**Week-4 Assessment**

Theoretical Questions:

**Q1.)** What are the major features in different versions of Spring Framework?

Ans.

* Spring 2.5: This version was released in 2007. It was the first version which supported annotations.
* Spring3.0: This version was released in 2009. It made full-fledged use of improvements in Java5 and also provided support to JEE6.
* Spring 4.0: This version was released in 2013. This was the first version to provide full support to Java 8.

Q2.) What is a Spring Framework?  
Ans.

1. Spring is a powerful open source, application framework created to reduce the complexity of enterprise application development.
2. It is light-weighted and loosely coupled.
3. It has layered architecture, which allows you to select the components to use, while also providing a cohesive framework for J2EE application development.
4. Spring framework is also called the framework of frameworks as it provides support to various other frameworks such as Struts, Hibernate, EJB, JSF etc.

Q3.) List the advantages of Spring Framework.  
Ans.

1. Because of Spring Frameworks layered architecture, you can use what you  
   need and leave which you don’t.
2. Spring Framework enables POJO (Plain Old Java Object)  
   Programming which in turn enables continuous integration and testability.
3. JDBC is simplified due to Dependency Injection and Inversion of Control.
4. It is open-source and has no vendor lock-in.

Q4.) What are the different features of Spring Framework?  
Ans.

Following are some of the major features of Spring Framework:

1. Lightweight: Spring is lightweight when it comes to size and transparency.
2. Inversion of control (IOC): The objects give their dependencies instead of  
   creating or looking for dependent objects. This is called Inversion Of  
   Control.
3. Aspect oriented Programming (AOP): Aspect oriented programming in  
   Spring supports cohesive development by separating application business  
   logic from system services.
4. Container: Spring Framework creates and manages the life cycle and  
   configuration of the application objects.
5. MVC Framework: Spring Framework’s MVC web application framework is  
   highly configurable. Other frameworks can also be used easily instead of  
   Spring MVC Framework.
6. Transaction Management: Generic abstraction layer for transaction  
   management is provided by the Spring Framework. Spring’s transaction  
   support can be also used in container less environments.
7. JDBC Exception Handling: The JDBC abstraction layer of the Spring offers an  
   exception hierarchy, which simplifies the error handling strategy.

Q5.) How many modules are there in Spring Framework and what are they?  
Ans.

There are around 20 modules which are generalized into Spring Core Container, Data Access/Integration, Web, AOP (Aspect Oriented Programming), Instrumentation  
and Test.

• **Spring Core Container –** This layer is basically the core of Spring Framework. It contains the following modules:  
a. Spring Core  
b. Spring Bean  
c. SPEL (Spring Expression Language)  
d. Spring Context  
• **Data Access/Integration –** This layer provides support to interact with the database. It contains the following modules:  
a. JDBC (Java Data Base Connectivity)  
b. ORM (Object Relational Mapping)  
c. OXM (Object XML Mappers)  
d. JMS (Java Messaging Service)  
e. Transaction  
• **Web –** This layer provides support to create web application. It contains the following modules:  
a. Web  
b. Web – MVC  
c. Web – Socket  
d. Web – Portlet  
• **Aspect Oriented Programming (AOP) –** In this layer you can use Advices,  
Pointcuts etc., to decouple the code.  
• **Instrumentation –** This layer provides support to class instrumentation and  
class loader implementations.  
• **Test –** This layer provides support to testing with JUnit and TestNG.  
**Few Miscellaneous modules are given below:**  
• **Messaging –** This module provides support for STOMP. It also supports an  
annotation programming model that is used for routing and processing  
STOMP messages from WebSocket clients.  
• **Aspects –** This module provides support to integration with AspectJ.

Q**6.) What is a Spring configuration file?**Ans.

A Spring configuration file is an XML file. This file mainly contains the classes  
information. It describes how those classes are configured as well as introduced to each other. The XML configuration files, however, are verbose and cleaner. If it’s not planned and written correctly, it becomes very difficult to manage in big projects.

**Q7. What are the different components of a Spring application?**

Ans.

The Spring framework consists of seven modules. These modules are:

1. Spring Core.
2. Spring AOP.
3. Spring Web MVC.
4. Spring DAO.
5. Spring ORM.
6. Spring context.
7. Spring Web flow.

These modules provide different platforms to develop different enterprise applications; for example, you can use Spring Web MVC module for developing MVC-based applications.

**Q8.) What are the various ways of using Spring Framework?**Ans.

Spring Framework can be used in various ways. They are listed as follows:

1. As a Full-fledged Spring web application.
2. As a third-party web framework, using Spring Frameworks middle-tier.
3. For remote usage.
4. As Enterprise Java Bean which can wrap existing POJOs (Plain Old Java  
   Objects).

Q9.) What is Spring IOC Container?

Ans.

At the core of the Spring Framework, lies the Spring container. The container creates the object, wires them together, configures them and manages their complete life cycle. The Spring container makes use of Dependency Injection to manage the components that make up an application. The container receives instructions for which objects to instantiate, configure, and assemble by reading the configuration metadata provided. This metadata can be provided either by XML, Java annotations or Java code.

**Q10.) What do you mean by Dependency Injection?**Ans.

In Dependency Injection, you do not have to create your objects but have to describe how they should be created. You don’t connect your components and services together in the code directly, but describe which services are needed by which components in the configuration file. The IoC container will wire them up together.

**Practical Questions:  
Q1. Write a code for YML configuration with JPA configuration?**

Ans.

server:

port: 8086 // Use this to specify the port on which you want to run the application

//DataBase Configuration

spring:

datasource:

driver-class-name: com.mysql.cj.jdbc.Driver //Use this property to specify the driver

url: jdbc:mysql://localhost:3306/demo?useSSL=false&serverTimezone=UTC&useLegacyDatetimeCode=false

username: root //Mention your database username here

password: krishna123//Mention your database password here.

//JPA Configuration

jpa:

database-platform: org.hibernate.dialect.MySQL8Dialect

generate-ddl: true

hibernate:

ddl-auto: update

show-sql: true

**Q2. Implement Spring Actuator to monitor one microservice?**

Ans.

Create a springboot project with actuator dependency in it.

**SpringBootActuatorDemoApplication**:

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringBootActuatorDemoApplication {

public static void main(String[] args) {

SpringApplication.run(SpringBootActuatorDemoApplication.class, args);

}

}

**ActuatorController:**

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

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

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

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

@RestController

@RequestMapping("/api")

public class ActuatorController {

@GetMapping("/greeting")

public String greeting() {

return "Welcome to MicroServices";

}

//@Value("${myname}") // Placeholder

@Value("krishna")

private String name;

@GetMapping("/greet")

public String greet() {

return name + "Welcome to MicroServices";

}

}

Then in application.properties file specify the below things.

server.port=8086

myname=Name

management.endpoints.web.exposure.include=\*

server.error.whitelabel.enabled=true

management.endpoint.health.group.custom.include=diskSpace,ping

management.endpoint.health.group.custom.show-components=always

# |when\_authorized

management.endpoint.health.group.custom.show-details=always

#when\_authorized

management.endpoint.health.group.custom.status.http-mapping.up=207

management.endpoint.shutdown.enabled=true

management.endpoints.web.exposure.exclude=loggers

endpoints.beans.id=springbeans

endpoints.beans.sensitive=false

endpoints.beans.enabled=true

endpoints.health.sensitive=false

info.app.name=Spring Boot Actuator Demo

info.app.description=This is a spring boot actuator demonstration application

info.app.version=1.0.0

#port used to expose actuator

management.port=8081

#CIDR allowed to hit actuator

management.address=127.0.0.1

#Whether security should be enabled or disabled altogether

management.security.enabled=true

security.user.name=admin

security.user.password=secret

management.security.role=SUPERUSER

After this open your project with localhost:8086/actuator specification in your browser

**Q3. Please create and deploy one microservice using Spring CLI?**

Ans.

In Command line after spring installation do the following:

To create a springboot maven project:

spring init --build maven  --version 1.0 --java-version 1.8 --dependencies web hello-springboot

mvn spring-boot:run

then create the project and save it as WebApp.java:

@RestController  
public class SampleRestController {  
    @GetMapping("/greeting")  
    public String greeting() {  
        return "Welcome to Microservices";  
    }

Then run this command.

spring run WebApp.java

**Q4. Please change default server port from 8080 to 9090?**

Ans.

Specify the server port number in application.properties file as below

Server.port = 9090

or if you have the application.yml file then mention the below

server:

port: 9090

**Q5. How to resolve white label error page in spring boot application?**

Ans.

Check for the @requestmapping path correctly and use that specified path only to

Run the operation. This should be the main cause of the error.

If we want to remove the whitelabel errors then we can do the following:

**Displaying custom page:**

Create a error.html page and put it into the **src/main/resources/templates** directory

**Using Application.properties method:**

#Disable Whitelabel Error Page server.error.whitelabel.enabled=false

**Q6. All Relationships example such as OneToOne, OneToMany, ManyTOne, ManyToMany using spring boot?**

Ans.

Create a springboot project and then create packages separately for model class, repository interfaces and restcontroller class.

**OneToOne**:

**Employee Class:**

import java.io.Serializable;

import javax.persistence.CascadeType;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.OneToOne;

@Entity

public class Employee implements Serializable {

/\*\*

\*

\*/

private static final long serialVersionUID = 4157709487427837224L;

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private Long eid;

private String name, address, phone;

@OneToOne(cascade = CascadeType.ALL)

private Laptop laptop;

public Long getEid() {

return eid;

}

public void setEid(Long eid) {

this.eid = eid;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getAddress() {

return address;

}

public void setAddress(String address) {

this.address = address;

}

public String getPhone() {

return phone;

}

public void setPhone(String phone) {

this.phone = phone;

}

public Laptop getLaptop() {

return laptop;

}

public void setLaptop(Laptop laptop) {

this.laptop = laptop;

}

}

**Laptop Class:**

import java.io.Serializable;

import java.util.Objects;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

@Entity

public class Laptop implements Serializable {

/\*\*

\*

\*/

private static final long serialVersionUID = -3467344934715695571L;

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private Long lid;

private String model, brand, processor, ram;

public Long getLid() {

return lid;

}

public void setLid(Long lid) {

this.lid = lid;

}

public String getModel() {

return model;

}

public void setModel(String model) {

this.model = model;

}

public String getBrand() {

return brand;

}

public void setBrand(String brand) {

this.brand = brand;

}

public String getProcessor() {

return processor;

}

public void setProcessor(String processor) {

this.processor = processor;

}

public String getRam() {

return ram;

}

public void setRam(String ram) {

this.ram = ram;

}

@Override

public int hashCode() {

return Objects.hash(brand, lid, model, processor, ram);

}

@Override

public boolean equals(Object obj) {

if (this == obj)

return true;

if (obj == null)

return false;

if (getClass() != obj.getClass())

return false;

Laptop other = (Laptop) obj;

return Objects.equals(brand, other.brand) && Objects.equals(lid, other.lid)

&& Objects.equals(model, other.model) && Objects.equals(processor, other.processor)

&& Objects.equals(ram, other.ram);

}

}

**Employee Repository:**

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

import org.springframework.stereotype.Repository;

import com.springboot\_hybernet\_one\_to\_one.model.Employee;

@Repository

public interface EmployeeRepository extends JpaRepository<Employee, Long>{

}

**Laptop Repository:**

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

import org.springframework.stereotype.Repository;

import com.springboot\_hybernet\_one\_to\_one.model.Laptop;

@Repository

public interface LaptopRepository extends JpaRepository<Laptop, Long>{

}

**Employee RestController:**

import java.util.List;

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

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

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

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.RequestMapping;

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

import com.springboot\_hybernet\_one\_to\_one.model.Employee;

import com.springboot\_hybernet\_one\_to\_one.repository.EmployeeRepository;

@RestController

@RequestMapping("/employee")

public class EmployeeRestController {

@Autowired

private EmployeeRepository empRepo;

@PostMapping("/")

public Employee addEmployee(@RequestBody Employee emp) {

return empRepo.save(emp);

}

@GetMapping("/")

public List<Employee> listEmployee(){

return empRepo.findAll();

}

@PutMapping("/")

public Employee updateEmployee(@RequestBody Employee emp) {

return empRepo.save(emp);

}

@DeleteMapping("/")

public String deleteEmployee(@RequestBody Employee emp) {

empRepo.delete(emp);

return "Deleted Employee";

}

}

**Laptop RestController:**

import java.util.List;

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

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

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

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.RequestMapping;

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

import com.springboot\_hybernet\_one\_to\_one.model.Laptop;

import com.springboot\_hybernet\_one\_to\_one.repository.LaptopRepository;

@RestController

@RequestMapping("/laptop")

public class LaptopRestController {

@Autowired

private LaptopRepository lapRepo;

@PostMapping("/")

public Laptop addLaptop(@RequestBody Laptop lap) {

return lapRepo.save(lap);

}

@GetMapping("/")

public List<Laptop> listLaptop() {

return lapRepo.findAll();

}

@PutMapping("/")

public Laptop updateLaptop(@RequestBody Laptop lap) {

return lapRepo.save(lap);

}

@DeleteMapping("/")

public String deleteLaptop(@RequestBody Laptop lap) {

lapRepo.delete(lap);

return "Deleted Laptop";

}

}

**One to Many:**

**Employee class:**

import java.io.Serializable;

//import java.util.List;

import java.util.Set;

import javax.persistence.CascadeType;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.OneToMany;

@Entity

public class Employee implements Serializable {

/\*\*

\*

\*/

private static final long serialVersionUID = 7108602029108606198L;

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private Long eid;

private String name, address;

@OneToMany(cascade = CascadeType.ALL)

private List<Phone> phones;

private Set<Phone> phones;

public Long getEid() {

return eid;

}

public void setEid(Long eid) {

this.eid = eid;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getAddress() {

return address;

}

public void setAddress(String address) {

this.address = address;

}

public Set<Phone> getPhones() {

return phones;

}

public void setPhones(Set<Phone> phones) {

this.phones = phones;

}

}

**Phone class:**

import javax.persistence.Entity;

import javax.persistence.EnumType;

import javax.persistence.Enumerated;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.ManyToOne;

@Entity

public class Phone {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private Long pid;

private String comment, phoneNumber;

@ManyToOne

private Employee employee;

public String getPhoneNumber() {

return phoneNumber;

}

public void setPhoneNumber(String phoneNumber) {

this.phoneNumber = phoneNumber;

}

@Enumerated(EnumType.STRING)

private PHONE\_TYPE phone\_TYPE;

public Long getPid() {

return pid;

}

public void setPid(Long pid) {

this.pid = pid;

}

public String getComment() {

return comment;

}

public void setComment(String comment) {

this.comment = comment;

}

public PHONE\_TYPE getPhone\_TYPE() {

return phone\_TYPE;

}

public void setPhone\_TYPE(PHONE\_TYPE phone\_TYPE) {

this.phone\_TYPE = phone\_TYPE;

}

}

**PHONE\_TYPE class:**

public enum PHONE\_TYPE {

*HOME*, *OFFICE*, *EMERGENCY*, *PERMANENT*;

}

**Employee Repository class:**

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

import org.springframework.stereotype.Repository;

import com.springboot\_hybernet\_one\_to\_many.model.Employee;

@Repository

public interface EmployeeRepository extends JpaRepository<Employee, Long>{

}

**Phone Repository Class:**

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

import org.springframework.stereotype.Repository;

import com.springboot\_hybernet\_one\_to\_many.model.Phone;

@Repository

public interface PhoneRepository extends JpaRepository<Phone, Long>{

}

**Employee RestController class:**

import java.util.List;

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

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

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

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.RequestMapping;

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

import com.springboot\_hybernet\_one\_to\_many.model.Employee;

import com.springboot\_hybernet\_one\_to\_many.repository.EmployeeRepository;

@RestController

@RequestMapping("/employee")

public class EmployeeRestController {

@Autowired

private EmployeeRepository empRepo;

@PostMapping("/")

public Employee addEmployee(@RequestBody Employee emp) {

return empRepo.save(emp);

}

@GetMapping("/")

public List<Employee> listEmployee() {

return empRepo.findAll();

}

@PutMapping("/")

public Employee updateEmployee(@RequestBody Employee emp) {

return empRepo.save(emp);

}

@DeleteMapping("/")

public String deleteEmployee(@RequestBody Employee emp) {

empRepo.delete(emp);

return "Deleted Employee Data";

}

}

**Phone RestController class:**

import java.util.List;

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

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

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

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.RequestMapping;

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

import com.springboot\_hybernet\_one\_to\_many.model.Phone;

import com.springboot\_hybernet\_one\_to\_many.repository.PhoneRepository;

@RestController

@RequestMapping("/phone")

public class PhoneRestController {

@Autowired

private PhoneRepository phnRepo;

@PostMapping("/")

public Phone addPhone(@RequestBody Phone phone) {

return phnRepo.save(phone);

}

@GetMapping("/")

public List<Phone> listPhone() {

return phnRepo.findAll();

}

@PutMapping("/")

public Phone updatePhone(@RequestBody Phone phone) {

return phnRepo.save(phone);

}

@DeleteMapping("/")

public String deletePhone(@RequestBody Phone ph) {

phnRepo.delete(ph);

return "Phone Data Deleted";

}

}

**ManyToMany:**

**Employee Class:**

import java.io.Serializable;

import java.util.HashSet;

import java.util.Set;

import javax.persistence.CascadeType;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.JoinColumn;

import javax.persistence.JoinTable;

import javax.persistence.ManyToMany;

@Entity

public class Employee implements Serializable {

/\*\*

\*

\*/

private static final long serialVersionUID = 7108602029108606198L;

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private Long eid;

private String name, address;

@ManyToMany(cascade = CascadeType.ALL)

@JoinTable(name = "Employee\_Project", joinColumns = { @JoinColumn(name = "employee\_eid") }, inverseJoinColumns = {

@JoinColumn(name = "project\_pid") })

private Set<Project> projects = new HashSet<Project>();

public Long getEid() {

return eid;

}

public void setEid(Long eid) {

this.eid = eid;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getAddress() {

return address;

}

public void setAddress(String address) {

this.address = address;

}

public Set<Project> getProjects() {

return projects;

}

public void setProjects(Set<Project> projects) {

this.projects = projects;

}

}

**Project Class:**

import java.io.Serializable;

import java.util.Set;

import javax.persistence.CascadeType;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

import javax.persistence.ManyToMany;

@Entity

public class Project implements Serializable {

/\*\*

\*

\*/

private static final long serialVersionUID = 6116602877170941471L;

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

private Long pid;

private String title;

@ManyToMany(cascade = CascadeType.ALL, mappedBy = "projects")

private Set<Employee> employees;

public Long getPid() {

return pid;

}

public void setPid(Long pid) {

this.pid = pid;

}

public String getTitle() {

return title;

}

public void setTitle(String title) {

this.title = title;

}

public Set<Employee> getEmployees() {

return employees;

}

public void setEmployees(Set<Employee> employees) {

this.employees = employees;

}

}

**Employee Repository:**

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

import org.springframework.stereotype.Repository;

import com.springboot\_hybernet\_many\_to\_many.model.Employee;

@Repository

public interface EmployeeRepository extends JpaRepository<Employee, Long>{

}

**Project Repository:**

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

import org.springframework.stereotype.Repository;

import com.springboot\_hybernet\_many\_to\_many.model.Project;

@Repository

public interface ProjectRepository extends JpaRepository<Project, Long>{

}

**Employee RestController:**

import java.util.List;

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

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

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

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.RequestMapping;

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

import com.springboot\_hybernet\_many\_to\_many.model.Employee;

import com.springboot\_hybernet\_many\_to\_many.repository.EmployeeRepository;

@RestController

@RequestMapping("/employee")

public class EmployeeRestController {

@Autowired

private EmployeeRepository empRepo;

@PostMapping("/")

public Employee addEmployee(@RequestBody Employee emp) {

return empRepo.save(emp);

}

@GetMapping("/")

public List<Employee> listEmployee() {

return empRepo.findAll();

}

@PutMapping("/")

public Employee updateEmployee(@RequestBody Employee emp) {

return empRepo.save(emp);

}

@DeleteMapping("/")

public String deleteEmployee(@RequestBody Employee emp) {

empRepo.delete(emp);

return "Deleted Employee Data";

}

}

**Project RestController:**

import java.util.List;

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

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

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

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.RequestMapping;

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

import com.springboot\_hybernet\_many\_to\_many.model.Project;

import com.springboot\_hybernet\_many\_to\_many.repository.ProjectRepository;

@RestController

@RequestMapping("/project")

public class ProjectRestController {

@Autowired

private ProjectRepository prjctRepo;

@PostMapping("/")

public Project addProject(@RequestBody Project prjt) {

return prjctRepo.save(prjt);

}

@GetMapping("/")

public List<Project> listProjects() {

return prjctRepo.findAll();

}

@PutMapping("/")

public Project updateProject(@RequestBody Project prjt) {

return prjctRepo.save(prjt);

}

@DeleteMapping("/")

public String deleteProject(@RequestBody Project prjt) {

prjctRepo.delete(prjt);

return "Project Data Deleted";

}

}