

# Student Management using Spring Boot with H2

---

**Submitted by: Ambati Jyothiraditya**

## Table of Contents

1. 1. Introduction
2. 2. Technologies Used
3. 3. application.properties Configuration
4. 4. Student Entity Class
5. 5. Student Repository
6. 6. Student Service Layer
7. 7. Student Controller
8. 8. Testing Endpoints
9. 9. H2 Console Screenshot
10. 10. Key Concepts Explained

## 1. Introduction

This project is a hands-on implementation of a Student Management System using Spring Boot with an H2 in-memory database. It demonstrates full CRUD functionality through REST APIs and integrates Spring Data JPA for ORM.

## 2. Technologies Used

- Java 17
- Spring Boot
- Spring Data JPA
- H2 Database
- Maven
- Postman

## 3. application.properties Configuration

# H2 Database Configuration

```
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
```

# JPA + Hibernate Configuration

```
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
```

# H2 Console Access

```
spring.h2.console.enabled=true
spring.h2.console.path=/h2-console
```

## 4. Student Entity Class

```
@Entity
```

```
@Table(name = "students")
```

```
public class Student {
```

```
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private Long id;
```

```
@Column(nullable = false)
private String name;

private String department;

// Getters and Setters
}
```

## 5. Student Repository

```
public interface StudentRepository extends JpaRepository<Student, Long> {
    List<Student> findByDepartment(String department);
}
```

## 6. Student Service Layer

```
@Service
public class StudentService {

    @Autowired
    private StudentRepository repository;

    public Student save(Student student) { return repository.save(student); }

    public Optional<Student> get(Long id) { return repository.findById(id); }

    public void delete(Long id) { repository.deleteById(id); }

    public List<Student> getByDepartment(String dept) {
        return repository.findByDepartment(dept);
    }
}
```

## 7. Student Controller

```
@RestController
```

```

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

    @Autowired
    private StudentService service;

    @PostMapping
    public Student create(@RequestBody Student student) { return service.save(student); }

    @GetMapping("/{id}")
    public ResponseEntity<Student> get(@PathVariable Long id) {
        return
service.get(id).map(ResponseEntity::ok).orElse(ResponseEntity.notFound().build());
    }

    @DeleteMapping("/{id}")
    public void delete(@PathVariable Long id) { service.delete(id); }

    @GetMapping("/department/{dept}")
    public List<Student> byDepartment(@PathVariable String dept) {
        return service.getByDepartment(dept);
    }
}

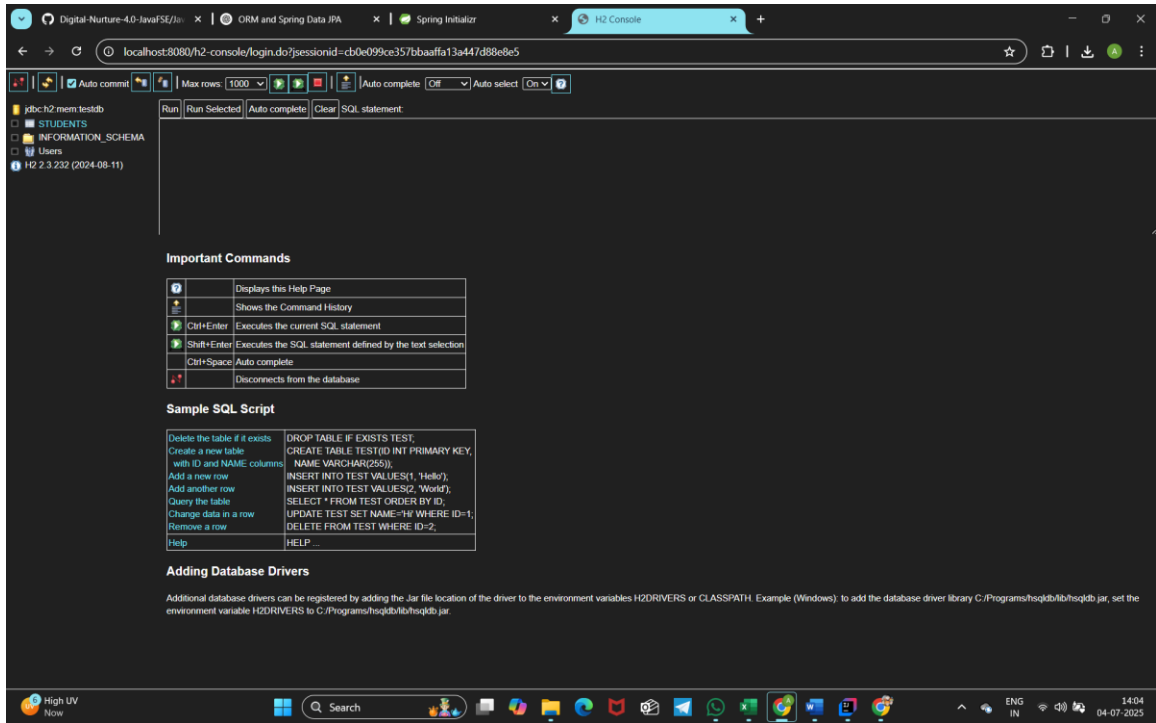
```

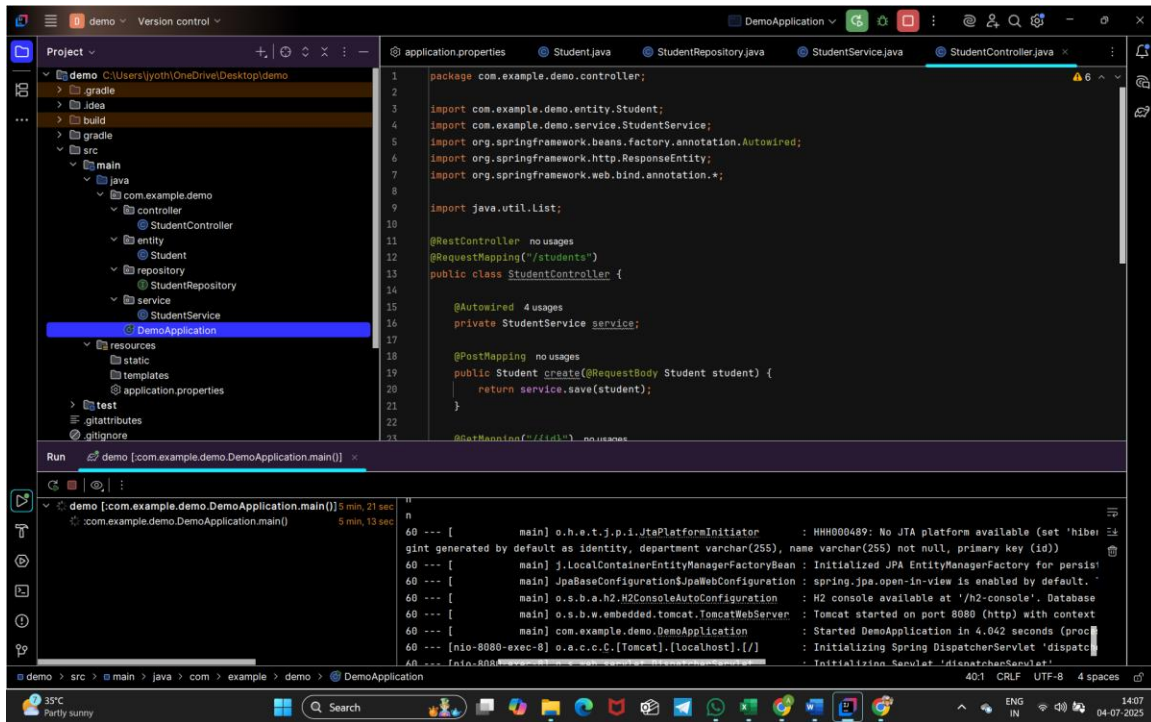
## 8. Testing Endpoints

Use Postman or Swagger to test the following endpoints:

- POST /students
- GET /students/{id}
- GET /students/department/{dept}
- DELETE /students/{id}

## 9. H2 Console Screenshot





## 10. Key Concepts Explained

- **ORM (Object Relational Mapping):** Maps Java classes to database tables automatically.
- **JPA (Java Persistence API):** A specification that simplifies data persistence.
- **Hibernate:** A JPA implementation used under Spring Boot for handling database interactions.
- **Spring Data JPA:** Simplifies the implementation of JPA-based repositories by eliminating boilerplate code.