TP1 – API REST et Microservices

1 Introduction

L'objectif de ce TP est d'explorer les principes fondamentaux des API REST et des microservices, en utilisant deux environnements différents : Java avec Spring Boot et Python. Dans un premier temps, nous avons implémenté une API REST simple permettant de manipuler des objets `Etudiant`. Dans un second temps, une version améliorée utilisant Spring Data JPA et une base de données relationnelle H2 a été réalisée afin d'introduire la notion de persistance.

Ce rapport synthétise l'ensemble des manipulations, l'architecture adoptée, les éléments techniques essentiels et les résultats obtenus.

2 API REST en Java - Spring Boot

La première partie du TP consiste à mettre en place une API REST capable de gérer une liste d'étudiants via différentes opérations HTTP (GET, POST, PUT, DELETE). Le projet utilise Maven pour la gestion des dépendances et Spring Boot pour la structure applicative et le serveur embarqué.

2.1 Classe Etudiant

La classe `Etudiant` définit les données manipulées par les différentes routes REST.

```
public class Etudiant {
    private int identifiant;
    private String nom;
    private double moyenne;
    // Constructeurs, getters et setters
}
```

2.2 Contrôleur REST : MyAPI

La classe `MyAPI` expose plusieurs endpoints permettant d'interagir avec la liste statique d'étudiants.

- /**lst** : retourne la liste complète des étudiants.
- /**get** : retourne un étudiant selon son identifiant.
- /**add** : ajoute un étudiant.
- /**put** : modifie un étudiant existant.
- /**del** : supprime un étudiant.
- Routes simples : `/bnj`, `/bns`, `/etd`, `/sum`.

```
@RestController
public class MyAPI {
    public static ArrayList<Etudiant> liste = new ArrayList<>();

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

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

2.3 Application Spring Boot

L'application démarre grâce à la classe principale annotée `@SpringBootApplication`.

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

3 API REST avec persistance - Spring Data JPA

La deuxième partie du TP introduit la persistance des données via JPA et une base de données embarquée H2. L'objectif est de stocker des objets `Adherent` dans une table générée automatiquement.

3.1 Entité Adherent

L'annotation `@Entity` permet à Spring et JPA de transformer la classe en table relationnelle.

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

3.2 Repository JPA

L'interface `AdherentRepository` hérite de `JpaRepository`, ce qui permet d'obtenir automatiquement toutes les opérations CRUD sans implémentation supplémentaire.

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

3.3 Initialisation de la base

Une méthode annotée `@Bean` et `CommandLineRunner` permet d'insérer des données au démarrage.

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

4 Conclusion

Ce TP a permis de comprendre concrètement le fonctionnement d'une API REST ainsi que la différence entre un service exposant simplement des données en JSON et un service issu d'une architecture microservices couplée à une base de données. Spring Boot et Spring Data JPA simplifient grandement la création de services professionnels et extensibles.

Cette expérience constitue une transition essentielle vers la manipulation de microservices plus avancés et l'intégration de plusieurs technologies backend.