AIM: Write a program to demonstrate RESTful Web Services with spring boot.

THEORY:

RESTful Web Services :

REST stands for REpresentational State Transfer. REST is web standards based architecture and uses HTTP Protocol. It revolves around resource where every component is a resource and a resource is accessed by a common interface using HTTP standard methods. REST was first introduced by Roy Fielding in 2000.

Restful Web Services is a lightweight, maintainable, and scalable service that is built on the REST architecture. Restful Web Service, expose API from your application in a secure, uniform, stateless manner to the calling client. The calling client can perform predefined operations using the Restful service. The underlying protocol for REST is HTTP. REST stands for REpresentational State Transfer.

The key elements of a RESTful implementation are as follows:

1. **Resources** – The first key element is the resource itself. Let assume that a web application on a server has records of several employees. Let's assume the URL of the web application is **http://demo.guru99.com**. Now in order to access an employee record resource via REST services, one can issue the command **http://demo.example.com/employee/1 -** This command tells the web server to please provide the details of the employee whose employee number is 1.
2. **Request Verbs -** These describe what you want to do with the resource. A browser issues a GET verb to instruct the endpoint it wants to get data. However, there are many other verbs available including things like POST, PUT, and DELETE. So in the case of the example **http://demo.example.com/employee/1** , the web browser is actually issuing a GET Verb because it wants to get the details of the employee record.
3. **Request Headers** – These are additional instructions sent with the request. These might define the type of response required or the authorization details.
4. **Request Body -** Data is sent with the request. Data is normally sent in the request when a POST request is made to the REST web services. In a POST call, the client actually tells the REST web services that it wants to add a resource to the server. Hence, the request body would have the details of the resource which is required to be added to the server.
5. **Response Body** – This is the main body of the response. So in our RESTful API example, if we were to query the web server via the request **http://demo.example.com/employee/1** , the web server might return an XML document with all the details of the employee in the Response Body.
6. **Response Status codes** – These codes are the general codes which are returned along with the response from the web server. An example is the code 200 which is normally returned if there is no error when returning a response to the client.

A) **Write a program to demonstrate RESTful Web Services with spring boot. [POST, PUT, GET, DELETE]**

SOURCE CODE:

Pom.xml:

```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
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>restful</groupId>
  <artifactId>narender</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>

  <name>narender</name>
  <!-- FIXME change it to the project's website -->
  <url>http://www.example.com</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.7</maven.compiler.source>
    <maven.compiler.target>1.7</maven.compiler.target>
  </properties>

  <parent>
     <groupId>org.springframework.boot</groupId>
     <artifactId>spring-boot-starter-parent</artifactId>
     <version>1.5.4.RELEASE</version>
     <relativePath/> <!-- lookup parent from repository -->
  </parent>
   <!-- project properties repository -->
  <dependencies>
     <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-web</artifactId>
     </dependency>

     <dependency>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-tomcat</artifactId>
        <scope>provided</scope>
     </dependency>
        <!-- Spring boot test depedency -->
  </dependencies>
</project>
```

Employee.java:

```
package restful.narender;

public class Employee {
public int id;
public String name;
public int age;
public double salary;

public Employee()
{
}

public Employee(int id, String name, int age, double salary) {
super();
this.id = id;
this.name = name;
this.age = age;
this.salary = salary;
}
public int getId() {
return id;
}
public void setId(int id) {
this.id = id;
}
public String getName() {
return name;
}
public void setName(String name) {
this.name = name;
}
public int getAge() {
return age;
}
public void setAge(int age) {
this.age = age;
}
public double getSalary() {
return salary;
}
public void setSalary(double salary) {
this.salary = salary;
}
}
```

EmployeeServiceController.java:

```
package restful.narender;

import java.util.HashMap;
import java.util.Map;

import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class EmployeeServiceController {

 private static Map<Integer, Employee> employeeRepo = new HashMap<>();
   static {
      Employee e1 = new Employee();
     e1.setId(1);
     e1.setName("Narender Keswani");
     e1.setAge(21);
     e1.setSalary(50000.00);
     employeeRepo.put(e1.getId(), e1);

     Employee e2 = new Employee();
     e2.setId(2);
     e2.setName("Neel Deshmukh");
     e2.setAge(20);
     e2.setSalary(70000.00);
     employeeRepo.put(e2.getId(), e2);
   }

   @RequestMapping(value = "/employees/{id}", method = RequestMethod.DELETE)
   public ResponseEntity<Object> delete(@PathVariable("id") Integer id) {
     employeeRepo.remove(id);
     return new ResponseEntity<Object>("Employee is deleted successsfully", HttpStatus.OK);
   }

   @RequestMapping(value = "/employees/{id}", method = RequestMethod.PUT)
   public ResponseEntity<Object> updateEmployee(@PathVariable("id") Integer id,
@RequestBody Employee employee) {
     employeeRepo.remove(id);
     employee.setId(id);
     employeeRepo.put(id, employee);
     return new ResponseEntity<Object>("Employee is updated successsfully",
HttpStatus.OK);
   }

   @RequestMapping(value = "/employees", method = RequestMethod.POST)
```

```
  public ResponseEntity<Object> createEmployee(@RequestBody Employee employee) {
    employeeRepo.put(employee.getId(), employee);
    return new ResponseEntity<Object>("Employee is created successfully",
HttpStatus.CREATED);
  }

  @RequestMapping(value = "/employees")
  public ResponseEntity<Object> getEmployee() {
    return new ResponseEntity<Object>(employeeRepo.values(), HttpStatus.OK);
  }

  @RequestMapping(value = "/employee/{id}", method = RequestMethod.GET)
  public ResponseEntity<Object> getSpecificeEmployee(@PathVariable("id") Integer id) {
    return new ResponseEntity<Object>(employeeRepo.get(id), HttpStatus.OK);
  }

  @RequestMapping(method=RequestMethod.GET, path="/")
  public String helloWorld() {
            return "Hello & Bye from Narender Keswani";
         }
}
```

App.java:

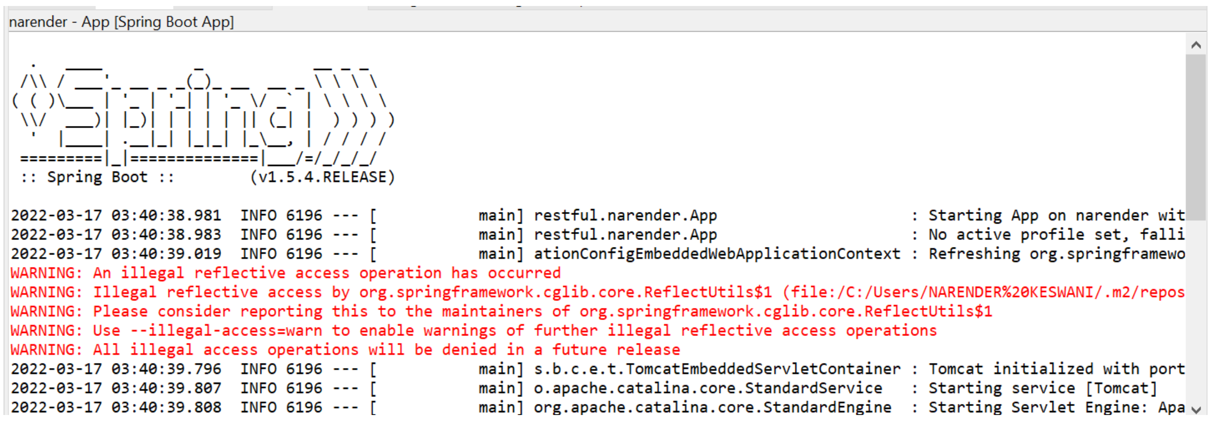
```
package restful.narender;

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

/**
 * Narender Keswani
 *
 */
@SpringBootApplication
public class App
{
   public static void main( String[] args )
   {
          SpringApplication.run(App.class, args);

   }
}
```

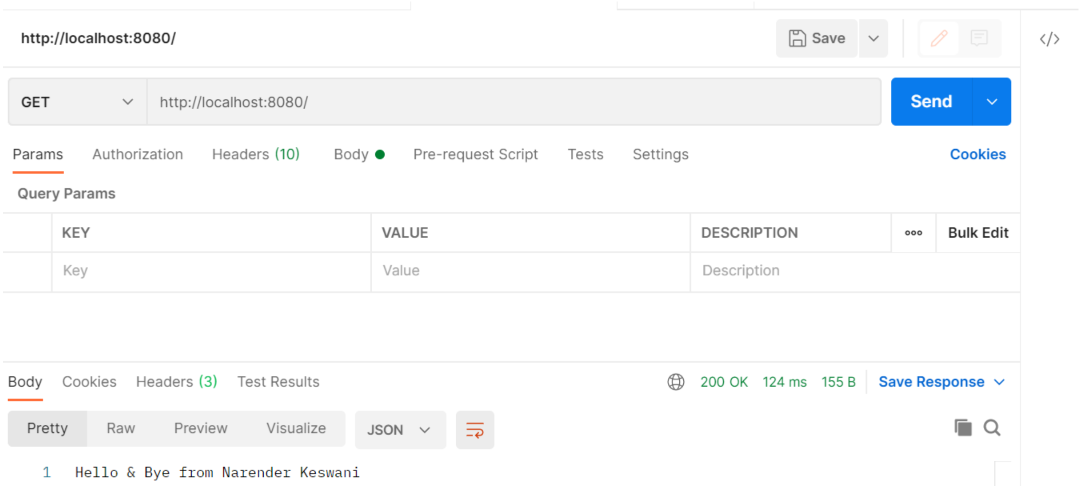
OUTPUT:


narender - App [Spring Boot App]

```
  .   ____          _            __ _ _
 /\\ / ___'_ __ _ _(_)_ __  __ _ \ \ \ \
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/
 :: Spring Boot ::        (v1.5.4.RELEASE)

2022-03-17 03:40:38.981  INFO 6196 --- [           main] restful.narender.App                     : Starting App on narender wit
2022-03-17 03:40:38.983  INFO 6196 --- [           main] restful.narender.App                     : No active profile set, falli
2022-03-17 03:40:39.019  INFO 6196 --- [           main] ationConfigEmbeddedWebApplicationContext : Refreshing org.springframewo
WARNING: An illegal reflective access operation has occurred
WARNING: Illegal reflective access by org.springframework.cglib.core.ReflectUtils$1 (file:/C:/Users/NARENDER%20KESWANI/.m2/repos
WARNING: Please consider reporting this to the maintainers of org.springframework.cglib.core.ReflectUtils$1
WARNING: Use --illegal-access=warn to enable warnings of further illegal reflective access operations
WARNING: All illegal access operations will be denied in a future release
2022-03-17 03:40:39.796  INFO 6196 --- [           main] s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat initialized with port
2022-03-17 03:40:39.807  INFO 6196 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2022-03-17 03:40:39.808  INFO 6196 --- [           main] org.apache.catalina.core.StandardEngine  : Starting Servlet Engine: Apa
```

Hello GET:



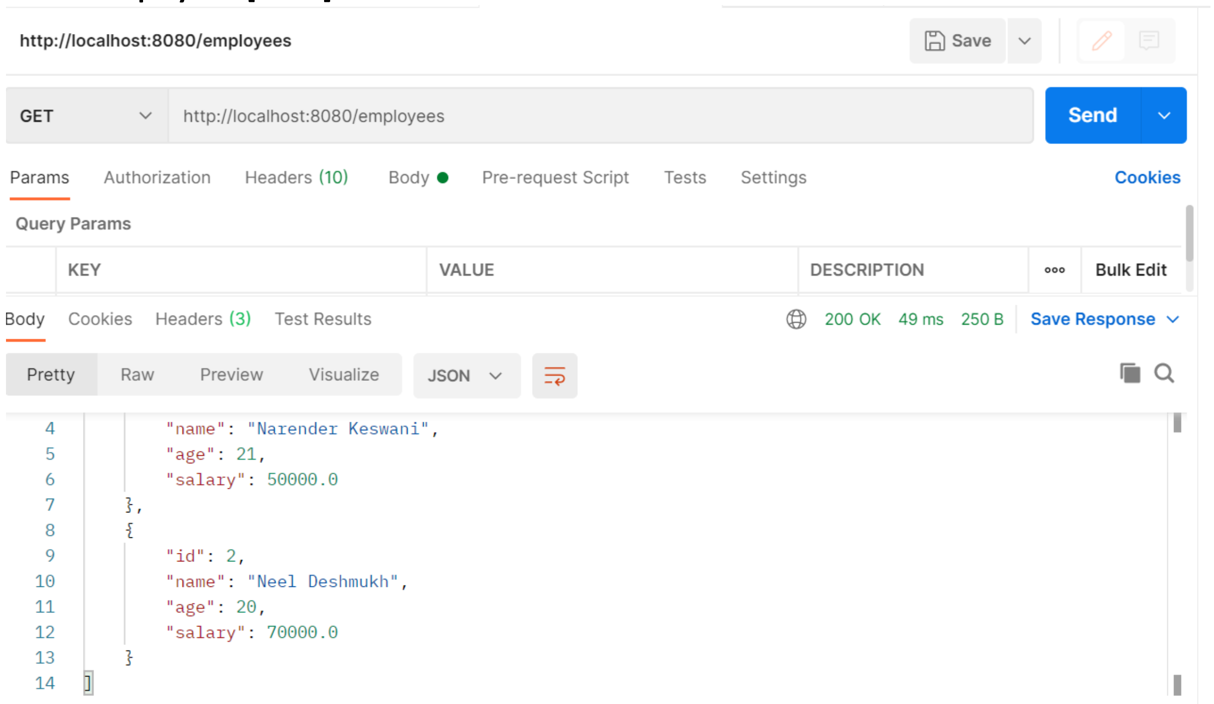
http://localhost:8080/

GET http://localhost:8080/

200 OK 124 ms 155 B

```
Hello & Bye from Narender Keswani
```

GET All Employees: [READ]



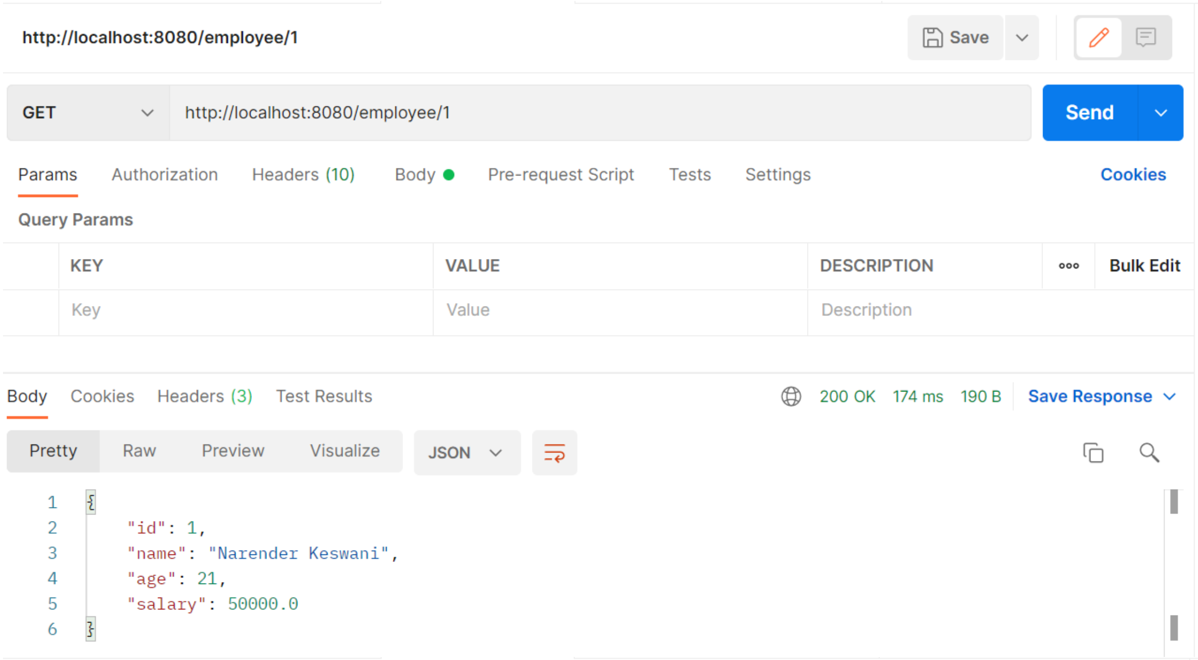
http://localhost:8080/employees

GET http://localhost:8080/employees

200 OK 49 ms 250 B

```
        "name": "Narender Keswani",
        "age": 21,
        "salary": 50000.0
    },
    {
        "id": 2,
        "name": "Neel Deshmukh",
        "age": 20,
        "salary": 70000.0
    }
]
```

GET Specific Employee:



http://localhost:8080/employee/1

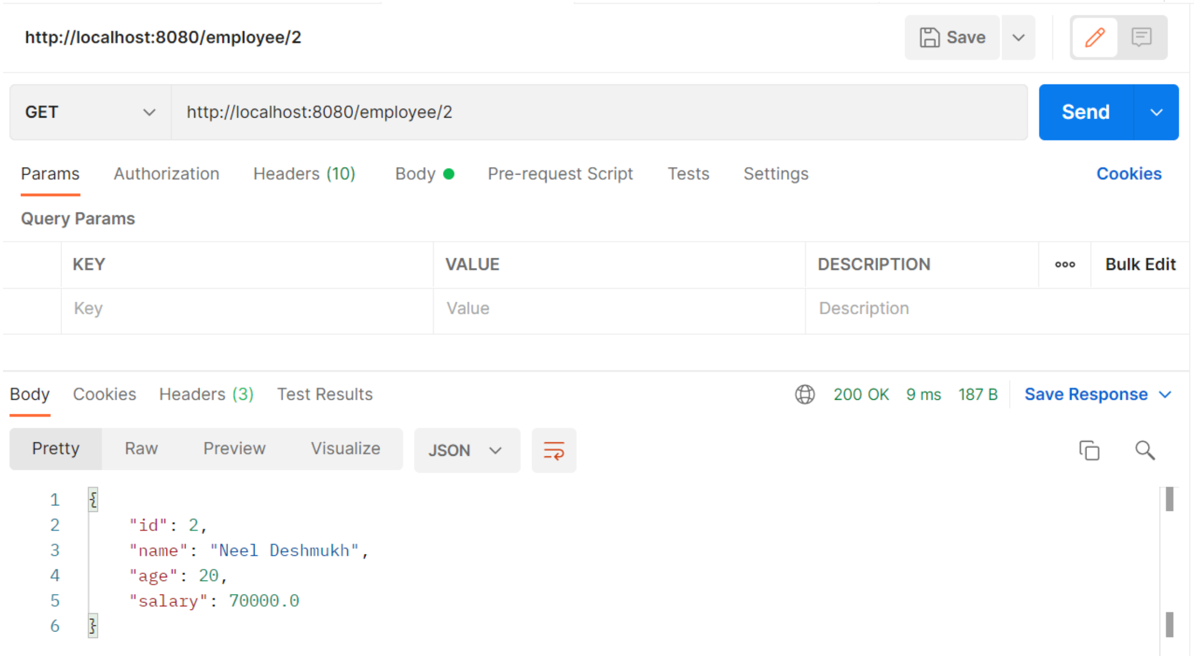
GET http://localhost:8080/employee/1

Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

200 OK 174 ms 190 B

```
{
    "id": 1,
    "name": "Narender Keswani",
    "age": 21,
    "salary": 50000.0
}
```



http://localhost:8080/employee/2

GET http://localhost:8080/employee/2

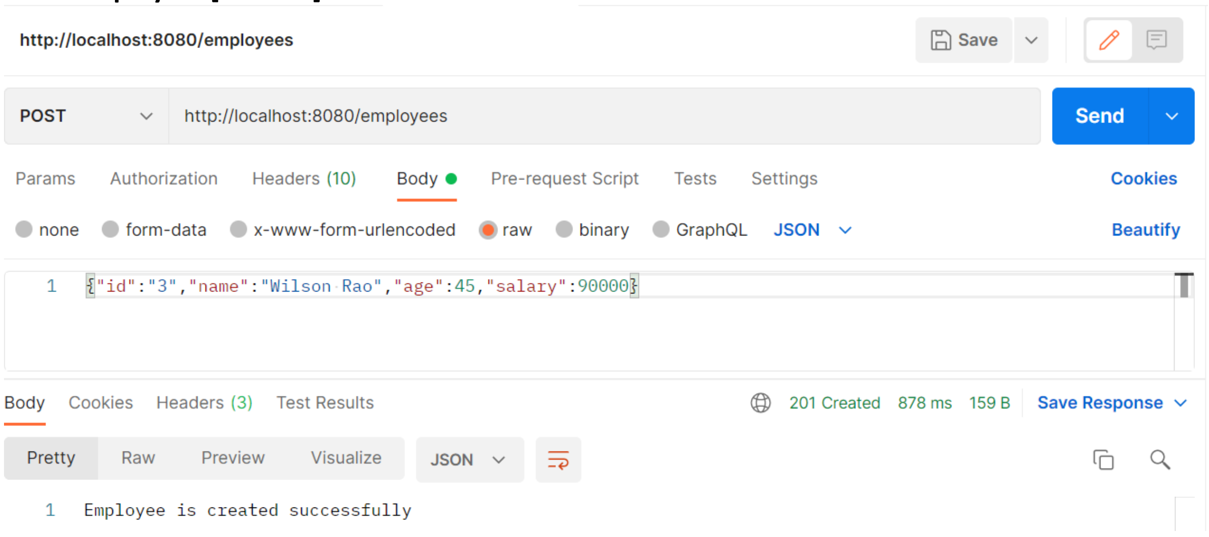
Query Params

KEY	VALUE	DESCRIPTION
Key	Value	Description

200 OK 9 ms 187 B

```
{
    "id": 2,
    "name": "Neel Deshmukh",
    "age": 20,
    "salary": 70000.0
}
```

POST Employee: [CREATE]


http://localhost:8080/employees

POST http://localhost:8080/employees

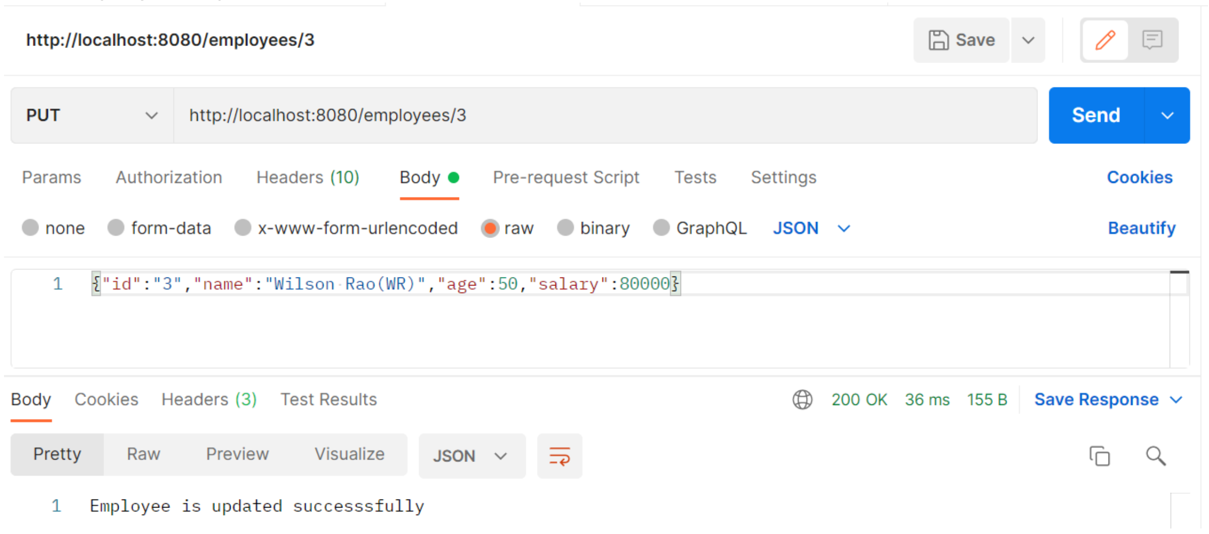
Body: raw JSON

```
{"id":"3","name":"Wilson Rao","age":45,"salary":90000}
```

201 Created 878 ms 159 B

```
Employee is created successfully
```

PUT Employee [Update]:


http://localhost:8080/employees/3

PUT http://localhost:8080/employees/3

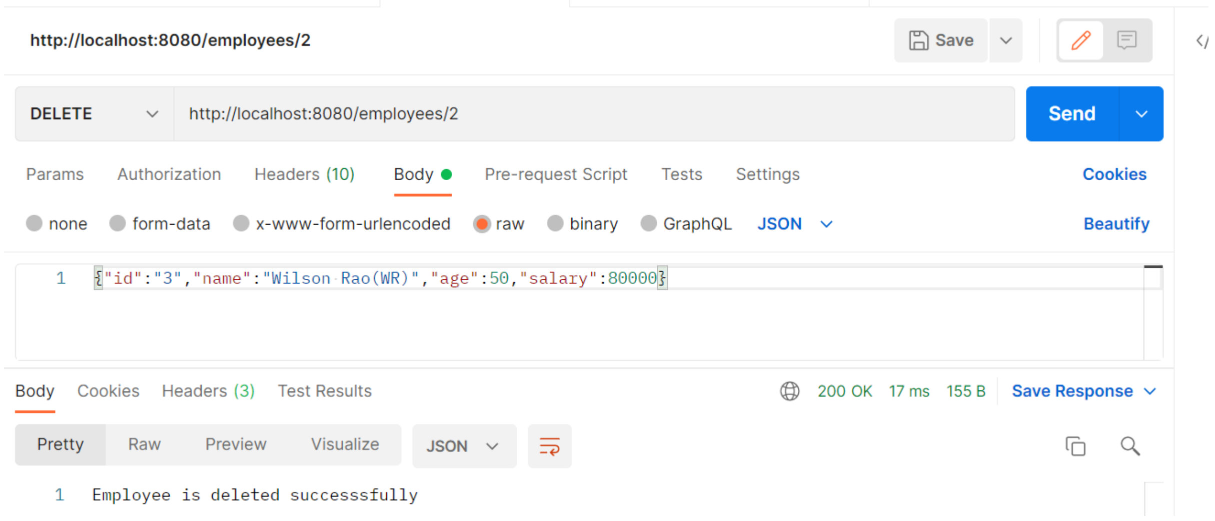
Body: raw JSON

```
{"id":"3","name":"Wilson Rao(WR)","age":50,"salary":80000}
```

200 OK 36 ms 155 B

```
Employee is updated successsfully
```

DELETE Employee:


http://localhost:8080/employees/2

DELETE http://localhost:8080/employees/2

Body: raw JSON

```
{"id":"3","name":"Wilson Rao(WR)","age":50,"salary":80000}
```

200 OK 17 ms 155 B

```
Employee is deleted successsfully
```

CONCLUSION:

From this practical, I have learned how to create and implement REST API in spring.