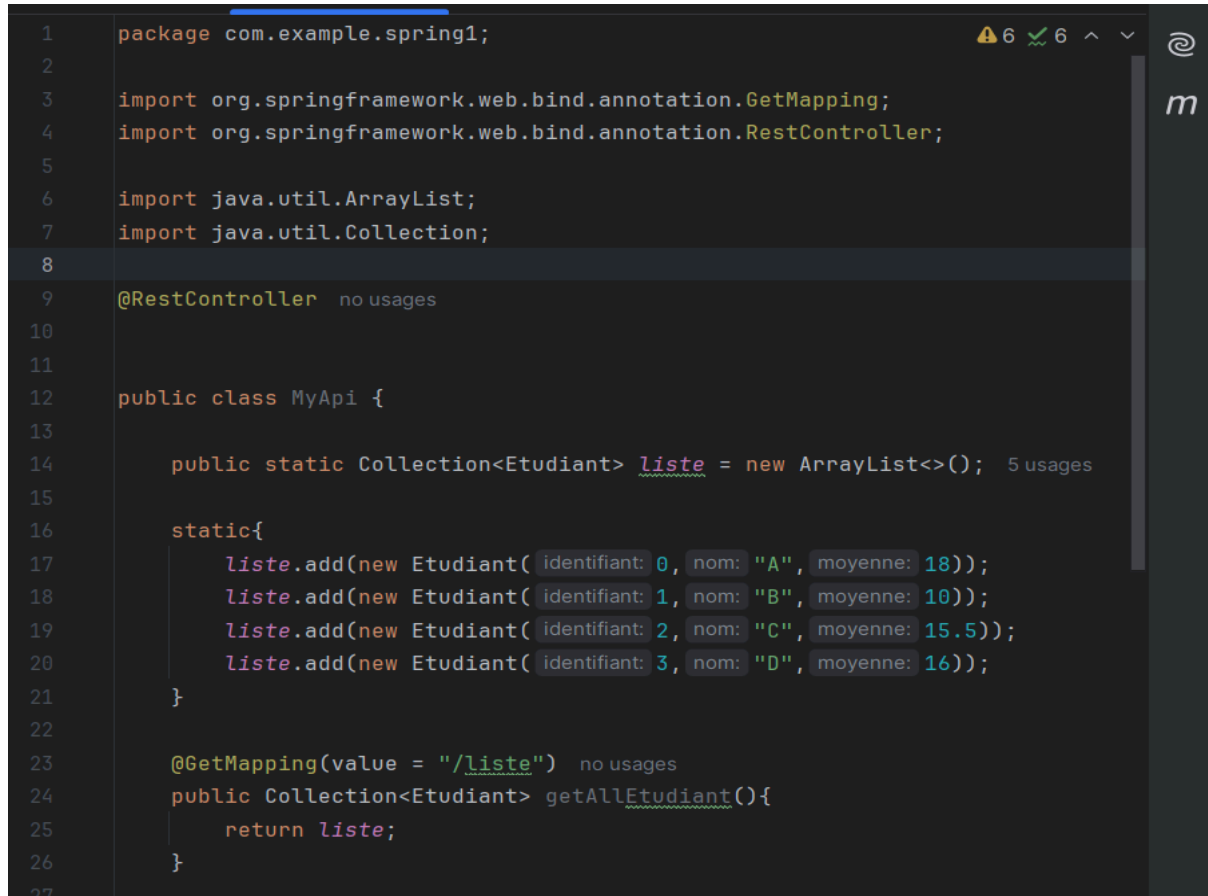
Pour vérifier le bon fonctionnement de l'API, on peut tester les endpoints avec Postman. Voici les résultats :



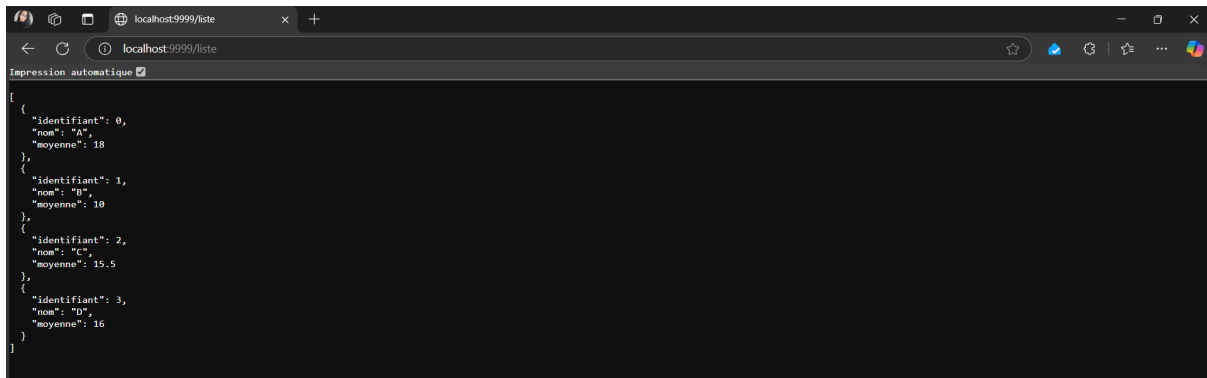


Partie 3 : Collection statique d'étudiants

- Création d'une collection statique dans la classe MyApi pour stocker les étudiants.
- Ajout Statique d'Étudiants : J'ai peuplé la liste avec des données initiales en utilisant un bloc statique pour garantir l'initialisation au démarrage de l'application.

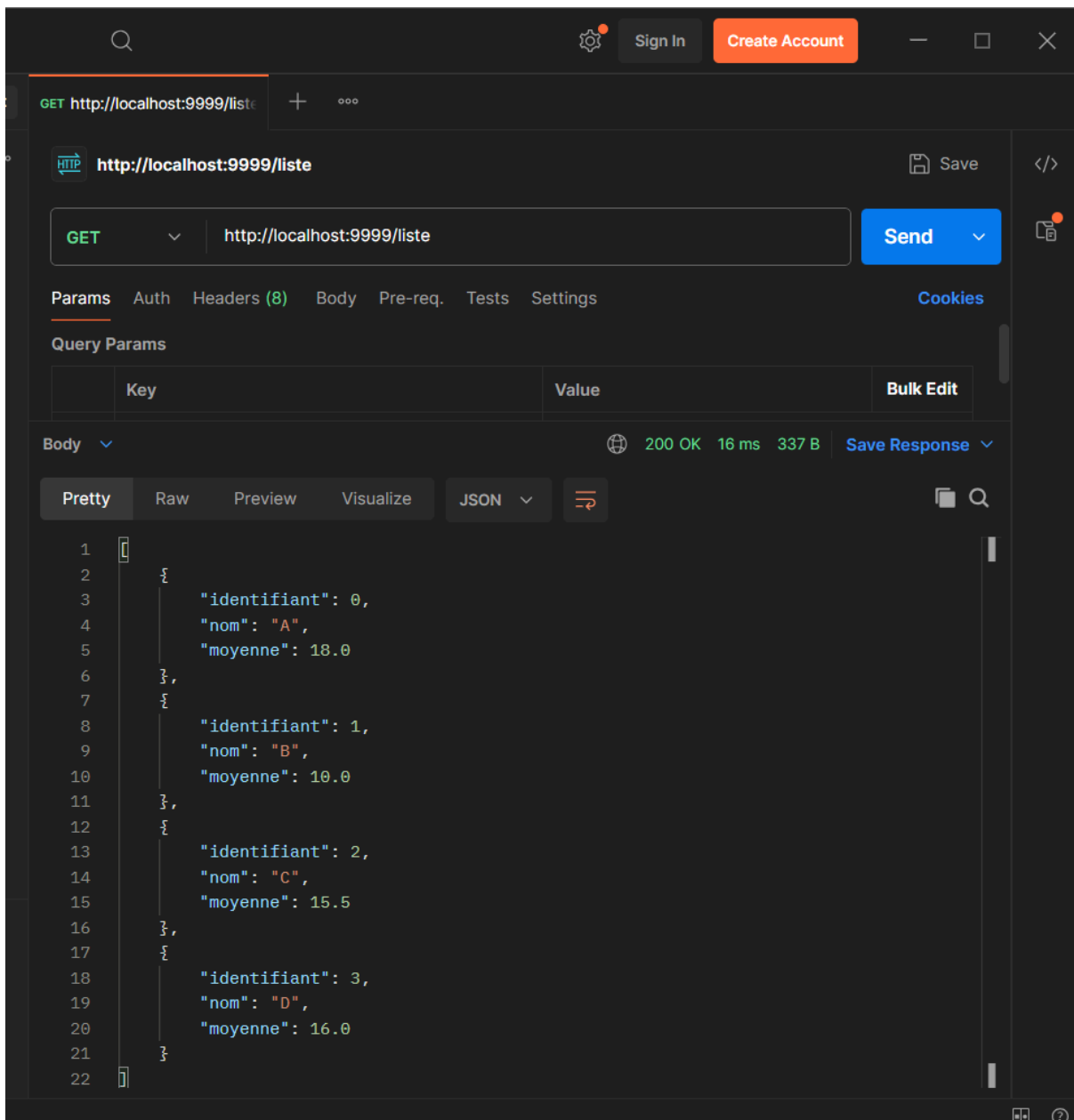


Test de l'ajout des étudiants :



```
[
  {
    "identifiant": 0,
    "nom": "A",
    "moyenne": 18
  },
  {
    "identifiant": 1,
    "nom": "B",
    "moyenne": 10
  },
  {
    "identifiant": 2,
    "nom": "C",
    "moyenne": 15.5
  },
  {
    "identifiant": 3,
    "nom": "D",
    "moyenne": 16
  }
]
```

Même test avec Postman :



Partie 4 : Méthodes GET et POST

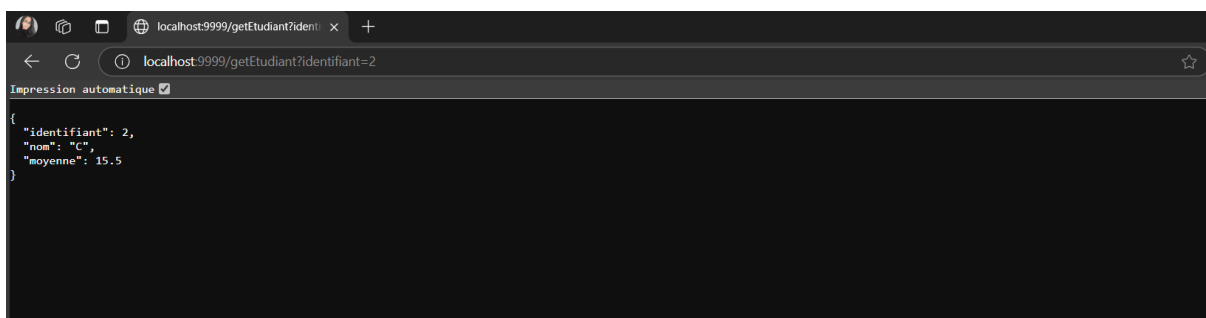
Implémentation de la méthode GET `getEtudiant` pour rechercher un étudiant par son identifiant dans la liste statique.

La méthode doit :

- Prendre un paramètre `identifiant`
- Parcourir la liste des étudiants
- Retourner l'étudiant correspondant ou null si non trouvé

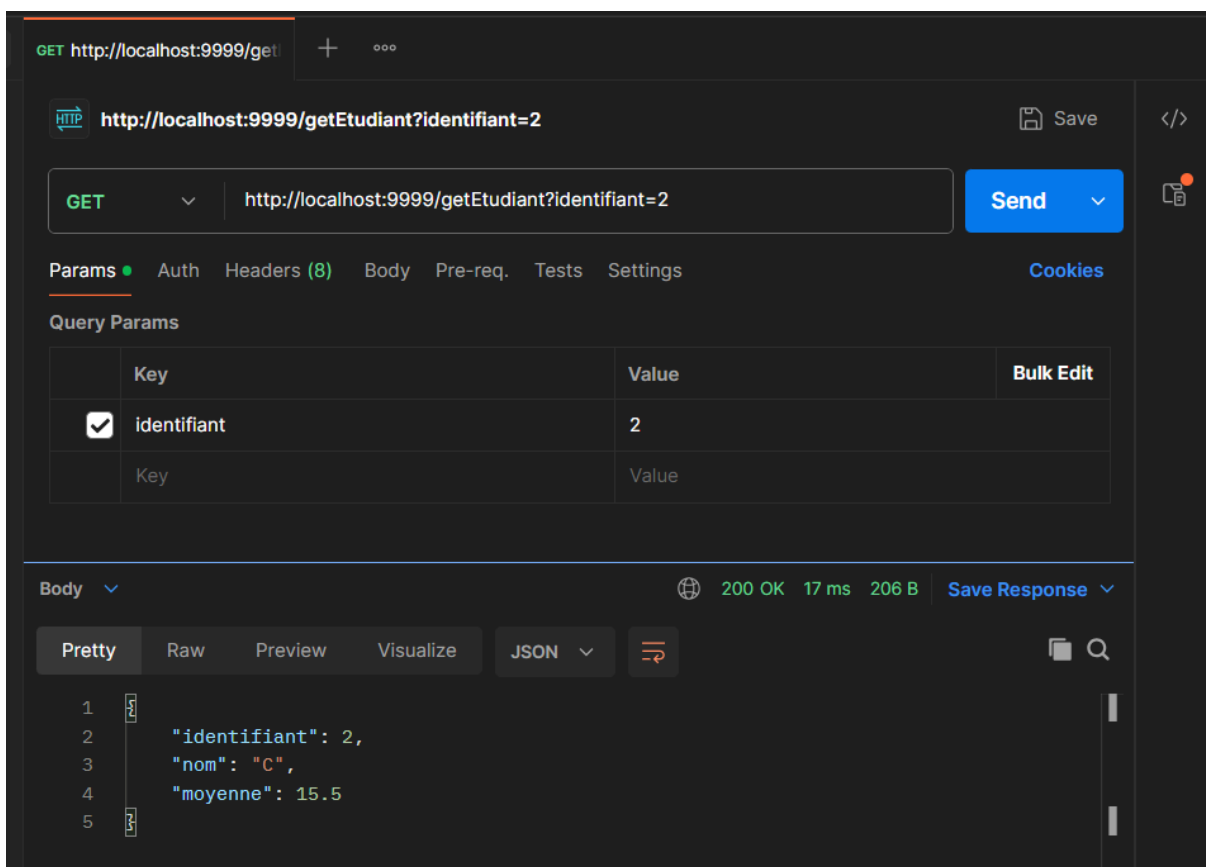
```
28     @GetMapping(value = "/getEtudiant") no usages
29     public Etudiant getEtudiant(int identifiant){
30         return liste.get(identifiant);
31     }
32 }
```

Tests de la méthode :



localhost:9999/getEtudiant?identifiant=2

```
{
  "identifiant": 2,
  "nom": "C",
  "moyenne": 15.5
}
```



GET http://localhost:9999/getEtudiant?identifiant=2

Send

Params Auth Headers (8) Body Pre-req. Tests Settings Cookies

Query Params

	Key	Value	Bulk Edit
<input checked="" type="checkbox"/>	identifiant	2	
	Key	Value	

Body

200 OK 17 ms 206 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "identifiant": 2,
3   "nom": "C",
4   "moyenne": 15.5
5 }
```


Implémentation de la méthode POST addEtudiant pour ajouter un étudiant :

```
33  
34  @PostMapping(value = "/addEtudiant") no usages  
35  public Etudiant addEtudiant(Etudiant etudiant){  
36      liste.add(etudiant);  
37      return etudiant;  
38  }  
39  
40
```

Test de la méthode POST avec Postman :

The screenshot shows the Postman interface for a POST request. The URL bar displays `http://localhost:9999/addEtudiant?identifiant=4&nom=E&moyenne=14`. The request is configured with the following query parameters:

Key	Value
identifiant	4
nom	E
moyenne	14

The response status is 200 OK, with a response time of 389 ms and a body size of 206 B. The response body is displayed in JSON format:

```
{  
  "identifiant": 4,  
  "nom": "E",  
  "moyenne": 14.0  
}
```

Partie 5 : Méthodes DELETE et PUT

Implémentation de la méthode DELETE delete pour supprimer un étudiant à partir de son identifiant :

```

41      @DeleteMapping(value = "/delete") no usages
42      public void supprimerEtudiant (int identifiant){
43          liste.remove(identifiant);
44      }
45

```

Test de la méthode DELETE :

The screenshot shows a REST client interface with the following details:

- URL:** `http://localhost:9999/delete?identifiant=2`
- Method:** `DELETE`
- Query Params:**

Key	Value
identifiant	2
- Response:**
 - Status: `200 OK`
 - Time: `316 ms`
 - Size: `123 B`

Vérification de la suppression :

The screenshot shows a REST client interface with the following details:

- URL:** `http://localhost:9999/liste`
- Method:** `GET`
- Response:**
 - Status: `200 OK`
 - Time: `39 ms`
 - Size: `294 B`
- Body (JSON):**

```

{
  "identifiant": 0,
  "nom": "A",
  "moyenne": 18.0
},
{
  "identifiant": 1,
  "nom": "B",
  "moyenne": 10.0
},
{
  "identifiant": 3,
  "nom": "D",
  "moyenne": 16.0
}

```

Implémentation de la méthode PUT update pour modifier un étudiant à partir de son identifiant :

```
43 @PutMapping(value="/update") no usages
44 public Etudiant modifierEtudiant(int identifiant,String nom){
45     liste.get(identifiant).setNom(nom);
46     return liste.get(identifiant);
47 }
```

Test de la méthode UPDATE :

HTTP PUT http://localhost:9999/update?identifiant=0&nom=test

Send

Params Auth Headers (9) Body Pre-req. Tests Settings Cookies

Key	Value	Bulk Edit
<input checked="" type="checkbox"/> identifiant	0	
<input checked="" type="checkbox"/> nom	test	
Key	Value	

Body 200 OK 408 ms 209 B Save Response

Pretty Raw Preview Visualize JSON

```
1 {
2   "identifiant": 0,
3   "nom": "test",
4   "moyenne": 18.0
5 }
```

Vérification de la modification avec l'affichage de la liste de tous les étudiants :

HTTP GET http://localhost:9999/liste

Send

Params Auth Headers (8) Body Pre-req. Tests Settings Cookies

Query Params

Key	Value	Bulk Edit
Key	Value	

Body 200 OK 26 ms 340 B Save Response

Pretty Raw Preview Visualize JSON

```
1 [
2   {
3     "identifiant": 0,
4     "nom": "test",
5     "moyenne": 18.0
6   },
7   {
8     "identifiant": 1,
9     "nom": "B",
10    "moyenne": 10.0
11  },
12  {
13    "identifiant": 2,
14    "nom": "C",
15    "moyenne": 15.5
16  },
17  ]
```

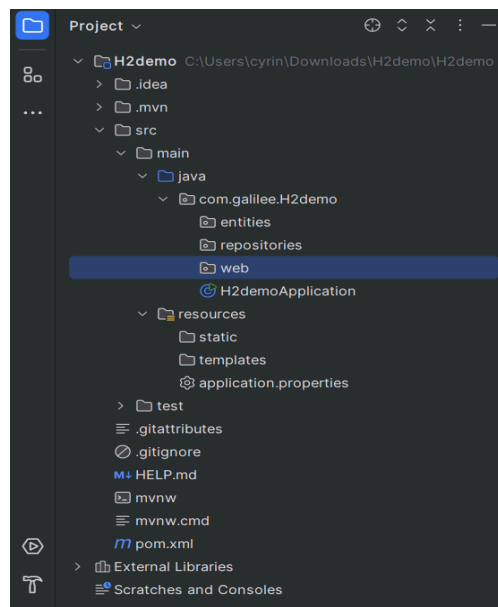
Partie 6 : H2 / MySQL

On commence par générer un nouveau projet Spring Boot via start.spring.io avec les dépendances suivantes :

Dépendance	Utilité
Spring Web	Pour développer des API REST (contrôleurs, endpoints)
Spring Data JPA	Pour la persistance des données et l'interaction avec la base de données
H2 Database	Base de données en mémoire pour le développement et les tests

Puis, on crée 3 sous-packages :

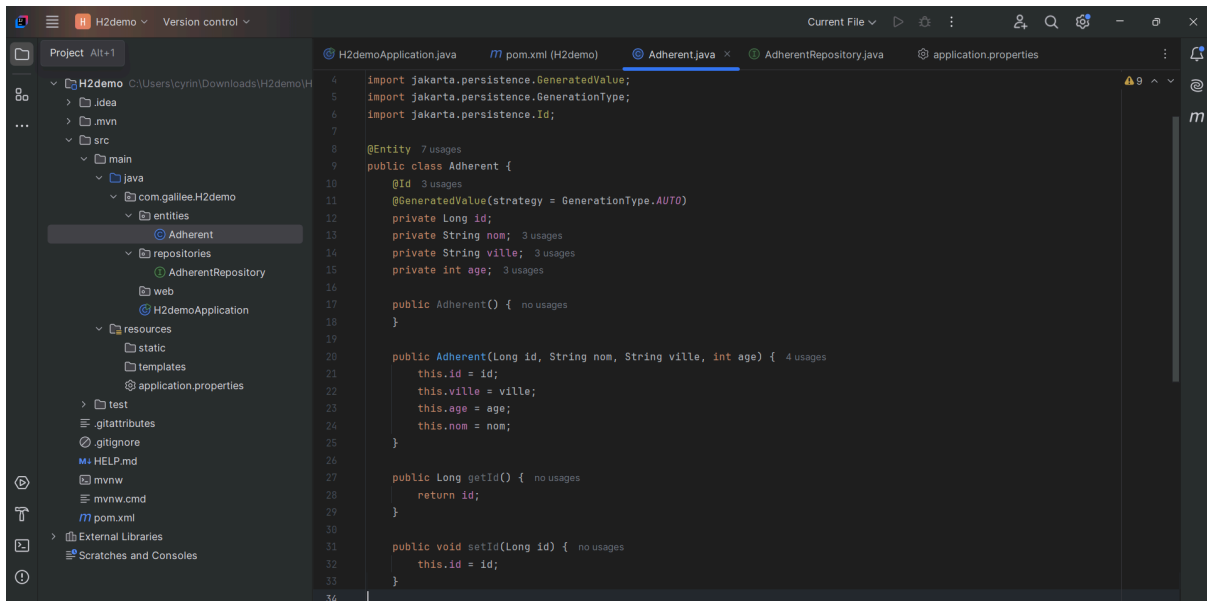
- **entities** : contient les classe smodèles (entités JPA)
- **repositories** : interfaces Spring Data JPA pour l'accès aux données
- **web** : controleurs REST (endpoints API)



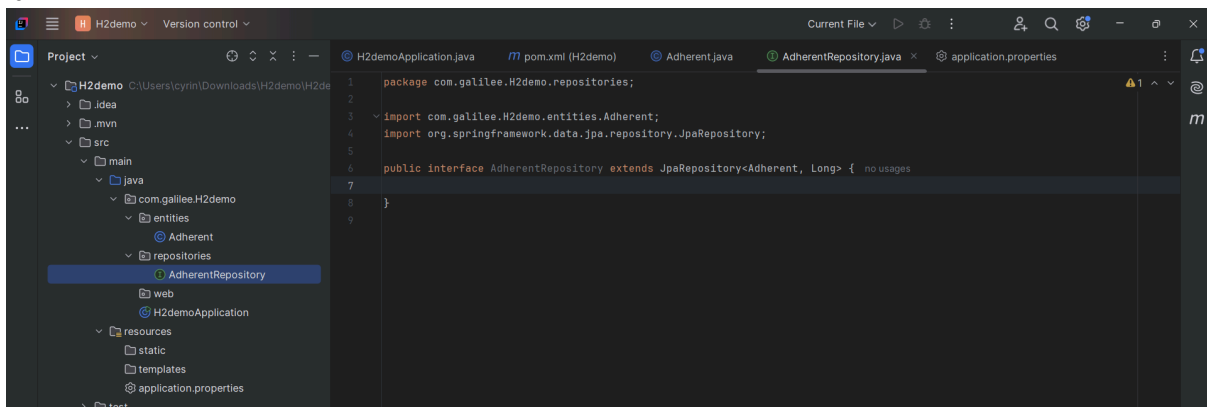
Nous allons utiliser Hibernate dans cette partie comme implémentation de JPA. Hibernate est automatiquement intégré via la dépendance Spring Data JPA. Aucune configuration supplémentaire n'est nécessaire

Création de l'Entité Adherent avec Hibernate/JPA : On ajoute une classe Adherent.java dans le package entities.

- L'annotation JPA/Hibernate **@Entity** transforme la classe en table SQL automatiquement (Nom de table par défaut : adherent).
- L'annotation **@Id** marque un champ comme clé primaire de l'entité, obligatoire pour toute entité JPA/Hibernate.
- L'annotation **@GeneratedValue** configure la génération automatique de la clé primaire.
- Les attributs (nom, ville,...) deviennent des colonnes en base de données.

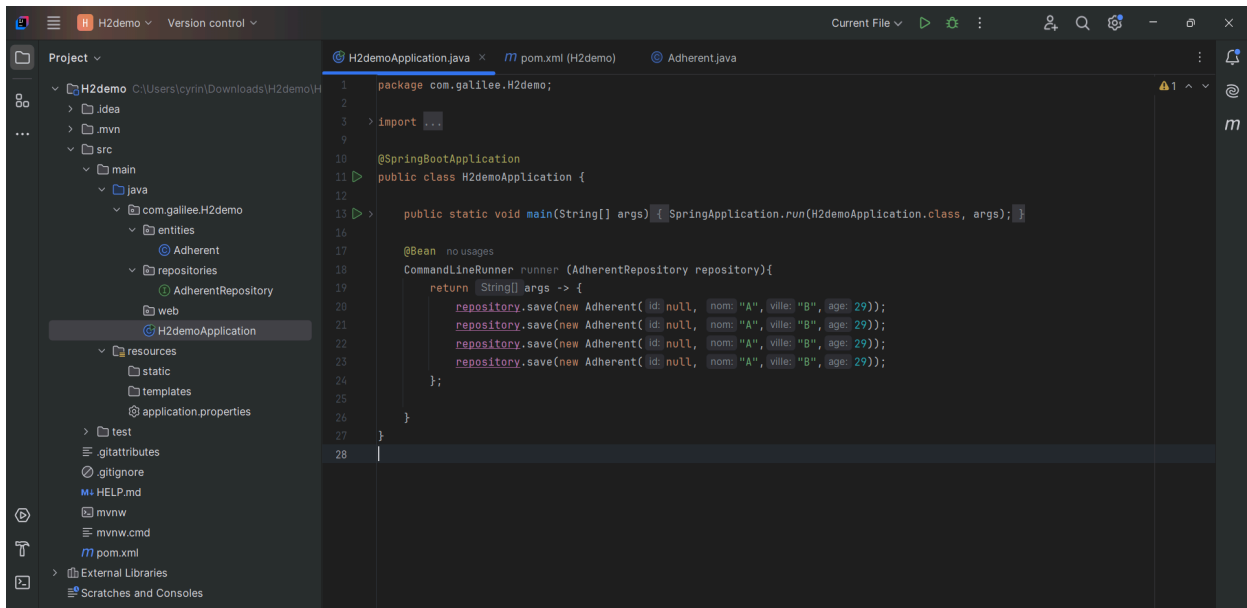


Ajout de l'interface AdherentRepository : On ajoute cette interface dans le package repositories. Elle fournit des méthodes CRUD prédéfinies (save(), findall(), ...) et on peut ajouter des méthodes personnalisées.

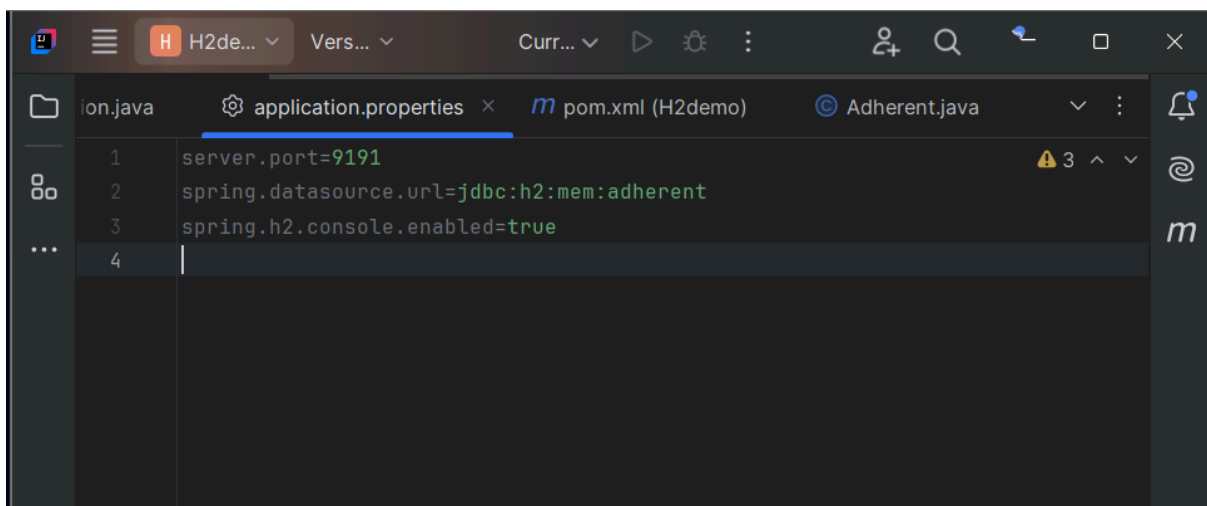


On implémente un CommandLineRunner dans la classe principale de notre application Spring Boot **H2demoApplication.java** :

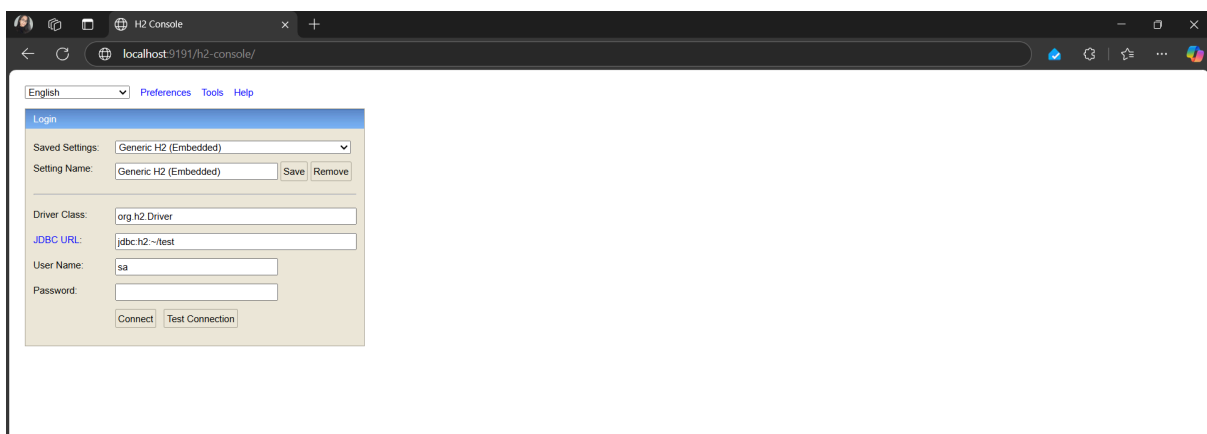
- Son rôle est d'insérer des données de test dans la base H2 au démarrage de l'application.
- L'annotation **@Bean** avant la méthode runner impose que Spring va appeler la méthode runner() au démarrage. Le retour (CommandLineRunner) est enregistré comme un bean Spring.



On configure le fichier application.properties avant d'exécuter une application Spring Boot avec H2 et JPA/Hibernate :



Une fois l'application exécutée, on peut accéder à la console H2 et interagir avec la base de données en mémoire :



On peut tester si l'application Spring Boot se connecte correctement à la base H2 en mémoire avec l'URL `jdbc:h2:mem:adherent` :

English Preferences Tools Help

Login

Saved Settings: Generic H2 (Embedded)

Setting Name: Generic H2 (Embedded) Save Remove

Driver Class: org.h2.Driver

JDBC URL: jdbc:h2:mem:adherent

User Name: sa

Password:

Connect Test Connection

Test successful

On se connecte à la base de données H2 : l'entité **Adherent** a bien été mappée en table SQL

H2 Console

localhost:9191/h2-console/login.do?sessionId=ae690672ae7d16f3e9809f1fd8a73b43

Auto commit Max rows: 1000 Auto complete Off Auto select On

jdbc:h2:mem:adherent

ADHERENT

INFORMATION_SCHEMA

Sequences

Users

H2 2.3.232 (2024-08-11)

Run Run Selected Auto complete Clear SQL statement:

Important Commands

?	Displays this Help Page
↑	Shows the Command History
Ctrl+Enter	Executes the current SQL statement
Shift+Enter	Executes the SQL statement defined by the text selection
Ctrl+Space	Auto complete
⏏	Disconnects from the database

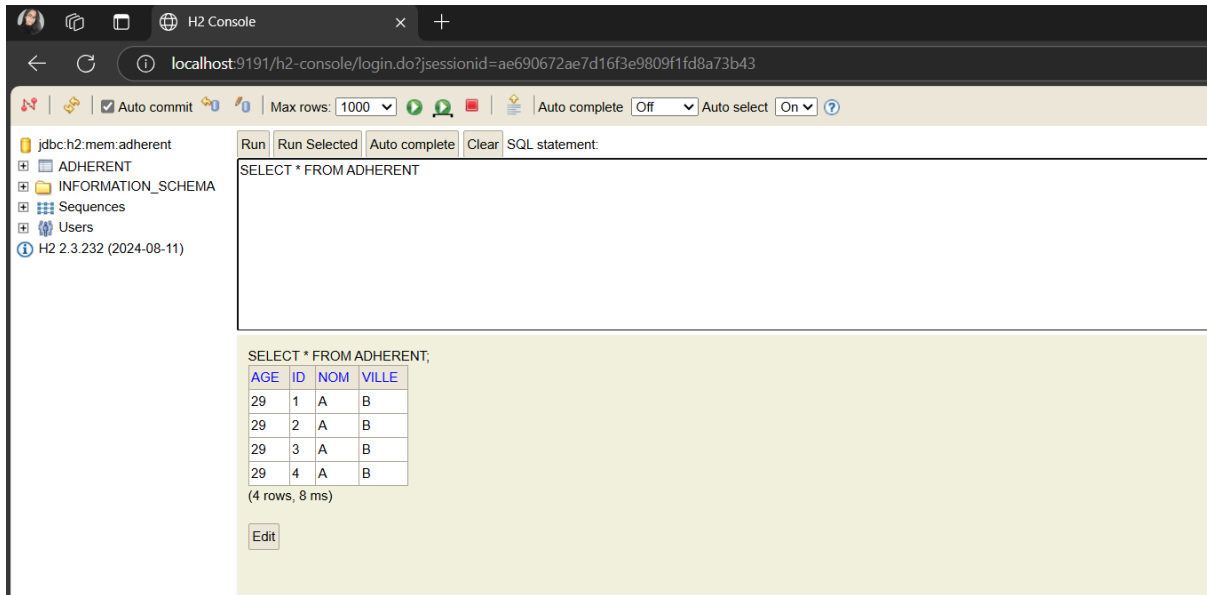
Sample SQL Script

Delete the table if it exists	DROP TABLE IF EXISTS TEST;
Create a new table with ID and NAME columns	CREATE TABLE TEST (ID INT PRIMARY KEY, NAME VARCHAR(255));
Add a new row	INSERT INTO TEST VALUES(1, 'Hello');
Add another row	INSERT INTO TEST VALUES(2, 'World');
Query the table	SELECT * FROM TEST ORDER BY ID;
Change data in a row	UPDATE TEST SET NAME='H' WHERE ID=1;
Remove a row	DELETE FROM TEST WHERE ID=2;
Help	HELP ...

Adding Database Drivers

Additional database drivers can be registered by adding the Jar file location of the driver to the environment variables H2DRIVERS or CLASSPATH. Example (Windows): to add the database driver library C:/Programs/hsqldb/lib/hsqldb.jar, set the environment variable H2DRIVERS to C:/Programs/hsqldb/lib/hsqldb.jar

On peut voir, au niveau de la table Adherent, les données insérées au moment de démarrage de l'application :



⇒ Cela montre le bon mapping entre la classe Java Adherent (Entity) et la table SQL ADHERENT via Hibernate/JPA .

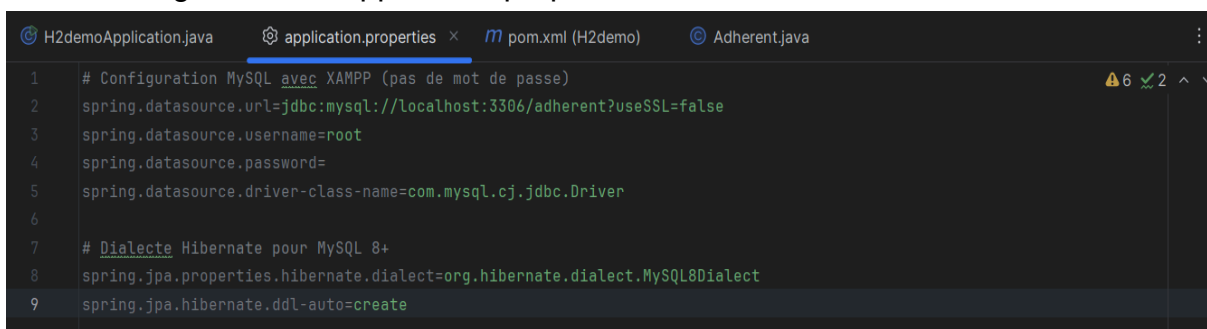
Migration de H2 (mémoire) vers MySQL

Voici les étapes complètes pour basculer notre application Spring Boot d'une base H2 en mémoire à MySQL :

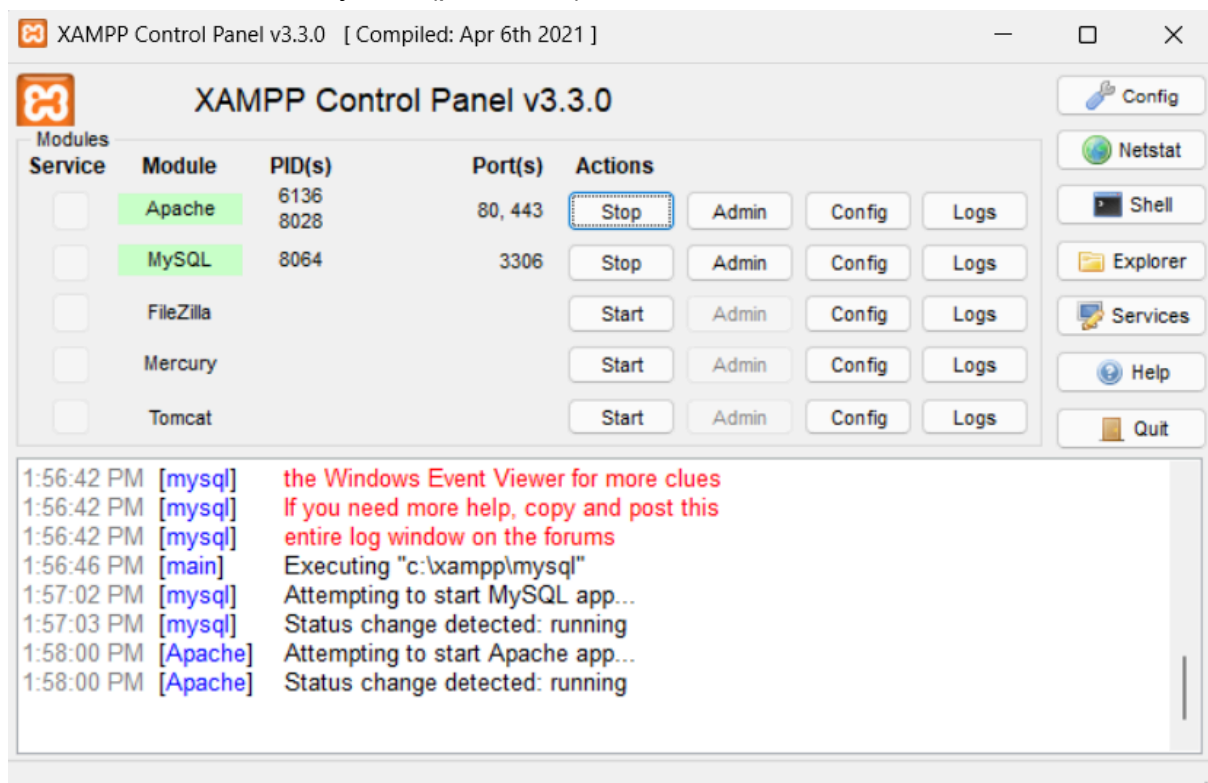
1. Modification dans pom.xml : on ajoute la dépendance MySQL



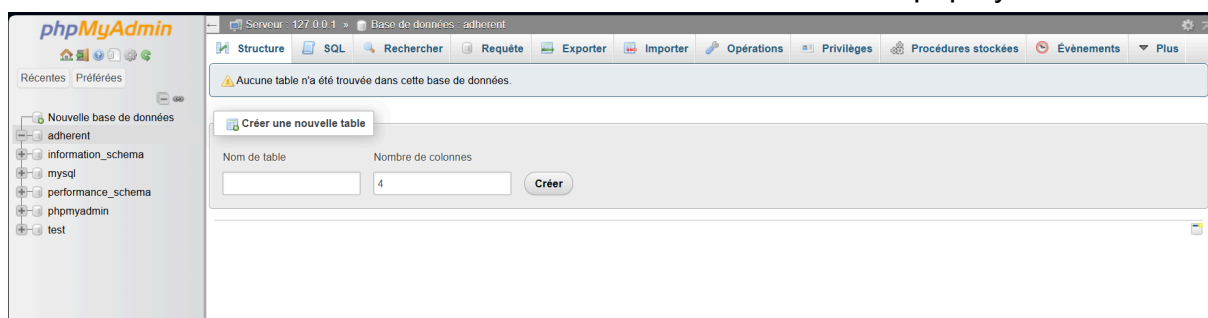
2. Configuration de application.properties :



3. Lancement de MySQL (port 3306)



4. Création d'une base de données 'adherent' au niveau de phpmyadmin :



Maintenant, on peut vérifier l'insertion des données dans la table adherent de MySQL après le démarrage de l'application Spring Boot :

