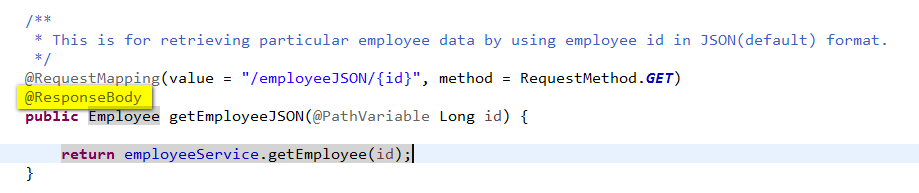
1. Spring frame provides simple but great way for creating Web Application using Java technology. Spring gives developer to ease of development using MVC design pattern.
2. MVC stands for Mode-View-Controller, in this approach we can divide out software development into 3 layers named model which is responsible for data related works, View layer is responsible for presentation logic and controller synchronized both view and model part.
3. Controller plays very important part in MVC. It takes care how to handle request or events. Do Security related stuff. How to send response, it is decided by controller? And so on…
4. There are many ways to create Spring Base Web application project. We choose SpringBoot with gradle build tool particularly.
5. Spring MVC offers two type of Controllers classes:
6. Annotated with @Controller
7. Annotated with @RestController
8. If we simply want to create a website using Spring MVC in which use a view technology (i.e. JSP/HTML pages), then first approach is better because @Controller provide both data as well as view.

But If want to create RESTful web services, in which we only want data independent from view, then Second approach is best because @RestController only deals in term of data. Suppose a client want to a web application for his business, then we should go for RESTful services. The reason behind this is that in future that particular client wants to make a mobile app (android app or iOS app), then we only need data from server not UI(view), because this data will show in native UIs.

1. We can also use @Controller for returning data only by using @ResponseBody annotation.



So we can say mathematically :

***@RestController =@Controller + @ResponseBody on every method***

1. Note That we use often as

*@RequestMapping (value = "<url-pattern>", method = RequestMethod.GET)*

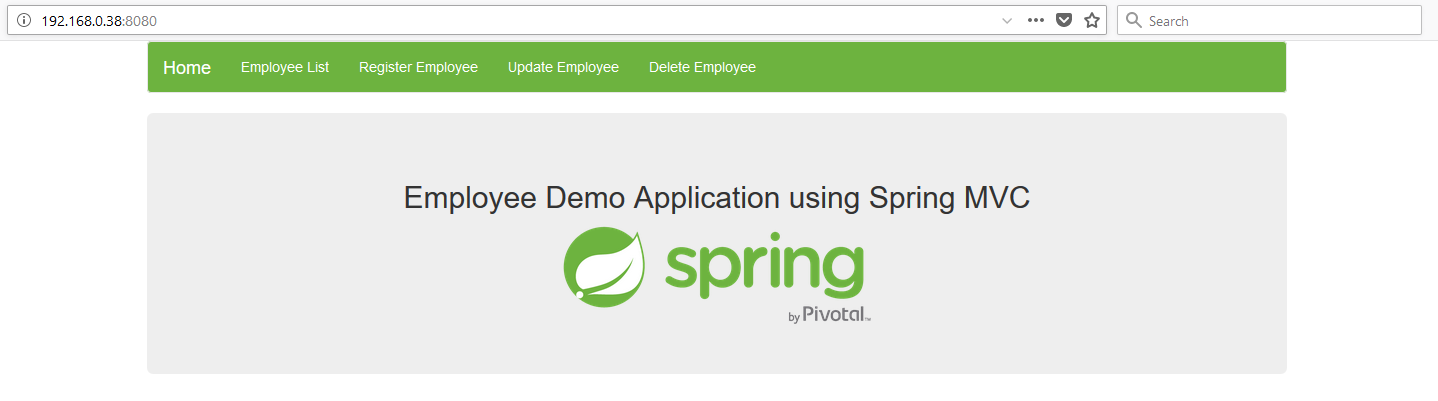
In Spring we do this with newer way as:

*@GetMapping(value="<url-patten>")*

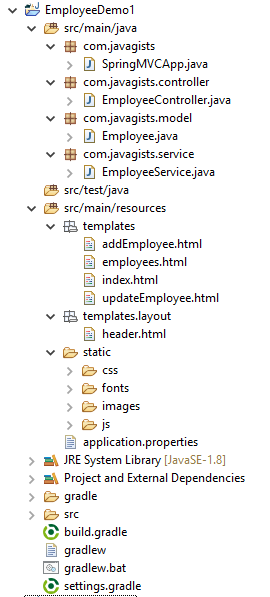
It will work for all method type i.e. POST, PUT, DELETE and so on…

1. We will cover both @Controller and @RestController below:
2. **Making Controller using @Controller:**

Now we are going to create a **SpringMVC** project using SpringBoot with Gradle build tool.



This is our project structure-



Please replace the content of your gradle build to this gradle build file.

Gradle build Script:

**buildscript** {

**ext** {

springBootVersion = '1.4.3.RELEASE'

}

**repositories** {

mavenCentral()

}

**dependencies** {

classpath("org.springframework.boot:spring-boot-gradle-plugin:2.0.1.RELEASE")

}

}

**apply** plugin: 'java'

**apply** plugin: 'eclipse'

**apply** plugin: 'idea'

**apply** plugin: 'org.springframework.boot'

**apply** plugin: 'io.spring.dependency-management'

jar {

baseName = 'boot-gradle'

**version** = '0.0.1-SNAPSHOT'

}

**sourceCompatibility** = 1.8

targetCompatibility = 1.8

**repositories** {

mavenCentral()

}

**dependencies** {

compile('org.springframework.boot:spring-boot-starter-web','org.apache.tomcat.embed:tomcat-embed-jasper'

,'javax.servlet:jstl')

compile('org.springframework.boot:spring-boot-starter-thymeleaf')

testCompile('org.springframework.boot:spring-boot-starter-test')

}

# This the starting place of our application. Its looks very simple and similar to normal java main method, but @SpringBootApplication contains all spring magic behind the scene. No Server configuration, no xml file configuration, SpringBoot will handle all things behind the scene.

SpringMVCApp.java

**package com.javagists;**

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

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

**@SpringBootApplication**

**public class SpringMVCApp {**

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

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

**}**

**}**

# This a model or POJO class which represents the employee and contains all employee related data. Each object of this class will represent an employee of real world.

Employee.java

**package** com.javagists.model;

/\*\*

\* **@author** shadab.khan

\*/

**public** **class** Employee {

**private** Long id;

**private** String name;

**private** Integer age;

**private** Float salary;

**public** Long getId() {

**return** id;

}

**public** **void** setId(Long id) {

**this**.id = id;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** Integer getAge() {

**return** age;

}

**public** **void** setAge(Integer age) {

**this**.age = age;

}

**public** Float getSalary() {

**return** salary;

}

**public** **void** setSalary(Float salary) {

**this**.salary = salary;

}

@Override

**public** String toString() {

**return** "Employee [id=" + id + ", name=" + name + ", age=" + age + ", salary=" + salary + "]";

}

}

# This an our SpringMVC controller class about which we discussed above. Any class annotated with @Controller with in the class path of Spring boot will treat as SpringMVC controller. Notice here all methods deals with view.

EmployeeController.java

**package com.javagists.controller;**

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

**import org.springframework.stereotype.Controller;**

**import org.springframework.ui.Model;**

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

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

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

**import com.javagists.model.Employee;**

**import com.javagists.service.EmployeeService;**

**@Controller**

**public class EmployeeController {**

**@Autowired**

**private EmployeeService employeeService;**

**/\*\***

**\* Shows addEmployee.html page with new employee having no data.**

**\***

**\* @return name of addEmployee page.**

**\*/**

**@RequestMapping(value = "/addEmployeePage")**

**public String getAddEmployeePage(Model model) {**

**model.addAttribute("employee", new Employee());**

**return "addEmployee";**

**}**

**/\*\***

**\* Save new employee's data and redirect to employees.html**

**\*/**

**@RequestMapping(value = "/employee", method = RequestMethod.POST)**

**public String addEmployee(Employee employee) {**

**employeeService.addEmployee(employee);**

**return "redirect:employees";**

**}**

**/\*\***

**\* Shows updateEmployee.html page with existing employee data retrieved by data**

**\* layer using employee id.**

**\***

**\* @return name of updateEmployee page.**

**\*/**

**@RequestMapping(value = "/updateEmployeePage/{id}")**

**public String getUpdateEmplyeePage(@PathVariable Long id, Model model) {**

**Employee emp = employeeService.getEmployee(id);**

**model.addAttribute("employee", emp);**

**return "updateEmployee";**

**}**

**/\*\***

**\* Update existing employee's data and redirect to employees.html**

**\*/**

**@RequestMapping(value = "/employee", method = RequestMethod.PUT)**

**public String updateEmployee(Employee employee) {**

**employeeService.updateEmployee(employee);**

**return "redirect:employees";**

**}**

**/\*\***

**\* Delete existing employee's data and redirect to employees.html**

**\*/**

**@RequestMapping(value = "/employee/{id}", method = RequestMethod.DELETE)**

**public String deleteEmployee(int id) {**

**employeeService.deleteEmployee(id);**

**return "redirect:employees";**

**}**

**/\*\***

**\* Find all employees data from data layer and will rendered on employees.html**

**\* page.**

**\***

**\* @return employees.html page name.**

**\*/**

**@RequestMapping(value = "/employees", method = RequestMethod.GET)**

**public String getAllEmployee(Model model) {**

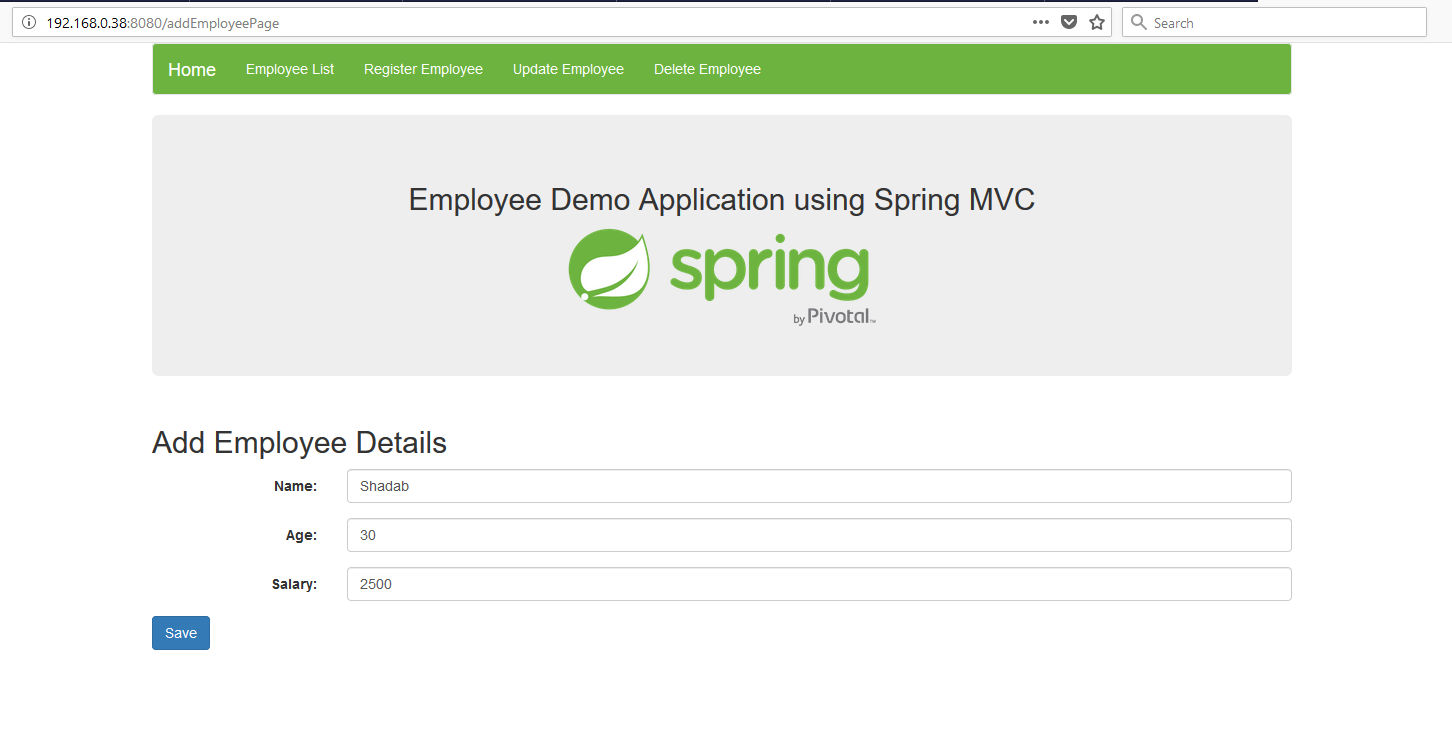
**model.addAttribute("employees", employeeService.getAllEmployee());**

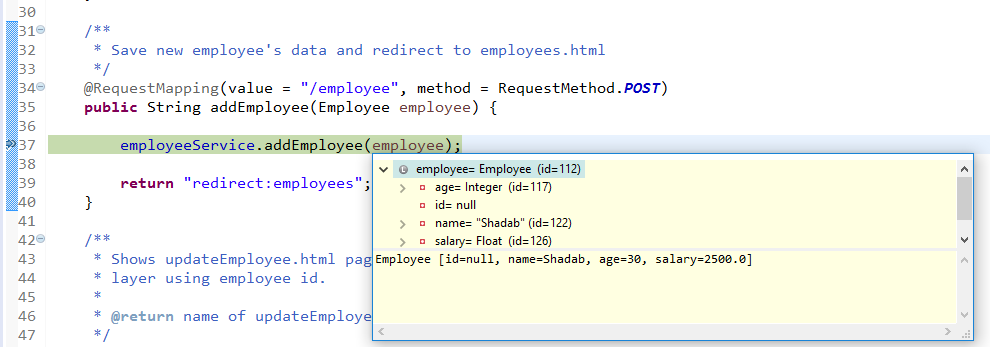
**return "employees";**

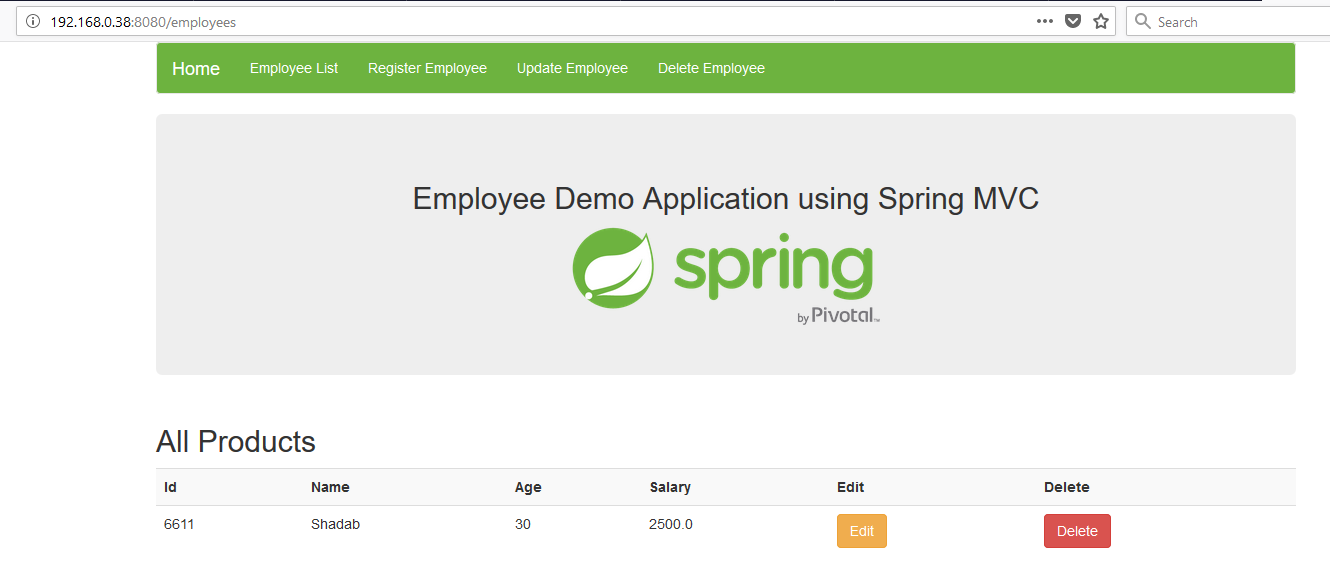
**}**

**}**

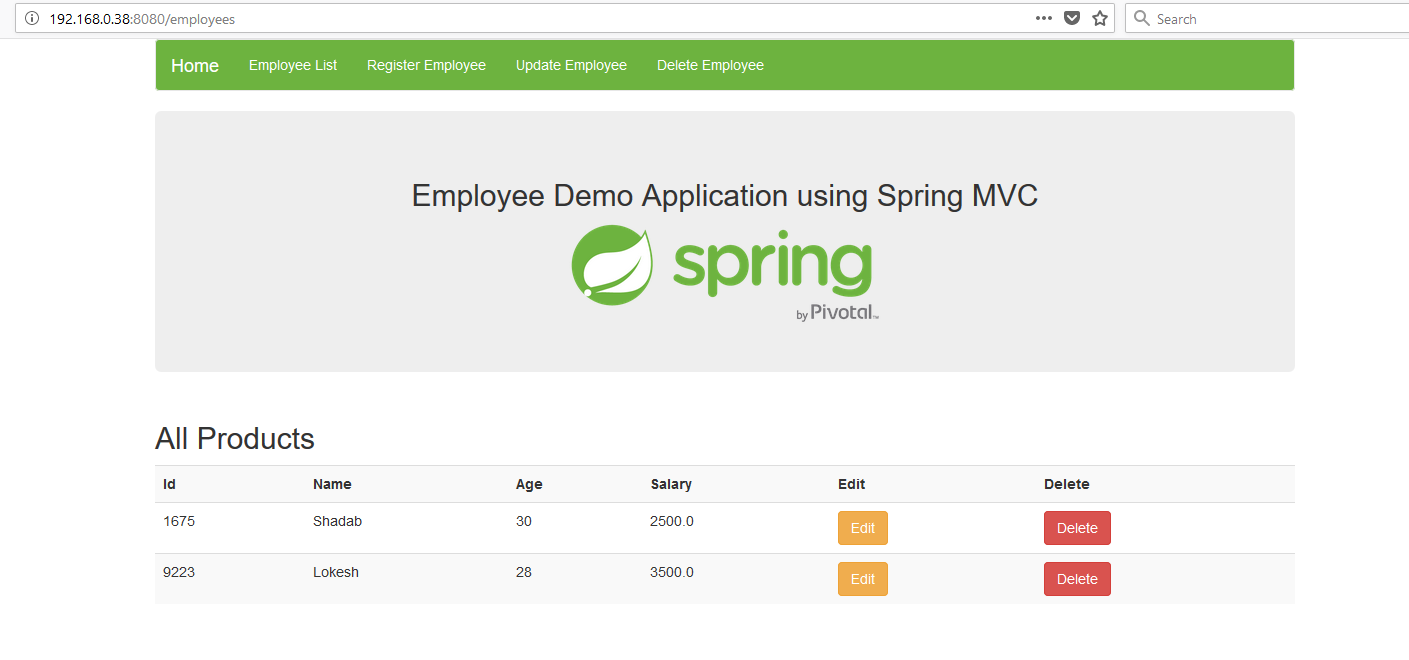
**Adding new Employee:**

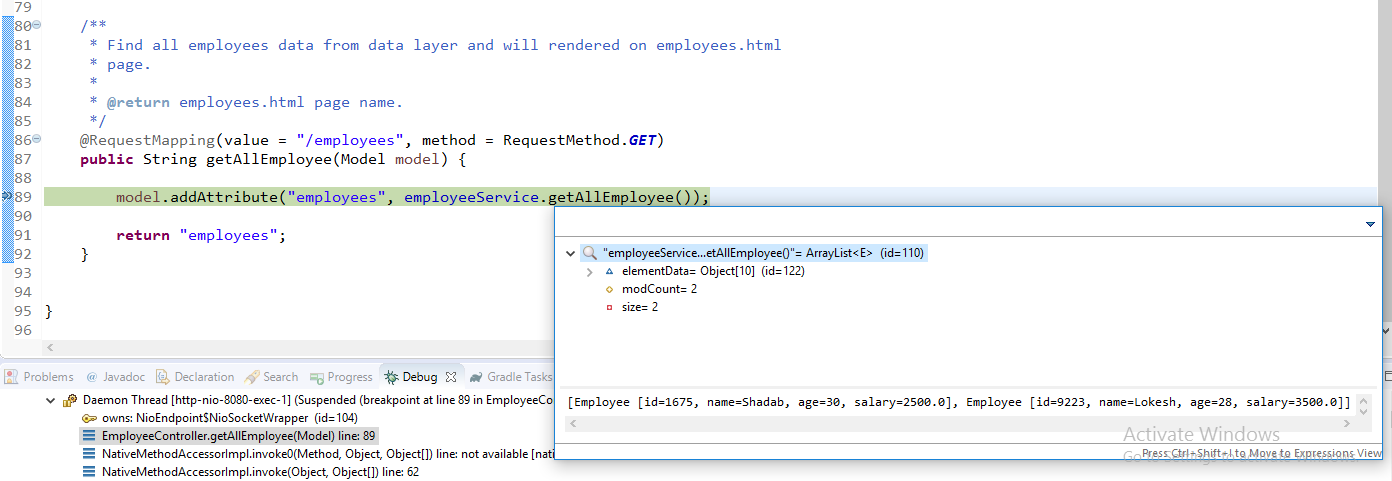




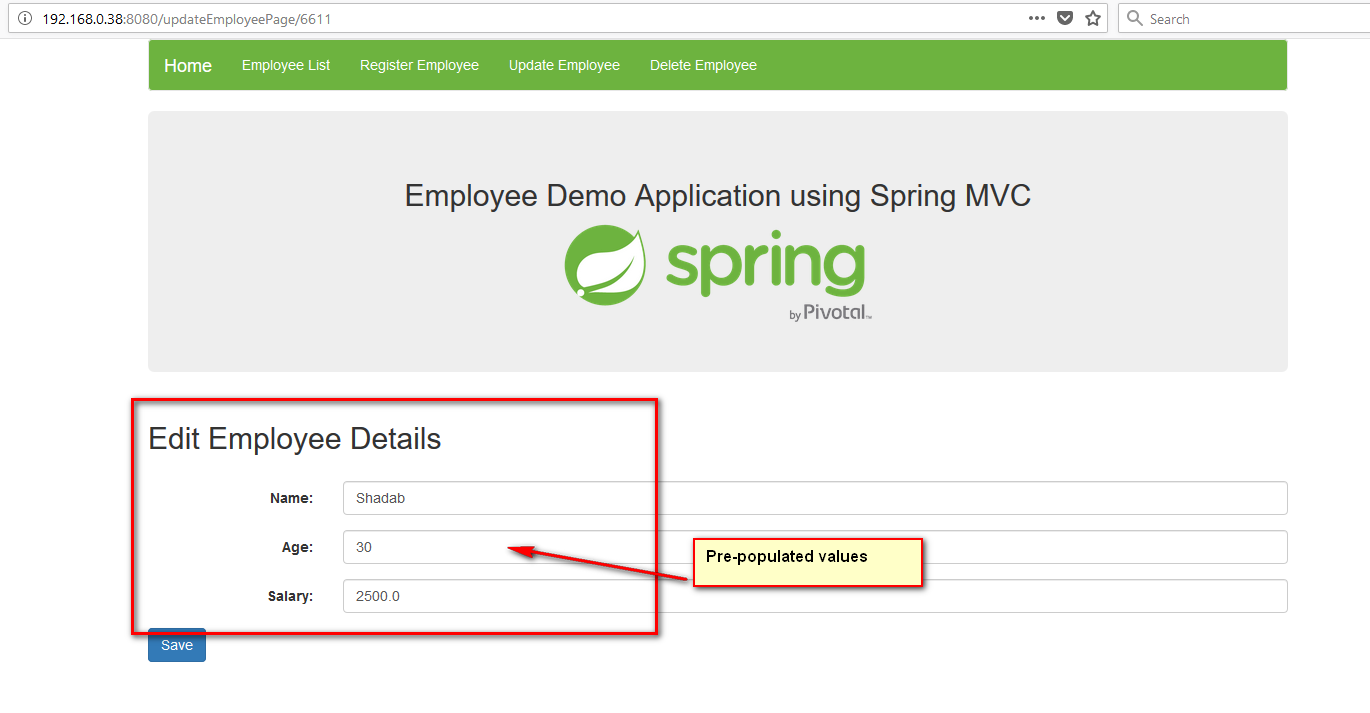


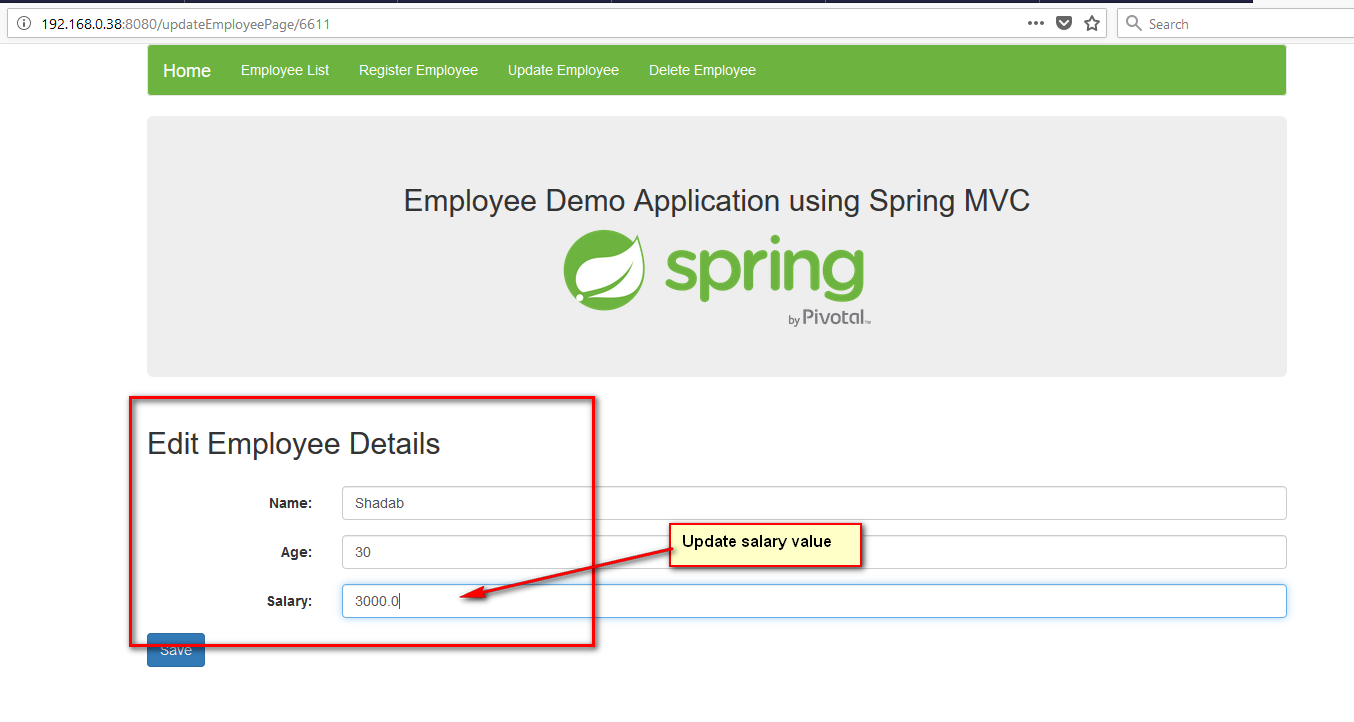
**Showing all Employees:**

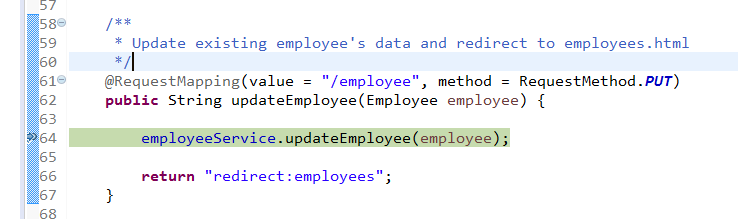




**Updating Employee:**



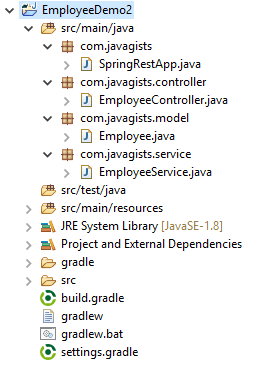




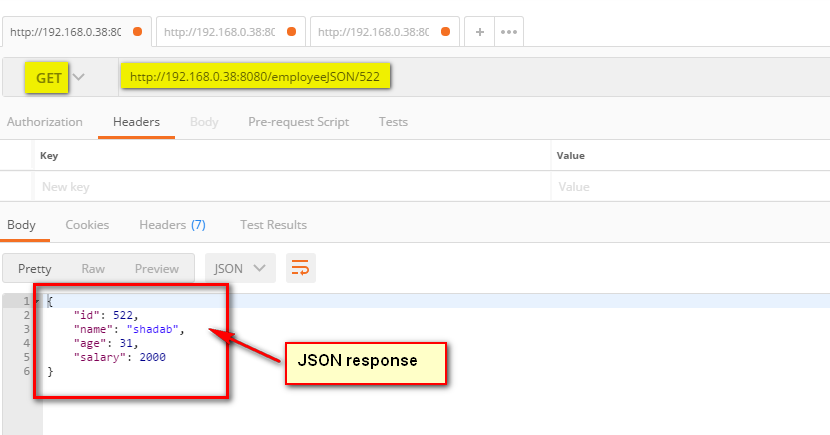
1. **Making Controller using @RestController:**

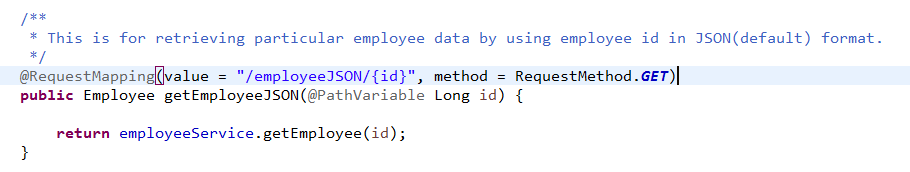
Now we are going to create a **Rest based SpringMVC** project using SpringBoot with Gradle build tool.

This is our project structure-

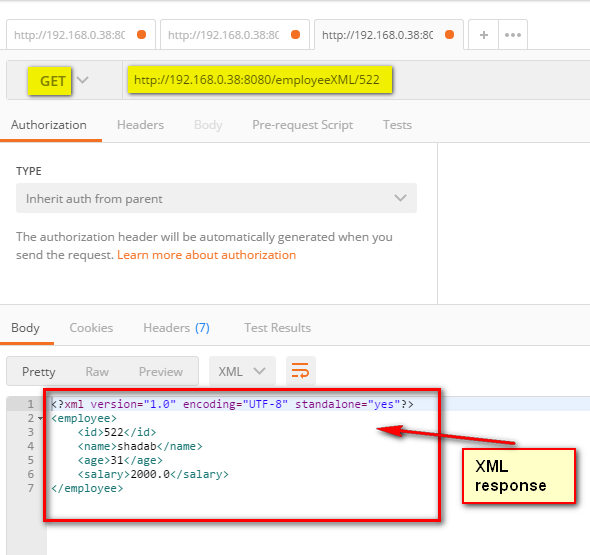


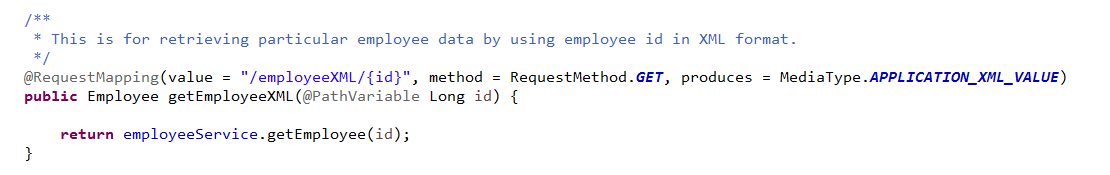
**Getting JSON data:**



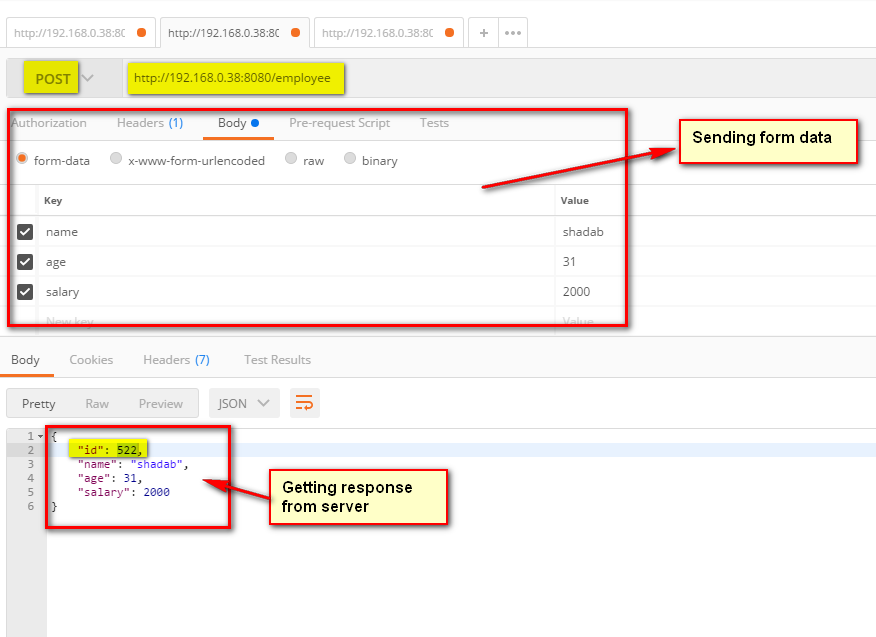


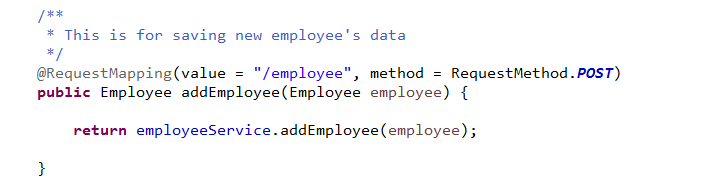
**Getting XML data:**





**Sending data:**





*SpringRestApp.java*

package com.javagists; package com.javagists;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringRestApp {

public static void main(String[] args) {

SpringApplication.run(SpringRestApp.class, args);

}

}

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringRestApp {

public static void main(String[] args) {

SpringApplication.run(SpringRestApp.class, args);

}

}

*EmployeeController.java*

**package** com.javagists.controller;

**import** java.util.List;

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

**import** org.springframework.http.MediaType;

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

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

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

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

**import** com.javagists.model.Employee;

**import** com.javagists.service.EmployeeService;

@RestController

**public** **class** EmployeeController {

@Autowired

**private** EmployeeService employeeService;

/\*\*

\* This is for saving new employee's data

\*/

@RequestMapping(value = "/employee", method = RequestMethod.***POST***)

**public** Employee addEmployee(Employee employee) {

**return** employeeService.addEmployee(employee);

}

/\*\*

\* This is for updating existing employee's data

\*/

@RequestMapping(value = "/employee", method = RequestMethod.***PUT***)

**public** **void** updateEmployee(Employee employee) {

employeeService.updateEmployee(employee);

}

/\*\*

\* This is for deleting existing employee's data

\*/

@RequestMapping(value = "/employee/{id}", method = RequestMethod.***DELETE***)

**public** **boolean** deleteEmployee(@PathVariable Long id) {

**return** employeeService.deleteEmployee(id);

}

/\*\*

\* This is for retrieving particular employee data by using employee id in JSON(default) format.

\*/

@RequestMapping(value = "/employeeJSON/{id}", method = RequestMethod.***GET***)

**public** Employee getEmployeeJSON(@PathVariable Long id) {

**return** employeeService.getEmployee(id);

}

/\*\*

\* This is for retrieving particular employee data by using employee id in XML format.

\*/

@RequestMapping(value = "/employeeXML/{id}", method = RequestMethod.***GET***, produces = MediaType.***APPLICATION\_XML\_VALUE***)

**public** Employee getEmployeeXML(@PathVariable Long id) {

**return** employeeService.getEmployee(id);

}

/\*\*

\* This is for retrieving particular employee data by using employee id.

\*/

@RequestMapping(value = "/employee", method = RequestMethod.***GET***, produces = MediaType.***APPLICATION\_XML\_VALUE***)

**public** List<Employee> getAllEmployees() {

**return** employeeService.getAllEmployee();

}

}

*Employee.java*

package com.javagists.model;

import java.io.Serializable;

import javax.xml.bind.annotation.XmlAccessType;

import javax.xml.bind.annotation.XmlAccessorType;

import javax.xml.bind.annotation.XmlRootElement;

/\*\*

\* @author shadab.khan

\*/

@XmlRootElement

@XmlAccessorType(XmlAccessType.FIELD)

public class Employee implements Serializable {

/\*\*

\*

\*/

private static final long serialVersionUID = -3866757643429936097L;

private Long id;

private String name;

private Integer age;

private Float salary;

public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public Integer getAge() {

return age;

}

public void setAge(Integer age) {

this.age = age;

}

public Float getSalary() {

return salary;

}

public void setSalary(Float salary) {

this.salary = salary;

}

@Override

public String toString() {

return "Employee [id=" + id + ", name=" + name + ", age=" + age + ", salary=" + salary + "]";

}

}