**PRACTICAL NO. 06**

**Getting Started with Spring Boot**

|  |  |
| --- | --- |
| LOB6 | Demonstrate Spring applications using Spring Boot and spring Boot RESTful Web Services. |
| LO6 | Develop Spring Boot Web Application and Spring Boot RESTful web services. |

Spring Boot is a recent addition to the Spring Framework. It helps developer to create stand-alone, production-grade Spring-based Applications that can user run. It makes coding and starting a Spring based project very easy.

Spring Boot is also an open-source Java-based framework used to create a Micro Service. It contains a infrastructure to support development of micro services and enables you to develop enterprise-ready applications that you can “just run”.

**What is REST?**

* REST stands for **RE**presentational **S**tate **T**ransfer. REST is an architectural style, or design pattern, for APIs and is commonly used to create interactive applications that use Web services.
* REST specifies a set of architectural constraints and a service that satisfies these constraints is called RESTful Service.
* REST has no hard and fast rules and uses Headers to describe requests and responses.
* REST embraces HTTP verbs (or methods, as they are POST, GET, PUT, PATCH, and DELETE).
* In the REST architectural style –
  + data and functionality are considered resources and
  + are accessed using Uniform Resource Identifiers (URIs), typically links on the Web.
* The five important constraints for RESTful Web Service are –
  + Client - Server : There should be a service producer and a service consumer.
  + The interface (URL) is uniform and exposing resources.
  + The service is stateless.
  + The service results should be Cacheable e.g. HTTP cache,
  + Service should assume a Layered architecture. Client should not assume direct connection to server - it might be getting info from a middle layer – cache.

The various operations that can be performed on a resource can be expressed using its URI along with the appropriate HTTP method (GET, POST, PUT, DELETE, etc.).

Assume for example that you’re building a REST API for a blog application. The resources that can be identified in a blog domain are post, comment, and user.

**RESTful Web Services - Resource -**

REST architecture treats every content as a resource and identifies them by URIs/ Global IDs.

These resources can be Text Files, Html Pages, Images, Videos or Dynamic Business Data.

REST Server simply provides access to resources and REST client accesses and modifies the resources.

REST uses Text, JSON, XML representations to represent a resource where most popular representations of resources are XML and JSON.

A REST resource is like Object in OOP, or it is like an Entity in a Database.

Once a resource is identified then its representation is to be decided using a standard format so that the server can send the resource in the above said format and client can understand the same format.

Example of a ”employee” as a resource which is represented using the JSON format −

{

”eid”:”1001”,

”name”: ”Rajesh Kumar”,

”designation”: ”Computer Programmer”

”salary”:”50000.00”

}

**RESTful Web Services – Messages -**

HTTP protocol is used as a medium of communication between client (HTTP Request) and server (HTTP Response) in RESTful Web Services. This technique is termed as Messaging.

These messages contain message data and metadata i.e. information about message itself.

**RESTful Web Services – Methods**

Below is the list of method that used while creating your RESTful web services –

* GET
* POST
* PUT
* DELETE
* PATCH

HTTP Method Summary for RESTful API

|  |  |  |
| --- | --- | --- |
| **HTTP Method** | **Operation** | **Comment** |
| GET | Read Operation only | Uses only for the read operation. GET should be idempotent |
| POST | Create new resource | Should only be used to create a new resource |
| PUT | Update / Replace Resource | Update an existing resource. Think of PUT method as putting a resource |
| DELETE | Delete Resource | To remove a given resource. DELETE operation is *idempotent* |
| PATCH | Partial Update / Modify | Partial update to a resource should happen through PATCH |

**Exercise:**

**Set 1 –**

1. **Create “HelloWorld” application using a Spring boot.**

package com.example.HelloSpringBootSB;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class HelloSpringBootSbApplication {

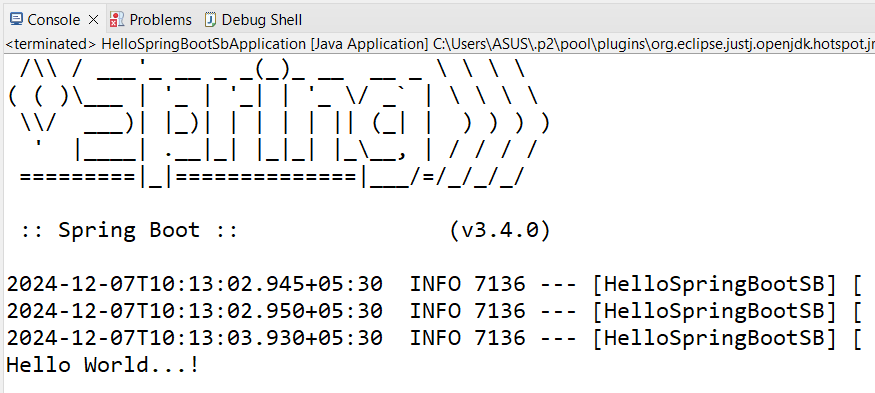
public static void main(String[] args) {

SpringApplication.run(HelloSpringBootSbApplication.class, args);

System.out.println("Hello World...!");

}

}



1. **Create a class to demonstrate the database connectivity using spring boot. Use the problem statement and question 3 of the practical number 5.**

**Department.java**

**package** com.example.springDepartmentDemo;

**public** **class** Department {

**private** **int** departmentId;

**private** String departmentName;

// Constructor

**public** Department() {}

**public** Department(**int** departmentId, String departmentName) {

**this**.departmentId = departmentId;

**this**.departmentName = departmentName;

}

// Getters and Setters

**public** **int** getDepartmentId() {

**return** departmentId;

}

**public** **void** setDepartmentId(**int** departmentId) {

**this**.departmentId = departmentId;

}

**public** String getDepartmentName() {

**return** departmentName;

}

**public** **void** setDepartmentName(String departmentName) {

**this**.departmentName = departmentName;

}

@Override

**public** String toString() {

**return** "Department [departmentId=" + departmentId + ", departmentName=" + departmentName + "]";

}

}

**DepartmentDAO.java**

**package** com.example.springDepartmentDemo;

**import** com.example.springDepartmentDemo.Department;

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

**import** org.springframework.jdbc.core.JdbcTemplate;

**import** org.springframework.stereotype.Repository;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.Map;

@Repository

**public** **class** DepartmentDAO {

**private** **final** JdbcTemplate jdbcTemplate;

@Autowired

**public** DepartmentDAO(JdbcTemplate jdbcTemplate) {

**this**.jdbcTemplate = jdbcTemplate;

}

// Method to insert a department record

**public** **int** saveDepartmentInfo(Department department) {

String sqlStr = "INSERT INTO department (department\_id, department\_name) VALUES (?, ?)";

**return** jdbcTemplate.update(sqlStr, department.getDepartmentId(), department.getDepartmentName());

}

// Method to get all department records as a list

**public** List<String> getAllDepartments() {

String sqlStr = "SELECT \* FROM department";

List<Map<String, Object>> departmentsList = jdbcTemplate.queryForList(sqlStr);

List<String> result = **new** ArrayList<>();

**for** (Map<String, Object> row : departmentsList) {

**int** departmentId = (**int**) row.get("department\_id");

String departmentName = (String) row.get("department\_name");

result.add("Department ID: " + departmentId + ", Department Name: " + departmentName);

}

**return** result;

}

}

**DepartmentService.java**

**package** com.example.springDepartmentDemo;

**import** com.example.springDepartmentDemo.DepartmentDAO;

**import** com.example.springDepartmentDemo.Department;

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

**import** org.springframework.stereotype.Service;

**import** java.util.List;

@Service

**public** **class** DepartmentService {

**private** **final** DepartmentDAO departmentDAO;

@Autowired

**public** DepartmentService(DepartmentDAO departmentDAO) {

**this**.departmentDAO = departmentDAO;

}

// Save department info

**public** **int** saveDepartment(Department department) {

**return** departmentDAO.saveDepartmentInfo(department);

}

// Get all departments

**public** List<String> getAllDepartments() {

**return** departmentDAO.getAllDepartments();

}

}

**SpringDepartmentDemoApplication.java**

**package** com.example.springDepartmentDemo;

**import** com.example.springDepartmentDemo.Department;

**import** com.example.springDepartmentDemo.DepartmentService;

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

**import** org.springframework.boot.CommandLineRunner;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

**import** java.util.List;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

**public** **class** SpringDepartmentDemoApplication **implements** CommandLineRunner{

@Autowired

**private** DepartmentService departmentService;

**public** **static** **void** main(String[] args) {

SpringApplication.*run*(SpringDepartmentDemoApplication.**class**, args);

}

@Override

**public** **void** run(String... args) **throws** Exception {

// **TODO** Auto-generated method stub

// Inserting a department record

Department department = **new** Department(107, "Home Science");

departmentService.saveDepartment(department);

// Fetching all department records

List<String> departments = departmentService.getAllDepartments();

departments.forEach(System.***out***::println);

}

}

application.properties

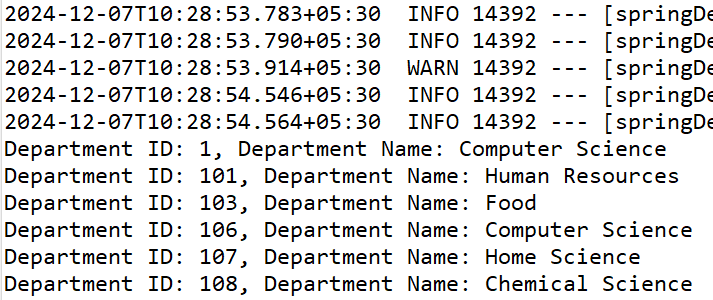
spring.application.name=springDepartmentDemo

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/javapractical5

spring.datasource.username=root

Output:-



1. **Create a maven-based project to demonstrate RESTful Web Services (without database) with spring boot. Consider the main class of the practical number 5.**

**Department.java**

**package** com.example.restDemo;

**public** **class** Department {

**private** **int** departmentId;

**private** String departmentName;

**public** Department(**int** departmentId, String departmentName) {

**this**.departmentId = departmentId;

**this**.departmentName = departmentName;

}

// Getters and Setters

**public** **int** getDepartmentId() {

**return** departmentId;

}

**public** **void** setDepartmentId(**int** departmentId) {

**this**.departmentId = departmentId;

}

**public** String getDepartmentName() {

**return** departmentName;

}

**public** **void** setDepartmentName(String departmentName) {

**this**.departmentName = departmentName;

}

@Override

**public** String toString() {

**return** "Department [departmentId=" + departmentId + ", departmentName=" + departmentName + "]";

}

}

**DepartmentController.java**

**package** com.example.restDemo;

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

**import** java.util.ArrayList;

**import** java.util.List;

@RestController

@RequestMapping("/api/departments")

**public** **class** DepartmentController {

**private** **final** List<Department> departments = **new** ArrayList<>();

// Add a new department

@PostMapping

**public** String addDepartment(@RequestBody Department department) {

departments.add(department);

**return** "Department added successfully!";

}

// Get all departments

@GetMapping

**public** List<Department> getAllDepartments() {

**return** departments;

}

}

package com.example.restDemo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class RestDemoApplication {

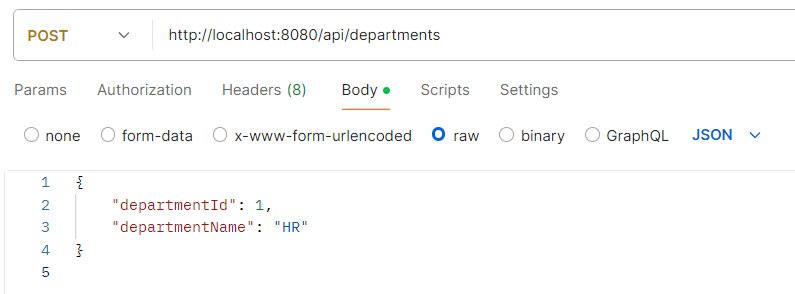
public static void main(String[] args) {

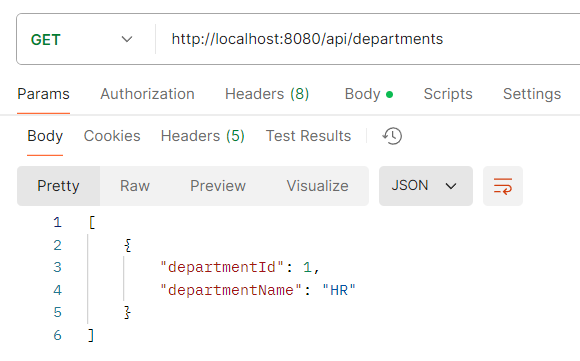
SpringApplication.run(RestDemoApplication.class, args);

}

}

**Output :-**

****

****