In springboot.

We should have 3 packages

Controller
Entity
Repository

- **Entity Class**: Representing the studentdetails table.
- **Repository**: To handle database operations.
- **Controller**: To manage HTTP requests.

In Entity..

We have getters and setters .

It is a class file. --

```
package com.example.demo.entity;

import jakarta.persistence.*;


@Entity
@Table(name = "studentdetails") // optional. If we have different classname
public class StudentDetails   {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO) // Auto increment
    private Long sid;


    @Column(name = "sname", nullable = false)
    private String sname;


    @Column(name = "class")
    private String studentClass;


    @Column(name = "dob")
```

```
    private String dob; // Use LocalDate for proper date handling


    @Column(name = "phone")
    private Long phone;


    // Getters and Setters
    public Long getSid() {
        return sid;
    }


    public void setSid(Long sid) {
        this.sid = sid;
    }


    public String getSname() {
        return sname;
    }
```

} // class ends

Repository - we have to create a interface

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

import com.example.demo.entity.StudentDetails;
import org.springframework.data.jpa.repository.JpaRepository;
```

public **interface** StudentDetailsRepository **extends** JpaRepository<**StudentDetails**, Long> {}

---

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

-> **this import brings in the JpaRepository interface from Spring Data JPA.**

public interface StudentDetailsRepository declares a public interface named StudentDetailsRepository.

It extends JpaRepository<StudentDetails, Long>.
This means that StudentDetailsRepository will inherit various methods for interacting with the StudentDetails entity.

Why Use a Repository?

Separation of Concerns:

- The repository pattern helps separate the logic that retrieves data from the underlying database from the rest of the application. This keeps your code organized and maintainable.

CRUD Operations:

- By extending JpaRepository, your repository automatically inherits several methods for common operations, such as:
  - save(S entity): Save an entity.
  - findById(ID id): Retrieve an entity by its ID.
  - findAll(): Retrieve all entities.
  - deleteById(ID id): Delete an entity by its ID.
- This reduces the amount of boilerplate code you need to write.

Custom Queries:

- You can define custom query methods by simply declaring methods in the interface. For example,
  if you wanted to find students by name, you could add:

```
List<StudentDetails> findBySname(String sname);
```

- **Repository Interface**: StudentDetailsRepository is an interface that provides an abstraction layer for data access.
- **Extending JpaRepository**: By extending JpaRepository, you gain access to a variety of built-in methods for CRUD operations without having to write any implementation code.
- **Automatic Query Generation**: Spring Data JPA automatically generates SQL queries based on the method names you define in the repository `interface`.

In Controller - create a class - to handle incoming request.

StudentDetails: This is the entity class that represents the studentdetails table in your database.

StudentDetailsRepository: This is the repository interface used to perform database operations on the StudentDetails entity.

@Autowired: This annotation is used for dependency injection, allowing Spring to automatically provide the required beans.

ResponseEntity: This class is used to represent the HTTP response, allowing you to customize the response status and body.

@RestController, @RequestMapping, @PostMapping: These are Spring

MVC annotations that help define RESTful web services.

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

import com.example.demo.entity.StudentDetails;

import com.example.demo.repository.StudentDetailsRepository;

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

import org.springframework.http.ResponseEntity;

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


@RestController

@RequestMapping("/api/students")

public class StudentDetailsController {
```

```
    @Autowired

    private StudentDetailsRepository studentDetailsRepository;

    @PostMapping

    public ResponseEntity<StudentDetails>
createStudent(@RequestBody StudentDetails studentDetails) {

        StudentDetails savedStudent =
studentDetailsRepository.save(studentDetails);

        return ResponseEntity.ok(savedStudent);

    }
```

}// Class ends

Class Declaration

```
@RestController
@RequestMapping("/api/students")

public class StudentDetailsController {
```

@RestController:
This annotation indicates that the class is a RESTful controller. It combines the functionality of @Controller and @ResponseBody, meaning that data returned from methods will be serialized directly to JSON and sent back in the HTTP response.

@RequestMapping("/api/students"):
This annotation defines the base URL for all endpoints in this controller. In this case, any request starting with /api/students will be handled by this controller.

Dependency Injection

```
@Autowired
```

private StudentDetailsRepository studentDetailsRepository;

Field Injection:

The @Autowired annotation tells Spring to inject an instance of StudentDetailsRepository into this controller.
This repository will be used to interact with the database for CRUD operations related to StudentDetails.

Endpoint for Creating a Student

```
@PostMapping
public ResponseEntity<StudentDetails>
createStudent(@RequestBody StudentDetails studentDetails) {
    StudentDetails savedStudent =
studentDetailsRepository.save(studentDetails);
    return ResponseEntity.ok(savedStudent);
}
```

@PostMapping: This annotation indicates that this method will handle HTTP POST requests sent to /api/students. Typically, a POST request is used to create a new resource.

Method Signature: The method createStudent takes a StudentDetails object as a parameter, which is populated from the request body.
@RequestBody: This annotation tells Spring to deserialize the incoming JSON request body into a StudentDetails object. The incoming data should match the structure of the StudentDetails entity.

Saving the Student:

```
StudentDetails savedStudent =
studentDetailsRepository.save(studentDetails);
```

This line calls the save method of the StudentDetailsRepository to persist the StudentDetails object in the database. The sid (ID) field will

be auto-incremented by the database.

Response Handling:

return ResponseEntity.ok(savedStudent);

This line creates a response entity with a status of 200 OK and includes the saved student object in the response body.

This allows the client to receive the newly created student details, including the auto-generated sid.

Purpose of the Controller

Handling HTTP Requests:

The primary purpose of this controller is to manage HTTP requests related to StudentDetails. In this case, it specifically handles the creation of new student entries via POST requests.

Interfacing with the Service Layer:

While this example does not include a service layer, in larger applications, the controller often communicates with a service layer for business logic. The repository is directly used here for simplicity.

Returning Responses:

The controller is responsible for crafting and sending HTTP responses back to the client, including the status and the data (in this case, the newly created student).