Rapport : API REST et Microservices avec Spring Boot

1 Introduction

Ce rapport présente les notions fondamentales liées aux API REST, aux microservices, ainsi que leur mise en pratique à travers un projet développé avec Spring Boot. L'objectif est réexpliquée ce que j'ai compris des vidéos et des manipulations effectuées durant le TP.

2 Qu'est-ce qu'une API REST ?

Une API REST (Representational State Transfer) est un ensemble de règles permettant à deux applications de communiquer entre elles à l'aide du protocole HTTP.

Principes

- des URLs pour identifier les ressources (ex : /liste, /getEtudiant)
- des méthodes HTTP pour effectuer des actions

Méthode HTTP	Rôle	Exemple
GET	Lire une donnée	Récupérer un étudiant
POST	Ajouter une donnée	Ajouter un étudiant
PUT	Modifier une donnée	Modifier le nom
DELETE	Supprimer une donnée	Supprimer un étudiant

Les données sont échangées au format JSON, simple et lisible.

3 Qu'est-ce qu'un microservice ?

Un microservice est une petite application indépendante qui réalise une tâche unique. Dans ce TP, notre microservice gère uniquement les informations d'étudiants.

Caractéristiques

- il peut fonctionner indépendamment des autres
- il possède son propre port
- il peut être développé, testé et déployé séparément

Spring Boot facilite grandement la création de microservices grâce à sa simplicité et à ses configurations automatiques.

4 Structure du projet Spring Boot

Le projet contient les fichiers essentiels suivants :

```
src/main/java/com/example/demo/
          DemoApplication.java        point de d marrage
          Etudiant.java               mod le de donn es
          MyApi.java                  contr leur REST
```

5 Description des classes

5.1 Classe principale : DemoApplication

Cette classe démarre le microservice.

```
@SpringBootApplication
public class DemoApplication {
    public static void main(String[] args) {
        SpringApplication.run(DemoApplication.class, args);
    }
}
```

5.2 Classe modèle : Etudiant

Elle représente un étudiant avec trois attributs : identifiant, nom et moyenne.

```
public class Etudiant {
    private int identifiant;
    private String nom;
    private double moyenne;

    public Etudiant() {}

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

5.3 Contrôleur REST : MyApi

Cette classe contient toutes les routes permettant de manipuler les étudiants. Les données sont stockées dans une liste en mémoire.

Initialisation des données

```
public static ArrayList<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));
}
```

Exemple de route GET

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

6 Modification du port

Spring Boot démarre par défaut sur le port 8080. Pour changer ce port :

```
# src/main/resources/application.properties
server.port=9999
```

7 Utilisation de Postman

Postman est un outil permettant de tester les API REST de manière simple.

Exemples de tests

Méthode	URL	Action
GET	http://localhost:9999/liste	Voir les étudiants
GET	http://localhost:9999/getEtudiant?identifiant=0	Voir un étudiant

8 Résumé de la partie 1

Cette partie m'a permis de découvrir les concepts d'API REST et de microservices, ainsi que leur implémentation avec Spring Boot. J'ai appris à :

- créer un microservice simple
- implémenter des routes REST
- manipuler une liste d'objets
- tester les routes avec Postman
- modifier la configuration de l'application

9 Partie 2

9.1 Introduction

Dans cette partie, nous allons explorer comment persister des données dans une base relationnelle avec Spring Boot et JPA/Hibernate. L'objectif est que même une personne n'ayant jamais vu ces notions puisse comprendre :

- le rôle d'une entité JPA,
- comment fonctionne un repository pour accéder à la base,
- comment initialiser des données avec CommandLineRunner,
- configurer la base H2 ou MySQL via application.properties,
- préparer une migration vers une API REST Python.

9.2 Entité JPA : Adherent

```
@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;
    }
}
```

Explications

- `@Entity` : indique que la classe correspond à une table en base.
- `@Id` et `@GeneratedValue` : la clé primaire est générée automatiquement.
- Les champs `nom`, `ville`, `age` sont stockés comme colonnes.

9.3 Repository : AdherentRepository

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

Explications

- JpaRepository fournit automatiquement toutes les méthodes CRUD : save, findById, findAll, delete.
- Cela simplifie l'accès aux données et permet de se concentrer sur la logique métier.

9.4 Initialisation des données : CommandLineRunner

```
@Bean
CommandLineRunner runner(AdherentRepository repository) {
    return args -> {
        repository.save(new Adherent(null,"A","B",29));
        repository.save(new Adherent(null,"A","B",29));
    };
}
```

Explications

- CommandLineRunner s'exécute au démarrage de l'application.
- Permet d'insérer des données initiales dans la base pour tester les endpoints REST.
- L'id est null pour que la base génère automatiquement la clé primaire.

9.5 Configuration de la base : application.properties

H2 (en mémoire)

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

- H2 est une base en mémoire, pratique pour tester rapidement.
- Le H2 console permet de visualiser la base via un navigateur (http://localhost:9191/h2-console).

MySQL (relationnelle)

```
spring.datasource.url=jdbc:mysql://localhost:8889/adherent
spring.datasource.username=root
spring.datasource.password=root
spring.jpa.hibernate.ddl-auto=create
```

- Crée automatiquement les tables au démarrage (`ddl-auto=create`).
- Les données peuvent être consultées via phpMyAdmin.

9.6 Notions importantes

- **JPA / Hibernate** : JPA est une interface pour gérer la persistance des objets. Hibernate est une implémentation concrète de JPA.
- **Microservices** : chaque application Spring Boot peut être un microservice autonome, avec sa propre base.
- **Repositories** : évitent d'écrire du SQL manuel et fournissent des méthodes CRUD prêtes à l'emploi.
- **CommandLineRunner** : exécute du code au démarrage et initialise les données pour tester l'API REST.

9.7 Résumé

Le microservice AdherentService utilise JPA pour persister les données. AdherentRepository permet d'accéder à la base sans SQL. Les données sont initialisées au démarrage avec CommandLineRunner. La configuration peut cibler H2 pour tests ou MySQL pour production. La structure permet de comprendre clairement la notion de microservice et de persistance.