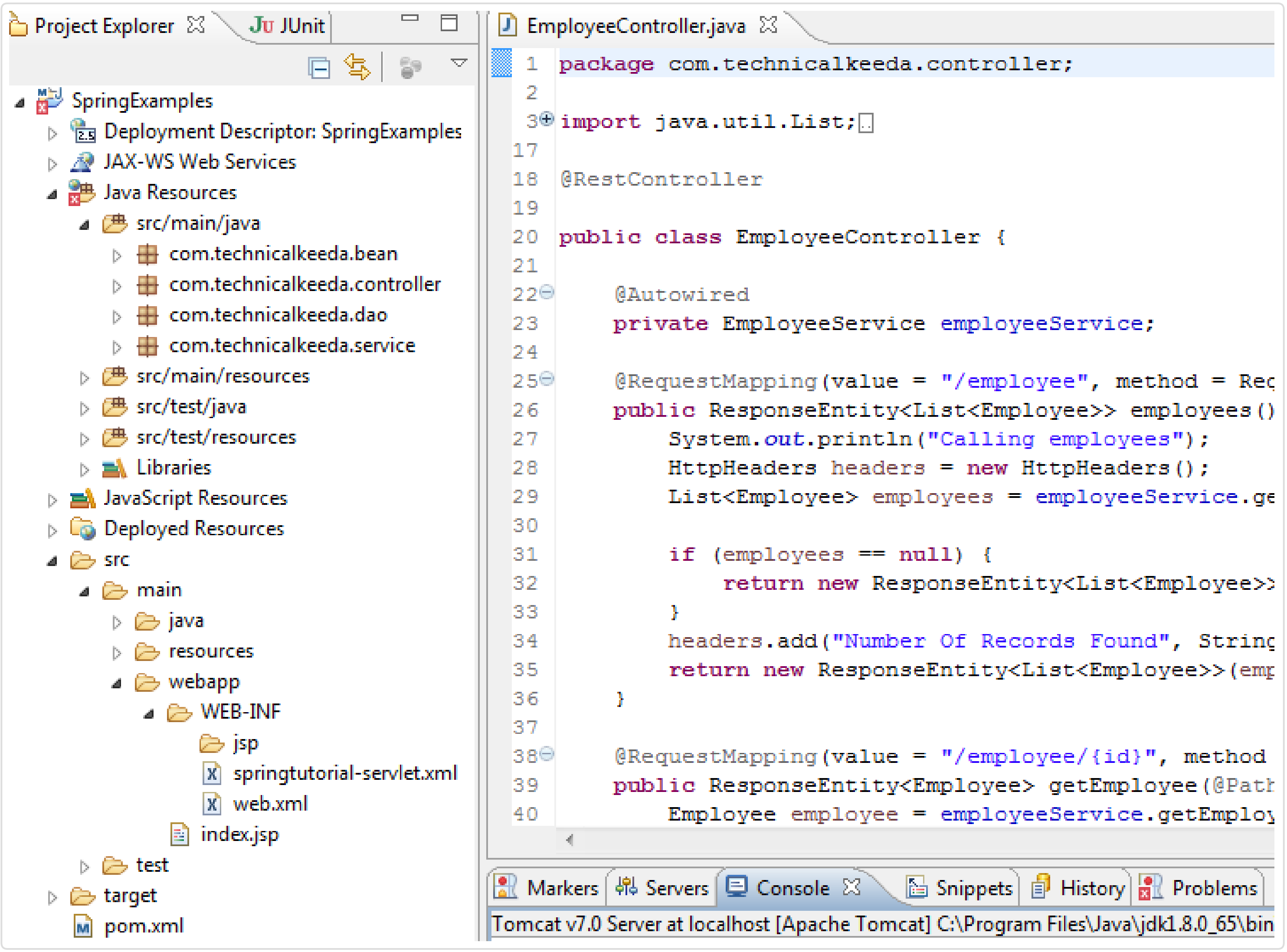
Complete Spring MVC RESTful Web Services Example

Posted On 2016-05-23 | Yashwant Chavan (https://plus.google.com/116241548760694569961?rel=author)

G+ 7 0

In this Spring MVC RESTful Web Services tutorial, We will learn how to build RESTFul APIs which will return the response object in JSON format. Here we will develop employee CRUD application using Spring Rest API. So Let start with step by step guide.

Spring Mvc Maven Project setup



```java
package com.technicalkeeda.controller;

import java.util.List;

@RestController

public class EmployeeController {

    @Autowired
    private EmployeeService employeeService;

    @RequestMapping(value = "/employee", method = Rec
    public ResponseEntity<List<Employee>> employees()
        System.out.println("Calling employees");
        HttpHeaders headers = new HttpHeaders();
        List<Employee> employees = employeeService.ge

        if (employees == null) {
            return new ResponseEntity<List<Employee>>
        }
        headers.add("Number Of Records Found", String
        return new ResponseEntity<List<Employee>>(emp
    }

    @RequestMapping(value = "/employee/{id}", method
    public ResponseEntity<Employee> getEmployee(@Path
        Employee employee = employeeService.getEmploy
```

```
Tomcat v7.0 Server at localhost [Apache Tomcat] C:\Program Files\Java\jdk1.8.0_65\bin
```

Tools and Technologies

1. Apache Maven 3.0.4
2. JDK 1.8
3. Spring 4.1.4.RELEASE

pom.xml

As we are using maven project. Let's define the spring specific maven dependencies.

```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instanc
 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>com.technicalkeeda</groupId>
 <artifactId>SpringExamples</artifactId>
 <packaging>war</packaging>
 <version>1.0</version>
 <name>SpringExamples</name>
 <description></description>
 <build>
  <plugins>
   <plugin>
    <artifactId>maven-compiler-plugin</artifactId>
    <configuration>
     <source>1.5</source>
     <target>1.5</target>
    </configuration>
   </plugin>
   <plugin>
    <artifactId>maven-war-plugin</artifactId>
    <version>2.2</version>
    <configuration>
     <source>1.7</source>
     <target>1.7</target>
     <failOnMissingWebXml>false</failOnMissingWebXml>
    </configuration>
   </plugin>
  </plugins>
 </build>

 <properties>
  <spring.version>4.1.4.RELEASE</spring.version>
 </properties>

 <dependencies>
  <dependency>
   <groupId>javax.servlet</groupId>
   <artifactId>servlet-api</artifactId>
   <version>2.5</version>
  </dependency>

  <dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-core</artifactId>
   <version>${spring.version}</version>
  </dependency>

  <dependency>
   <groupId>org.springframework</groupId>
```

```
    <artifactId>spring-web</artifactId>
    <version>${spring.version}</version>
   </dependency>

   <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>${spring.version}</version>
   </dependency>

   <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>${spring.version}</version>
   </dependency>

   <dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.4.1</version>
   </dependency>

   <dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>5.1.9</version>
   </dependency>

   <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-jdbc</artifactId>
    <version>${spring.version}</version>
   </dependency>
   <dependency>
    <groupId>commons-dbcp</groupId>
    <artifactId>commons-dbcp</artifactId>
    <version>1.2.2</version>
   </dependency>
  </dependencies>
 </project>
```

web.xml

Define the dispatcher Servlet which is front controller in spring mvc.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app id="WebApp_ID" version="2.4"
 xmlns="http://java.sun.com/xml/ns/j2ee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
 <display-name>SpringExamples</display-name>
 <welcome-file-list>
  <welcome-file>index.jsp</welcome-file>
 </welcome-file-list>

 <servlet>
  <servlet-name>springtutorial</servlet-name>
  <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
 </servlet>
 <servlet-mapping>
  <servlet-name>springtutorial</servlet-name>
  <url-pattern>/</url-pattern>
 </servlet-mapping>

 <welcome-file-list>
  <welcome-file>index.jsp</welcome-file>
 </welcome-file-list>
</web-app>
```

Spring datasource configuration

```
jdbc.driverClassName=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/technicalkeeda
jdbc.username=root
jdbc.password=
```

Spring Configuration File

Lets define the spring specific configurations in springexamples-servlet.xml file. This file is located under `/WEB-INF/..` folder.

Here we are using annotation to define the RestController class and its request handler. To process all the annotation we have provided base package *com.technicalkeeda*

`DefaultAnnotationHandlerMapping` is used to map request with class and/or methods that are annotated with `@RequestMapping` annotation

```
<beans xmlns="http://www.springframework.org/schema/beans"
 xmlns:context="http://www.springframework.org/schema/context"
 xmlns:mvc="http://www.springframework.org/schema/mvc" xmlns:xsi="http://www.w3.org/2001/XMLSchema-ins
 xsi:schemaLocation=" http://www.springframework.org/schema/beans http://www.springframework.org/schem

 <mvc:annotation-driven />
 <context:component-scan base-package="com.technicalkeeda" />

 <bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource" destroy-method="close">
  <property name="driverClassName" value="${jdbc.driverClassName}" />
  <property name="url" value="${jdbc.url}" />
  <property name="username" value="${jdbc.username}" />
  <property name="password" value="${jdbc.password}" />
 </bean>

 <context:property-placeholder location="classpath:jdbc.properties" />

 <bean class="org.springframework.web.servlet.mvc.annotation.DefaultAnnotationHandlerMapping" />
 <bean class="org.springframework.web.servlet.mvc.annotation.AnnotationMethodHandlerAdapter" />
 <bean id="viewResolver" class="org.springframework.web.servlet.view.InternalResourceViewResolver">
  <property name="prefix"><value>/WEB-INF/jsp/</value></property>
  <property name="suffix"><value>.jsp</value></property>
 </bean>
</beans>
```

Employee RestController

This is simple Employee Rest Controller class which accept HTTP web-service requests and respond with a JSON representation. To build the RESTful web services using Spring you need specify the rest Controller using `@RestController` annotation along with implementation methods.

Each method is declared with `@RequestMapping` annotation which ensures that HTTP request is mapped to respected method implementation. e.g. "/employee" is mapped to employees() method.

```
package com.technicalkeeda.controller;

import java.util.List;

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

import com.technicalkeeda.bean.Employee;
import com.technicalkeeda.service.EmployeeService;

@RestController
public class EmployeeController {

 @Autowired
 private EmployeeService employeeService;

 @RequestMapping(value = "/employee", method = RequestMethod.GET, produces = "application/json")
 public ResponseEntity<List<Employee>> employees() {

  HttpHeaders headers = new HttpHeaders();
  List<Employee> employees = employeeService.getEmployees();

  if (employees == null) {
   return new ResponseEntity<List<Employee>>(HttpStatus.NOT_FOUND);
  }
  headers.add("Number Of Records Found", String.valueOf(employees.size()));
  return new ResponseEntity<List<Employee>>(employees, headers, HttpStatus.OK);
 }

 @RequestMapping(value = "/employee/{id}", method = RequestMethod.GET)
 public ResponseEntity<Employee> getEmployee(@PathVariable("id") Long employeeId) {
  Employee employee = employeeService.getEmployee(employeeId);
  if (employee == null) {
   return new ResponseEntity<Employee>(HttpStatus.NOT_FOUND);
  }
  return new ResponseEntity<Employee>(employee, HttpStatus.OK);
 }

 @RequestMapping(value = "/employee/delete/{id}", method = RequestMethod.DELETE)
 public ResponseEntity<Employee> deleteEmployee(@PathVariable("id") Long employeeId) {
  HttpHeaders headers = new HttpHeaders();
  Employee employee = employeeService.getEmployee(employeeId);
```

```
   if (employee == null) {
    return new ResponseEntity<Employee>(HttpStatus.NOT_FOUND);
   }
   employeeService.deleteEmployee(employeeId);
   headers.add("Employee Deleted - ", String.valueOf(employeeId));
   return new ResponseEntity<Employee>(employee, headers, HttpStatus.NO_CONTENT);
  }

  @RequestMapping(value = "/employee", method = RequestMethod.POST,produces = "application/json")
  public ResponseEntity<Employee> createEmployee(@RequestBody Employee employee) {
   HttpHeaders headers = new HttpHeaders();
   if (employee == null) {
    return new ResponseEntity<Employee>(HttpStatus.BAD_REQUEST);
   }
   employeeService.createEmployee(employee);
   headers.add("Employee Created  - ", String.valueOf(employee.getEmployeeId()));
   return new ResponseEntity<Employee>(employee, headers, HttpStatus.CREATED);
  }

  @RequestMapping(value = "/employee/{id}", method = RequestMethod.PUT)
  public ResponseEntity<Employee> updateEmployee(@PathVariable("id") Long employeeId, @RequestBody Empl
   HttpHeaders headers = new HttpHeaders();
   Employee isExist = employeeService.getEmployee(employeeId);
   if (isExist == null) {
    return new ResponseEntity<Employee>(HttpStatus.NOT_FOUND);
   } else if (employee == null) {
    return new ResponseEntity<Employee>(HttpStatus.BAD_REQUEST);
   }
   employeeService.updateEmployee(employee);
   headers.add("Employee Updated  - ", String.valueOf(employeeId));
   return new ResponseEntity<Employee>(employee, headers, HttpStatus.OK);
  }

 }
```

Employee Bean

Simple Employee Pojo class with getter setter methods.

```
package com.technicalkeeda.bean;

import com.fasterxml.jackson.annotation.JsonCreator;
import com.fasterxml.jackson.annotation.JsonIgnoreProperties;
import com.fasterxml.jackson.annotation.JsonProperty;

@JsonIgnoreProperties(ignoreUnknown = true)
public class Employee {

 private Long employeeId;
 private String firstName;
 private String lastName;
 private Long age;

 @JsonCreator
 public Employee(@JsonProperty("employeeId") Long employeeId, @JsonProperty("firstName") String firstN
   @JsonProperty("lastName") String lastName, @JsonProperty("age") Long age) {
  this.employeeId = employeeId;
  this.firstName = firstName;
  this.lastName = lastName;
  this.age = age;
 }

 public Employee() {

 }

 public Long getEmployeeId() {
  return employeeId;
 }

 public void setEmployeeId(Long employeeId) {
  this.employeeId = employeeId;
 }

 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 Long getAge() {
  return age;
 }

 public void setAge(Long age) {
  this.age = age;
 }


 @Override
 public String toString() {
  StringBuilder str = new StringBuilder();
  str.append("Employee Id:- " + getEmployeeId());
  str.append(" First Name:- " + getFirstName());
  str.append(" Last Name:- " + getLastName());
  str.append(" Age:- " + getAge());
  return str.toString();
 }

}
```

Employee Service Interface

Let's declare the Employee Service Interface methods.

```
package com.technicalkeeda.service;

import java.util.List;

import com.technicalkeeda.bean.Employee;

public interface EmployeeService {
 public List<Employee> getEmployees();
 public Employee getEmployee(Long employeeId);
 public int deleteEmployee(Long employeeId);
 public int updateEmployee(Employee employee);
 public int createEmployee(Employee employee);
}
```

Employee Service Implementation

Employee Service Implementation class , which is used to make call to Employee DAO method.

```
package com.technicalkeeda.service;

import java.util.List;

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

import com.technicalkeeda.bean.Employee;
import com.technicalkeeda.dao.EmployeeDao;

@Service("employeeService")
public class EmployeeServiceImpl implements EmployeeService {
 @Autowired
 private EmployeeDao employeeDao;

 public List<Employee> getEmployees() {
  List<Employee> employees = employeeDao.getEmployees();
  return employees;
 }

 public Employee getEmployee(Long employeeId) {
  Employee employee = employeeDao.getEmployee(employeeId);
  return employee;
 }

 public int deleteEmployee(Long employeeId) {
  return employeeDao.deleteEmployee(employeeId);
 }

 public int updateEmployee(Employee employee) {
  return employeeDao.updateEmployee(employee);
 }

 public int createEmployee(Employee employee) {
  return employeeDao.createEmployee(employee);
 }
}
```

Employee Dao Interface

Employee Dao Interface methods.

```
package com.technicalkeeda.dao;

import java.util.List;

import com.technicalkeeda.bean.Employee;

public interface EmployeeDao {
 public List<Employee> getEmployees();
 public Employee getEmployee(Long employeeId);
 public int deleteEmployee(Long employeeId);
 public int updateEmployee(Employee employee);
 public int createEmployee(Employee employee);
}
```

Employee Dao Implementation

Employee Dao implementation class which perform the CRUD operation on trn_employee table.

```
package com.technicalkeeda.dao;

import java.util.List;

import javax.sql.DataSource;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.dao.DataAccessException;
import org.springframework.jdbc.core.BeanPropertyRowMapper;
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.stereotype.Repository;

import com.technicalkeeda.bean.Employee;

@Repository("employeeDao")
public class EmployeeDaoImpl implements EmployeeDao {

 private JdbcTemplate jdbcTemplate;

 @Autowired
 public void setDataSource(DataSource dataSource) {
  this.jdbcTemplate = new JdbcTemplate(dataSource);
 }

 public List<Employee> getEmployees() {
  List<Employee> employees = null ;

  try {
   employees = jdbcTemplate.query("SELECT * FROM trn_employee",new BeanPropertyRowMapper<Employee>(Emp
  } catch (DataAccessException e) {
   e.printStackTrace();
  }
  return employees;
 }

 public Employee getEmployee(Long employeeId) {
  Employee employee = null;
  try {
   employee = jdbcTemplate.queryForObject("SELECT * FROM trn_employee WHERE employee_id = ?",
     new Object[] { employeeId }, new BeanPropertyRowMapper<Employee>(Employee.class));
  } catch (DataAccessException e) {
   e.printStackTrace();
  }
  return employee;

 }

 public int deleteEmployee(Long employeeId) {
  int count = jdbcTemplate.update("DELETE from trn_employee WHERE employee_id = ?", new Object[] { emp
  return count;
 }

 public int updateEmployee(Employee employee) {
  int count = jdbcTemplate.update(
    "UPDATE trn_employee set first_name = ? , last_name = ? , age = ? where employee_id = ?", new Obje
      employee.getFirstName(), employee.getLastName(), employee.getAge(), employee.getEmployeeId() });
  return count;
 }

 public int createEmployee(Employee employee) {
  int count = jdbcTemplate.update(
    "INSERT INTO trn_employee(employee_id,first_name, last_name, age)VALUES(?,?,?,?)", new Object[] {
      employee.getEmployeeId(), employee.getFirstName(), employee.getLastName(), employee.getAge() });
  return count;
 }

}
```

Output

To verify this Rest API, I am using POSTMAN rest client.

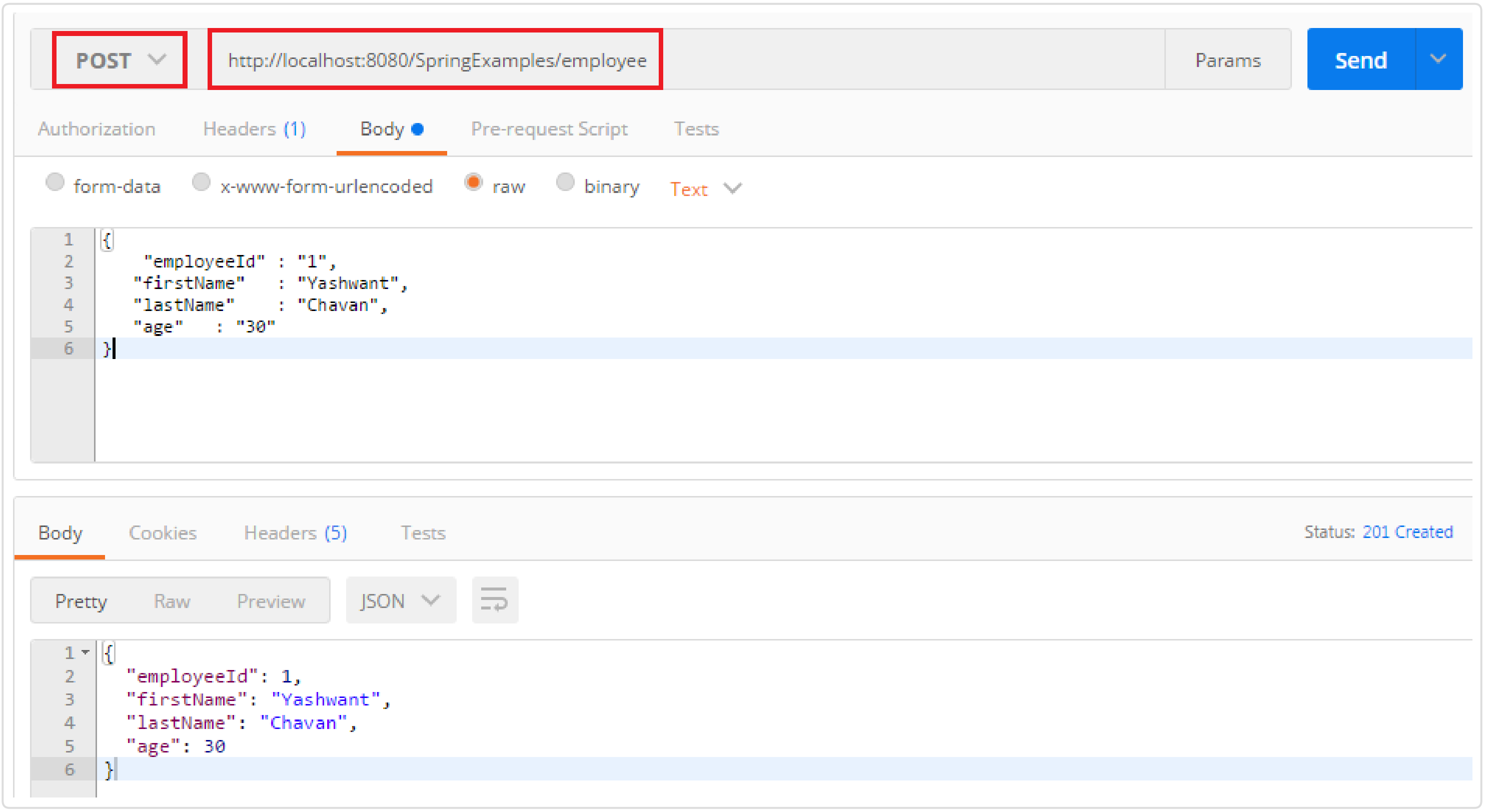
1] Create Employee

Please follow the below instructions to send a POST request using POSTMAN client.

1. Choose **POST** to be the selected HTTP method.
2. Specify the request URL as **http://localhost:8080/SpringExamples/employee**
3. Add request header value as **Content-Type** as **application/json**
4. Specify Employee data in JSON format.
5. Click on Send Button.
6. Once you submitted the request it will return the response. Same way you can create the multiple Employee records.

Sample employee data in JSON format.

```
{
    "employeeId" : "1",
   "firstName"   : "Yashwant",
   "lastName"    : "Chavan",
   "age"   : "30"
}
```

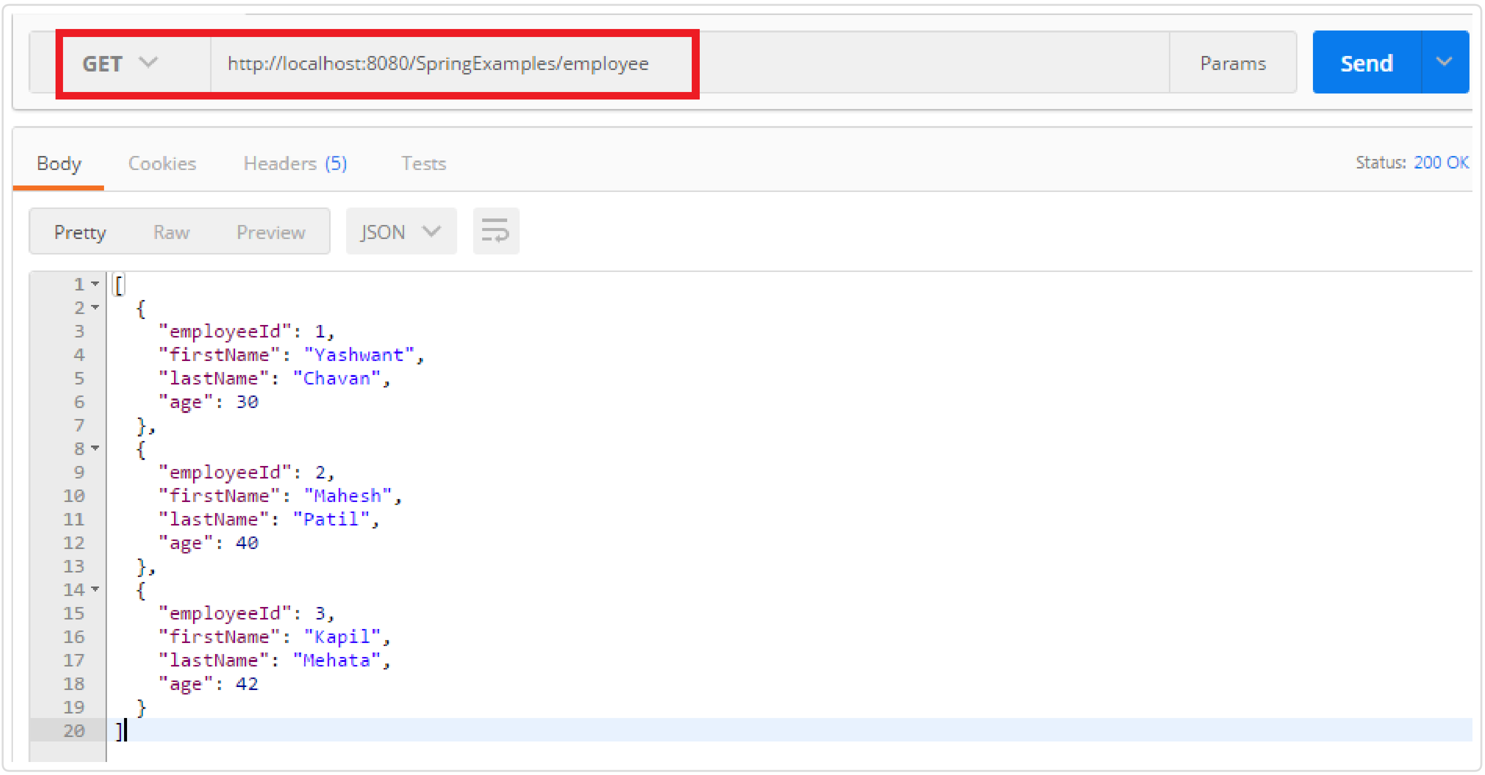


```
{
   "employeeId" : "1",
  "firstName"   : "Yashwant",
  "lastName"    : "Chavan",
  "age"   : "30"
}
```

```
{
  "employeeId": 1,
  "firstName": "Yashwant",
  "lastName": "Chavan",
  "age": 30
}
```

2] Retrieve all employees

1. Choose **GET** to be the selected HTTP method.
2. Specify the request URL as **http://localhost:8080/SpringExamples/employee**
3. Add request header value as **Content-Type** ad **application/json**
4. Click on Send Button.
5. It will display all records from the database using Spring rest controller method. Refer below image.

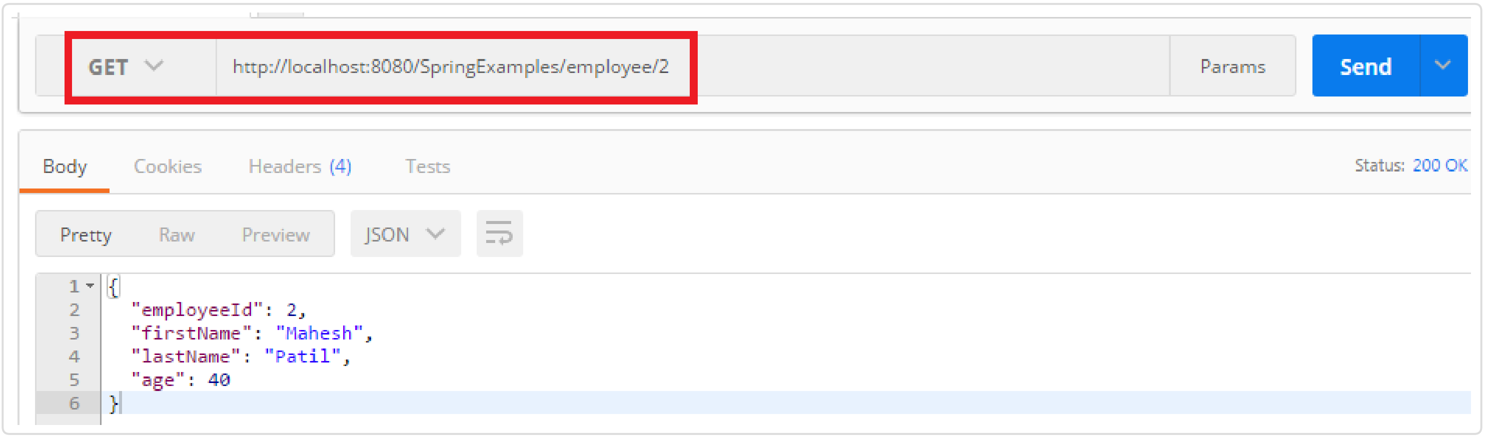


http://localhost:8080/SpringExamples/employee

```
[
  {
    "employeeId": 1,
    "firstName": "Yashwant",
    "lastName": "Chavan",
    "age": 30
  },
  {
    "employeeId": 2,
    "firstName": "Mahesh",
    "lastName": "Patil",
    "age": 40
  },
  {
    "employeeId": 3,
    "firstName": "Kapil",
    "lastName": "Mehata",
    "age": 42
  }
]
```

3] Get By Employee Id

1. Choose **GET** to be the selected HTTP method.
2. Specify the request URL as **http://localhost:8080/SpringExamples/employee/2**
3. Add request header value as **Content-Type** as **application/json**
4. Click on Send Button.
5. It will return single Employee record whose employeeId is 2. Refer below image.



http://localhost:8080/SpringExamples/employee/2

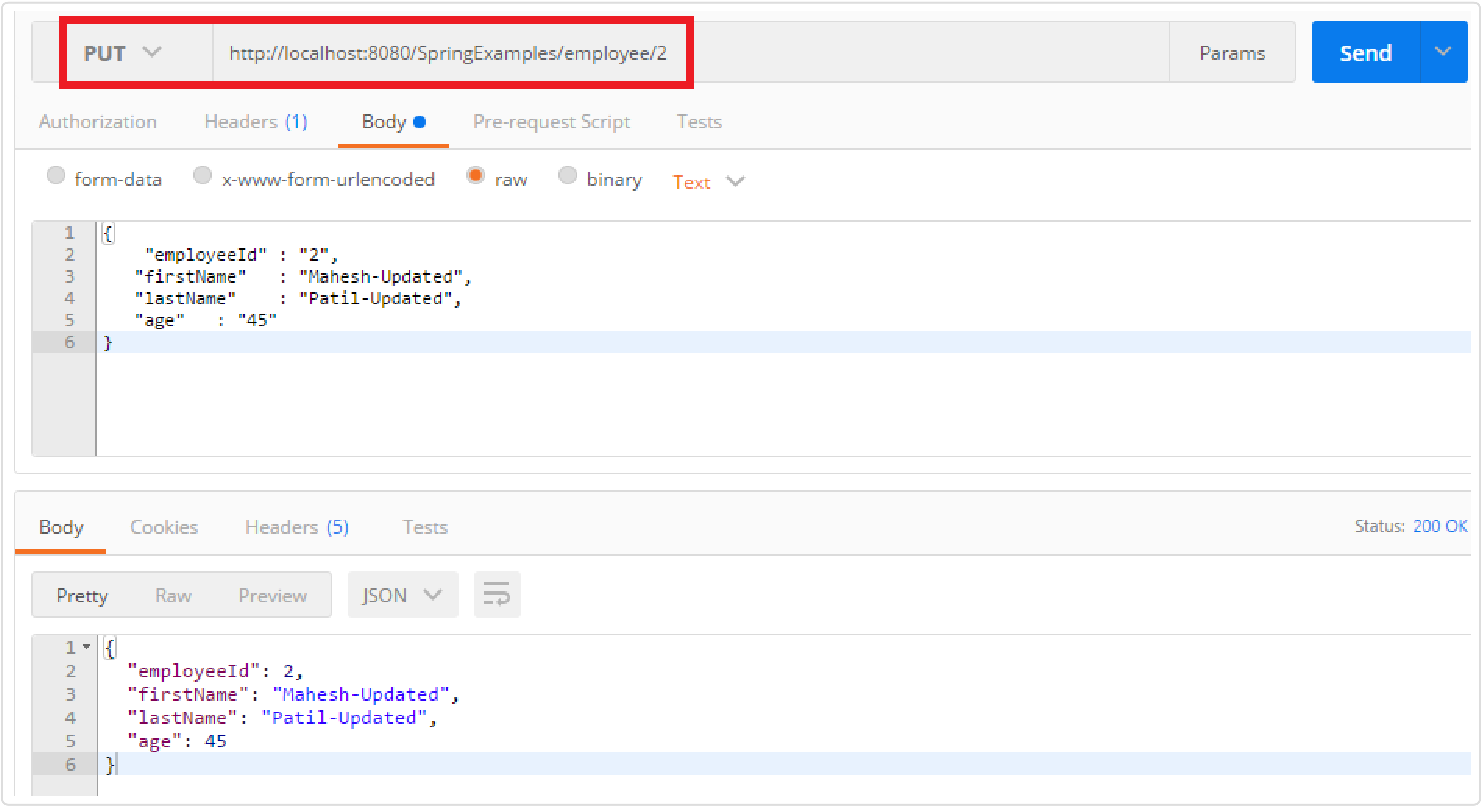
```
{
  "employeeId": 2,
  "firstName": "Mahesh",
  "lastName": "Patil",
  "age": 40
}
```

4] Update Employee

1. Choose **PUT** to be the selected HTTP method.
2. Specify the request URL as **http://localhost:8080/SpringExamples/employee/2**
3. Add request header value as **Content-Type** as **application/json**

4. Specify Employee data in JSON format which you want to update.
5. Click on Send Button.
6. Refer below image and verify the changes in the database.

```
{
    "employeeId" : "2",
   "firstName"   : "Mahesh-Updated",
   "lastName"    : "Patil-Updated",
   "age"    : "45"
}
```



PUT http://localhost:8080/SpringExamples/employee/2

raw Text

```
{
    "employeeId" : "2",
   "firstName"   : "Mahesh-Updated",
   "lastName"    : "Patil-Updated",
   "age"    : "45"
}
```

Body Status: 200 OK

JSON

```
{
  "employeeId": 2,
  "firstName": "Mahesh-Updated",
  "lastName": "Patil-Updated",
  "age": 45
}
```

5] Delete Employee

1. Choose **DELETE** to be the selected HTTP method.
2. Specify the request URL as **http://localhost:8080/SpringExamples/employee/delete/3**
3. Add request header value as **Content-Type** as **application/json**
4. Click on Send Button.
5. It will delete the record from database.