TP Spring Boot - Rest API

Étape 1: Ajouter Spring Boot à Eclipse

1.1 Créer un projet Spring Boot dans Eclipse

Pour commencer avec Spring Boot, vous devez d'abord installer Eclipse IDE pour Java et ajouter le support Spring. Voici les étapes :

1. Ouvrir Eclipse et aller dans le menu **File > New > Project**.
2. Sélectionner **Spring > Spring Starter Project**.
3. Donner un nom à votre projet, par exemple : **StudentManagement**.
4. Choisir la version de Spring Boot et les dépendances nécessaires comme **Spring Web** et **Spring Data JPA**.
5. Cliquez sur **Finish** pour créer le projet.

Une fois le projet créé, Eclipse génère les fichiers de base pour le projet Spring Boot, y compris un fichier **Application.java** dans le répertoire de base.

Étape 2: Fichiers à créer dans le projet

2.1 Structure des fichiers

Voici la structure du projet et les fichiers à créer pour l'implémentation du modèle, contrôleur, service et le fichier **application.properties** :

```
src/
 └── main/
      └── java/
           └── com/
                └── example/
                     └── studentmanagement/
                          ├── Application.java
                          ├── model/
                          │    └── Student.java
```

```
                        ├── controller/
                        │    ├── StudentController.java
                        │    └── StudentRestController.java
                        ├── repository/
                        │    └── StudentRepository.java
                        └── service/
                             └── StudentService.java
└── resources/
     └── application.properties
     └── templates/
          └── students.jsp
```

Étape 3: Fichier `application.properties`

3.1 Configuration de la base de données

```
# Configuration de la base de données H2
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.h2.console.enabled=true
spring.jpa.hibernate.ddl-auto=update
```

Étape 4: Code Java

4.1 Classe Application.java

Voici le point d'entrée principal pour votre application Spring Boot :

```
package com.example.studentmanagement;


import org.springframework.boot.SpringApplication;

import
org.springframework.boot.autoconfigure.SpringBootApplication;


@SpringBootApplication

public class Application {


    public static void main(String[] args) {

        SpringApplication.run(Application.class, args);

    }

}
```

4.2 Modèle Student.java

Voici la classe modèle représentant un étudiant :

```
package com.example.studentmanagement.model;


import javax.persistence.Entity;

import javax.persistence.Id;
```

```java
@Entity
public class Student {

    @Id
    private Long id;
    private String name;
    private String email;

    // Getters et setters
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
```

```
    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }
}
```

4.3 Repository StudentRepository.java

Le repository JPA pour interagir avec la base de données :

```
package com.example.studentmanagement.repository;

import com.example.studentmanagement.model.Student;
import org.springframework.data.jpa.repository.JpaRepository;

public interface StudentRepository extends JpaRepository {
    // Des méthodes personnalisées peuvent être ajoutées ici si nécessaire
}
```

4.4 Service StudentService.java

Le service pour gérer les opérations métiers sur les étudiants :

```java
package com.example.studentmanagement.service;


import com.example.studentmanagement.model.Student;

import
com.example.studentmanagement.repository.StudentRepository;

import
org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;


import java.util.List;


@Service

public class StudentService {


    @Autowired

    private StudentRepository studentRepository;


    public List getAllStudents() {

        return studentRepository.findAll();

    }


    public void saveStudent(Student student) {

        studentRepository.save(student);

    }
```

```
    public void deleteStudent(Long id) {
        studentRepository.deleteById(id);
    }
}
```

4.5 Contrôleur REST StudentRestController.java

Le contrôleur REST pour exposer des API pour ajouter et supprimer des étudiants :

```
package com.example.studentmanagement.controller;

import com.example.studentmanagement.model.Student;
import com.example.studentmanagement.service.StudentService;
import
org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
@RequestMapping("/api/students")
public class StudentRestController {

    @Autowired
    private StudentService studentService;
```

```
    @GetMapping
    public List getAllStudents() {
        return studentService.getAllStudents();
    }

    @PostMapping
    public void addStudent(@RequestBody Student student) {
        studentService.saveStudent(student);
    }

    @DeleteMapping("/{id}")
    public void deleteStudent(@PathVariable Long id) {
        studentService.deleteStudent(id);
    }
}
```

4.6 Contrôleur Web StudentController.java

Le contrôleur web qui charge la page JSP et gère l'ajout des étudiants :

```
package com.example.studentmanagement.controller;

import com.example.studentmanagement.model.Student;
import com.example.studentmanagement.service.StudentService;
import
org.springframework.beans.factory.annotation.Autowired;
```

```java
import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.*;


@Controller

public class StudentController {


    @Autowired

    private StudentService studentService;


    @GetMapping("/acceuil")

    public String getAllStudents(Model model) {

        model.addAttribute("students",
studentService.getAllStudents());

        return "students"; // Nom du fichier JSP

    }


    @PostMapping("/acceuil")

    public String addStudent(@RequestParam String name,
@RequestParam String email) {

        Student student = new Student();

        student.setName(name);

        student.setEmail(email);

        studentService.saveStudent(student);

        return "redirect:/acceuil"; // Redirige vers la page
d'accueil
```

```
    }


    @PostMapping("/students/{id}")
    public String deleteStudent(@PathVariable Long id) {
        studentService.deleteStudent(id);
        return "redirect:/acceuil";
    }
}
```

Étape 5: Lancer l'application

5.1 Lancer l'application

Une fois que vous avez créé tous les fichiers nécessaires, vous pouvez lancer votre application Spring Boot. Dans Eclipse, faites un clic droit sur le fichier `Application.java` et sélectionnez **Run As > Java Application**.

L'application démarrera sur le port 8080. Vous pouvez accéder à l'interface Web via http://localhost:8080/acceuil.