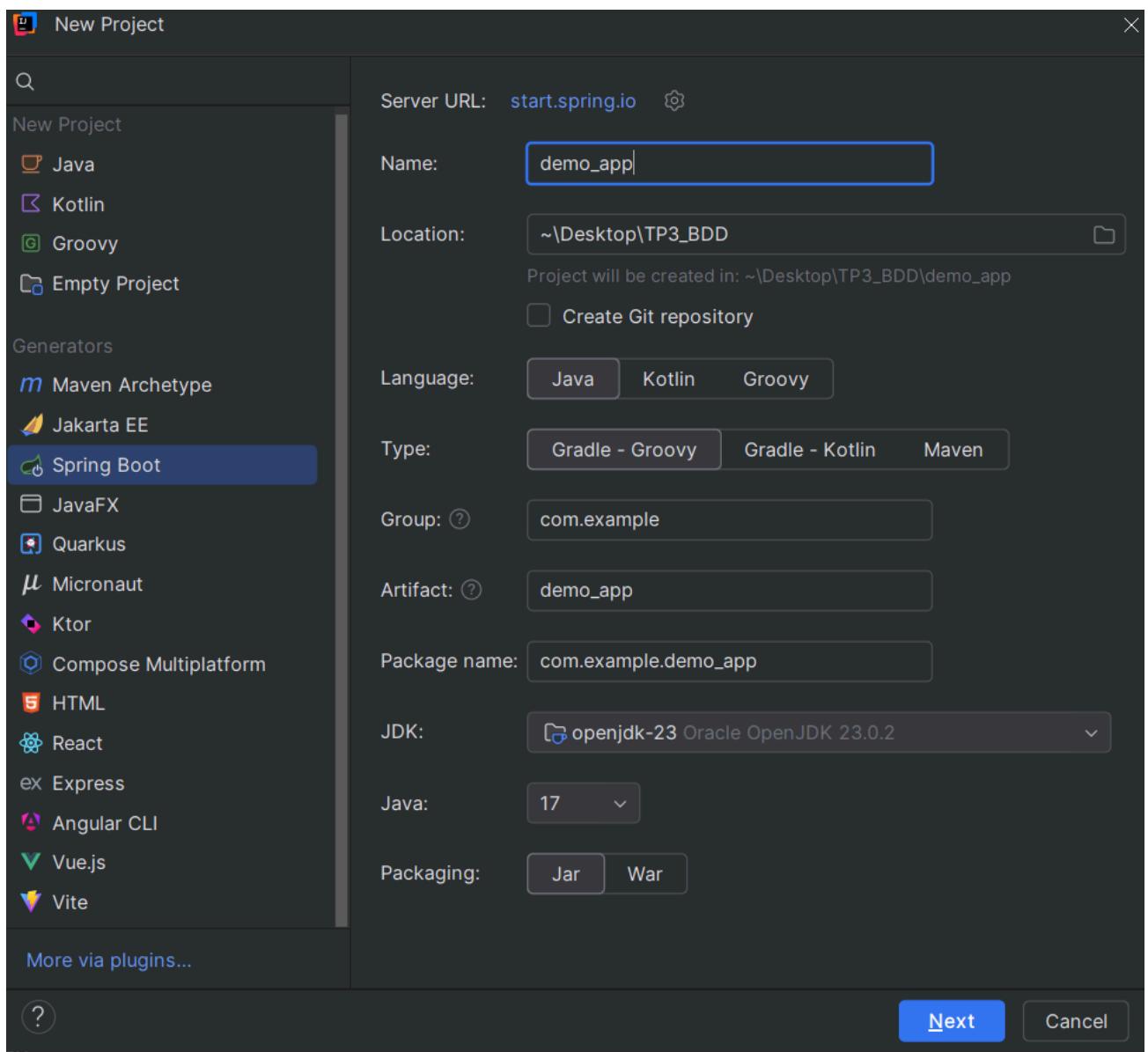
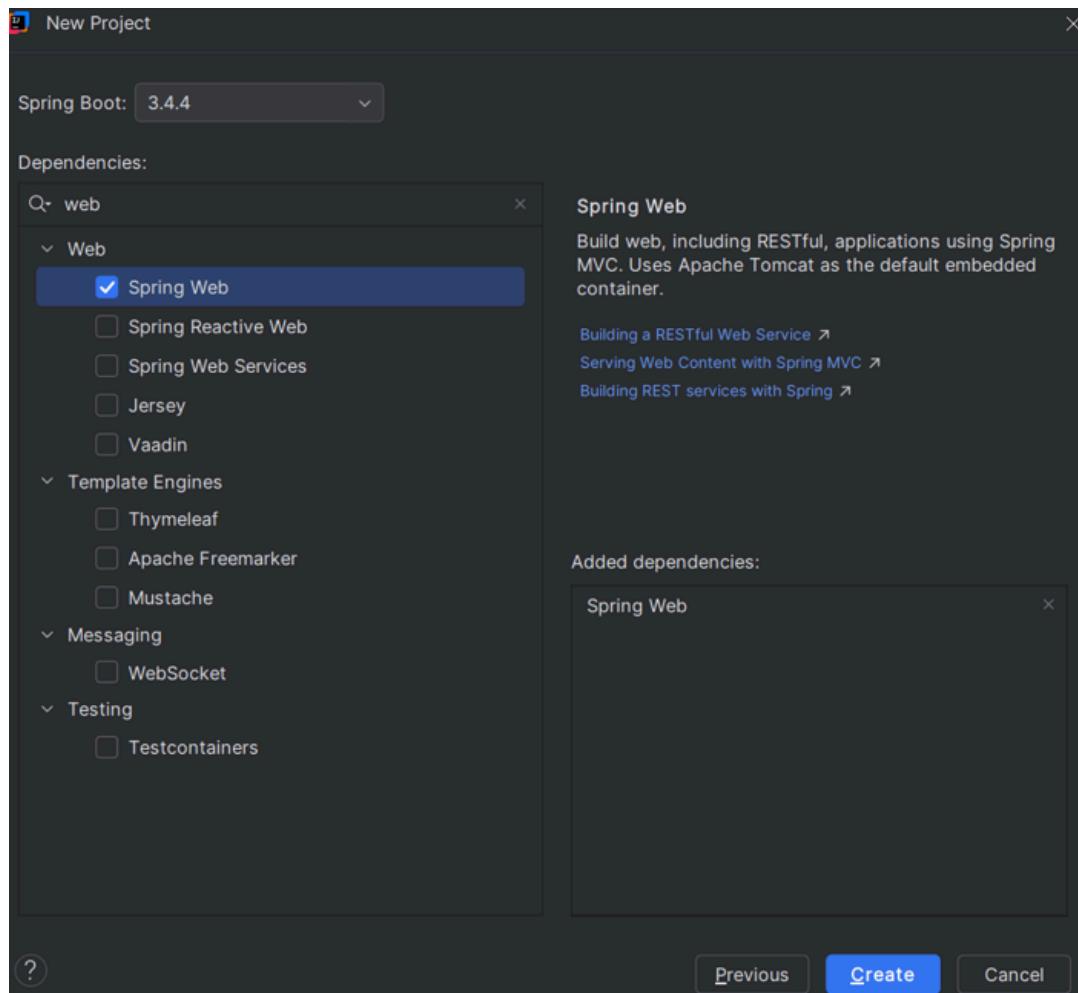


COMPTE-RENDU TP3 BDA



Dans cette étape, nous créons une application Spring Boot nommée `spring_app` avec Maven comme outil de build. Le projet sera sauvegardé dans le dossier `TP3_BDD` sur le bureau. Nous utilisons Java comme langage, avec le JDK 23 et une compatibilité Java 17. Le groupe est défini comme `com.example`, l'artifact comme `spring_app`, et le package résultant est `com.example.spring_app`. Cette configuration servira de base pour développer une application web légère et modulaire.



Nous sélectionnons ici la dépendance Spring Web pour notre projet, afin de développer une application web RESTful avec Spring MVC. Cette dépendance inclut un serveur Tomcat intégré, ce qui permet de lancer l'application sans configuration supplémentaire. Cela constitue la base pour créer des contrôleurs et exposer des API web. Ensuite, nous cliquons sur le bouton “Create” pour créer l'application Spring Boot.

```

Project ▾
  spring_app C:\Users\sadde\Desktop\TP3_BDD\spring_app
    .idea
    .mvn
    src
      main
        java
          com.example.spring_app
            SpringAppApplication
    resources
    test
    .gitattributes
    .gitignore
    HELP.md
    mvnw
    mvnw.cmd
    pom.xml
  External Libraries
  Scratches and Consoles

M+ HELP.md C SpringAppApplication.java x
1 package com.example.spring_app;
2
3 > import ...
4
5
6 @SpringBootApplication
7 public class SpringAppApplication {
8
9 >   public static void main(String[] args) { SpringApplication.run(SpringAppApplication.class, args); }
10
11 }
12
13
14

```

The screenshot shows the IntelliJ IDEA interface with the 'spring_app' project open. The project tree on the left lists files like .idea, .mvn, src, pom.xml, mvnw, and mvnw.cmd. The code editor on the right displays the 'SpringAppApplication.java' file, which contains the following code:

```

package com.example.spring_app;
import ...
@SpringBootApplication
public class SpringAppApplication {
    public static void main(String[] args) { SpringApplication.run(SpringAppApplication.class, args); }
}

```

Suite à la création du projet, une classe principale SpringAppApplication est automatiquement générée. Elle contient la méthode main, point d'entrée de l'application, qui utilise SpringApplication.run() pour démarrer le contexte Spring Boot. Cette classe, annotée

avec `@SpringBootApplication`, configure et initialise l'application web en intégrant toutes les dépendances Spring nécessaires. C'est à partir de cette base que nous allons ajouter nos propres contrôleurs et logiques métier.

The screenshot shows the IntelliJ IDEA interface. In the Project tool window, the file `SpringAppApplication.java` is selected. The code contains the standard Spring Boot main class definition:

```

6  @SpringBootApplication
7  public class SpringAppApplication {
8
9  >   public static void main(String[] args) { SpringApplication.run(SpringAppApplication.class, args); }
10
11
12
13
14

```

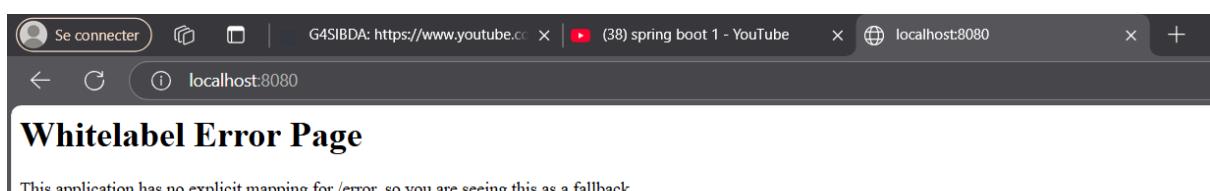
In the Run tool window, the application is running under Tomcat. The logs show the startup process:

```

2025-03-28T14:39:01.299+01:00 INFO 14332 --- [spring_app] [main] c.e.spring_app.SpringApplication : Starting SpringAppApplication using Java 23.0.2 with Tomcat initialized with port 8080 (http)
2025-03-28T14:39:01.304+01:00 INFO 14332 --- [spring_app] [main] c.e.spring_app.SpringApplication : No active profile set, falling back to 1 default profile
2025-03-28T14:39:02.227+01:00 INFO 14332 --- [spring_app] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8080 (http)
2025-03-28T14:39:02.253+01:00 INFO 14332 --- [spring_app] [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2025-03-28T14:39:02.254+01:00 INFO 14332 --- [spring_app] [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.39]
2025-03-28T14:39:02.297+01:00 INFO 14332 --- [spring_app] [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2025-03-28T14:39:02.298+01:00 INFO 14332 --- [spring_app] [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization complete
2025-03-28T14:39:02.674+01:00 INFO 14332 --- [spring_app] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8080 (http) with context path
2025-03-28T14:39:02.680+01:00 INFO 14332 --- [spring_app] [main] c.e.spring_app.SpringApplication : Started SpringAppApplication in 1.96 seconds (process time)

```

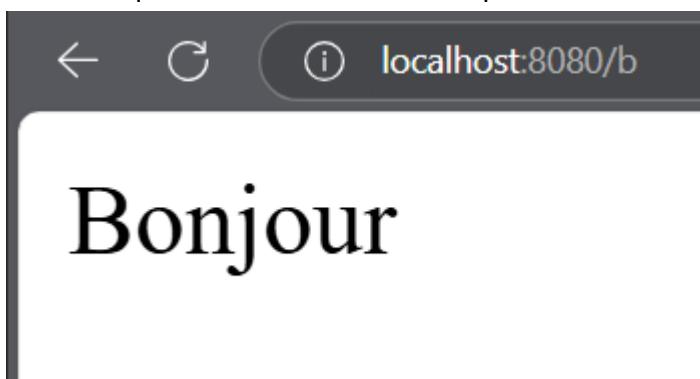
Après avoir exécuté l'application, le serveur Tomcat intégré démarre automatiquement sur le port 8080, comme indiqué dans la console. Le message "Started SpringAppApplication" confirme que l'application Spring Boot s'est lancée avec succès. Elle est maintenant prête à recevoir des requêtes HTTP, ce qui permet de commencer le développement de nos fonctionnalités web.



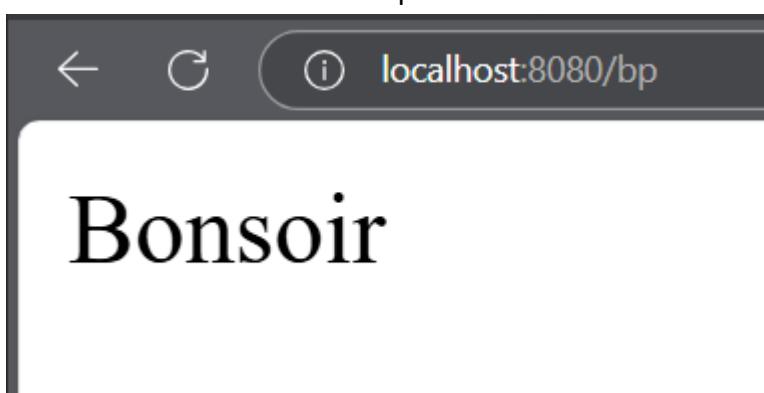
Lorsqu'on accède à `localhost:8080`, une page d'erreur Whitelabel Error Page s'affiche. Cela signifie que l'application fonctionne, mais qu'aucun contrôleur n'a encore été défini pour gérer la racine `/`. Le serveur répond avec une erreur 404 (Not Found) car aucune route n'est encore configurée. Il faut maintenant créer un contrôleur pour définir les premières routes de l'application.

```
import org.springframework.web.bind.annotation.RestController;
import org.springframework.web.bind.annotation.GetMapping;
public class MyApi {
    @GetMapping(value = "/b")
    public String bonjour(){
        return "Bonjour";
    }
    @GetMapping(value = "/bp")
    public String bonsoir(){
        return "Bonsoir";
    }
}
```

Pour corriger l'erreur 404, un contrôleur nommé MyApi a été ajouté. Ce contrôleur est annoté avec `@RestController`, ce qui permet de gérer des requêtes HTTP. Deux routes ont été définies : `/b` qui retourne "Bonjour" et `/bp` qui retourne "Bonsoir". En accédant à `localhost:8080/b` ou `localhost:8080/bp`, l'application renverra ces réponses textuelles, montrant que l'API fonctionne. Il faut par la suite relancer le serveur.



En accédant à `localhost:8080/b`, l'application répond correctement avec le message "Bonjour", ce qui confirme que le contrôleur MyApi fonctionne et que la route a bien été définie. L'API est maintenant opérationnelle.



De même, en accédant à `localhost:8080/bp`, l'application renvoie bien le texte "Bonsoir". Cela confirme que les deux routes de l'API fonctionnent correctement et que le contrôleur répond comme attendu.

The screenshot shows the IntelliJ IDEA interface. The left sidebar displays the project structure for 'spring_app' with files like .idea, .mvn, src (containing main/java/com/example/spring_app/MyApi.java and main/resources/application.properties), test, target, .gitattributes, and .gitignore. The right pane shows the 'application.properties' file with the content:

```
spring.application.name=spring_app
server.port=9999
```

The 'Run' tab is selected, showing the 'SpringAppApplication' configuration. The 'Console' tab displays the application's startup logs:

```
2025-03-28T14:51:25.179+01:00 INFO 5720 --- [spring_app] [main] c.e.spring_app.SpringApplication : Starting SpringAppApplication using Java 25.0.2 with Tomcat initialized with port 9999 (http)
2025-03-28T14:51:25.951+01:00 INFO 5720 --- [spring_app] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Starting service [Tomcat]
2025-03-28T14:51:25.977+01:00 INFO 5720 --- [spring_app] [main] o.apache.catalina.core.StandardService : Starting Servlet Engine: [Apache Tomcat/10.1.39]
2025-03-28T14:51:26.017+01:00 INFO 5720 --- [spring_app] [main] o.a.c.c.C.[Tomcat].[localhost].[] : Initializing Spring embedded WebApplicationContext
2025-03-28T14:51:26.018+01:00 INFO 5720 --- [spring_app] [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed
2025-03-28T14:51:26.348+01:00 INFO 5720 --- [spring_app] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 9999 (http) with context path
2025-03-28T14:51:26.354+01:00 INFO 5720 --- [spring_app] [main] c.e.spring_app.SpringApplication : Started SpringAppApplication in 1.627 seconds (process time)
```

Le fichier application.properties a été modifié pour personnaliser le port de l'application. Le port par défaut 8080 a été remplacé par 9999 avec la ligne server.port=9999. Après redémarrage, le serveur Tomcat intégré s'est bien lancé sur le port 9999, comme confirmé dans la console. Désormais, l'application est accessible via localhost:9999.

The screenshot shows the IntelliJ IDEA interface with the 'Etudiant.java' file open. The code defines a class 'Etudiant' with attributes 'identifiant', 'nom', and 'moyenne', and methods for getting and setting 'identifiant'. The code is as follows:

```
package com.example.spring_app;

public class Etudiant {
    private int identifiant; 3 usages
    private String nom; 3 usages
    private double moyenne; 3 usages
    public Etudiant() {} no usages
    public Etudiant(int identifiant, String nom, double moyenne) { no usages
        this.identifiant = identifiant;
        this.nom = nom;
        this.moyenne = moyenne;
    }

    public int getIdentifiant() { no usages
        return identifiant;
    }

    public void setIdentifiant(int identifiant) { no usages
        this.identifiant = identifiant;
    }
}
```

Une nouvelle classe Etudiant a été créée pour représenter un étudiant avec trois attributs : identifiant, nom, et moyenne. La classe comprend un constructeur par défaut, un constructeur avec paramètres, ainsi que des méthodes get et set pour l'attribut identifiant. Cette classe pourra être utilisée pour manipuler des objets étudiants dans l'API, comme par exemple les envoyer ou les recevoir via des routes REST.

```

public void setIdentifiant(int identifiant) { no usages
    this.identifiant = identifiant;
}

public String getNom() { no usages
    return nom;
}

public void setNom(String nom) { no usages
    this.nom = nom;
}

public double getMoyenne() { no usages
    return moyenne;
}

public void setMoyenne(double moyenne) { no usages
    this.moyenne = moyenne;
}
}

```

La classe Etudiant a été complétée avec les getters et setters pour tous les attributs : nom et moyenne, en plus de l'identifiant. Ces méthodes permettent de lire et modifier les valeurs de chaque propriété, facilitant ainsi l'échange de données entre le backend et d'éventuels clients via une API REST. Cette classe est maintenant pleinement utilisable comme modèle dans l'application Spring Boot.

The screenshot shows a Java IDE interface with several tabs at the top: HELP.md, SpringAppApplication.java, MyApi.java (which is currently selected), application.properties, and Etudiant.java.

The code in MyApi.java is as follows:

```

1 package com.example.spring_app;
2 import org.springframework.web.bind.annotation.GetMapping;
3 import org.springframework.web.bind.annotation.RestController;
4
5 @RestController
6 public class MyApi {
7     @GetMapping(value="/bonjour")
8     public String bonjour(){
9         return "Bonjour";
10    }
11    @GetMapping(value="/bonsoir")
12    public String bonsoir(){
13        return "Bonsoir";
14    }
15    @GetMapping(value="/etudiant")
16    public Etudiant getEtudiant(){
17        return new Etudiant( identifiant: 1, nom: "A", moyenne: 19);
18    }
19 }

```

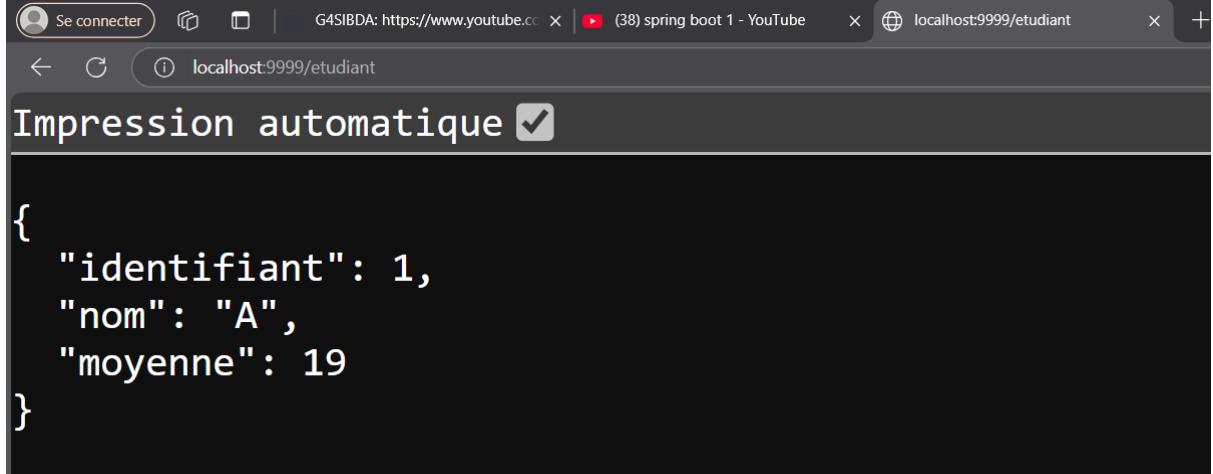
Below the code editor is the SpringAppApplication run console, which displays the following log output:

```

Run SpringAppApplication
Console Beans Health Mappings Environment
2025-03-28T15:07:43.457+01:00 INFO 14220 --- [spring_app] [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2025-03-28T15:07:43.457+01:00 INFO 14220 --- [spring_app] [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.39]
2025-03-28T15:07:43.496+01:00 INFO 14220 --- [spring_app] [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2025-03-28T15:07:43.496+01:00 INFO 14220 --- [spring_app] [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization complete
2025-03-28T15:07:43.808+01:00 INFO 14220 --- [spring_app] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 9999 (http) with context path
2025-03-28T15:07:43.815+01:00 INFO 14220 --- [spring_app] [main] c.e.spring_app.SpringAppApplication : Started SpringAppApplication in 1.644 seconds (prod)

```

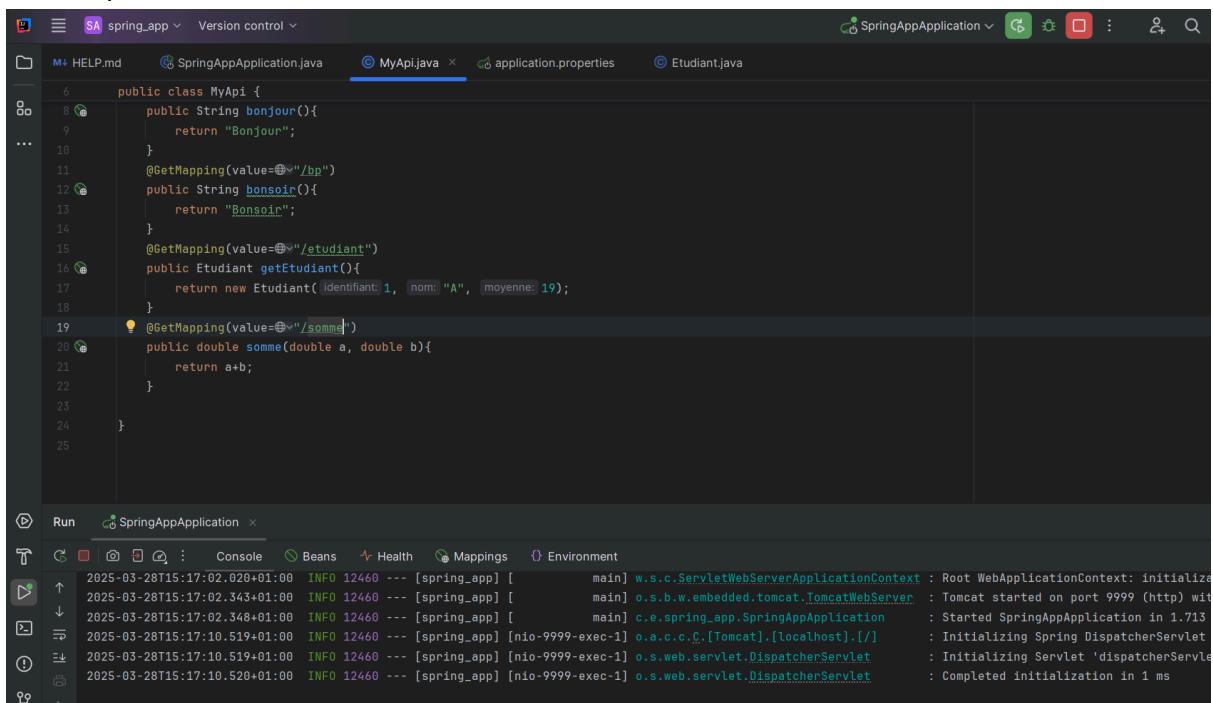
Une nouvelle route /etudiant a été ajoutée au contrôleur MyApi. Cette méthode retourne un objet Etudiant avec des valeurs prédéfinies (identifiant : 1, nom : "A", moyenne : 19). Grâce à @RestController, Spring Boot convertira automatiquement cet objet en JSON lorsqu'il sera retourné, permettant de l'utiliser comme une API REST. On relance ainsi le serveur.



The screenshot shows a browser window with the address bar set to localhost:9999/etudiant. The page content displays the JSON representation of an Etudiant object:

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

En accédant à localhost:9999/etudiant, l'API renvoie bien un objet JSON représentant un étudiant. On retrouve les données : identifiant : 1, nom : "A", moyenne : 19, ce qui confirme que l'objet Etudiant est correctement sérialisé par Spring Boot. L'API REST fonctionne donc comme prévu.

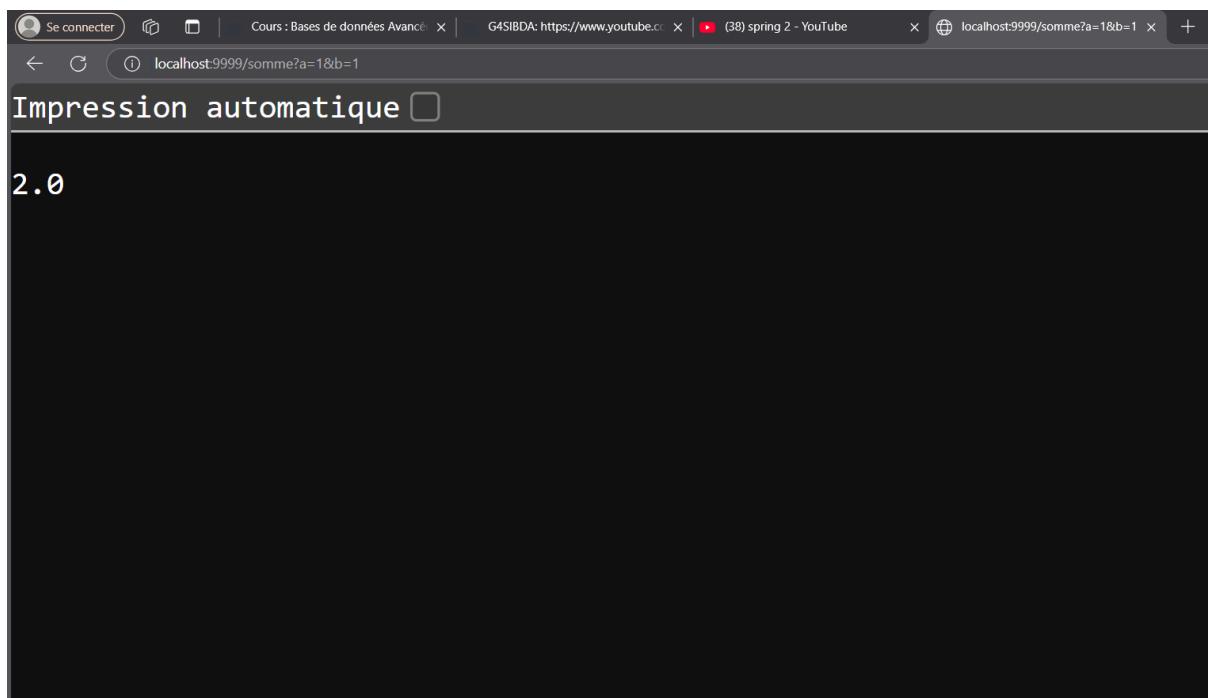


The screenshot shows an IDE interface with the following details:

- Code View:** The file MyApi.java is open, showing Java code for a controller. It includes methods for bonjour, bonnejour, getEtudiant, and somme.
- Log View:** The bottom pane shows the application's log output:

```
2025-03-28T15:17:02.020+01:00 INFO 12460 --- [spring_app] [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialized at 2025-03-28T15:17:02.343+01:00 INFO 12460 --- [spring_app] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 9999 (http) with context '/SpringAppApplication' at 2025-03-28T15:17:02.519+01:00 INFO 12460 --- [spring_app] [nio-9999-exec-1] o.a.c.c.C.[Tomcat].[localhost].[] : Started SpringAppApplication in 1.713 seconds (average processing time 1 ms) at 2025-03-28T15:17:10.519+01:00 INFO 12460 --- [spring_app] [nio-9999-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet' at 2025-03-28T15:17:10.520+01:00 INFO 12460 --- [spring_app] [nio-9999-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
```

Une nouvelle route /somme a été ajoutée, qui prend deux paramètres a et b de type double et retourne leur somme. Cette méthode permet de tester une API avec des paramètres, mais il faut encore spécifier que ces valeurs doivent être passées via des query parameters (avec @RequestParam) pour que Spring puisse les lier correctement lors de l'appel HTTP. On relance le serveur pour calculer la somme.



L'appel à `localhost:9999/somme?a=1&b=1` fonctionne et retourne bien 2.0, ce qui prouve que les paramètres `a` et `b` sont correctement pris en compte dans la méthode.

A screenshot of the Postman application interface. The top navigation bar shows "GET" and the URL "http://localhost:9999/b". Below the URL, there are tabs for "Params", "Authorization", "Headers (6)", "Body", "Scripts", and "Settings". The "Params" tab is selected, showing a table for "Query Params" with one row containing "Key" and "Value". The "Headers" tab shows five entries. The "Body" tab is selected, showing the response body: "1 Bonjour". The status bar at the bottom indicates "200 OK" and "26 ms".

À ce niveau, nous utilisons Postman pour tester et valider les différentes routes exposées par notre API REST. Cet outil permet d'envoyer des requêtes HTTP de manière contrôlée et d'analyser les réponses retournées par le serveur. Ici, une requête GET a été effectuée vers `localhost:9999/b`, et la réponse "Bonjour" avec le code 200 OK confirme que l'endpoint fonctionne comme attendu.

The screenshot shows the Postman interface. At the top, there is a header bar with 'GET' and the URL 'http://localhost:9999/somme?a=1&b=1'. Below this is a navigation bar with tabs for 'Params', 'Authorization', 'Headers (6)', 'Body', 'Scripts', and 'Settings'. On the right side of the navigation bar, there are buttons for 'Cookies' and 'Send'. Under the 'Params' tab, there is a table with two rows: 'a' with value '1' and 'b' with value '1'. Below the table, there is a section for 'Query Params' with a table for 'Key' and 'Value'. The main body of the Postman window shows a 'Body' tab with a JSON response: { "result": 2.0 }. Above the body, there is a status bar with '200 OK', '6 ms', '167 B', and a 'Save Response' button.

On utilise notamment Postman pour tester qu'une requête GET a été envoyée sur localhost:9999/somme avec les paramètres a=1 et b=1. La réponse obtenue est 2.0 avec un statut 200 OK, ce qui confirme que la route calcule correctement la somme des deux valeurs fournies.

The screenshot shows the IntelliJ IDEA interface for a Spring Boot application named 'spring_app'. The code editor displays the 'MyApi.java' file, which contains Java code for a REST API. The run console at the bottom shows the application's logs:

```

2025-03-28T15:18:25.885+01:00 INFO 5748 --- [spring_app] [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed
2025-03-28T15:18:26.204+01:00 INFO 5748 --- [spring_app] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 9999 (http) with context path
2025-03-28T15:18:26.210+01:00 INFO 5748 --- [spring_app] [main] c.e.spring.app.SpringApplication : Started SpringApplication in 1.681 seconds (process)
2025-03-28T15:18:28.568+01:00 INFO 5748 --- [spring_app] [nio-9999-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2025-03-28T15:18:28.568+01:00 INFO 5748 --- [spring_app] [nio-9999-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2025-03-28T15:18:28.569+01:00 INFO 5748 --- [spring_app] [nio-9999-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms

```

Une collection statique liste de type Collection<Etudiant> a été ajoutée à la classe MyApi. Elle contient plusieurs objets Etudiant initialisés avec des valeurs identiques.

The screenshot shows an IDE interface with several tabs at the top: HELP.md, SpringAppApplication.java, MyApi.java (which is currently selected), application.properties, and Etudiant.java. The MyApi.java code defines a class with methods for addition and a collection of students. A new method `getallEtudiant` has been added, which returns a collection of `Etudiant` objects. The `application.properties` file contains the line `server.port=9999`. The bottom section shows the application's logs and a terminal window where the application is running.

```
public class MyApi {  
    ...  
    @GetMapping(value = "/somme")  
    public double somme(double a, double b){  
        return a+b;  
    }  
    public static Collection<Etudiant> liste = new ArrayList<>();  
    static{  
        liste.add(new Etudiant( identifiant: 0, nom: "A", moyenne: 19));  
        liste.add(new Etudiant( identifiant: 0, nom: "A", moyenne: 19));  
        liste.add(new Etudiant( identifiant: 0, nom: "A", moyenne: 19));  
        liste.add(new Etudiant( identifiant: 0, nom: "A", moyenne: 19));  
    }  
    @GetMapping(value = "/liste")  
    public Collection<Etudiant> getAllEtudiant(){  
        return liste;  
    }  
}  
Run SpringAppApplication
```

```
2025-03-28T15:18:25.885+01:00 INFO 5748 --- [spring_app] [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed  
2025-03-28T15:18:26.204+01:00 INFO 5748 --- [spring_app] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 9999 (http) with context path  
2025-03-28T15:18:26.210+01:00 INFO 5748 --- [spring_app] [main] c.e.spring_app.SpringApplication : Started SpringApplication in 1.681 seconds (proce  
2025-03-28T15:18:28.568+01:00 INFO 5748 --- [spring_app] [nio-9999-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherSer  
2025-03-28T15:18:28.569+01:00 INFO 5748 --- [spring_app] [nio-9999-exec-1] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'  
2025-03-28T15:18:28.569+01:00 INFO 5748 --- [spring_app] [nio-9999-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
```

Une nouvelle route /liste a été ajoutée. Elle permet de retourner l'ensemble des objets Etudiant présents dans la collection statique liste. L'appel à cette route renverra un tableau JSON contenant tous les étudiants initialisés.

The screenshot shows a browser window with multiple tabs. The active tab displays a JSON array of four student objects, each with an identifier of 0, a name of "A", and a grade of 19. This corresponds to the four entries in the `liste` collection defined in the code.

```
[  
  {  
    "identifiant": 0,  
    "nom": "A",  
    "moyenne": 19  
  },  
  {  
    "identifiant": 0,  
    "nom": "A",  
    "moyenne": 19  
  },  
  {  
    "identifiant": 0,  
    "nom": "A",  
    "moyenne": 19  
  },  
  {  
    "identifiant": 0,  
    "nom": "A",  
    "moyenne": 19  
  }]
```

L'appel à localhost:9999/liste retourne un tableau JSON contenant quatre objets Etudiant avec les mêmes valeurs (identifiant : 0, nom : "A", moyenne : 19), ce qui confirme que la route expose correctement l'ensemble de la collection définie.

The screenshot shows the Postman interface with a collection named "SpringApp". A new request is being made to the endpoint `http://localhost:9999/liste`. The response body is a JSON array with four elements, each representing a student with an identifier of 0, a name of "A", and a grade of 19.0. The status code is 200 OK.

```

[{"identifiant": 0, "nom": "A", "moyenne": 19.0}, {"identifiant": 0, "nom": "A", "moyenne": 19.0}, {"identifiant": 0, "nom": "A", "moyenne": 19.0}, {"identifiant": 0, "nom": "A", "moyenne": 19.0}]
  
```

La même requête GET a été envoyée via Postman sur localhost:9999/liste. La réponse obtenue est aussi un tableau JSON contenant quatre objets Etudiant avec les valeurs identifiant : 0, nom : "A", moyenne : 19.0, et le statut retourné est 200 OK.

The screenshot shows the IntelliJ IDEA interface with a project named "spring_app". The `MyApi.java` file is open, showing the following code:

```

public class MyApi {
    ...
    @GetMapping(value = "/somme")
    public double somme(double a, double b) {
        return a+b;
    }
    public static Collection<Etudiant> liste = new ArrayList<>(); static{
        liste.add(new Etudiant(0, "A", 19));
        liste.add(new Etudiant(1, "A", 19));
        liste.add(new Etudiant(2, "A", 19));
        liste.add(new Etudiant(3, "A", 19));
    }
    ...
    @GetMapping(value = "/liste")
    public Collection<Etudiant> getAllEtudiant(){
        return liste;
    }
}
  
```

The `Run` tab shows the application is running. The `Console` tab displays the application logs, which include initialization messages for the Spring context and Tomcat server.

Les valeurs de la collection liste ont été modifiées : les objets Etudiant ont maintenant des identifiants uniques allant de 0 à 3, tandis que le nom et la moyenne restent inchangés.

```
[{"identifiant": 0, "nom": "A", "moyenne": 19}, {"identifiant": 1, "nom": "A", "moyenne": 19}, {"identifiant": 2, "nom": "A", "moyenne": 19}, {"identifiant": 3, "nom": "A", "moyenne": 19}]
```

L'appel à localhost:9999/liste affiche désormais une liste JSON avec quatre étudiants ayant des identifiants distincts de 0 à 3, ce qui confirme que la mise à jour des données dans la collection a bien été prise en compte.

HTTP SpringApp / New Request

GET http://localhost:9999/liste

Send

Params	Authorization	Headers (6)	Body	Scripts	Settings	Cookies					
Query Params											
<table border="1"> <thead> <tr> <th>Key</th> <th>Value</th> <th>Description</th> <th>Bulk Edit</th> </tr> </thead> <tbody> <tr> <td>Key</td> <td>Value</td> <td>Description</td> <td></td> </tr> </tbody> </table>	Key	Value	Description	Bulk Edit	Key	Value	Description				
Key	Value	Description	Bulk Edit								
Key	Value	Description									

Body Cookies Headers (5) Test Results

200 OK 7 ms 337 B Save Response

{ } JSON ▾ Preview Visualize

```

1 [
2   {
3     "identifiant": 0,
4     "nom": "A",
5     "moyenne": 19.0
6   },
7   {
8     "identifiant": 1,
9     "nom": "A",
10    "moyenne": 19.0
11  },
12  {
13    "identifiant": 2,
14    "nom": "A",
15    "moyenne": 19.0
16  },
17  {
18    "identifiant": 3,
19    "nom": "A",
20    "moyenne": 19.0
21  }
22 ]

```

On obtient la même réponse via Postman qui renvoie un tableau JSON contenant quatre objets Etudiant avec des identifiants allant de 0 à 3, un nom égal à "A" et une moyenne de 19.0. Le serveur répond avec le statut 200 OK, confirmant le bon fonctionnement de la route.

The screenshot shows a Java application running in a Spring Boot environment. The code editor displays `MyApi.java` with the following content:

```
public class MyApi {
    ...
    @GetMapping(value="/etudiant")
    public Etudiant getEtudiant(){
        return new Etudiant( identifiant: 1, nom: "A", moyenne: 19);
    }
    @GetMapping(value="/somme")
    public double somme(double a, double b){
        return a+b;
    }
    public static ArrayList<Etudiant> liste = new ArrayList<>();  6 usages
    static{
        liste.add(new Etudiant( identifiant: 0, nom: "A", moyenne: 19));
        liste.add(new Etudiant( identifiant: 1, nom: "A", moyenne: 19));
        liste.add(new Etudiant( identifiant: 2, nom: "A", moyenne: 19));
        liste.add(new Etudiant( identifiant: 3, nom: "A", moyenne: 19));
    }
    ...
    @GetMapping(value="/liste")
    public Collection<Etudiant> getAllEtudiant(){
    }
}
```

The line `public static ArrayList<Etudiant> liste = new ArrayList<>();` is highlighted with a red box.

The terminal window shows the application's logs:

```
2025-03-28T15:30:40.687+01:00 INFO 12580 --- [spring_app] [main] < Performance
2025-03-28T15:30:41.034+01:00 INFO 12580 --- [spring_app] [main]
2025-03-28T15:30:41.040+01:00 INFO 12580 --- [spring_app] [main]
2025-03-28T15:30:42.214+01:00 INFO 12580 --- [spring_app] [nio-9999-exec-1]
2025-03-28T15:30:42.215+01:00 INFO 12580 --- [spring_app] [nio-9999-exec-1]
2025-03-28T15:30:42.216+01:00 INFO 12580 --- [spring_app] [nio-9999-exec-1]
```

The terminal also shows the current working directory and encoding information:

```
> java > com > example > spring_app > MyApi > liste 26:28 CRLF UTF-8 4 spaces
```

Le type de la collection liste a été modifié pour être explicitement un `ArrayList<Etudiant>` au lieu d'une interface `Collection<Etudiant>`.

The screenshot shows a Java IDE interface. The top bar includes tabs for 'HELP.md', 'SpringApplication.java', 'MyApi.java' (which is currently selected), and 'application.properties'. The code editor displays the 'MyApi.java' file:

```
public class MyApi {
    static{
        liste.add(new Etudiant( identifiant: 3, nom: "A", moyenne: 19));
    }

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

    public Etudiant getEtudiant(int identifiant){ no usages
        return liste.get(identifiant);
    }
}
```

The bottom section shows the 'Run' tab with the application name 'SpringApplication'. The 'Console' tab displays log entries:

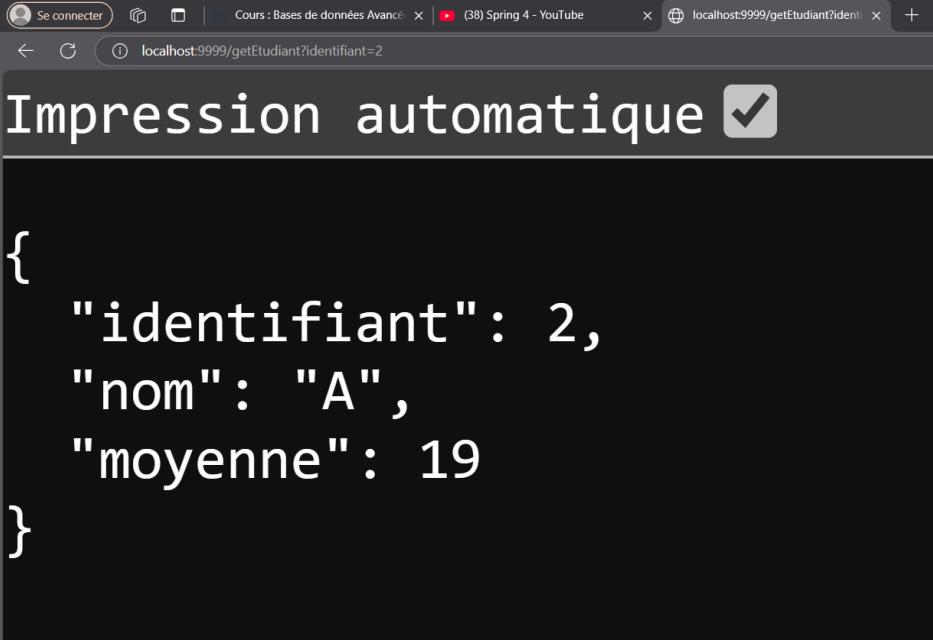
Time	Level	Logger	Message
2025-03-28T15:30:40.687+01:00	INFO	12580	-- [spring_app] main
2025-03-28T15:30:41.034+01:00	INFO	12580	-- [spring_app] main
2025-03-28T15:30:41.040+01:00	INFO	12580	-- [spring_app] main
2025-03-28T15:30:42.214+01:00	INFO	12580	-- [spring_app] [nio-9999-exec-1]
2025-03-28T15:30:42.215+01:00	INFO	12580	-- [spring_app] [nio-9999-exec-1]
2025-03-28T15:30:42.216+01:00	INFO	12580	-- [spring_app] [nio-9999-exec-1]

Une nouvelle méthode `getEtudiant(int identifiant)` a été ajoutée, qui retourne un étudiant depuis la liste en fonction de son index. Elle utilise `liste.get(identifiant)` pour retourner l'identifiant correspondant à la position de l'objet dans la liste.

The screenshot shows a browser window with the URL `localhost:9999/getEtudiant?identifiant=1`. The page content is a JSON object:

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

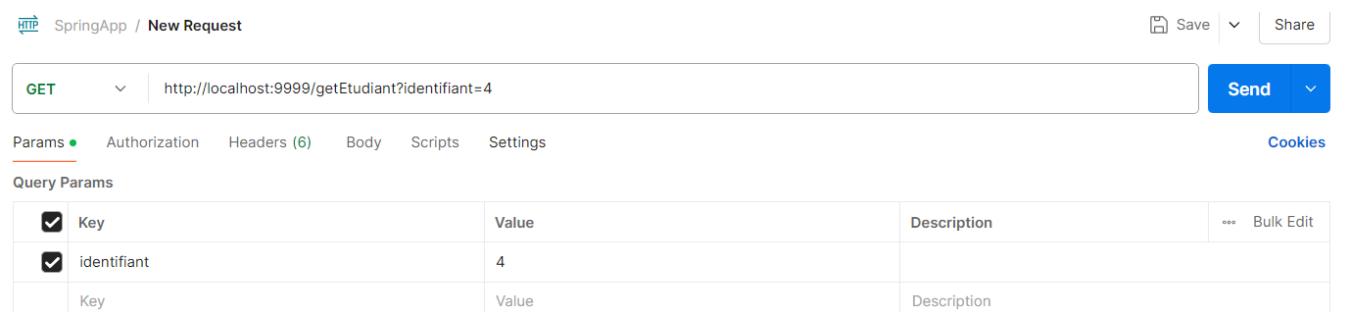
L'appel à localhost:9999/getEtudiant?identifiant=1 renvoie l'objet Etudiant avec identifiant : 1, nom : "A", moyenne : 19, ce qui confirme que la méthode getEtudiant récupère correctement l'étudiant correspondant à l'index 1 dans la liste.



The screenshot shows a browser window with the title "Impression automatique" and a checkmark icon. The main content area displays the following JSON object:

```
{  
    "identifiant": 2,  
    "nom": "A",  
    "moyenne": 19  
}
```

L'appel à localhost:9999/getEtudiant?identifiant=2 renvoie correctement l'objet Etudiant avec identifiant : 2, nom : "A", moyenne : 19, confirmant que l'accès par index fonctionne également pour cette valeur.



The screenshot shows the SpringApp interface with a "New Request" button. The request method is set to "GET" and the URL is "http://localhost:9999/getEtudiant?identifiant=4". The "Params" tab is selected, showing the following query parameters:

Key	Value	Description	Bulk Edit
identifiant	4		
Key	Value	Description	



The screenshot shows the SpringApp interface with a "500 Internal Server Error" message. The "Body" tab is selected, displaying the following JSON error response:

```
1 {  
2     "timestamp": "2025-03-28T14:39:29.417+00:00",  
3     "status": 500,  
4     "error": "Internal Server Error",  
5     "path": "/getEtudiant"  
6 }
```

En exécutant l'API suivante, on obtient une erreur 500.

```

        liste.add(new Etudiant( identifiant: 2 , nom: "A" , moyenne: 17));
        liste.add(new Etudiant( identifiant: 3 , nom: "A" , moyenne: 19));

    }

    @GetMapping(value=@"/liste")
    public Collection<Etudiant> getAllEtudiant(){
        return liste;
    }
    @GetMapping(value=@"/getEtudiant")
    public Etudiant getEtudiant(int identifiant){}


```

SpringApplication

Console Beans Health Mappings Environment

2025-03-28T15:39:29.415+01:00 ERROR 12548 --- [spring_app] [nio-9999-exec-5] o.a.c.c.C.[.].[/].[dispatcherServlet] : Serv

```

> java.lang.IndexOutOfBoundsException Create breakpoint: Index 4 out of bounds for length 4 <3 internal lines>
  at java.base/java.util.Objects.checkIndex(Objects.java:365) ~[na:na]
  at java.base/java.util.ArrayList.get(ArrayList.java:428) ~[na:na]
>  at com.example.spring_app.MyApi.getEtudiant(MyApi.java:41) ~[classes/:na] <12 internal lines>
>  at jakarta.servlet.http.HttpServlet.service(HttpServlet.java:564) ~[tomcat-embed-core-10.1.39.jar:6.0] <1 internal line>
>  at jakarta.servlet.http.HttpServlet.service(HttpServlet.java:658) ~[tomcat-embed-core-10.1.39.jar:6.0] <33 internal lines>
```

Cela s'explique par le fait que l'index 4 est hors limites pour la liste, qui ne contient que 4 éléments indexés de 0 à 3.

GET Send

Params • Authorization Headers (6) Body Scripts Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
<input checked="" type="checkbox"/> identifiant	1		
Key	Value	Description	

Body Cookies Headers (5) Test Results

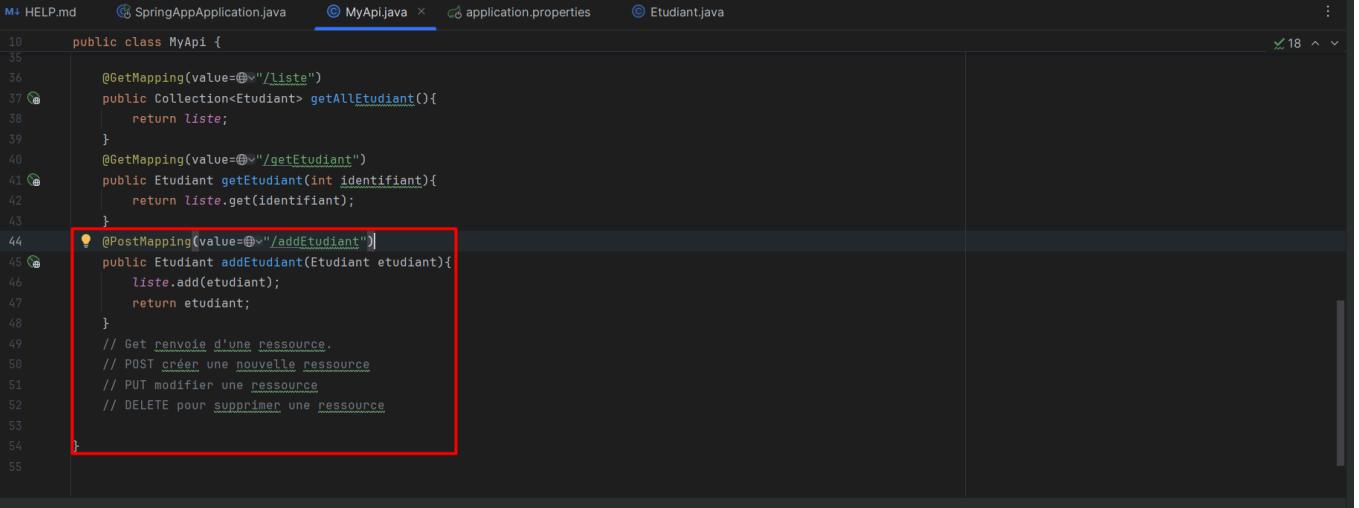
200 OK • 6 ms • 206 B • Save Response

{ } JSON ▾ Preview Visualize

```

1 {
2     "identifiant": 1,
3     "nom": "A",
4     "moyenne": 19.0
5 }
```

La requête GET envoyée sur localhost:9999/getEtudiant?identifiant=1 en utilisant Postman retourne à nouveau l'objet Etudiant avec identifiant : 1, nom : "A", moyenne : 19.0, et le statut HTTP est 200 OK, confirmant le bon fonctionnement pour cet index valide.



```

10  public class MyApi {
11
12      @GetMapping(value=@value"/liste")
13      public Collection<Etudiant> getAllEtudiant(){
14          return liste;
15      }
16
17      @GetMapping(value=@value"/getEtudiant")
18      public Etudiant getEtudiant(int identifiant){
19          return liste.get(identifiant);
20      }
21
22      @PostMapping(value=@value"/addEtudiant")
23      public Etudiant addEtudiant(Etudiant etudiant){
24          liste.add(etudiant);
25          return etudiant;
26      }
27
28      // Get renvoie d'une ressource.
29      // POST créer une nouvelle ressource
30      // PUT modifier une ressource
31      // DELETE pour supprimer une ressource
32  }
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55

```

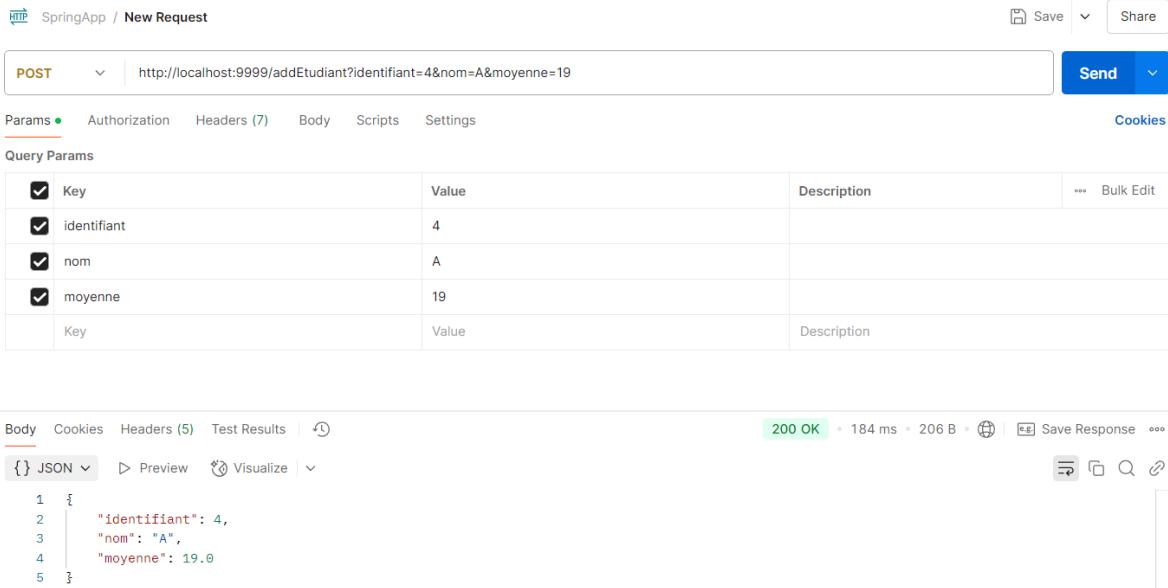
The screenshot shows the MyApi.java file in a Java IDE. The code defines a MyApi class with several methods: getAllEtudiant, getEtudiant, and addEtudiant. The addEtudiant method is annotated with @PostMapping. A red box highlights this annotation. Below the code, there is a Run tab showing the application's logs.

```

Run SpringAppApplication x
Console Beans Health Mappings Environment
2025-03-28T15:45:12.331+01:00 INFO 12016 --- [spring_app] [main] o.a.c.c.C.[Tomcat].[localhost].[]: Starting Servlet Engine: Apache Tomcat/10.1.0
2025-03-28T15:45:12.332+01:00 INFO 12016 --- [spring_app] [main] w.s.c.ServletWebServerApplicationContext: Initializing Spring embedded WebApplicationContext: initialization complete
2025-03-28T15:45:12.661+01:00 INFO 12016 --- [spring_app] [main] o.s.b.w.embedded.tomcat.TomcatWebServer: Tomcat started on port 9999 (http) with context path
2025-03-28T15:45:12.667+01:00 INFO 12016 --- [spring_app] [main] c.e.spring_app.SpringAppApplication: Started SpringAppApplication in 1.675 seconds (proc)

```

Un nouveau point de terminaison POST a été ajouté à l'URL /addEtudiant. Cette méthode prend un objet Etudiant en paramètre, l'ajoute à la liste existante, puis renvoie cet objet.



HTTP SpringApp / New Request

POST <http://localhost:9999/addEtudiant?identifiant=4&nom=A&moyenne=19>

Params • Authorization Headers (7) Body Scripts Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
identifiant	4		
nom	A		
moyenne	19		
Key	Value	Description	

Body Cookies Headers (5) Test Results

200 OK • 184 ms • 206 B • Save Response

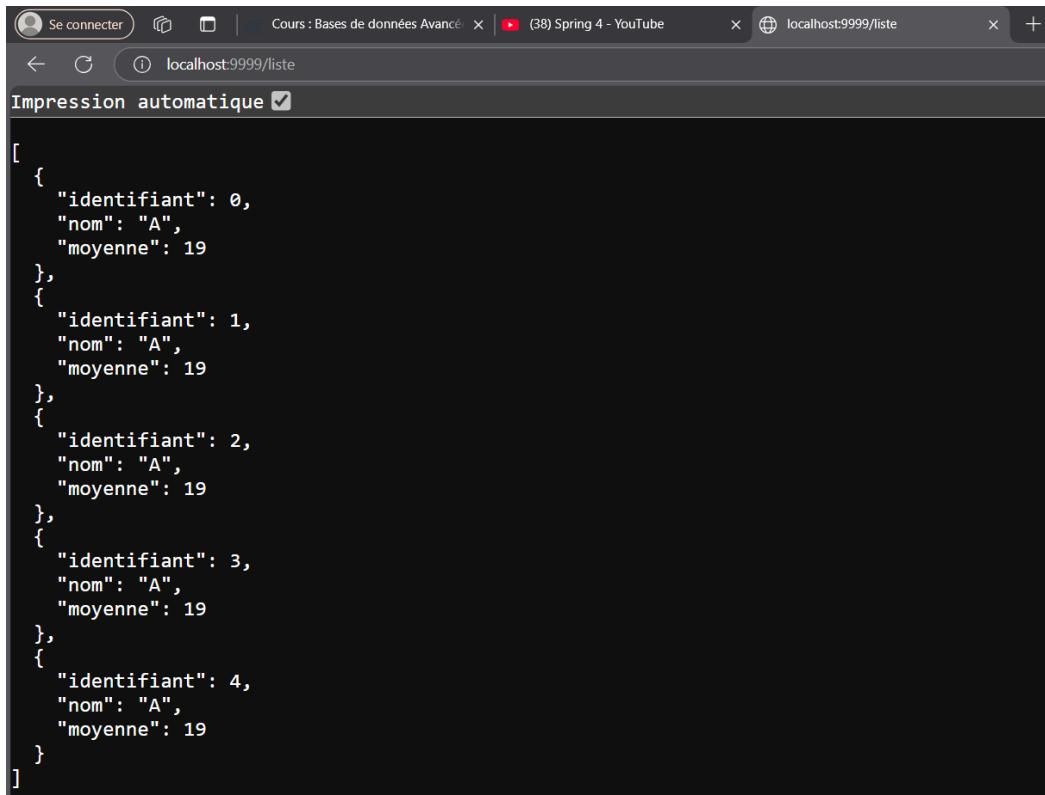
{ } JSON ▶ Preview ⚡ Visualize

```

1  {
2     "identifiant": 4,
3     "nom": "A",
4     "moyenne": 19.0
5 }

```

L'envoi d'une requête POST vers l'URL /addEtudiant avec les paramètres identifiant=4, nom=A, et moyenne=19 permet d'ajouter un nouvel étudiant à la collection. La réponse confirme l'ajout avec le retour de l'objet JSON correspondant à l'étudiant inséré.

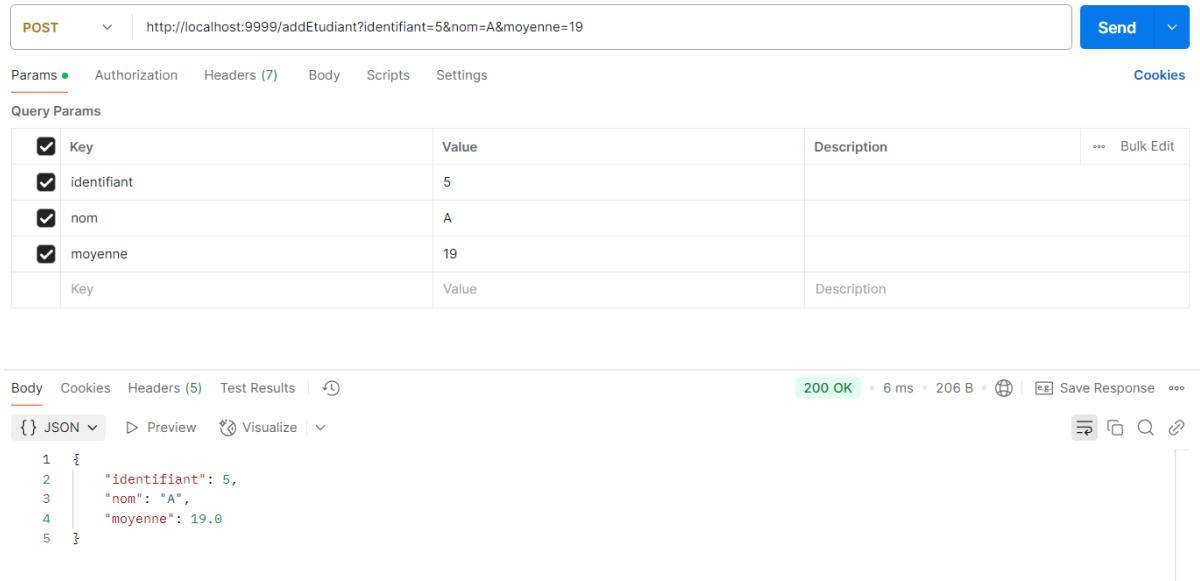


```

[{"identifiant": 0, "nom": "A", "moyenne": 19}, {"identifiant": 1, "nom": "A", "moyenne": 19}, {"identifiant": 2, "nom": "A", "moyenne": 19}, {"identifiant": 3, "nom": "A", "moyenne": 19}, {"identifiant": 4, "nom": "A", "moyenne": 19}]

```

L'appel à l'endpoint /liste montre que l'étudiant avec identifiant=4 a bien été ajouté à la collection, confirmant que la requête POST /addEtudiant a été traitée avec succès. La liste contient désormais cinq étudiants avec des identifiants allant de 0 à 4.



POST <http://localhost:9999/addEtudiant?identifiant=5&nom=A&moyenne=19>

Params	Authorization	Headers (7)	Body	Scripts	Settings	Cookies
<input checked="" type="checkbox"/> Query Params						
<input checked="" type="checkbox"/> identifiant			5			
<input checked="" type="checkbox"/> nom			A			
<input checked="" type="checkbox"/> moyenne			19			
<input type="checkbox"/> Key			Value		Description	*** Bulk Edit

Body Cookies Headers (5) Test Results 200 OK 6 ms 206 B Save Response

```

1 {
2   "identifiant": 5,
3   "nom": "A",
4   "moyenne": 19.0
5 }

```

L'étudiant avec identifiant = 5 a été ajouté avec succès à la collection, comme confirmé par la réponse 200 OK contenant les données envoyées. La requête POST a été réalisée avec Postman.

```

[{"identifiant": 0, "nom": "A", "moyenne": 19}, {"identifiant": 1, "nom": "A", "moyenne": 19}, {"identifiant": 2, "nom": "A", "moyenne": 19}, {"identifiant": 3, "nom": "A", "moyenne": 19}, {"identifiant": 4, "nom": "A", "moyenne": 19}, {"identifiant": 5, "nom": "A", "moyenne": 19}]

```

L'étudiant avec identifiant = 5 est bien présent dans la liste renvoyée, ce qui confirme son ajout réussi à la collection.

```

public class MyApi {
    @GetMapping(value=@"/getEtudiant")
    public Etudiant getEtudiant(int identifiant){
        return liste.get(identifiant);
    }
    @PostMapping(value=@"/addEtudiant")
    public Etudiant addEtudiant(Etudiant etudiant){
        liste.add(etudiant);
        return etudiant;
    }
    @DeleteMapping(value=@"/delete")
    public void supprimerEtudiant(int identifiant){
        liste.remove(identifiant);
    }
}

```

Run SpringAppApplication

```

2025-03-28T15:54:09.582+01:00 INFO 16752 --- [spring_app] [main] o.apache.catalina.core.StandardService : Starting service [Tomcat]
2025-03-28T15:54:09.582+01:00 INFO 16752 --- [spring_app] [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/9.0.62]
2025-03-28T15:54:09.623+01:00 INFO 16752 --- [spring_app] [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2025-03-28T15:54:09.624+01:00 INFO 16752 --- [spring_app] [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialized at [2025-03-28T15:54:09.624+01:00]
2025-03-28T15:54:09.948+01:00 INFO 16752 --- [spring_app] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 9999 (http://localhost:9999)
2025-03-28T15:54:09.954+01:00 INFO 16752 --- [spring_app] [main] c.e.spring_app.SpringApplication : Started SpringApplication in 1.119 seconds (local)

```

On ajoute `@DeleteMapping` pour supprimer un étudiant à partir de son identifiant. Cette méthode utilise la fonction `remove` de la liste pour supprimer l'étudiant correspondant.

The screenshot shows the Postman interface. At the top, it says "HTTP SpringApp / New Request". Below that, a "DELETE" button is selected, and the URL is "http://localhost:9999/delete?identifiant=1". Under "Params", there is a table with one row containing "identifiant" and the value "1". In the "Body" tab, the response is shown as a single character "1". The status bar at the bottom indicates "200 OK" and "24 ms".

L'API utilise la méthode `@DeleteMapping` pour supprimer un étudiant de la liste à l'aide de son identifiant. Lors de l'envoi d'une requête `DELETE` avec l'identifiant 1, l'étudiant correspondant est retiré de la liste et la réponse `200 OK` confirme que l'opération s'est déroulée avec succès.

The screenshot shows a browser window with the address bar set to "localhost:9999/liste". The page content displays a JSON array of student objects:

```
[{"identifiant": 0, "nom": "A", "moyenne": 19}, {"identifiant": 2, "nom": "A", "moyenne": 19}, {"identifiant": 3, "nom": "A", "moyenne": 19}]
```

On peut voir que l'étudiant avec l'identifiant 1 a bien été supprimé de la liste, car il n'apparaît plus dans la réponse JSON à la requête GET sur `/liste`. Les étudiants avec les identifiants 0, 2 et 3 sont toujours présents.

The screenshot shows the Postman interface. At the top, it says "SpringApp / New Request". Below that, the method is set to "DELETE" and the URL is "http://localhost:9999/delete?identifiant=0". Under "Params", there is a "Query Params" section with two entries: "identifiant" set to "0" and "Key" set to "Value". On the right side, there are buttons for "Save", "Share", "Send", and other options like "Cookies" and "Headers". Below the main request area, there's a preview of the response: "Body" shows "1", "Cookies" shows "", "Headers (4)" shows "Content-Type: application/json", "Test Results" shows "200 OK", and "Save Response" is available.

On supprime l'étudiant avec l'id 0.

The screenshot shows a browser window with the address bar at "localhost:9999/liste". The page content displays a JSON array with two elements:

```
[
  {
    "identifiant": 2,
    "nom": "A",
    "moyenne": 19
  },
  {
    "identifiant": 3,
    "nom": "A",
    "moyenne": 19
  }
]
```

On remarque notamment qu'il n'existe plus dans le retour de /liste.

The screenshot shows the IntelliJ IDEA interface. In the top navigation bar, "spring_app" is selected. The code editor shows "MyApi.java" with the following content:

```

public class MyApi {
    ...
    @GetMapping(value = "/addEtudiant")
    public Etudiant addEtudiant(Etudiant etudiant) {
        liste.add(etudiant);
        return etudiant;
    }

    @DeleteMapping(value = "/delete")
    public void supprimerEtudiant(int identifiant) {
        liste.remove(identifiant);
    }

    ...
}

```

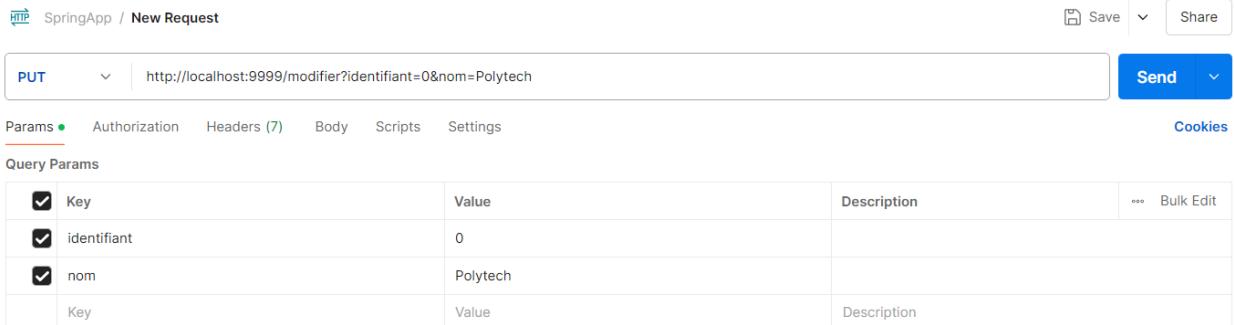
Below the code editor is the "Run" tool window, which shows the "SpringApplication" configuration and the logs. The logs output the following information:

```

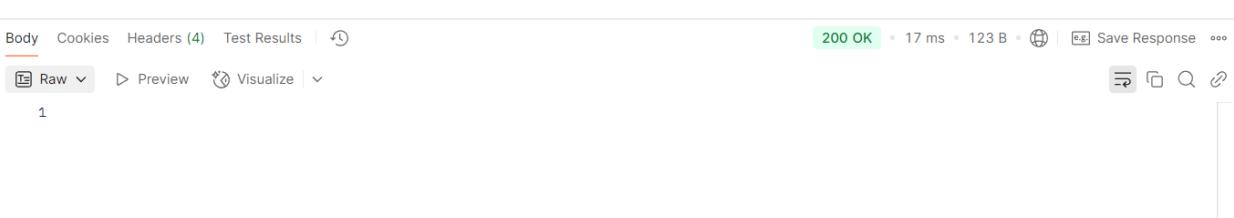
2025-03-28T15:54:09.948+01:00 INFO 16752 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 9999 (http) with context path ''
2025-03-28T15:54:09.954+01:00 INFO 16752 --- [main] c.e.spring_app.SpringApplication : Started SpringApplication in 1.68 seconds (process time)
2025-03-28T15:55:14.466+01:00 INFO 16752 --- [nio-9999-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Servlet 'dispatcherServlet'
2025-03-28T15:55:14.467+01:00 INFO 16752 --- [nio-9999-exec-1] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
2025-03-28T15:55:27.732+01:00 WARN 16752 --- [nio-9999-exec-4] .w.s.m.s.DefaultHandlerExceptionResolver : Resolved [org.springframework.web.HttpRequestMethodNotSupportedException]

```

On réalise la création d'une nouvelle API en ajoutant une méthode PUT qui permet de modifier un étudiant existant dans la liste. Cette méthode est associée à l'URL /modifier et prend en paramètres l'identifiant de l'étudiant ainsi qu'un nouveau nom. Le code récupère l'étudiant correspondant à l'identifiant fourni et met à jour son nom en utilisant la méthode setNom().



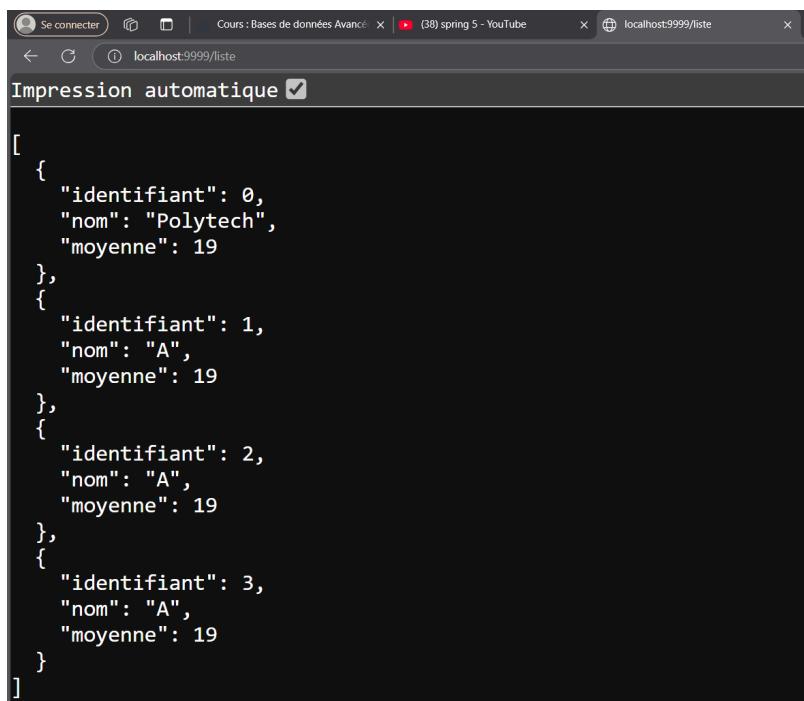
The screenshot shows a POSTMAN interface. The URL is `http://localhost:9999/modifier?identifiant=0&nom=Polytech`. The method is `PUT`. The `Query Params` section contains two entries: `identifiant` with value `0` and `nom` with value `Polytech`.



The screenshot shows the response details. The status is `200 OK` with a response time of `17 ms` and a size of `123 B`. The response body is a JSON array:

```
[{"identifiant": 0, "nom": "Polytech", "moyenne": 19}, {"identifiant": 1, "nom": "A", "moyenne": 19}, {"identifiant": 2, "nom": "A", "moyenne": 19}, {"identifiant": 3, "nom": "A", "moyenne": 19}]
```

Nous réalisons une requête PUT pour modifier les informations d'un étudiant dans la liste. La méthode PUT est utilisée pour mettre à jour un étudiant existant. Dans ce cas, l'identifiant de l'étudiant à modifier est 0 et son nom est mis à jour pour "Polytech". La requête est envoyée à l'URL /modifier avec les paramètres nécessaires : identifiant=0 et nom=Polytech. La réponse indique un statut 200 OK, confirmant que la modification a été effectuée avec succès.



The screenshot shows a browser window with the URL `localhost:9999/liste`. The page displays a JSON array of student records:

```
[{"identifiant": 0, "nom": "Polytech", "moyenne": 19}, {"identifiant": 1, "nom": "A", "moyenne": 19}, {"identifiant": 2, "nom": "A", "moyenne": 19}, {"identifiant": 3, "nom": "A", "moyenne": 19}]
```

On remarque que suite à la mise à jour effectuée précédemment via la méthode PUT, le nom de l'étudiant avec l'identifiant 0 a bien été modifié pour "Polytech", comme attendu. Les autres étudiants conservent leurs informations inchangées, ce qui montre que la modification s'est limitée à l'étudiant spécifié dans la requête. Cela confirme que l'API fonctionne correctement pour effectuer des mises à jour ciblées.

The screenshot shows the Postman interface with the following details:

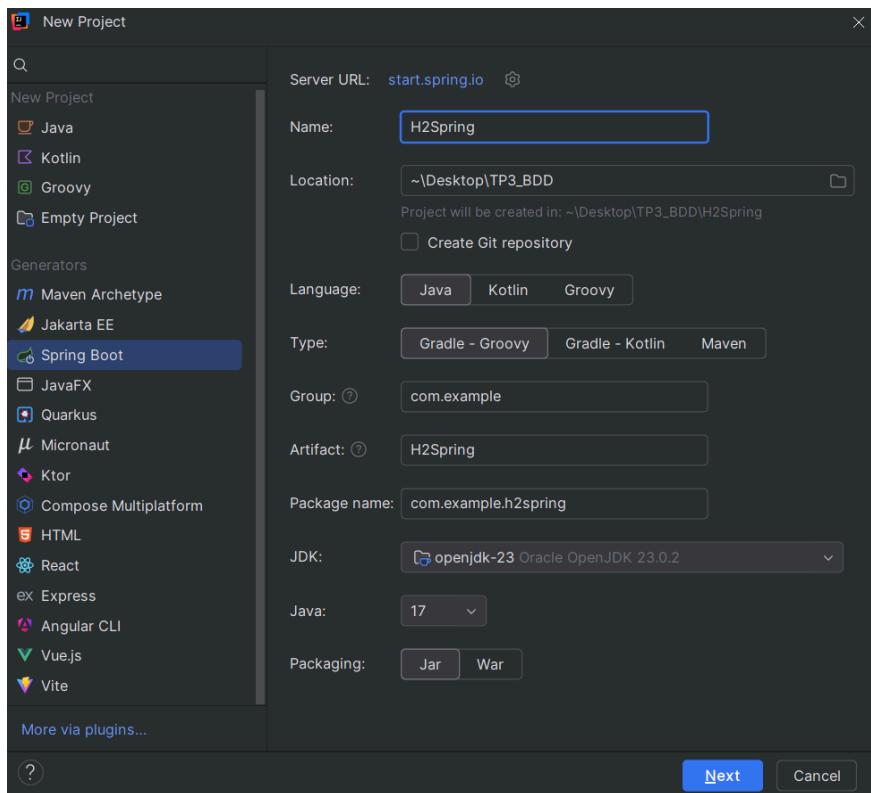
- Method:** PUT
- URL:** <http://localhost:9999/modifier?identifiant=1&nom=Polytech>
- Params:** Key, identifiant (Value: 1), nom (Value: Polytech)
- Response Status:** 200 OK
- Response Time:** 140 ms
- Response Size:** 123 B
- Content:** A single digit '1' indicating the success of the update.

On modifie le nom de l'étudiant avec l'identifiant 1 par "Polytech". Le statut 200 indique la bonne exécution de la requête.

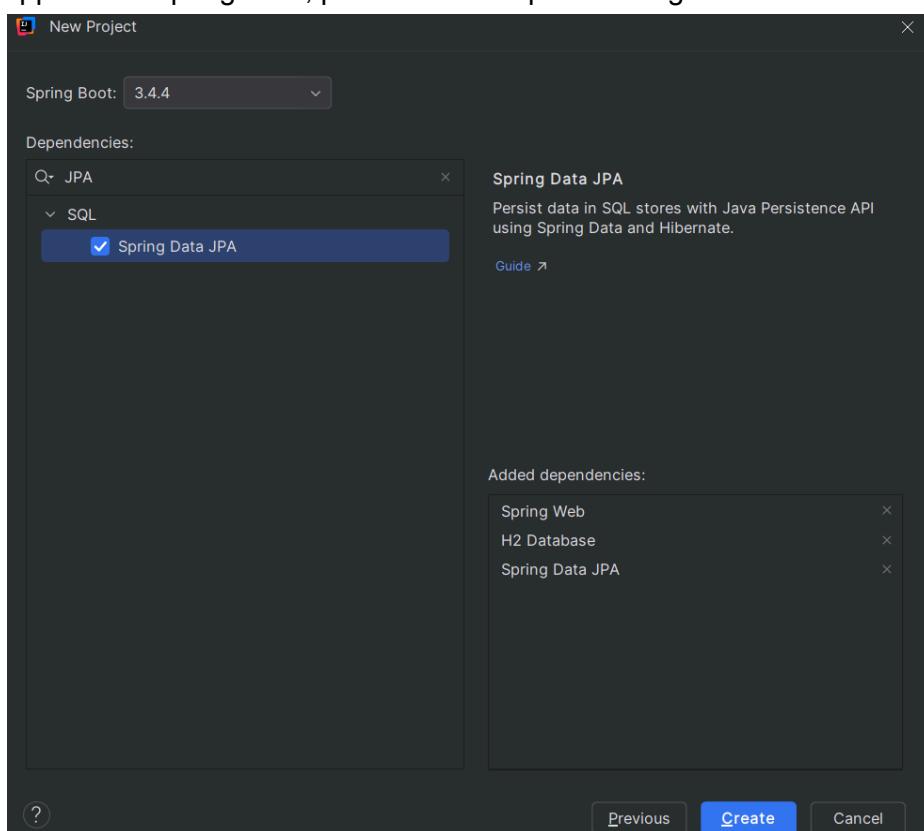
The browser window displays the following JSON response:

```
[
  {
    "identifiant": 0,
    "nom": "A",
    "moyenne": 19
  },
  {
    "identifiant": 1,
    "nom": "Polytech",
    "moyenne": 19
  },
  {
    "identifiant": 2,
    "nom": "A",
    "moyenne": 19
  },
  {
    "identifiant": 3,
    "nom": "A",
    "moyenne": 19
  }
]
```

L'étudiant avec l'id 1 dispose ainsi du nom "Polytech".



On crée un nouveau projet Spring Boot intitulé "H2Spring", en utilisant Java et Maven comme système de build. Le projet est configuré avec le groupe com.example et l'artefact nommé "H2Spring". Le packaging est défini en JAR, et Java 17 avec OpenJDK 23.0.2 est sélectionné comme version du JDK. Ce projet sera utilisé comme base pour développer une application Spring Boot, potentiellement pour interagir avec une base de données H2.



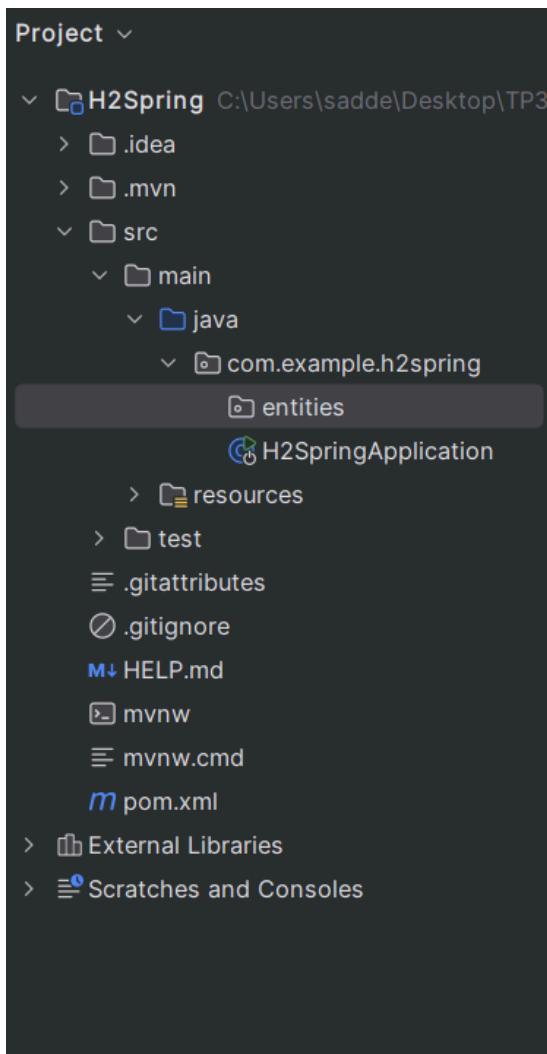
Dans cette étape, on configure le projet Spring Boot en utilisant la version 3.4.4. On sélectionne les dépendances nécessaires, à savoir "Spring Web" pour le développement d'applications web avec Spring MVC, "Spring Data JPA" pour faciliter l'accès aux données et la gestion des entités, ainsi que "H2 Database" pour intégrer une base de données en mémoire H2.

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>

<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
</dependencies>

<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
        </plugin>
    </plugins>
```

On vérifie le contenu du fichier pom.xml. On y trouvera ainsi les dépendances ajoutées précédemment ainsi que la dépendance de test et un plugin maven.



On crée un package entities qui contiendra les entités du projet.

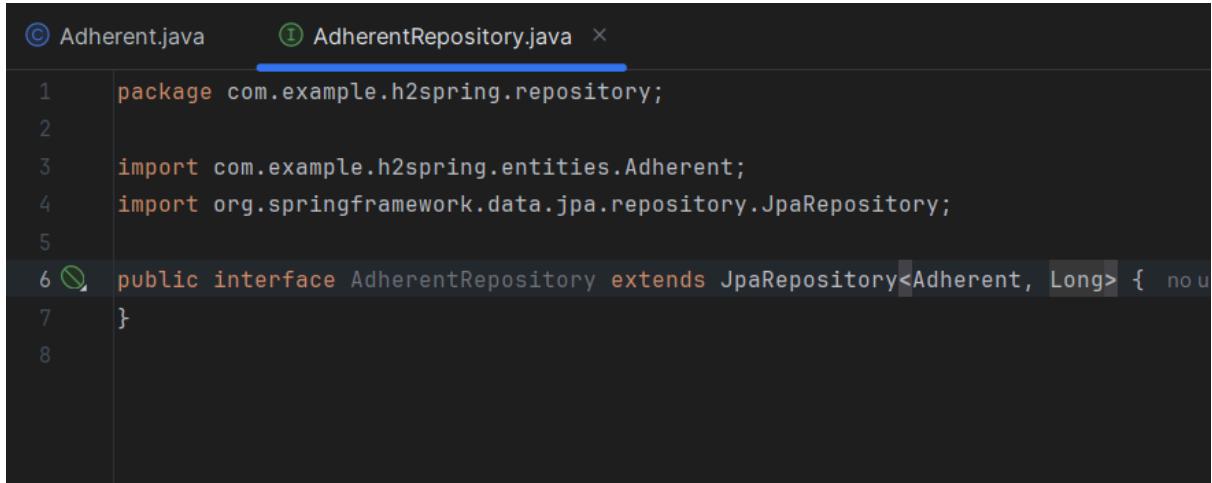
The screenshot shows the IntelliJ IDEA code editor with two tabs: "Adherent.java" and "AdherentRepository.java". The "Adherent.java" tab is active, displaying the following code:

```
1 package com.example.h2spring.entities;
2 // Hibernate une des implémentations de JPA
3 import jakarta.persistence.Entity;
4 import jakarta.persistence.Id;
5
6 @Entity
7 public class Adherent {
8     @Id
9     private Long id;
10    private String nom; no usages
11    private String ville; no usages
12    private int age; no usages
13 }
14
```

The "Adherent" class is annotated with `@Entity` and `@Id`. The "AdherentRepository" class is shown in the adjacent tab.

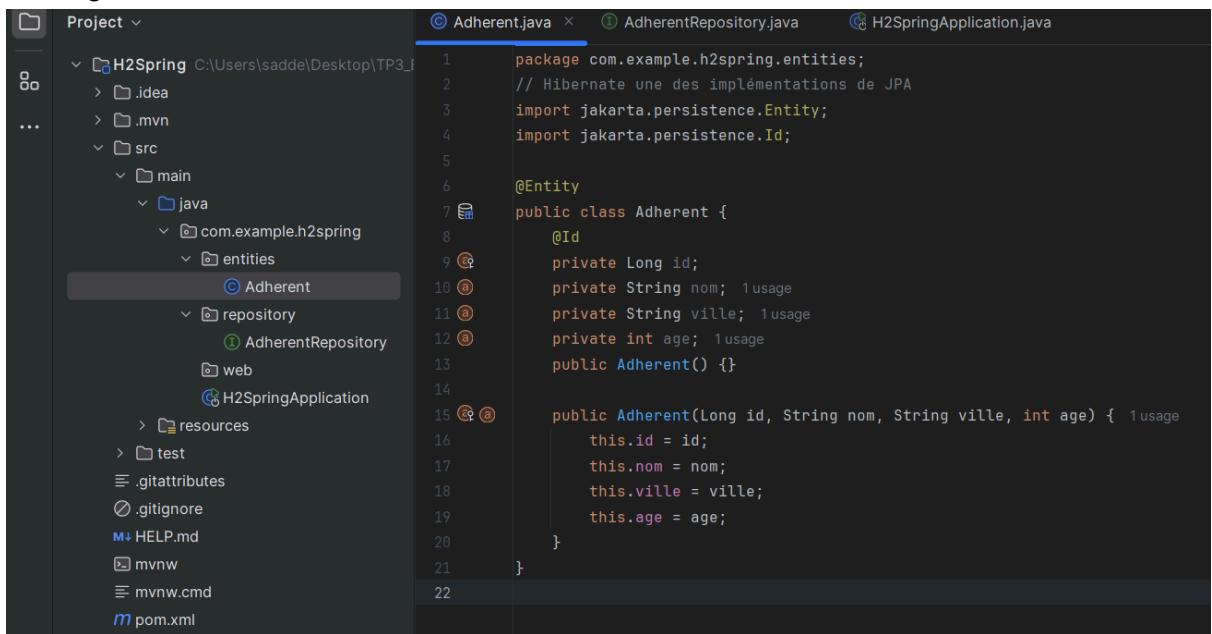
On crée une entité Adherent avec les champs id, nom, ville et âge. L'annotation `@Entity` est utilisée pour indiquer que cette classe représente une entité JPA, et l'annotation `@Id` est

appliquée au champ id pour en faire la clé primaire.



```
1 package com.example.h2spring.repository;
2
3 import com.example.h2spring.entities.Adherent;
4 import org.springframework.data.jpa.repository.JpaRepository;
5
6 public interface AdherentRepository extends JpaRepository<Adherent, Long> { no u
7 }
8
```

On crée un AdherentRepository qui étend l'interface JpaRepository fournie par Spring Data JPA. Cela permet de bénéficier d'un ensemble de méthodes prédéfinies pour effectuer des opérations de base sur l'entité Adherent, comme la récupération, la suppression et la mise à jour des adhérents dans la base de données. L'interface JpaRepository prend deux paramètres génériques : l'entité à gérer, ici Adherent, et le type de la clé primaire de l'entité, ici Long.

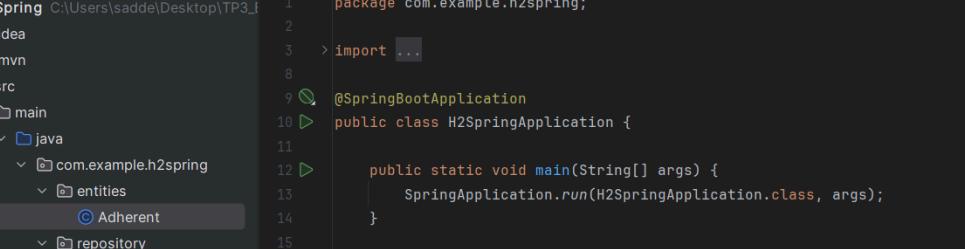


Project

- H2Spring
- .idea
- .mvn
- src
 - main
 - java
 - com.example.h2spring
 - entities
 - Adherent
 - repository
 - AdherentRepository
 - web
 - H2SpringApplication
 - resources
 - test
 - .gitattributes
 - .gitignore
 - HELP.md
 - mvnw
 - mvnw.cmd
 - pom.xml

```
1 package com.example.h2spring.entities;
2 // Hibernate une des implémentations de JPA
3 import jakarta.persistence.Entity;
4 import jakarta.persistence.Id;
5
6 @Entity
7 public class Adherent {
8     @Id
9     private Long id;
10    private String nom; 1usage
11    private String ville; 1usage
12    private int age; 1usage
13    public Adherent() {}
14
15    public Adherent(Long id, String nom, String ville, int age) { 1usage
16        this.id = id;
17        this.nom = nom;
18        this.ville = ville;
19        this.age = age;
20    }
21}
```

On ajoute un constructeur par défaut et un constructeur paramétré à l'adhérent.



The screenshot shows a Java development environment with the following details:

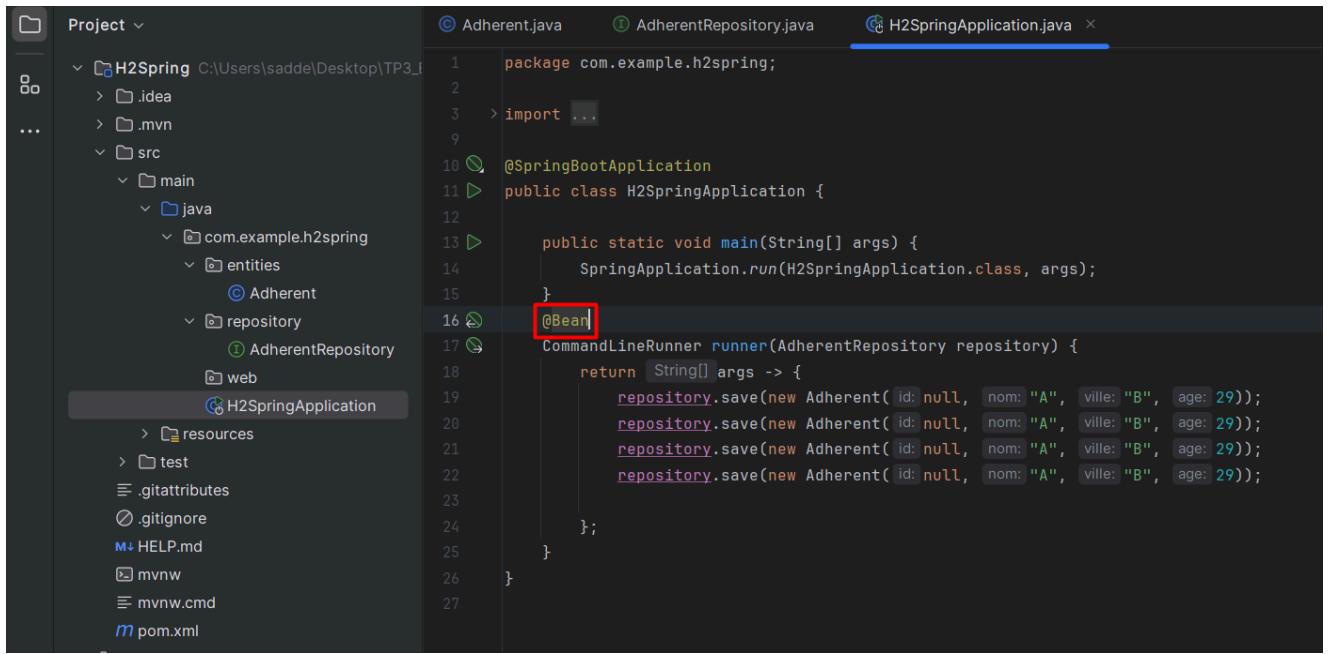
- Project Tree:** The project structure is displayed on the left, showing files like `H2Spring`, `.idea`, `.mvn`, `src/main/java/com/example/h2spring/entities/Adherent`, `repository/AdherentRepository`, `web/H2SpringApplication`, `resources`, `test`, `.gitattributes`, `.gitignore`, `HELP.md`, `mvnw`, `mvnw.cmd`, and `pom.xml`.
- Code Editor:** The main window displays the `H2SpringApplication.java` file. The cursor is positioned at line 21, where the code is as follows:

```
repository.save(new Adherent(id: null, nom: "A", ville: "B", age: 29));  
repository.save(new Adherent(id: null, nom: "A", ville: "B", age: 29));  
repository.save(new Adherent(id: null, nom: "A", ville: "B", age: 29));  
repository.save(new Adherent(id: null, nom: "A", ville: "B", age: 29));
```
- Toolbars and Status:** The top bar includes icons for file operations and tabs for `Adherent.java`, `AdherentRepository.java`, and `H2SpringApplication.java`. The status bar on the right shows the file name `H2SpringApplication.java`.

Au niveau de la classe principale H2SpringApplication qui est annotée avec @SpringBootApplication. On initialise un CommandLineRunner, qui est une interface fournie par Spring pour exécuter un code une fois que le contexte de l'application est entièrement initialisé. Dans ce cas, on utilise le repository.save() pour enregistrer plusieurs objets Adherent dans la base de données H2. Chaque objet Adherent est créé avec un id de valeur null, ce qui laisse à Spring le soin d'assigner un identifiant unique (automatiquement généré grâce à la configuration de JPA). Ces objets sont ensuite enregistrés avec des informations sur le nom, la ville et l'âge. Cela permet de tester la fonctionnalité de persistance dans la base de données après le démarrage de l'application.

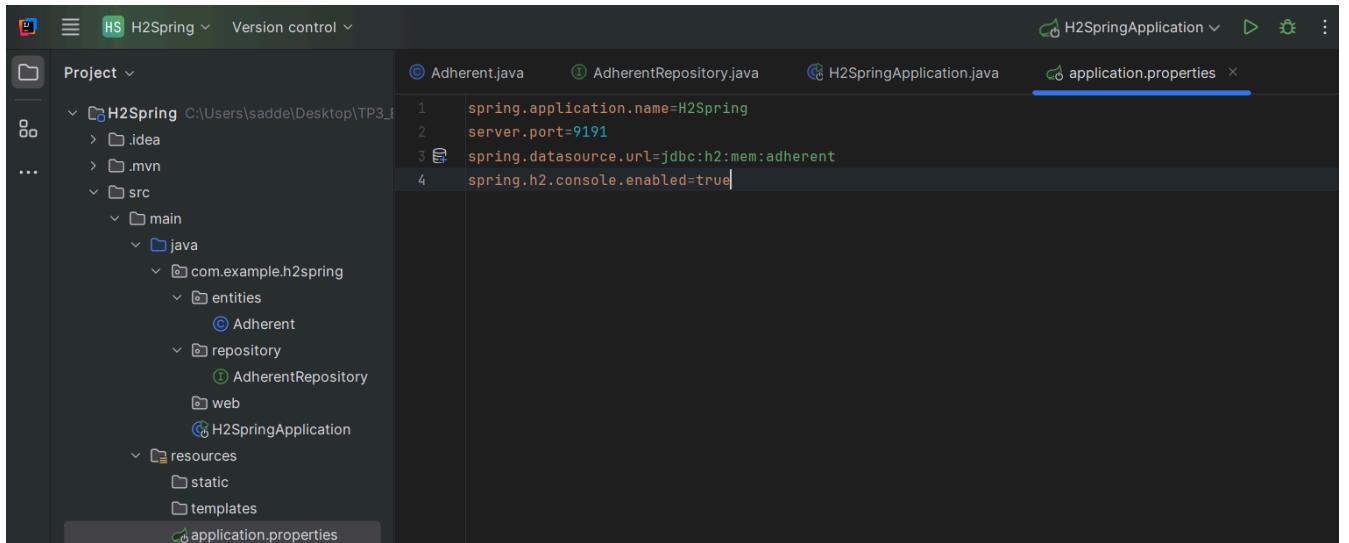
```
④ Adherent.java x ⑤ AdherentRepository.java ⑥ H2SpringApplication.java
1 package com.example.h2spring.entities;
2 // Hibernate une des implémentations de JPA
3 import jakarta.persistence.Entity;
4 import jakarta.persistence.GeneratedValue;
5 import jakarta.persistence.GenerationType;
6 import jakarta.persistence.Id;
7
8 @Entity
9
10 @Id
11 @GeneratedValue(strategy = GenerationType.AUTO)
12 private Long id;
13 private String nom; 1 usage
14 private String ville; 1 usage
15 private int age; 1 usage
16 public Adherent() {}
17
18 public Adherent(Long id, String nom, String ville, int age) { 4 usages
19     this.id = id;
20     this.nom = nom;
21     this.ville = ville;
22     this.age = age;
23 }
24
25 }
```

On ajoute l'annotation `@GeneratedValue(strategy = GenerationType.AUTO)` à la propriété `id` de la classe `Adherent`. Cette annotation permet à la base de données de générer automatiquement une valeur unique pour chaque nouvel enregistrement d'adhérent.



```
1 package com.example.h2spring;
2
3 import ...
4
5 @SpringBootApplication
6 public class H2SpringApplication {
7
8     public static void main(String[] args) {
9         SpringApplication.run(H2SpringApplication.class, args);
10    }
11
12    @Bean
13    CommandLineRunner runner(AdherentRepository repository) {
14        return String[] args -> {
15            repository.save(new Adherent(id: null, nom: "A", ville: "B", age: 29));
16            repository.save(new Adherent(id: null, nom: "A", ville: "B", age: 29));
17            repository.save(new Adherent(id: null, nom: "A", ville: "B", age: 29));
18            repository.save(new Adherent(id: null, nom: "A", ville: "B", age: 29));
19        };
20    }
21
22    }
23
24    }
25
26    }
27}
```

On ajoute l'annotation `@Bean` avant la méthode `CommandLineRunner runner` dans la classe `H2SpringApplication`. Cette annotation permet de déclarer cette méthode comme un bean Spring, ce qui permet à Spring de l'exécuter au démarrage de l'application.



```
1 spring.application.name=H2Spring
2 server.port=9191
3 spring.datasource.url=jdbc:h2:mem:adherent
4 spring.h2.console.enabled=true
```

On configure les propriétés de l'application dans le fichier `application.properties`. Tout d'abord, on définit le nom de l'application avec `spring.application.name=H2Spring`. Ensuite, on spécifie le port du serveur avec `server.port=9191`. Pour connecter l'application à la base de données H2 en mémoire, on utilise `spring.datasource.url=jdbc:h2:mem:adherent`. Finalement, on active la console H2 avec `spring.h2.console.enabled=true`, permettant ainsi d'accéder à l'interface H2 pour interagir avec la base de données depuis un navigateur.

```

2025-03-28T16:50:16.230+01:00 INFO 11576 --- [H2Spring] [           main] o.h.c.internal.RegionFactoryInitiator : HHH000026: Second-level cache disabled
2025-03-28T16:50:16.544+01:00 INFO 11576 --- [H2Spring] [           main] o.s.o.j.p.SpringPersistenceUnitInfo : No LoadTimeWeaver setup: ignoring JPA class transform
2025-03-28T16:50:16.627+01:00 INFO 11576 --- [H2Spring] [           main] org.hibernate.orm.connections.pooling : HHH10001005: Database info:
Database JDBC URL [Connecting through datasource 'HikariDataSource (HikariPool-1)']
Database driver: undefined/unknown
Database version: 2.3.232
Autocommit mode: undefined/unknown
Isolation level: undefined/unknown
Minimum pool size: undefined/unknown
Maximum pool size: undefined/unknown
2025-03-28T16:50:17.462+01:00 INFO 11576 --- [H2Spring] [           main] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000489: No JTA platform available (set 'hibernate.
2025-03-28T16:50:17.493+01:00 INFO 11576 --- [H2Spring] [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence
2025-03-28T16:50:17.735+01:00 WARN 11576 --- [H2Spring] [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Theref
2025-03-28T16:50:18.010+01:00 INFO 11576 --- [H2Spring] [           main] o.s.b.a.h2.H2ConsoleAutoConfiguration : H2 console available at '/h2-console'. Database avail
2025-03-28T16:50:18.075+01:00 INFO 11576 --- [H2Spring] [           main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 9191 (http) with context path
2025-03-28T16:50:18.082+01:00 INFO 11576 --- [H2Spring] [           main] c.example.h2spring.H2SpringApplication : Started H2SpringApplication in 4.154 seconds (process

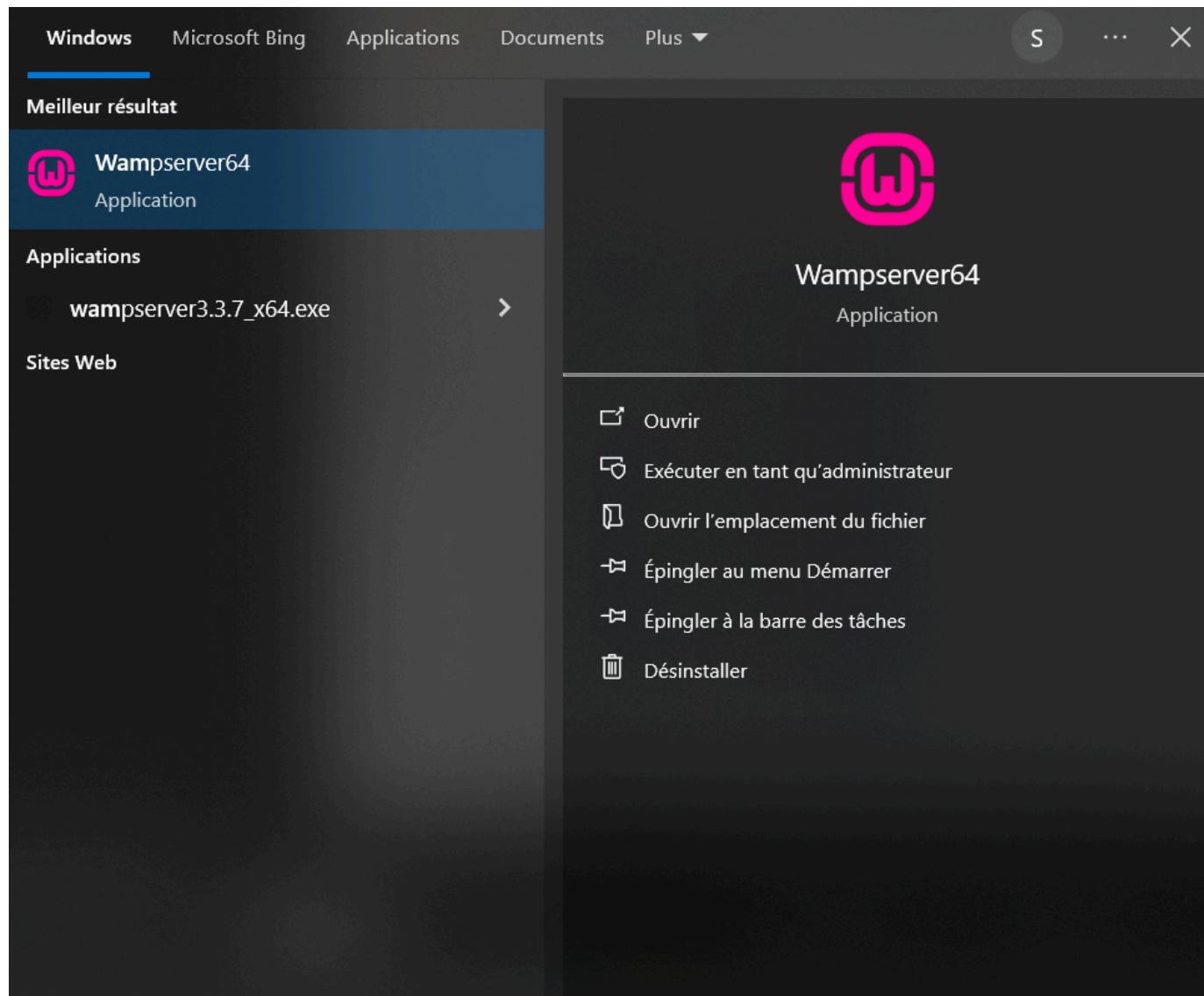
```

On peut observer dans les logs que le serveur Tomcat démarre sur le port 9191 et que la console H2 est activée et accessible via le chemin /h2-console.

On accède à la console H2 via l'URL localhost:9191/h2-console. Le test de connexion est réussi, ce qui indique que la connexion à la base de données en mémoire s'est bien établie. La configuration utilisée inclut le pilote H2, l'URL jdbc:h2:mem:adherent.

AGE	ID	NOM	VILLE
29	1	A	B
29	2	A	B
29	3	A	B
29	4	A	B

On observe que la base de données H2 contient une table appelée ADHERENT, avec quatre enregistrements. Chaque enregistrement a les colonnes AGE, ID, NOM, et VILLE. Les données affichées montrent que tous les adhérents ont l'âge de 29 ans, avec les mêmes valeurs pour le nom (A) et la ville (B). Cela confirme que l'insertion des données s'est déroulée correctement via le code de l'application, et que les informations sont stockées dans la base de données H2.



Pour mettre en place une communication avec une base de données MySQL, on accède tout d'abord à WampServer.

The screenshot shows the Maven Repository page for the MySQL Connector-J artifact. Key details include:

- Ranking:** #72 in MvnRepository (See Top Artifacts)
- Used By:** 8,201 artifacts
- Vulnerabilities:** Direct vulnerabilities: CVE-2023-22102
- This artifact was moved to:** com.mysql > mysql-connector-j
- Maven Coordinates:** <!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
 <dependency>
 <groupId>mysql</groupId>
 <artifactId>mysql-connector-java</artifactId>
 <version>8.0.33</version>
 </dependency>
- Scope:** Compile
- Include backlinks:** checked

On recherche la dépendance de MySQL à partir du mavenrepository pour pouvoir interagir avec la base de données MySQL en utilisant l'application Spring Boot.

The screenshot shows an IDE interface with several files open:

- Adherent.java
- AdherentRepository.java
- H2SpringApplication.java
- application.properties
- mvnw.cmd
- pom.xml (H2Spring) - This file is currently selected.

The pom.xml content is as follows:

```

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <dependencies> <!-- Add Starters... -->
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.33</version>
        </dependency>
        <dependency>
            <groupId>com.h2database</groupId>
            <artifactId>h2</artifactId>
            <scope>runtime</scope>
        </dependency>
    </dependencies>

```

A red box highlights the MySQL dependency line (line 42). The bottom part of the screen shows the application's logs in the terminal:

```

2023-09-29T14:29:48.929+00:00 [H2Spring] [main] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000489: NO JTA platform available (set 'hibernate.transaction.jta.platform' to 'true' if you want to use JBoss Seam or similar)
2023-09-29T14:29:48.929+00:00 [H2Spring] [main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2023-09-29T14:29:49.029+00:00 [H2Spring] [main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may be performed during view rendering. Explicitly configure to disable this behavior
2023-09-29T14:29:49.029+00:00 [H2Spring] [main] o.s.b.a.h2.H2ConsoleAutoConfiguration : H2 console available at '/h2-console'. Database available at 'jdbc:h2:mem:SpringDataJpa'
2023-09-29T14:29:49.029+00:00 [H2Spring] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 9191 (http) with context path ''
2023-09-29T14:29:49.029+00:00 [H2Spring] [main] c.example.h2spring.H2SpringApplication : Started H2SpringApplication in 3.978 seconds (process PID 14448)
2023-09-29T14:29:49.029+00:00 [nio-9191-exec-6] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2023-09-29T14:29:49.029+00:00 [nio-9191-exec-6] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2023-09-29T14:29:49.029+00:00 [nio-9191-exec-6] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms

```

On ajoute la dépendance au niveau du fichier pom.xml et on installe par la suite les dépendances ajoutées.



On crée par la suite la base de données adherent.

The screenshot shows the IntelliJ IDEA interface. The left sidebar shows a project structure with files like Adherent.java, AdherentRepository.java, H2SpringApplication.java, application.properties, mvnw.cmd, and pom.xml. The application.properties file is open, showing the following configuration:

```
server.port=9191
spring.datasource.url=jdbc:mysql://localhost:3306/adherent
spring.datasource.username=root
spring.datasource.password=
spring.jpa.hibernate.ddl-auto=create
```

A red box highlights this configuration. The bottom part of the screen shows the "Run" tab with the application running. The console output shows the application starting up and connecting to the MySQL database:

```
2025-04-14T13:56:42.634+02:00 INFO 8984 --- [main] o.s.o.j.p.SpringPersistenceUnitInfo : No LoadTimeWeaver setup: ignoring JPA class transformer
2025-04-14T13:56:42.660+02:00 INFO 8984 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2025-04-14T13:56:43.015+02:00 INFO 8984 --- [main] com.zaxxer.hikari.pool.HikariPool : HikariPool-1 - Added connection com.mysql.cj.jdbc.ConnectionImpl@...
2025-04-14T13:56:43.019+02:00 INFO 8984 --- [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2025-04-14T13:56:43.120+02:00 INFO 8984 --- [main] org.hibernate.orm.connections.Pooling...
Database JDBC URL [Connecting through datasource 'HikariDataSource (HikariPool-1)']
Database driver: undefined/unknown
Database version: 9.1
Autocommit mode: undefined/unknown
Isolation level: undefined/unknown
Minimum pool size: undefined/unknown
Maximum pool size: undefined/unknown
2025-04-14T13:56:43.904+02:00 INFO 8984 --- [main] o.h.e.t.j.p.i.JtaPlatformInitiator : HHH000489: No JTA platform available (set 'hibernate.transaction...
2025-04-14T13:56:43.990+02:00 INFO 8984 --- [main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'defau...
2025-04-14T13:56:44.219+02:00 WARN 8984 --- [main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database...
2025-04-14T13:56:44.511+02:00 INFO 8984 --- [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 9191 (http) with context path '/'
2025-04-14T13:56:44.517+02:00 INFO 8984 --- [main] c.example.h2spring.H2SpringApplication : Started H2SpringApplication in 3.904 seconds (process running for...
```

On modifie la configuration de l'application pour utiliser une base de données MySQL au lieu de H2. Les informations de connexion sont spécifiées dans le fichier application.properties, où l'URL de la base de données, le nom d'utilisateur et le mot de passe ont été définis. L'application utilise désormais un serveur MySQL local (localhost:3306) avec un utilisateur root. De plus, la propriété spring.jpa.hibernate.ddl-auto=create est définie, ce qui permet à Hibernate de générer automatiquement les tables lors de l'exécution de l'application. Le processus de démarrage de l'application a été effectué avec succès et la base de données MySQL est désormais connectée à l'application Spring Boot.

The screenshot shows the phpMyAdmin interface for a MySQL server. The current database is 'adherent'. The 'adherent' table is selected, displaying 4 rows of data:

	id	nom	ville	age
<input type="checkbox"/>	1	A	B	29
<input type="checkbox"/>	2	A	B	29
<input type="checkbox"/>	3	A	B	29
<input type="checkbox"/>	4	A	B	29

Below the table, there are several buttons for managing the data: Éditer (Edit), Copier (Copy), Supprimer (Delete), Tout cocher (Select All), and options for filtering and exporting the results.

On remarque que l'application fonctionne avec une base de données MySQL, et les données des étudiants sont affichées via l'interface phpMyAdmin. La table adherent contient les informations des étudiants, notamment leur id, nom, ville, et âge. Les données relatives au CommandLineRunner configuré dans la classe principale ont été ajoutées avec succès.