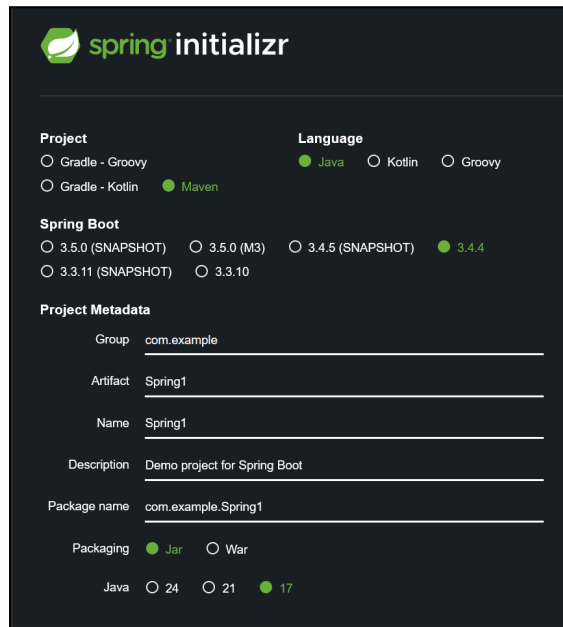


## Compte Rendu Spring Boot

Nous avons commencé par la création d'un projet Spring à l'aide de l'outil Spring initializr disponible en ligne comme le montre la figure suivante :



The image shows the Spring Initializr web form. It has a dark theme with green accents. The form is divided into several sections: Project, Language, Spring Boot, Project Metadata, and Packaging. The Project section has radio buttons for Gradle - Groovy, Gradle - Kotlin, and Maven (selected). The Language section has radio buttons for Java (selected), Kotlin, and Groovy. The Spring Boot section has radio buttons for 3.5.0 (SNAPSHOT), 3.5.0 (M3), 3.4.5 (SNAPSHOT), 3.4.4 (selected), 3.3.11 (SNAPSHOT), and 3.3.10. The Project Metadata section has input fields for Group (com.example), Artifact (Spring1), Name (Spring1), Description (Demo project for Spring Boot), and Package name (com.example.Spring1). The Packaging section has radio buttons for Jar (selected) and War. The Java section has radio buttons for 24, 21, and 17 (selected).

**Project**

☐ Gradle - Groovy ☐ Gradle - Kotlin ☒ Maven

**Language**

☒ Java ☐ Kotlin ☐ Groovy

**Spring Boot**

☐ 3.5.0 (SNAPSHOT) ☐ 3.5.0 (M3) ☐ 3.4.5 (SNAPSHOT) ☒ 3.4.4 ☐ 3.3.11 (SNAPSHOT) ☐ 3.3.10

**Project Metadata**

Group

Artifact

Name

Description

Package name

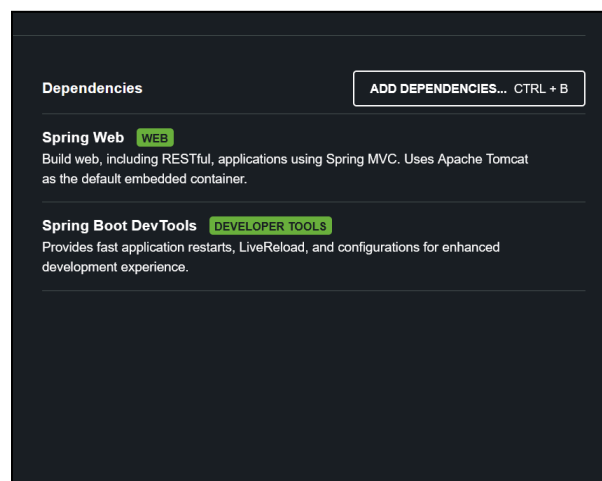
**Packaging**

☒ Jar ☐ War

**Java**

☐ 24 ☐ 21 ☒ 17

A côté, on a la barre des dépendances, où on va installer les dépendances nécessaires :



The image shows the Dependencies section of the Spring Initializr web form. It has a dark theme with green accents. The section is titled 'Dependencies' and has a button 'ADD DEPENDENCIES... CTRL + B'. Below the title, there are two dependency cards: 'Spring Web' with a 'WEB' tag and 'Spring Boot DevTools' with a 'DEVELOPER TOOLS' tag. Each card has a description of the dependency.

**Dependencies** [ADD DEPENDENCIES... CTRL + B](#)

**Spring Web** **WEB**  
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

**Spring Boot DevTools** **DEVELOPER TOOLS**  
Provides fast application restarts, LiveReload, and configurations for enhanced development experience.

Et enfin on n'a plus qu'à installer le projet et l'exploiter.

### Développement de l'application :

#### Premiers pas :

On a commencé par la création d'une classe MyApi, qui va contenir deux méthodes qui vont retourner bonjour et bonsoir et qui sont référencées par **@GetMapping** :

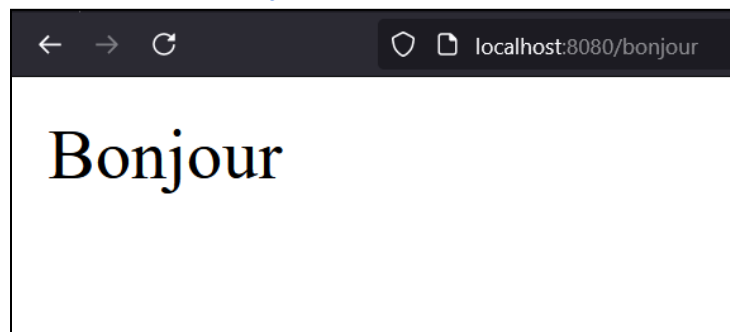
```
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

no usages
@RestController
public class MyApi {

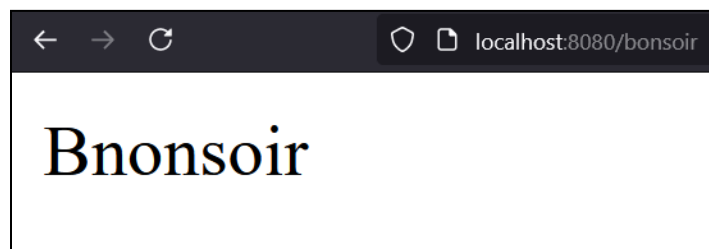
    no usages
    @GetMapping(value="/bonjour")
    public String bonjour(){
        return "Bonjour";
    }

    no usages
    @GetMapping(value = "/bonsoir")
    public String bonsoir(){
        return "Bonsoir";
    }
}
```

En exécutant "<http://localhost:8080/bonjour>" :

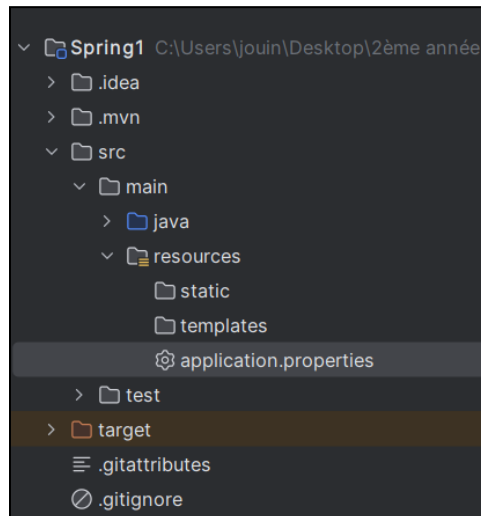


En exécutant "<http://localhost:8080/bonsoir>" :

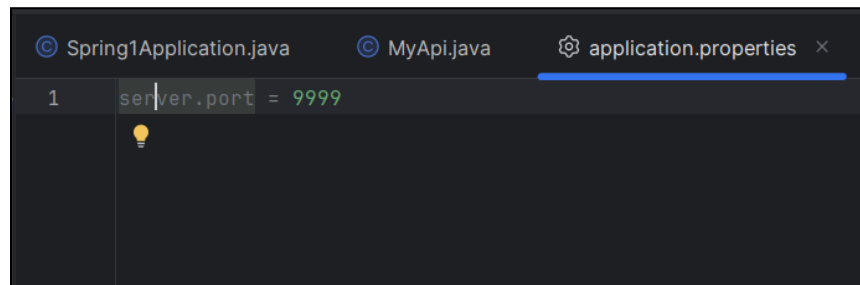


### Changement de port d'exécution de l'application :

Afin de changer le port d'exécution de l'application, il faut accéder au fichier **application.properties** disponible dans l'arborescence du projet comme le montre la figure suivante :



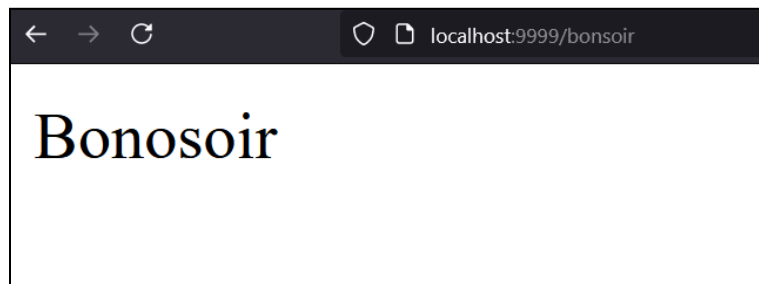
Dedans, on spécifie le numéro de port :



Après redémarrage, on remarque que l'application ne tourne plus sur le port 8080:



Elle tourne sur le port 9999:



### Programmation Objet :

On a commencé par la création d'une classe nommée "Etudiant" :

```
Spring1Application.java  MyApi.java  Etudiant.java  application.p
1  package com.example.Spring1;
2
3  2 usages
4  public class Etudiant {
5      3 usages
6      private int id ;
7      3 usages
8      private String nom;
9      3 usages
10     private double moyenne;
11
12     no usages
13     public Etudiant() {
14     }
15 }
```

On a crée deux constructeurs, un par défaut et l'autre prenant une valeur pour chaque attribut :

```
no usages
public Etudiant() {
}

1 usage
public Etudiant(int id, String nom, double moyenne) {
    this.id = id;
    this.nom = nom;
    this.moyenne = moyenne;
}
```

Pour chaque attribut de cette classe, on a créé les accesseur nécessaires : get et set :  
Exemple pour id :

```

no usages
public int getId() {
    return id;
}

no usages
public void setId(int id) {
    this.id = id;
}

```

Dans la classe **MyApi.java**, on a ajouté une méthode nommée “**getEtudiant**” qui va créer et nous retourner un étudiant:

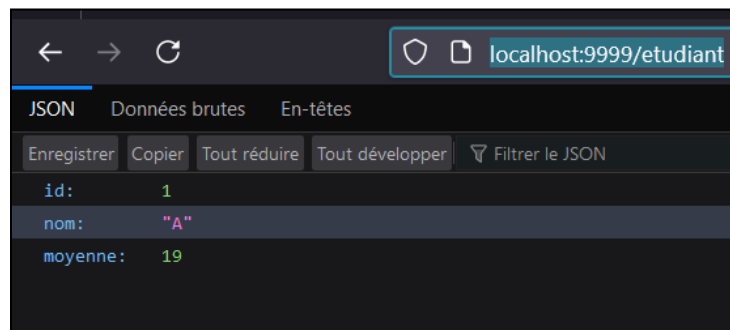
```

no usages
public Etudiant() {
}

1 usage
public Etudiant(int id, String nom, double moyenne) {
    this.id = id;
    this.nom = nom;
    this.moyenne = moyenne;
}

```

En exécutant “<http://localhost:9999/etudiant>” :



Paramètres dans l'URL :

On a créée une méthode somme qui prend deux entiers :

```

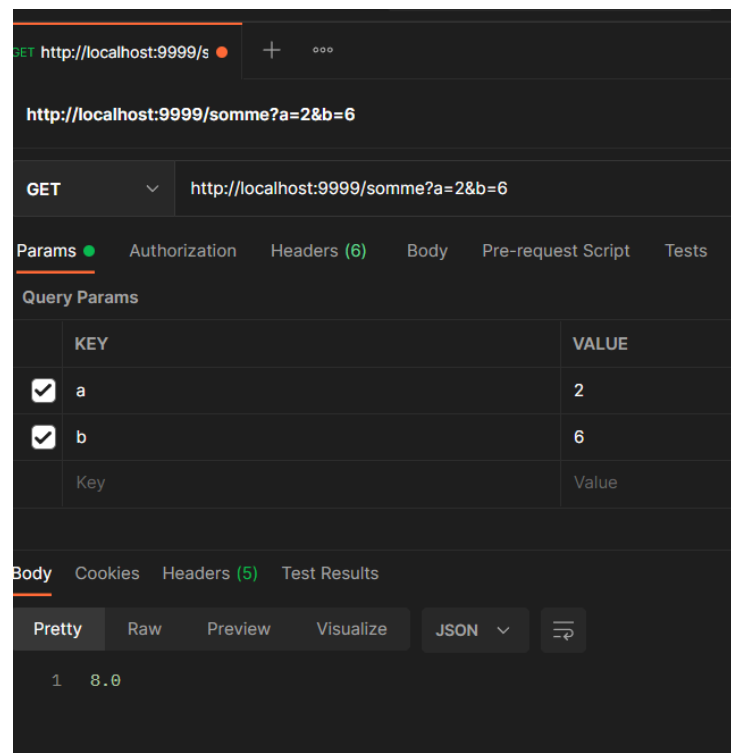
no usages
@GetMapping(value = "/somme ")
public double somme (double a, double b){
    return a+b;
}
}

```

Pour passer les valeurs à travers l'URL, on utilise “?” :

On a exécuté : “<http://localhost:9999/somme?a=2&b=6>”

RQ : on a utilisé Postman :



### Liste d'étudiants :

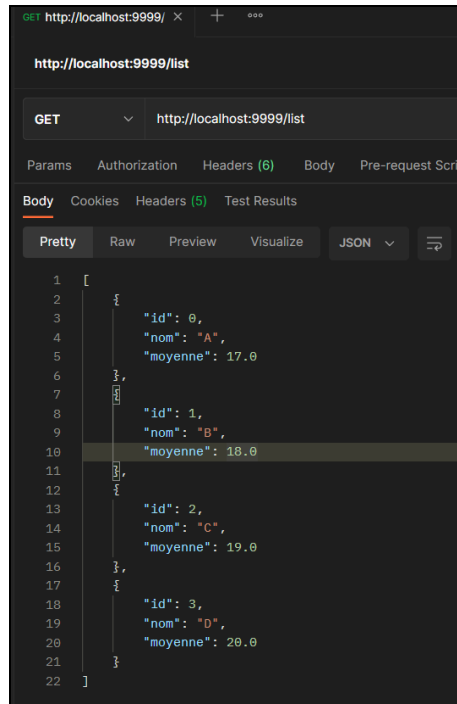
On a commencé par la création d'une liste statique d'étudiants et on lui a ajouté des objets :

```
no usages
public static Collection<Etudiant> liste = new ArrayList<>();
static{
    liste.add(new Etudiant( id: 0, nom: "A", moyenne: 17));
    liste.add(new Etudiant( id: 1, nom: "B", moyenne: 18));
    liste.add(new Etudiant( id: 2, nom: "C", moyenne: 19));
    liste.add(new Etudiant( id: 3, nom: "D", moyenne: 20));
}
```

On a ajouté une méthode “**getAllEtudiants**” qui va nous retourner tous les étudiants :

```
no usages
@GetMapping(value = "/list")
public Collection<Etudiant> getAllEtudiants(){
    return liste;
}
```

En exécutant “<http://localhost:9999/list>” :



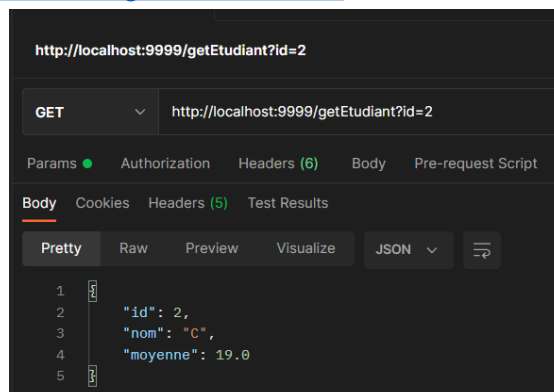
**Retrouver un étudiant avec un id :**

On a créé une méthode qui prend en paramètre un id de l'étudiant :

```
no usages

@GetMapping(value="/getEtudiant")
public Etudiant getEtudiant(int id) {
    Etudiant res=null;
    for(int i=0; i< liste.size();i++)
        if( liste.get(i).getId()==id)
            res=liste.get(i);
    return res;
}
```

En exécutant "<http://localhost:9999/getEtudiant?id=2>" :



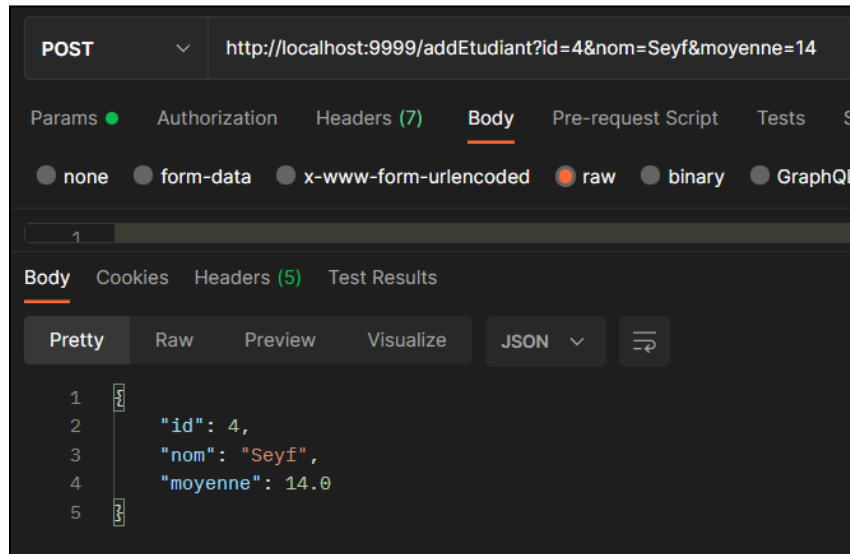
### Ajout d'un étudiant :

On a ajouté une méthode qui ajoute un étudiant :

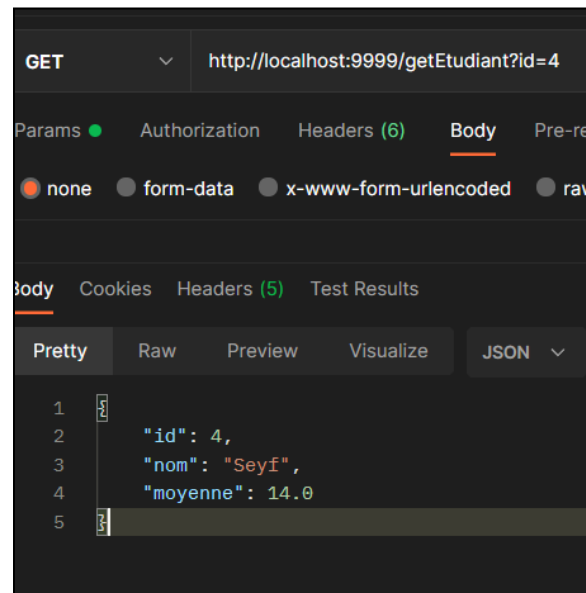
```
@PostMapping(value = "/addEtudiant")
public Etudiant addEtudiant(Etudiant etudiant){
    liste.add(etudiant);
    return etudiant;
}
```

Cette méthode prend les valeurs des attributs à partir de l'URL.

En exécutant "<http://localhost:9999/addEtudiant?id=4&nom=Seyf&moyenne=14>" :



Vérification de l'ajout :





## Suppression d'un étudiant :

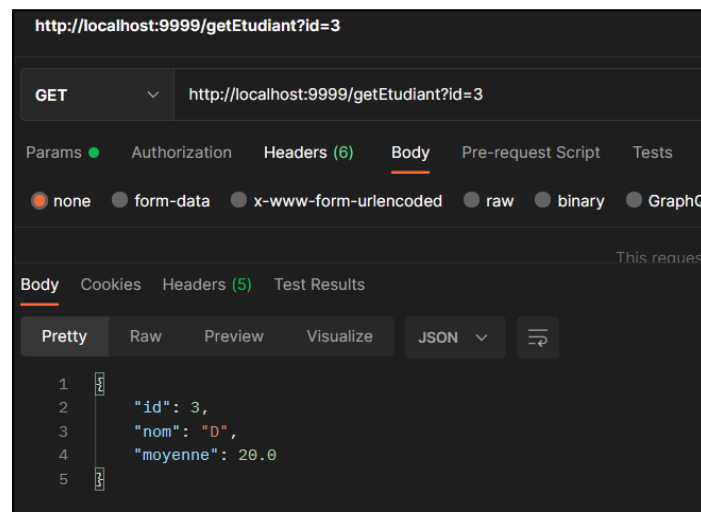
On a ajouté la méthode **deleteEtudiant** qui va prendre en paramètre l'identifiant et supprimer l'étudiant:

```
no usages
@DeleteMapping(value = "/deleteEtudiant")

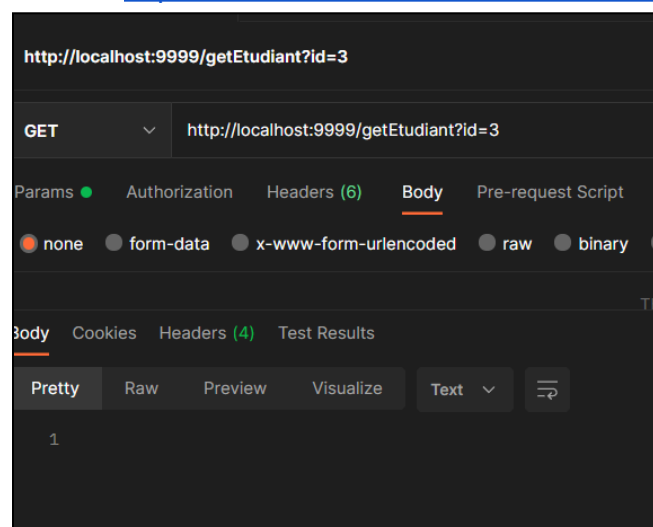
public void supprimerEtudiant(int id){
    for (int i= 0 ; i<liste.size();i++)
        if(liste.get(i).getId()==id)
            liste.remove(i);
}
```

On va supprimer l'étudiant id=3 :

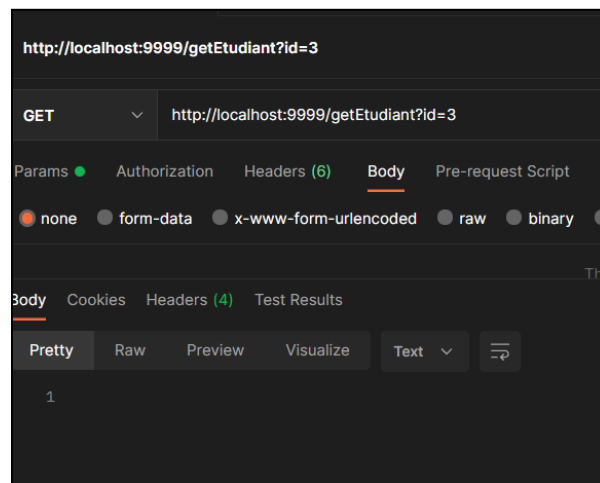
On commence par vérifier son existence :



On le supprime en exécutant : “<http://localhost:9999/deleteEtudiant?id=3>” :



On vérifie que l'étudiant a bien été supprimé :

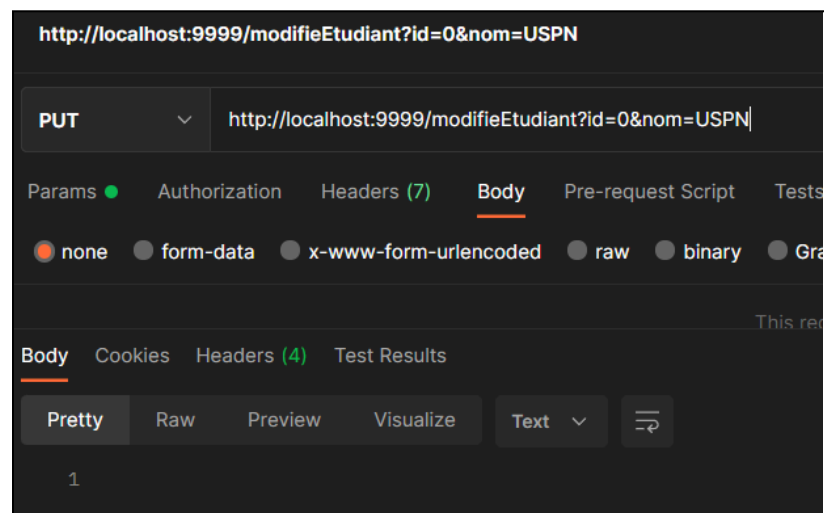


### Modification d'un étudiant :

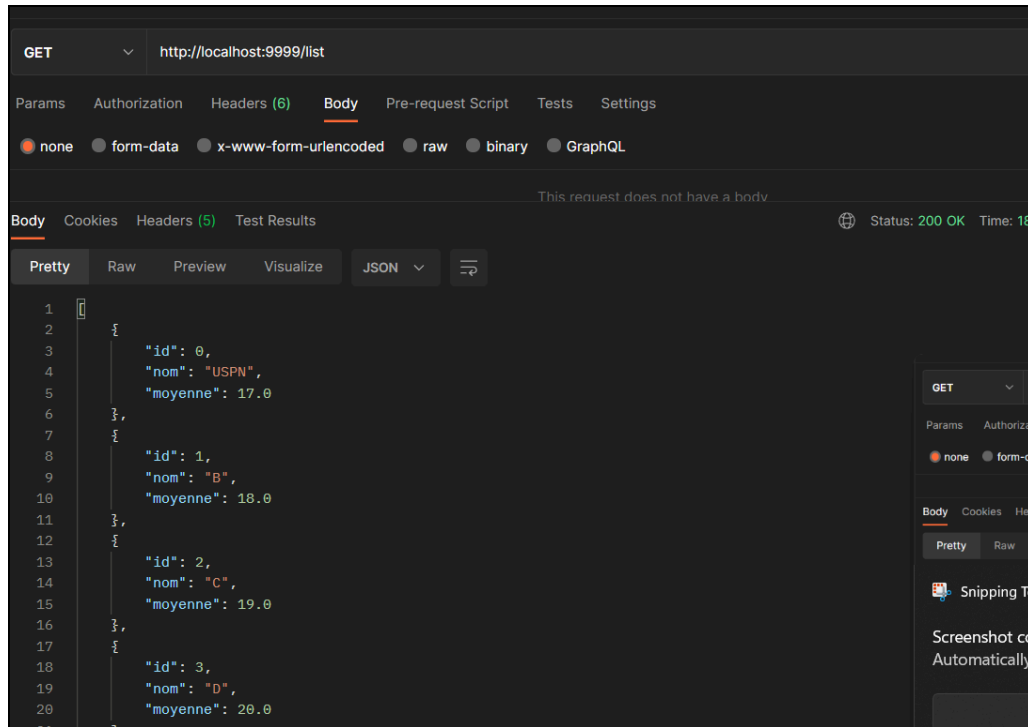
On a ajouté une méthode `modifierEtudiant` :

```
no usages
@PutMapping(value = "/modifierEtudiant")
public void modifierEtudiant (int id, String nom){
    for (int i= 0 ; i<liste.size();i++)
        if(liste.get(i).getId()==id)
            liste.get(i).setNom(nom);
}
```

On exécute "<http://localhost:9999/modifieEtudiant?id=0&nom=USPN>" :

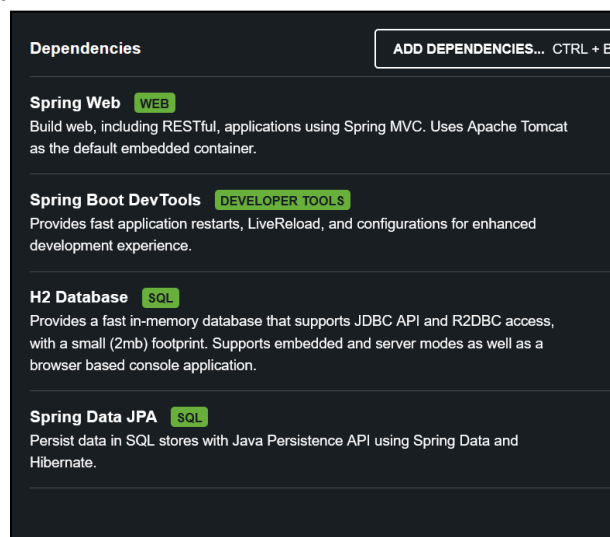


On vérifie la modification :



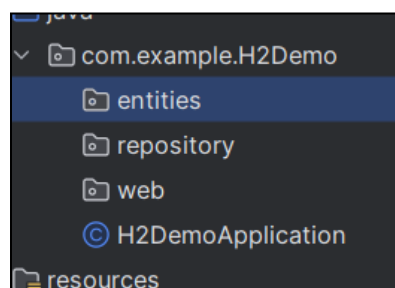
## Utilisation d'une base de données en mémoire : H2 :

On a créé un autre projet en installant les dépendances nécessaires :



## Création de packages :

On a créé 3 packages :



On a créé une classe **Adherent** dont les attributs sont les suivants :

```
no usages
public class Adherent {
    no usages
    private Long id;
    no usages
    private String nom;
    no usages
    private String ville;
    no usages
    private int age ;
    |
}
```

RQ : Créer les constructeurs et les accesseurs.

#### Création de l'interface de l'adhérent :

Une interface qui hérite de JpaRepository avec en paramètre la classe et le type de l'identifiant.

```
7 pom.xml (H2Demo)  Adherent.java  AdherentRepository.java x
1 package com.example.H2Demo.repository;
2
3 import com.example.H2Demo.entities.Adherent;
4 import org.springframework.data.jpa.repository.JpaRepository;
5
6 no usages
7 public interface AdherentRepository extends JpaRepository<Adherent,Long> {
8 }
9
```

Automatisation de la génération de l'identifiant des adhérents :

```
7 usages
@Entity
public class Adherent {
    3 usages
    @GeneratedValue(strategy = GenerationType.AUTO)
    @Id
    private Long id;
    3 usages
    private String nom;
```

## Ajout d'un CommandLineRunner :

Un CommandLineRunner est un concept qui permet d'exécuter un bean pour faire un traitement au moment du démarrage de notre application.

Ce code doit être dans le main :

```
public static void main(String[] args) { SpringApplication.run(H2DemoApplication.class, args); }  
no usages  
@Bean  
CommandLineRunner runner (AdherentRepository repository) {  
    return args -> {  
        repository.save(new Adherent( id: null, nom: "A", ville: "B", age: 29));  
        repository.save(new Adherent( id: null, nom: "B", ville: "B", age: 19));  
        repository.save(new Adherent( id: null, nom: "C", ville: "B", age: 49));  
        repository.save(new Adherent( id: null, nom: "D", ville: "B", age: 89));  
    };  
}
```

## Configuration de l'application :

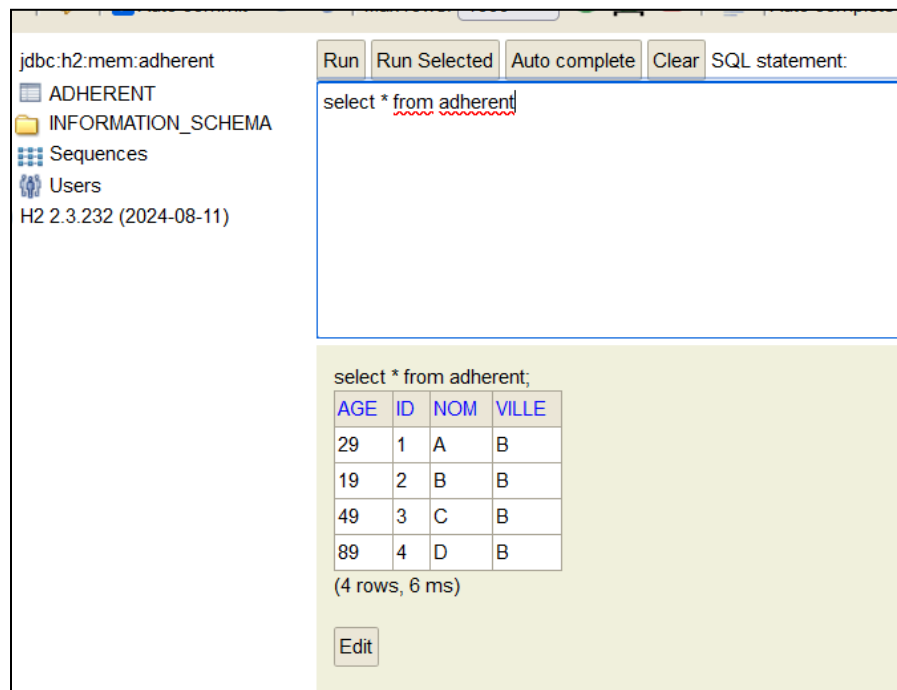
```
server.port= 9191  
spring.datasource.url = jdbc:h2:mem:adherent  
spring.h2.console.enabled=true
```

## Exécution :

En exécutant "<http://localhost:9191/h2-console/>" :

The screenshot shows the H2 Console web interface. At the top, there is a language dropdown set to 'English' and links for 'Preferences', 'Tools', and 'Help'. The main section is titled 'Login' and contains a 'Saved Settings' dropdown set to 'Generic H2 (Embedded)'. Below this, there is a 'Setting Name' field with the value 'Generic H2 (Embedded)' and 'Save' and 'Remove' buttons. The 'Driver Class' field is set to 'org.h2.Driver'. The 'JDBC URL' field is set to 'jdbc:h2:mem:adherent'. The 'User Name' field is set to 'sa'. The 'Password' field is empty. There are 'Connect' and 'Test Connection' buttons. At the bottom, a green banner displays the message 'Test successful'.

## Requête dans la base H2 :



## Utilisation d'une base de données MySql :

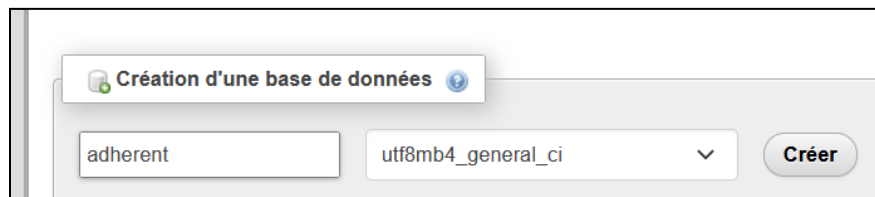
La première chose à faire, est la configuration de l'accès à la base de données :

Dans le fichier **application.properties**, on insère l'accès à la base.

RQ : Port par défaut : 3306.

```
spring.datasource.url = jdbc:mysql://localhost:3306/adherent
spring.datasource.username=root
spring.datasource.password=
```

On accède à PhpMyAdmin et on crée une base de données :



On configure le fonctionnement de la relation entre l'application et la base de données à travers hibernate.

Un aperçu des différentes options à partir de la documentation :

- create – Hibernate first drops existing tables, then creates new tables
- update – the object model created based on the mappings (annotations or XML) is compared with the existing schema, and then Hibernate updates the schema according to the diff. It never deletes the existing tables or columns even if they are no more required by the application
- create-drop – similar to create, with the addition that Hibernate will drop the database after all operations are completed. Typically used for unit testing
- validate – Hibernate only validates whether the tables and columns exist, otherwise it throws an exception
- none – this value effectively turns off the DDL generation

On a choisi l'option create :

```
spring.datasource.url = jdbc:mysql://localhost:3306/adherent
spring.datasource.username=root
spring.datasource.password=
spring.jpa.hibernate.ddl-auto = create
```

On a installé la dépendance : mysql pour un projet maven :

```
<dependency>
  <groupId>com.mysql</groupId>
  <artifactId>mysql-connector-j</artifactId>
  <version>9.2.0</version>
</dependency>
```

En exécutant l'application et en allant à l'interface de PhpMyAdmin, on constate la création de la table adhérents et l'insertion des nuplets.

✓ Amortage des lignes 0 - 3 (total de 4, traitement en 0,0001 seconde(s).)

`SELECT * FROM `adherent``

☐ Profilage [ Éditer en ligne ] [ Éditer ] [ Expliquer SQL ] [ Créer le code source PHP ] [ Actualiser ]

☐ Tout afficher | Nombre de lignes : 25 ▼ | Filtrer les lignes: Chercher dans

Options supplémentaires

				age	id	nom	ville
<input type="checkbox"/>	✎ Éditer	📋 Copier	🗑 Supprimer	29	1	A	B
<input type="checkbox"/>	✎ Éditer	📋 Copier	🗑 Supprimer	19	2	B	B
<input type="checkbox"/>	✎ Éditer	📋 Copier	🗑 Supprimer	49	3	C	B
<input type="checkbox"/>	✎ Éditer	📋 Copier	🗑 Supprimer	89	4	D	B

⬆ ☐ Tout cocher Avec la sélection : ✎ Éditer 📋 Copier 🗑 Supprimer