**Step 1: Create the Spring Boot Project**

1. **Go to Spring Initializr**: <https://start.spring.io/>.
2. **Set the following project settings**:
   * **Project**: Maven Project
   * **Language**: Java
   * **Spring Boot Version**: 2.x (latest stable version)
   * **Group**: com.example
   * **Artifact**: studentapp
   * **Name**: studentapp
   * **Package Name**: com.example.studentapp
   * **Packaging**: Jar
   * **Java Version**: 8 or above
3. **Add Dependencies**:
   * **Spring Web**
   * **Spring Data JPA**
   * **MySQL Driver**
   * **Spring Boot DevTools** (Optional, for auto-reloading during development)
   * **Spring Boot Starter Test** (Optional, for testing)
4. Click **Generate** and download the project.

**Step 2: Extract the Project**

1. Unzip the downloaded zip file.
2. Open the project in your IDE (IntelliJ, Eclipse, etc.).

**Step 3: Configure Database Connection (application.properties)**

1. In src/main/resources/application.properties, add the following database configuration:

properties

Copy code

# MySQL Database Configuration

spring.datasource.url=jdbc:mysql://localhost:3306/student\_db?useSSL=false&serverTimezone=UTC

spring.datasource.username=root

spring.datasource.password=root\_password

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

Replace root\_password with your MySQL root password.

1. **Create Database**:
   * Open MySQL Command Line or MySQL Workbench and run the following:

sql

Copy code

CREATE DATABASE student\_db;

**Step 4: Create the Student Entity**

1. Create a new Java class Student.java in src/main/java/com/example/studentapp/model/.

java

Copy code

package com.example.studentapp.model;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

@Entity

public class Student {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

// Getters and 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;

}

}

**Step 5: Create the Repository Interface**

1. Create a new Java interface StudentRepository.java in src/main/java/com/example/studentapp/repository/.

java

Copy code

package com.example.studentapp.repository;

import com.example.studentapp.model.Student;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

@Repository

public interface StudentRepository extends JpaRepository<Student, Long> {

}

This interface extends JpaRepository to provide CRUD operations for the Student entity.

**Step 6: Create the Service Layer**

1. Create a new Java class StudentService.java in src/main/java/com/example/studentapp/service/.

java

Copy code

package com.example.studentapp.service;

import com.example.studentapp.model.Student;

import com.example.studentapp.repository.StudentRepository;

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

import org.springframework.stereotype.Service;

import java.util.List;

import java.util.Optional;

@Service

public class StudentService {

@Autowired

private StudentRepository studentRepository;

public List<Student> getAllStudents() {

return studentRepository.findAll();

}

public Student addStudent(Student student) {

return studentRepository.save(student);

}

public void deleteStudent(Long id) {

studentRepository.deleteById(id);

}

public Optional<Student> getStudentById(Long id) {

return studentRepository.findById(id);

}

}

**Step 7: Create the Controller**

1. Create a new Java class StudentController.java in src/main/java/com/example/studentapp/controller/.

java

Copy code

package com.example.studentapp.controller;

import com.example.studentapp.model.Student;

import com.example.studentapp.service.StudentService;

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

import org.springframework.http.ResponseEntity;

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

import java.util.List;

import java.util.Optional;

@RestController

@RequestMapping("/students")

public class StudentController {

@Autowired

private StudentService studentService;

@GetMapping

public List<Student> getAllStudents() {

return studentService.getAllStudents();

}

@PostMapping

public Student addStudent(@RequestBody Student student) {

return studentService.addStudent(student);

}

@DeleteMapping("/{id}")

public ResponseEntity<String> deleteStudent(@PathVariable Long id) {

Optional<Student> student = studentService.getStudentById(id);

if (student.isPresent()) {

studentService.deleteStudent(id);

return ResponseEntity.ok("Student deleted successfully.");

}

return ResponseEntity.status(404).body("Student not found.");

}

}

**Step 8: Run the Application**

1. Open the terminal or IDE and navigate to the root of your project.
2. Run the Spring Boot application:

bash

Copy code

mvn spring-boot:run

Alternatively, if you are using an IDE, you can run StudentAppApplication.java directly.

**Step 9: Verify the Endpoints**

You can use Postman or any HTTP client to verify the REST API.

1. **List All Students** (GET /students)
   * **Request**:

http

Copy code

GET http://localhost:8080/students

1. **Add a New Student** (POST /students)
   * **Request Body (JSON)**:

json

Copy code

{

"name": "John Doe",

"email": "john.doe@example.com"

}

* + **Request**:

http

Copy code

POST http://localhost:8080/students

1. **Delete a Student** (DELETE /students/{id})
   * **Request**:

http

Copy code

DELETE http://localhost:8080/students/1

**Step 10: Verify Data in MySQL**

1. Log into MySQL:

bash

Copy code

mysql -u root -p

1. Select the student\_db database:

sql

Copy code

USE student\_db;

1. List the students in the student table:

sql

Copy code

SELECT \* FROM student;

**Step 11: Testing in Postman**

1. **GET /students**:
   * Use **GET** request in Postman to fetch the list of students:

http

Copy code

GET http://localhost:8080/students

1. **POST /students**:
   * Use **POST** request in Postman to add a student with JSON data:

json

Copy code

{

"name": "Jane Doe",

"email": "jane.doe@example.com"

}

http

Copy code

POST http://localhost:8080/students

1. **DELETE /students/{id}**:
   * Use **DELETE** request in Postman to delete a student by ID:

http

Copy code

DELETE http://localhost:8080/students/1

**Final Project Structure**

css

Copy code

studentapp/

│

├── src/

│ └── main/

│ ├── java/

│ │ └── com/

│ │ └── example/

│ │ └── studentapp/

│ │ ├── controller/

│ │ │ └── StudentController.java

│ │ ├── model/

│ │ │ └── Student.java

│ │ ├── repository/

│ │ │ └── StudentRepository.java

│ │ └── service/

│ │ └── StudentService.java

│ └── resources/

│ ├── application.properties

│

├── pom.xml

└── mvnw