**\*SpringBoot Example In Eclipse\***

**Step 1**: Set Up Eclipse and Install Necessary Tools

Install Eclipse if you haven't done so already.

Install Spring Tools Suite (STS) in Eclipse:

Go to Help > Eclipse Marketplace, search for "Spring Tools 4", and install it.

Install MySQL if you don’t have it yet. Make sure the MySQL server is running.

**Step 2:** Create a New Spring Boot Project

1. Open Eclipse, then go to File > New > Spring Starter Project.
2. Fill in the details:

Name: SpringBootCrudExample

Type: Maven Project

Group: com.example

Artifact: demo

1. Click Next, and in the next screen, select dependencies:

Spring Web

Spring Data JPA

MySQL Driver

Spring Boot DevTools (optional, for auto-reload during development)

1. Create Database :

Database name: crud\_demo

**Step 3:** Configure MySQL Database in application.properties

Open the src/main/resources/application.properties file and configure the database connection.

# MySQL database connection

spring.application.name=crud-demo

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

spring.datasource.username=sanjeev

spring.datasource.password=sanjeev123

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

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

server.port=8081

**Step 4**: Create the JPA Entity

Create a Java class that represents the table in the database. Let's create an Employee entity.

Right-click on src/main/java > New > Package, name it com.example.demo.model.

Create a new Java class in the package called Employee.java.

package com.example.demo.model;

import javax.persistence.\*;

@Entity

@Table(name = "employees")

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private long id;

@Column(name = "first\_name")

private String firstName;

@Column(name = "last\_name")

private String lastName;

@Column(name = "email")

private String email;

// Getters and Setters

public long getId() {

return id;

}

public void setId(long id) {

this.id = id;

}

public String getFirstName() {

return firstName;

}

public void setFirstName(String firstName) {

this.firstName = firstName;

}

public String getLastName() {

return lastName;

}

public void setLastName(String lastName) {

this.lastName = lastName;

}

public String getEmail() {

return email;

}

public void setEmail(String email) {

this.email = email;

}

}

**Step 5:** Create the Repository Interface

Create a repository interface that extends Spring Data JPA’s JpaRepository interface.

Right-click on src/main/java > New > Package, name it com.example.demo.repository.

Create a new Java interface in the package called EmployeeRepository.java.

package com.example.demo.repository;

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

import com.example.demo.model.Employee;

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

}

**Step 6**: Create the Service Class

Create a service layer to interact with the repository.

Right-click on src/main/java > New > Package, name it com.example.demo.service.

Create a new Java class called EmployeeService.java.

package com.example.demo.service;

import com.example.demo.model.Employee;

import java.util.List;

public interface EmployeeService {

Employee createEmployee(Employee employee);

Employee getEmployeeById(Long id);

List<Employee> getAllEmployees();

Employee updateEmployee(Long id, Employee employeeDetails);

void deleteEmployee(Long id);

}

EmployeeServiceImpl.java

package com.example.demo.service.impl;

import com.example.demo.model.Employee;

import com.example.demo.repository.EmployeeRepository;

import com.example.demo.service.EmployeeService;

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

import org.springframework.stereotype.Service;

import java.util.List;

import java.util.Optional;

@Service

public class EmployeeServiceImpl implements EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Override

public Employee createEmployee(Employee employee) {

return employeeRepository.save(employee);

}

@Override

public Employee getEmployeeById(Long id) {

Optional<Employee> employee = employeeRepository.findById(id);

return employee.orElse(null);

}

@Override

public List<Employee> getAllEmployees() {

return employeeRepository.findAll();

}

@Override

public Employee updateEmployee(Long id, Employee employeeDetails) {

Employee employee = getEmployeeById(id);

if (employee != null) {

employee.setName(employeeDetails.getName());

employee.setEmail(employeeDetails.getEmail());

employee.setDepartment(employeeDetails.getDepartment());

return employeeRepository.save(employee);

}

return null;

}

@Override

public void deleteEmployee(Long id) {

employeeRepository.deleteById(id);

}

}

**Step 7:** Create the Controller

Create a controller EmployeeController.java in the com.example.demo.controller package to handle HTTP requests.

package com.example.demo.controller;

import com.example.demo.model.Employee;

import com.example.demo.service.EmployeeService;

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

import org.springframework.http.ResponseEntity;

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

import java.util.List;

@RestController

@RequestMapping("/api/employees")

public class EmployeeController {

@Autowired

private EmployeeService employeeService;

// Create Employee

@PostMapping

public Employee createEmployee(@RequestBody Employee employee) {

return employeeService.createEmployee(employee);

}

// Get All Employees

@GetMapping

public List<Employee> getAllEmployees() {

return employeeService.getAllEmployees();

}

// Get Employee By ID

@GetMapping("/{id}")

public ResponseEntity<Employee> getEmployeeById(@PathVariable Long id) {

Employee employee = employeeService.getEmployeeById(id);

if (employee != null) {

return ResponseEntity.ok(employee);

} else {

return ResponseEntity.notFound().build();

}

}

// Update Employee

@PutMapping("/{id}")

public ResponseEntity<Employee> updateEmployee(@PathVariable Long id, @RequestBody Employee employeeDetails) {

Employee updatedEmployee = employeeService.updateEmployee(id, employeeDetails);

if (updatedEmployee != null) {

return ResponseEntity.ok(updatedEmployee);

} else {

return ResponseEntity.notFound().build();

}

}

// Delete Employee

@DeleteMapping("/{id}")

public ResponseEntity<Void> deleteEmployee(@PathVariable Long id) {

employeeService.deleteEmployee(id);

return ResponseEntity.noContent().build();

}

}

**Step 8:** Run the Application

Right-click on the project and choose Run As -> Spring Boot App.

The application will start, and you can access the CRUD operations at <http://localhost:8081/api/employees>.

**Step 9:** Test the CRUD Operations

1. Create a new Employee using a POST request:

POST http://localhost:8081/api/employees

Content-Type: application/json

{

"name": "John Doe",

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

"department": "IT"

}

1. Get all Employees using a GET request:

GET http://localhost:8081/api/employees

1. Get a specific Employee by ID using a GET request:

GET http://localhost:8081/api/employees/{id}

1. Update an Employee using a PUT request:

PUT http://localhost:8081/api/employees/{id}

Content-Type: application/json

{

"name": "Jane Doe",

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

"department": "HR"

}

1. Delete an Employee using a DELETE request:

DELETE http://localhost:8081/api/employees/{id}

**Pom.xml**

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.3.3</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com.example</groupId>

<artifactId>crud-demo</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>crud-demo</name>

<description>Demo project for Spring Boot</description>

<url/>

<licenses>

<license/>

</licenses>

<developers>

<developer/>

</developers>

<scm>

<connection/>

<developerConnection/>

<tag/>

<url/>

</scm>

<properties>

<java.version>1.8.0</java.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>