Spring Boot – CRUD operations in Spring Boot and Spring Data JPA

Description

1. Introduction

In this post we will see how to perform CRUD operations using Spring Data JPA. We will use the in memory database i.e. h2 and try out. If you plan to use some other database then the only thing you have to do is change the database specific connector dependency and the DB connection details.

2. Project Download

For creating the Spring Boot project, head over to https://start.spring.io. We need to add “Spring Web” , Spring Data JPA and H2 dependency. Depending on when you are accessing the site, you will find “Add dependencies” button. Add these dependencies one by one and download the project.

3. Database Connectivity

```
server.port=9009
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.h2.console.enabled=true
```

In your application.properties file add the above configuration. The first line is self explanatory that our application will run on 9009 port and the rest are database connectivity details. If you want to see the SQL that gets executed on the eclipse console then add the last line.

4. Creating the Model class

We will use the Employee details to Save, Update, Read and Delete the details. Create a POJO named Employee.java. This Employee table will be mapped to Employee table, lets annotate the class with @Entity and annotate the primary key or the id field with @Id and @GeneratedValue annotation.

```
package com.springmicroservices.model;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

@Entity
public class Employee {

 @Id
```

```
 @GeneratedValue
 public Integer empId;
 public String empName;
 public String empCity;
 public Integer empSalary;

 public Integer getEmpId() {
  return empId;
 }
 public void setEmpId(Integer empId) {
  this.empId = empId;
 }
 public String getEmpName() {
  return empName;
 }
 public void setEmpName(String empName) {
  this.empName = empName;
 }
 public String getEmpCity() {
  return empCity;
 }
 public void setEmpCity(String empCity) {
  this.empCity = empCity;
 }
 public Integer getEmpSalary() {
  return empSalary;
 }
 public void setEmpSalary(Integer empSalary) {
  this.empSalary = empSalary;
 }
}
```


5. Creating the Repository Interface

Lets create the EmployeeRepository interface using which we will be performing the database operations. To enable this capability we will extend the EmployeeRepository with the CrudRepository interface.

If you see the below code, in the angular brackets we have given the Employee class name followed by Integer. We have given Employee class name because this repository is responsible to perform operation against the Employee table and the Integer is the datatype of the primary key.

```
package com.springmicroservices.repository;
import org.springframework.data.repository.CrudRepository;
import com.springmicroservices.model.Employee;

public interface EmployeeRepository extends CrudRepository<Employee, Integer>
}
```

6. Creating the Controller Class

Lets create EmployeeController class. It will contain endpoints which we will invoke through POSTMAN tool. From this EmployeeController class we will invoke the methods of EmployeeRepository. Lets autowire EmployeeRepository class inside the controller class and invoke each method from POSTMAN.

Method 1 :- getEmployees() controller method will return all the employees present in the database. In order to get all the records we will use the **employeeRepository.findAll()** method which Spring JPA provides.

Method 2 :- getEmployee() controller method will return record based on the provided Id value. This Id field will be used as a search criteria. From the database querying perspective we will use **employeeRepository.findById(empID)** method from Spring JPA.

Method 3 :- updateEmployee() controller method will update the existing record. Before updating the records, we will have to fetch the record we want to change, make the change and then save the object. We will use **employeeRepository.save(dbEmployee)** method from Spring JPA.

Method 4 :- saveEmployee() controller method will save the new record. We will use **employeeRepository.save(employee)** method from Spring JPA to save the record.

Method 5 :- deleteEmployee() controller method will delete the existing record from the underlying database. **employeeRepository.delete(employee)** will be used to delete the db record.


```
package com.springmicroservices.controller;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.util.ObjectUtils;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

import com.springmicroservices.model.Employee;
import com.springmicroservices.repository.EmployeeRepository;

@RestController
public class EmployeeController {

 @Autowired
 EmployeeRepository employeeRepository;

 @GetMapping("/employees")
 public  ResponseEntity<?> getEmployees() {

  Iterable<Employee> employees = employeeRepository.findAll();
  if(ObjectUtils.isEmpty(employees)) {

   return new ResponseEntity<Object>("No records exists for Employee table", H
```

```
 }
 return new ResponseEntity<Object>(employees, HttpStatus.OK);
}

@GetMapping("/employees/{empID}")
public ResponseEntity<Object> getEmployee(@PathVariable("empID") Integer empI

 Employee dbEmployee = null;

 try {
  dbEmployee = employeeRepository.findById(empID).get();
  if(ObjectUtils.isEmpty(dbEmployee)) {

   return new ResponseEntity<Object>("The Employee object does not exist", Ht
  }
 }
 catch(Exception e) {
  return new ResponseEntity<Object>("Something Went Wrong", HttpStatus.INTERN
 }
 return new ResponseEntity<Object>(dbEmployee, HttpStatus.OK);
}

@PutMapping("/employees")
public ResponseEntity<?> updateEmployee(@RequestBody Employee employee) {

 Employee dbEmployee = null;

 try {
  dbEmployee = employeeRepository.findById(employee.getEmpId()).get();

  if(ObjectUtils.isEmpty(dbEmployee)) {
   return new ResponseEntity<Object>("The Employee Object does not exist", Ht
  }
  dbEmployee.setEmpSalary(100000);
  employeeRepository.save(dbEmployee);
 }
 catch(Exception e) {
  return new ResponseEntity<Object>("Something went wrong", HttpStatus.INTERN
 }
 return new ResponseEntity<Object>(dbEmployee, HttpStatus.OK);
}

@PostMapping("/employees")
public ResponseEntity<String> saveEmployee(@RequestBody Employee employee) {

 employeeRepository.save(employee);
 return new ResponseEntity<String>("Employee Saved Successfully", HttpStatus.
}

@DeleteMapping("/employees")
public ResponseEntity<?> deleteEmployee(@RequestBody Employee employee) {

 Employee dbEmployee = employeeRepository.findById(employee.getEmpId()).get()

 if(ObjectUtils.isEmpty(dbEmployee)) {
  return new ResponseEntity<Object>("The Object does not exist", HttpStatus.I
```

```
  } else {
   employeeRepository.delete(employee);
  }
  return new ResponseEntity<Object>("Employee Deleted Successfully", HttpStatu
 }
}
```

You can download the project from the below GIT repository. Detailed documentation is provided on how to execute the project.

Git Repository – https://github.com/RavikiranKada/H2-CRUDOperationsSpringJPA

Category

1. Hands on
2. Spring Boot

Date Created
April 27, 2021
Author
kk-ravi144gmail-com