📄 CRUD Application with Spring Boot, Spring Data JPA, and Swagger

📝 Requirements

TO CREATE A SIMPLE CRUD (CREATE, READ, UPDATE, DELETE) APPLICATION USING SPRING BOOT, SPRING DATA JPA, AND SWAGGER, YOU'LL NEED THE FOLLOWING:

Java Development Kit (JDK)

- Ensure you have JDK 8 or higher installed.

Maven

- Maven is a build automation tool used for Java projects.

Spring Boot

- A framework that makes it easy to create stand-alone, production-grade Spring-based applications.

Spring Data JPA

- A part of the Spring Data project that makes it easy to implement JPA-based repositories.

Spring Web

- To build web applications, including RESTful services.

SpringDoc OpenAPI Starter WebMVC UI (Swagger)

- A tool to document and test APIs easily.

IDE (Integrated Development Environment)

- Examples include IntelliJ IDEA or Eclipse.

Postman

- For testing the RESTful APIs.

🚀 Step-by-Step

TO CREATE A SIMPLE CRUD (CREATE, READ, UPDATE, DELETE) APPLICATION THEN,
WE NEED TO FOLLOW THE STEPS:

1. Generate a Spring Boot Project

Go to Spring Initializr and choose the following options:

- Project: Maven
- Language: Java
- Spring Boot Version: 2.7.0 or higher
- Dependencies: Add the following
  - Spring Web
  - Spring Data JPA
  - H2 Database (for simplicity)
  - SpringDoc OpenAPI Starter WebMVC UI (Swagger)
- Click on Generate to download the project zip file.
- Extract and open it in your IDE.

2. Configure Database

Set Up application.properties

- Open src/main/resources/application.properties.
- Add the following configuration for H2 database:

```
spring.application.name=demo // name of the application

spring.datasource.url=jdbc:h2:mem:testdb // testdb is database name
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=test // username
spring.datasource.password=123 // password
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
```

3. Create the Student Entity

Define the Entity

Create a new Java class Student in src/main/java/com/example/demo/model:

```java
package com.example.demo.model;

import jakarta.persistence.Entity;
import jakarta.persistence.GeneratedValue;
import jakarta.persistence.GenerationType;
import jakarta.persistence.Id;

@Entity
public class Student {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;
    private String name;
    private String classname;
    private String subject;

    // 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 getClassname() {
        return classname;
    }

    public void setClassname(String classname) {
        this.classname = classname;
    }

    public String getSubject() {
        return subject;
    }

    public void setSubject(String subject) {
        this.subject = subject;
    }
}
```

4. Create the Repository

Define the Repository Interface

Create a new interface StudentRepository in src/main/java/com/example/demo/repository:

```
package com.example.demo.repository;

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

public interface StudentRepository extends JpaRepository<Student, Long> {
    // No additional methods are required as JpaRepository provides
    // all the basic CRUD operations out of the box. Custom query methods
    // can be added here if needed.
}
```

5. Create the Service Layer

Implement the Service

Create a new class StudentService in src/main/java/com/example/demo/service:

```
package com.example.demo.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.example.demo.model.Student;
import com.example.demo.repository.StudentRepository;
import java.util.List;
import java.util.Optional;

@Service
public class StudentService {

    @Autowired
    private StudentRepository studentRepository; // Injects StudentRepository dependency

    public List<Student> getAllStudents() {
        return studentRepository.findAll(); // Retrieves all students
    }

    public Optional<Student> getStudentById(Long id) {
        return studentRepository.findById(id); // Retrieves a student by ID
    }

    public Student createStudent(Student student) {
        return studentRepository.save(student); // Saves a new student
    }

    public Student updateStudent(Long id, Student studentDetails) {
        return studentRepository.findById(id).map(student -> {
            student.setName(studentDetails.getName()); // Updates student's name
            student.setClassname(studentDetails.getClassname()); // Updates student's class name
            student.setSubject(studentDetails.getSubject()); // Updates student's subject
            return studentRepository.save(student); // Saves the updated student
        }).orElse(null); // Returns null if student not found
    }

    public void deleteStudent(Long id) {
        studentRepository.deleteById(id); // Deletes a student by ID
    }
}
```

6. Create the Controller

Define the REST Controller

Create a new class StudentController in src/main/java/com/example/demo/controller:

```java
package com.example.demo.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import com.example.demo.model.Student;
import com.example.demo.service.StudentService;
import java.util.List;

@RestController
@RequestMapping("/students")
public class StudentController {

    @Autowired
    private StudentService studentService; // Injects StudentService dependency

    @GetMapping
    public List<Student> getAllStudents() {
        return studentService.getAllStudents(); // Retrieves all students
    }

    @GetMapping("/{id}")
    public ResponseEntity<Student> getStudentById(@PathVariable Long id) {
        return studentService.getStudentById(id) // Retrieves student by ID
                .map(ResponseEntity::ok) // Returns 200 OK with student
                .orElse(ResponseEntity.notFound().build()); // Returns 404 if not found
    }

    @PostMapping
    public Student createStudent(@RequestBody Student student) {
        return studentService.createStudent(student); // Creates a new student
    }

    @PutMapping("/{id}")
    public ResponseEntity<Student> updateStudent(@PathVariable Long id, @RequestBody Student studentDetails) {
        Student updatedStudent = studentService.updateStudent(id, studentDetails); // Updates student by ID
        if (updatedStudent != null) {
            return ResponseEntity.ok(updatedStudent); // Returns 200 OK with updated student
        } else {
            return ResponseEntity.notFound().build(); // Returns 404 if student not found
        }
    }

    @DeleteMapping("/{id}")
    public ResponseEntity<Void> deleteStudent(@PathVariable Long id) {
        studentService.deleteStudent(id); // Deletes student by ID
        return ResponseEntity.noContent().build(); // Returns 204 No Content
    }
}
```

NOW WE HAVE TO RUN THE JAVA APPLICATION
NOW WE HAVE TO RUN THE JAVA APPLICATION
NOW WE HAVE TO RUN THE JAVA APPLICATION
NOW WE HAVE TO RUN THE JAVA APPLICATION
NOW WE HAVE TO RUN THE JAVA APPLICATION