TP1 : Développement d'une API REST avec Spring Boot et H2

Riham Kaddour Bakir

23 novembre 2025

Table des matières

1 Introduction

Le but de ce TP est de développer une API REST simple en utilisant **Spring Boot 4**, **Spring Data JPA** et une base de données **H2 en mémoire**. Nous allons créer une application capable de gérer des *Adhérents*, avec des opérations de lecture via un endpoint REST et une console web H2 pour visualiser les données.

Les notions principales abordées sont :

— Création d'un projet Spring Boot avec Maven.
— Utilisation de JPA pour la gestion des entités et de la persistance.
— Utilisation d'une base H2 en mémoire pour faciliter le développement.
— Création d'API REST pour exposer les données.
— Configuration de la console H2 pour visualiser la base en temps réel.

2 Configuration du projet

2.1 Structure Maven

Le projet est basé sur Maven et utilise le parent `spring-boot-starter-parent` version 4.0.0. Les dépendances principales incluent :

— `spring-boot-starter-web` : pour créer des services REST.
— `spring-boot-starter-data-jpa` : pour utiliser JPA/Hibernate.
— `spring-boot-starter-validation` : pour la validation des données.
— `com.h2database:h2` : base de données en mémoire.

2.2 application.properties

Pour configurer le projet, nous avons utilisé le fichier `application.properties` suivant :

```
server.port=8081

# Configuration H2
spring.datasource.url=jdbc:h2:mem:adherent
spring.datasource.username=sa
spring.datasource.password=
spring.datasource.driver-class-name=org.h2.Driver

spring.h2.console.enabled=true
spring.h2.console.path=/h2-console

# Configuration JPA/Hibernate
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
```

Explications :

— `spring.datasource.url` : URL de la base H2 en mémoire.
— `spring.h2.console.enabled=true` : active la console H2 web.
— `spring.jpa.hibernate.ddl-auto=update` : Hibernate crée automatiquement les tables.

— `spring.jpa.show-sql=true` : affiche les requêtes SQL dans la console.

3 Création de l'entité JPA

Nous avons créé une entité `Adherent` pour représenter les adhérents :

```java
package com.H2.H2.entity;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

@Entity
public class Adherent {
    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String nom;
    private String ville;
    private int age;

    public Adherent() {}
    public Adherent(Long id, String nom, String ville, int age) {
        this.id = id;
        this.nom = nom;
        this.ville = ville;
        this.age = age;
    }

    // Getters et Setters
}
```

Explications :
— `@Entity` : indique que la classe est une entité JPA.
— `@Id` et `@GeneratedValue` : pour gérer automatiquement la clé primaire.

4 Création du repository

Le repository est une interface qui hérite de `JpaRepository` :

```java
package com.H2.H2.repository;

import com.H2.H2.entity.Adherent;
import org.springframework.data.jpa.repository.JpaRepository;

public interface AdherentRepository extends JpaRepository<Adherent,
    Long> {}
```

Explications : Spring Data JPA fournit automatiquement les méthodes CRUD (save, findAll, findById, delete, etc.) sans avoir à les implémenter.

5 Classe principale H2Application

La classe principale initialise l'application et pré-remplit la base avec quelques adhérents :

```
package com.H2.H2;

import com.H2.H2.entity.Adherent;
import com.H2.H2.repository.AdherentRepository;
import org.springframework.boot.CommandLineRunner;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication
    ;
import org.springframework.context.annotation.Bean;

@SpringBootApplication
public class H2Application {

    public static void main(String[] args) {
        SpringApplication.run(H2Application.class, args);
    }

    @Bean
    CommandLineRunner run(AdherentRepository adherentRepository) {
        return args -> {
            adherentRepository.save(new Adherent(null, "riham", "
               Algerie", 22));
            adherentRepository.save(new Adherent(null, "ali", "
               Tunisie", 30));
            adherentRepository.save(new Adherent(null, "sara", "
               Maroc", 25));
        };
    }
}
```

Explications : Le `CommandLineRunner` permet d'exécuter du code au démarrage de l'application pour pré-remplir la base.

6 Création du contrôleur REST

Pour exposer les adhérents via une API REST, nous créons un contrôleur :

```
package com.H2.H2.controller;

import com.H2.H2.entity.Adherent;
import com.H2.H2.repository.AdherentRepository;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

import java.util.List;

@RestController
```

```java
public class AdherentController {

    private final AdherentRepository repository;

    public AdherentController(AdherentRepository repository) {
        this.repository = repository;
    }

    @GetMapping("/adherents")
    public List<Adherent> getAll() {
        return repository.findAll();
    }
}
```

Explications :

— `@RestController` : indique que cette classe expose des endpoints REST.

— `@GetMapping("/adherents")` : mappe la méthode sur l'URL '/adherents'.

—

7 Test de l'application

1. on Lance l'application : `mvn spring-boot:run`.
2. on Accéde à l'API REST : http://localhost:8081/adherents On obtient la liste des adhérents en JSON.
3. on Accéde à la console H2 : http://localhost:8081/h2-console/ JDBC URL : `jdbc:h2:mem:adhere` user : `sa`, mot de passe vide.

Note : Il est possible de remplacer H2 par une base MySQL en modifiant le fichier `application.properties`. Dans ce cas, il faudra activer la configuration MySQL (URL, utilisateur, mot de passe, driver et dialect Hibernate). Cette option permet d'utiliser une base persistante plutôt qu'une base en mémoire pour le développement ou la production. (la configuration de MYSQL est commenté dans le code rendu)

8 Conclusion

Ce TP constitue une introduction pratique au développement d'API REST avec Spring Boot et à l'utilisation de bases de données en mémoire.