Rapport – API REST & Microservices (Spring Boot + Java)

Objectif général

Ce TP avait pour but de comprendre les principes d'une API REST et de les mettre en pratique au sein d'un microservice développé avec Spring Boot.
L'objectif était de manipuler les routes HTTP, les entités, les opérations CRUD, la sérialisation JSON, ainsi que la persistance en mémoire puis en base de données.

PARTIE 1 — API REST AVEC SPRING BOOT

1.1 Génération du projet Spring Boot

Le projet a été créé via Spring Initializr dans IntelliJ IDEA.
Contrairement aux anciennes versions utilisées en cours, j'ai travaillé en :

- **Java 17**
- **Spring Boot 3.x**
- **Packaging JAR**
- **Dépendances : Spring Web, Spring Data JPA, H2 Database**

Spring Boot fournit automatiquement :

- **un serveur Tomcat intégré,**
- **un point d'entrée unique,**
- **une structure Maven standard,**
- **un fichier de configuration unique (application.properties).**

1.2 Création des premières routes REST

Une première série de routes REST a été créée pour comprendre le fonctionnement de @RestController et des annotations comme @GetMapping.

Les tests ont été réalisés via navigateur et via Postman.
Cela a servi d'introduction au fonctionnement d'une API REST (exposition de routes, paramètres, réponses JSON).

PARTIE 2 — COUCHE DONNÉES : ADHERENT + JPA

2.1 Mise en place d'une entité : Adherent

Le modèle principal du TP est la classe Adherent, contenant :

- **un identifiant,**
- **un nom,**
- **une ville,**
- **un âge.**

Cette classe est annotée comme entité JPA, ce qui permet de la relier à une table automatiquement générée par Spring lors du démarrage.

2.2 Création du Repository

Un repository a été ajouté pour permettre les opérations CRUD sur la table correspondante.
Grâce à Spring Data JPA, il n'a pas été nécessaire d'écrire du SQL : les opérations courantes sont générées automatiquement.

2.3 Configuration de l'initialisation des données

Un composant Spring a été utilisé pour remplir la base dès le démarrage de l'application, avec plusieurs adhérents créés automatiquement.

Cette étape permettait de tester la persistance dès la première exécution.

PARTIE 3 — TESTS AVEC H2

3.1 Configuration H2

La première base utilisée est H2, une base en mémoire.
L'avantage est qu'elle ne nécessite aucune installation externe.

La configuration contenait notamment :

- **le port du serveur,**
- **l'URL JDBC H2,**
- **l'activation de la console H2.**

3.2 Accès via la console H2

Une interface graphique accessible depuis le navigateur permettait :

- **d'afficher les tables générées,**
- **de visualiser les données insérées par Spring,**
- **de valider la bonne création de la table ADHERENT.**

Les tests ont confirmé que la connexion, la génération du schéma et l'insertion initiale fonctionnaient correctement.

PARTIE 4 — PASSAGE À MYSQL

4.1 Ajout du connecteur MySQL

Une dépendance a été ajoutée pour permettre à Spring Boot de communiquer avec MySQL.

4.2 Modification de la configuration

C'est la seule modification nécessaire pour passer de H2 à MySQL :
il suffit de remplacer l'URL JDBC et les identifiants dans la configuration Spring.

4.3 Création de la base de données

La base MySQL a été créée localement, puis Spring Boot s'est chargé :

- **de créer automatiquement la table adherent,**
- **d'insérer les données prévues au démarrage.**

4.4 Validation

Les tests via un client SQL ont confirmé que :

- **les enregistrements étaient bien insérés,**
- **la structure correspondait à l'entité JPA,**
- **aucune modification de code métier n'a été nécessaire.**