CRUD Restful Web Service with Spring Boot Example

1- Objective of Example

This document is based on:

- **Spring Boot 2.x**
- **Eclipse 4.7 Oxygen**

See more:

- *What are RESTful Web Services?*

In this post, I will show you how to create a **Restful Web Service** application using **Spring Boot** and having the 4 functions of**Create**, **Read**, **Update**, **Delete** (CRUD).

Read (GET method)

We will build an **URI** that is assigned to return the user an employee list and defines another **URI** that returns the user the information of a particular employee. The data that the user will be received is in **XML** or **JSON** format. These URIs only accept the requests with **GET** method.

- GET http://localhost:8080/employees
- GET http://localhost:8080/employee/E01

Update (PUT method).

Build an **URI** to process the request for changing an employee's information. This **URI** accepts only the requests with **PUT**method. The data attached with the request is the new information of the employee, which is in **XML** or **JSON** format.

- PUT http://localhost:8080/employee

Create (POST method)

Build a **URI** to handle the request for creating an employee. This **URI** accepts only the requests with the **POST** method. The data attached to the request is the information of the employee to be created. It is in **XML** format or **JSON** format.

- POST http://localhost:8080/employee

Delete (DELETE method).

Build an **URI** to handle the request for deleting an employee. This **URI** accepts only the requests with the **DELETE** method.

*Note: No data is enclosed with the request in this case (Like the data attached with **POST** method), because the request with**DELETE** method can not be attached with data. The employee's information to be deleted will be located on the **URI** or the **QueryString** of the **URL**.*

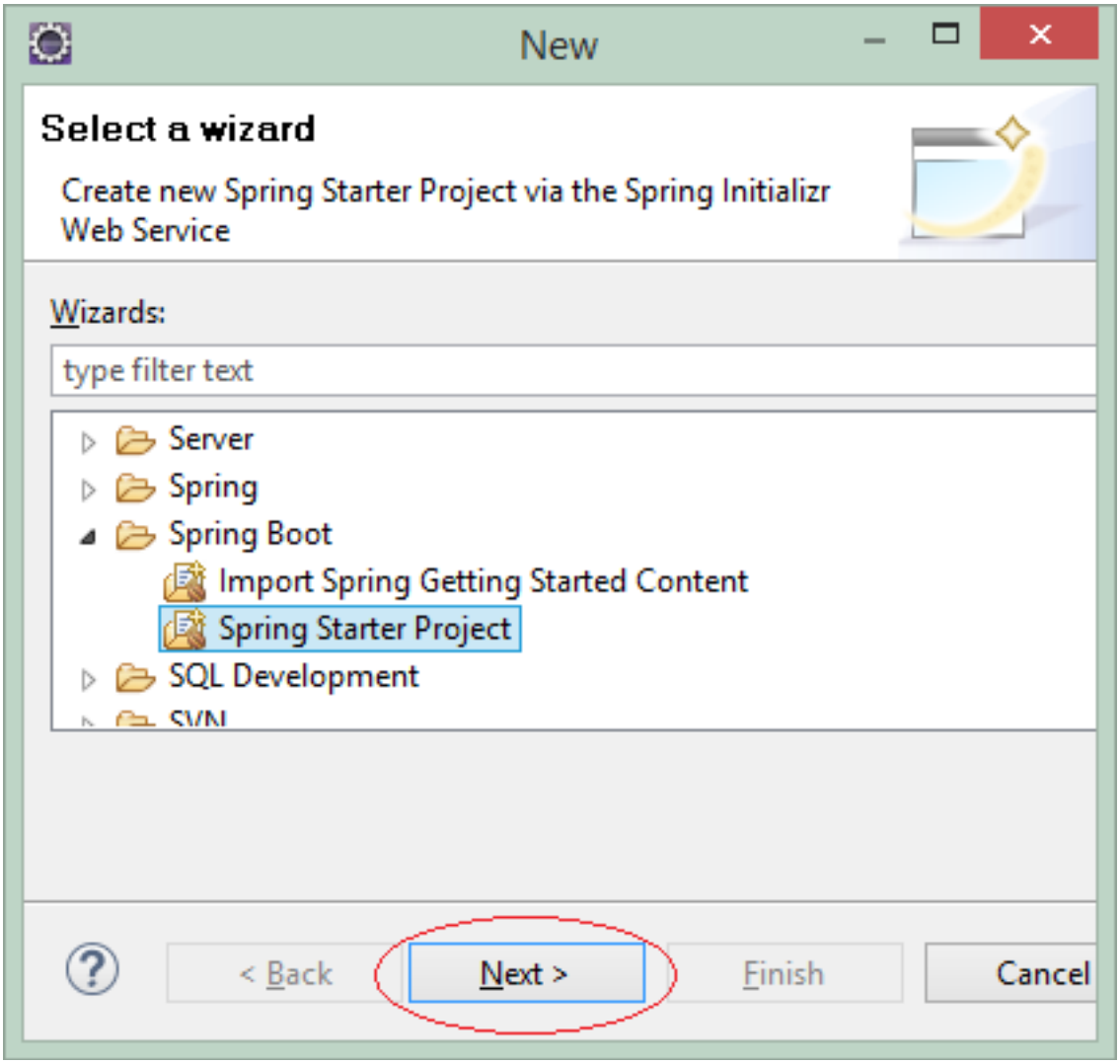
- DELETE http://localhost:8080/employee/{empNo}

2- Create Spring Boot project

- *Install Spring Tool Suite into Eclipse*

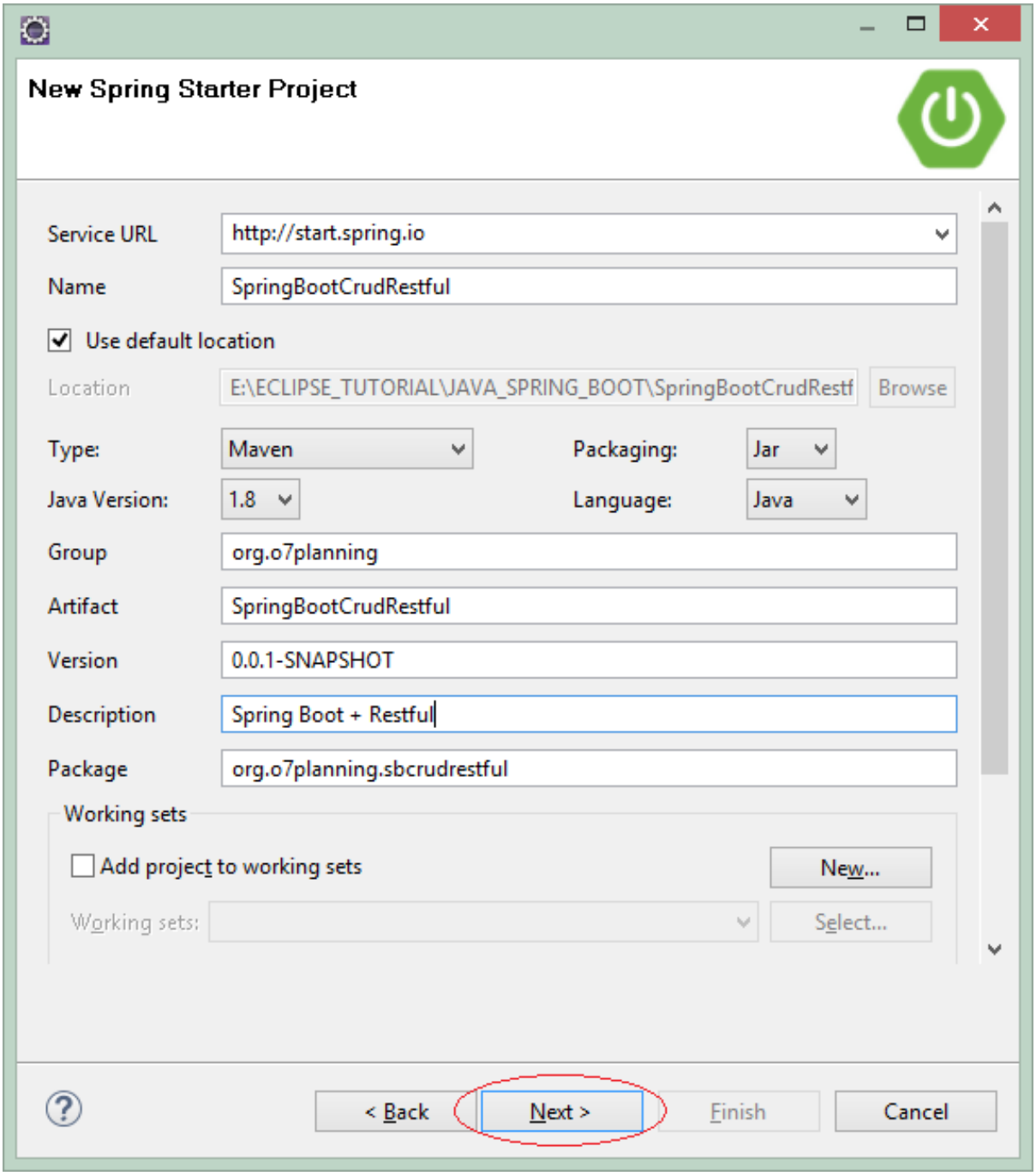
On the **Eclipse,** select:

- **File/New/Other..**

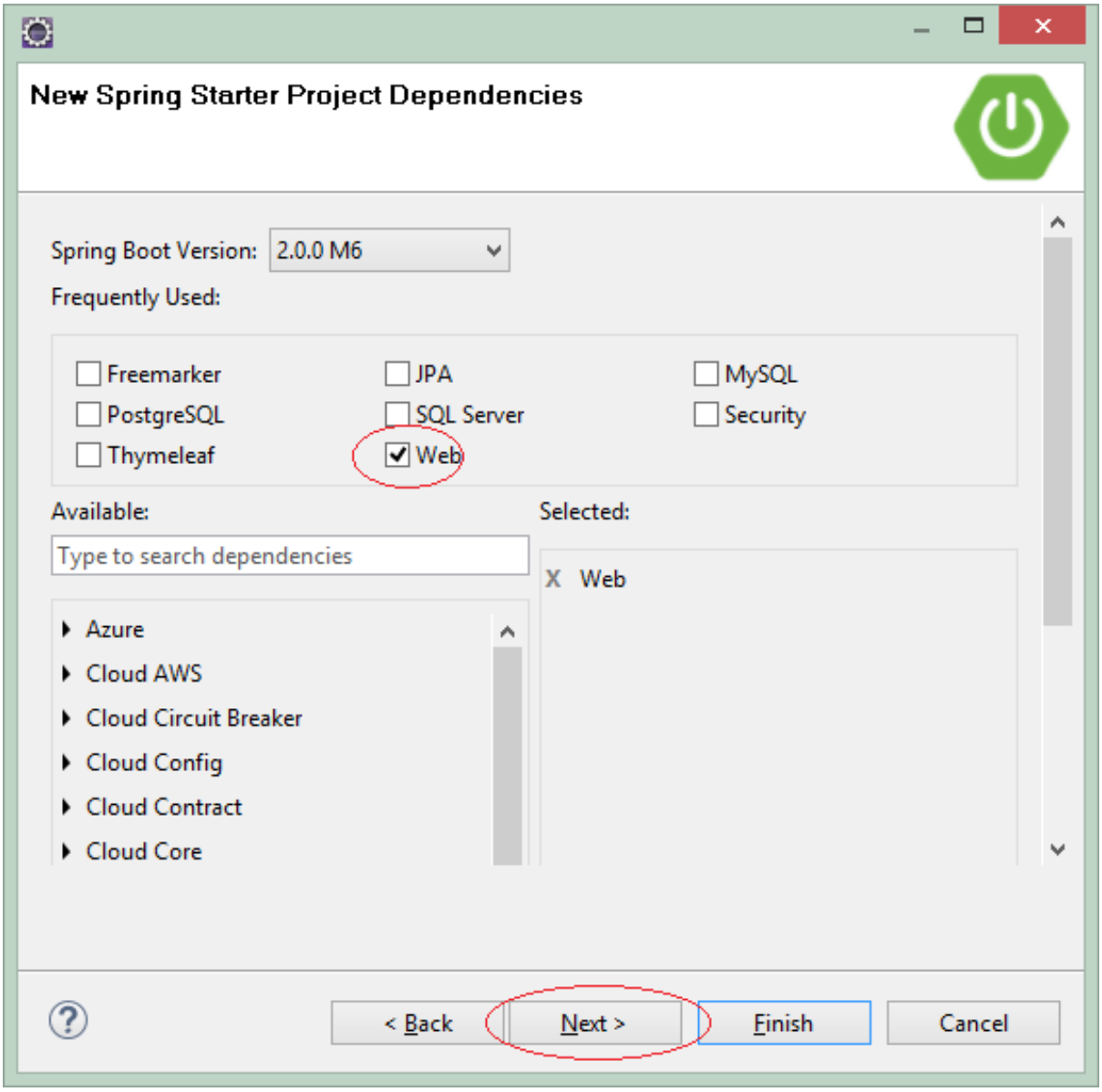


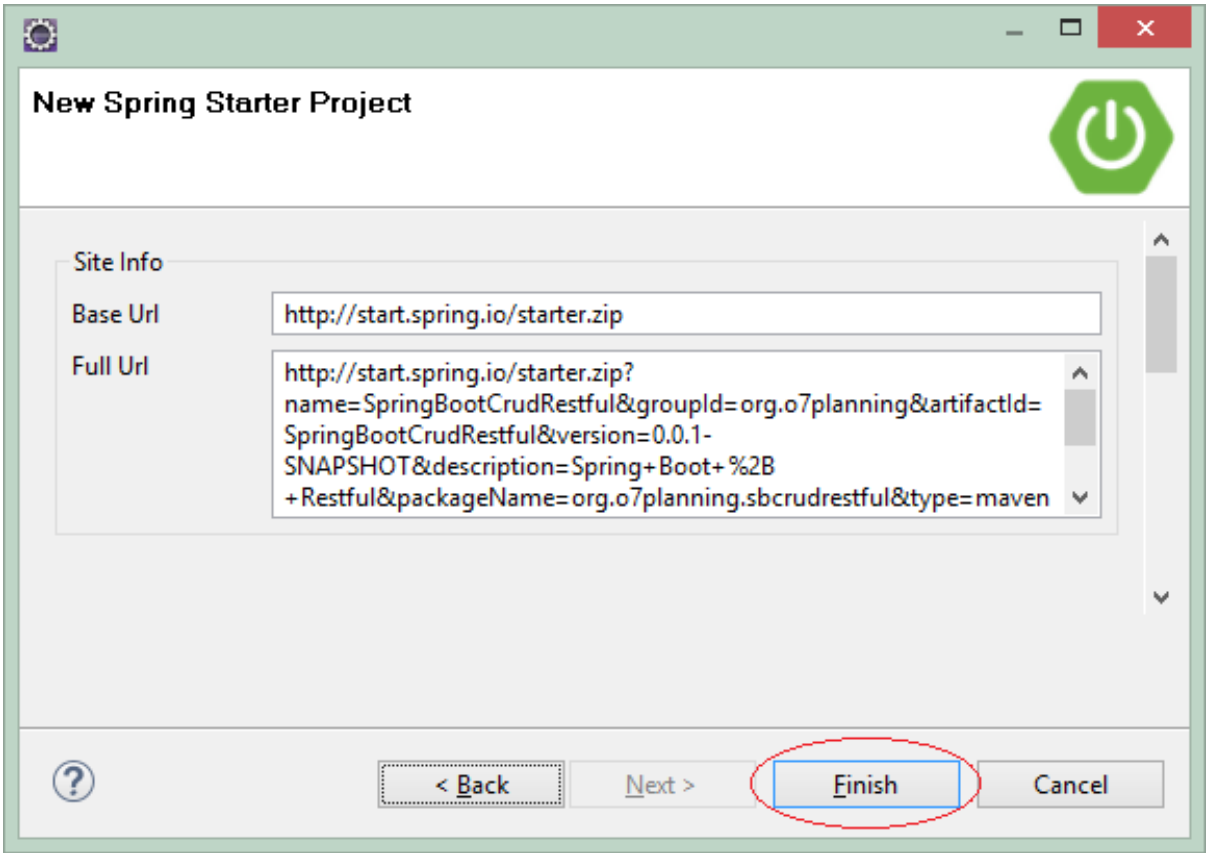
Enter:

- **Name**: SpringBootCrudRestful
- **Group**: org.o7planning
- **Package**: org.o7planning.sbcrudrestful

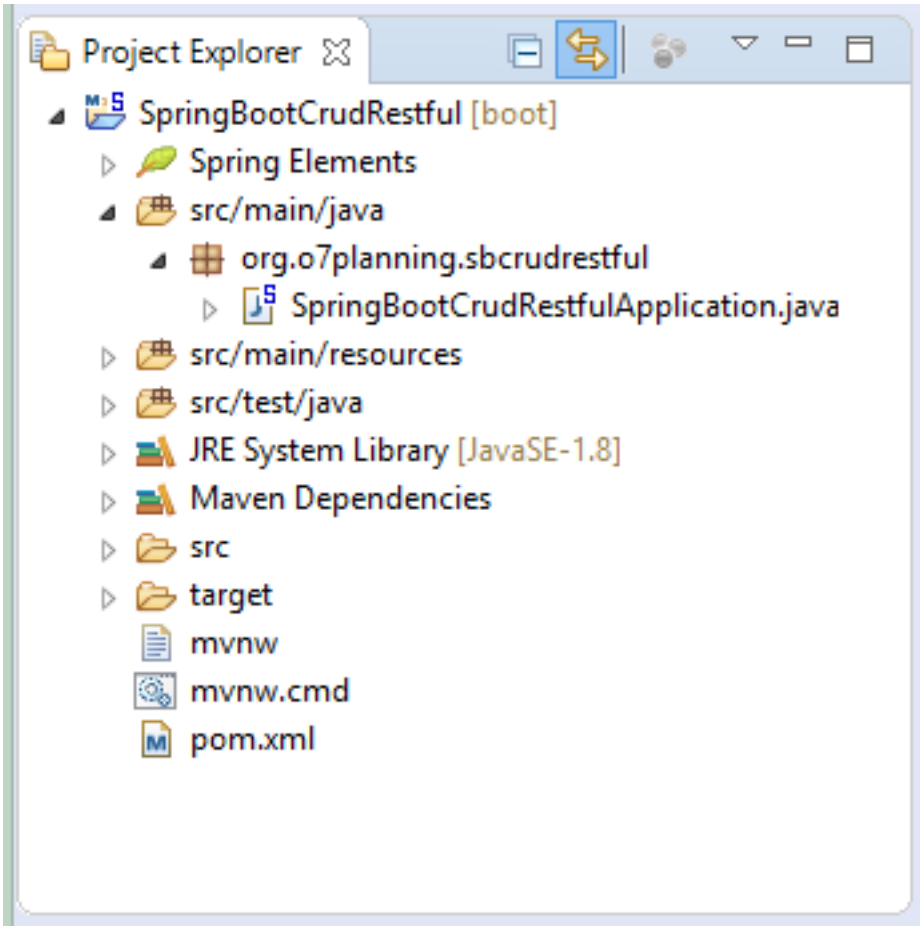


In the next step, you need to select the technologies to be used.


New Spring Starter Project Dependencies
Spring Boot Version:
2.0.0 M6
Frequently Used:
Freemarker
JPA
MySQL
PostgreSQL
SQL Server
Security
Thymeleaf
Web
Available:
Selected:
Type to search dependencies
X Web
Azure
Cloud AWS
Cloud Circuit Breaker
Cloud Config
Cloud Contract
Cloud Core
< Back
Next >
Finish
Cancel



OK, the Project has been created.



3- Configure pom.xml

In this example, we need to a library to convert the **XML** object into a **Java** one and vice versa and another library to convert **JSON** into **Java** and vice versa.

JSON <==> Java

The **spring-boot-starter-web** has built in **jackson-databind**, which helps to convert **JSON** into **Java object** and vice versa.

```
public class Employee {
      private String empNo;
      private String empName;
      private String position;
}
```

```
Employee emp1 =
    new Employee("E01",
                 "Smith", "Clerk");
```

JSON ⇨ jackson-databind

```
{"empNo":"E01","empName":"Smith","position":"Clerk"}
```

```
List<Employee> list;
```

JSON ⇨ jackson-databind

```
[{"empNo":"E02","empName":"Allen","position":"Salesman"},
 {"empNo":"E01","empName":"Smith","position":"Clerk"},
 {"empNo":"E03","empName":"Jones","position":"Manager"}]
```

XML <==> Java

The **Spring Boot** uses **JAXB** (available in **JDK**) as a default library to convert **XML** and **Java**. However, Java classes need to be annotated by **@XmlRootElement**,... Therefore, my advice is that you should use the **jackson-dataformat-xml** as a library to convert **XML** and **Java**. To use the **jackson-dataformat-xml**, you need to declare it in the **pom.xml** file:

** pom.xml **

?

```
...

<dependencies>

        ...

    <dependency>
        <groupId>com.fasterxml.jackson.dataformat</groupId>
        <artifactId>jackson-dataformat-xml</artifactId>
    </dependency>

    ...

</dependencies>

...
```

Jackson

```
// No need annotation
public class Employee {
      private String empNo;
      private String empName;
      private String position;
}
```

XML

```
Employee emp1 =
    new Employee("E01",
                 "Smith", "Clerk");
```

```
<Employee>
  <empNo>E01</empNo>
  <empName>Smith</empName>
  <position>Clerk</position>
</Employee>
```

XML

```
List<Employee> list;
```

```
▼<List>
  ▼<item>
     <empNo>E02</empNo>
     <empName>Allen</empName>
     <position>Salesman</position>
   </item>
  ▼<item>
     <empNo>E01</empNo>
     <empName>Smith</empName>
     <position>Clerk</position>
   </item>
  ▼<item>
     <empNo>E03</empNo>
     <empName>Jones</empName>
     <position>Manager</position>
   </item>
</List>
```

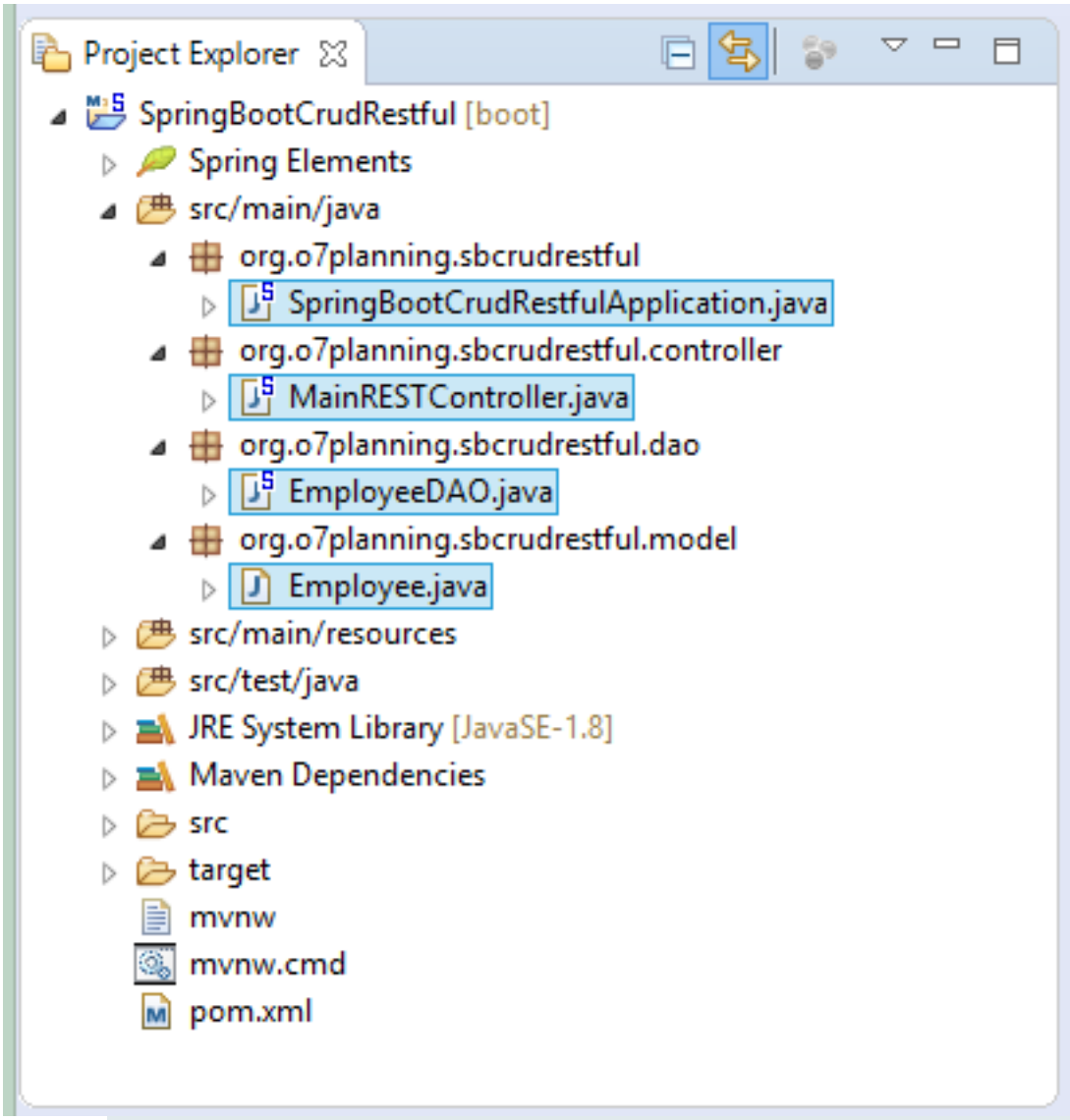
pom.xml

?

```
<?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>org.o7planning</groupId>
    <artifactId>SpringBootCrudRestful</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <packaging>jar</packaging>

    <name>SpringBootCrudRestful</name>
    <description>Spring Boot + Restful</description>

    <parent>
        <groupId>org.springframework.boot</groupId>
```

```
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.0.0.RELEASE</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>

    <properties>
        <project.build.sourceEncoding>UTF-
8</project.build.sourceEncoding>
        <project.reporting.outputEncoding>UTF-
8</project.reporting.outputEncoding>
        <java.version>1.8</java.version>
    </properties>

    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>com.fasterxml.jackson.dataformat</groupId>
            <artifactId>jackson-dataformat-xml</artifactId>
        </dependency>

        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-
plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

4- Code of application



SpringBootCrudRestfulApplication.java

?

```
package org.o7planning.sbcrudrestful;

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

@SpringBootApplication
public class SpringBootCrudRestfulApplication {

    public static void main(String[] args) {
        SpringApplication.run(SpringBootCrudRestfulApplication.class,
args);
    }
}
```

Employee class represents for an employee.

Employee.java

?

```
package org.o7planning.sbcrudrestful.model;
```

```
public class Employee {

    private String empNo;
    private String empName;
    private String position;

    public Employee() {

    }

    public Employee(String empNo, String empName,
String position) {
        this.empNo = empNo;
        this.empName = empName;
        this.position = position;
    }

    public String getEmpNo() {
        return empNo;
    }

    public void setEmpNo(String empNo) {
        this.empNo = empNo;
    }

    public String getEmpName() {
        return empName;
    }

    public void setEmpName(String empName) {
        this.empName = empName;
    }

    public String getPosition() {
        return position;
    }

    public void setPosition(String position) {
        this.position = position;
    }

}
```

The **EmployeeDAO** class is annotated by **@Repository** to notify to the **Spring** that it is a **Spring BEAN**. This class includes the methods helping query a list of employees, create employees, update employee's information, and delete employees.

EmployeeDAO.java

?

```
package org.o7planning.sbcrudrestful.dao;


import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.o7planning.sbcrudrestful.model.Employee;
import org.springframework.stereotype.Repository;

@Repository
public class EmployeeDAO {

    private static final Map<String, Employee> empMap =
new HashMap<String, Employee>();

    static {
        initEmps();
    }

    private static void initEmps() {
        Employee emp1 = new Employee("E01", "Smith",
"Clerk");
        Employee emp2 = new Employee("E02", "Allen",
"Salesman");
        Employee emp3 = new Employee("E03", "Jones",
"Manager");

        empMap.put(emp1.getEmpNo(), emp1);
        empMap.put(emp2.getEmpNo(), emp2);
        empMap.put(emp3.getEmpNo(), emp3);
    }

    public Employee getEmployee(String empNo) {
        return empMap.get(empNo);
    }

    public Employee addEmployee(Employee emp) {
        empMap.put(emp.getEmpNo(), emp);
        return emp;
    }
```

```
    public Employee updateEmployee(Employee emp) {
        empMap.put(emp.getEmpNo(), emp);
        return emp;
    }

    public void deleteEmployee(String empNo) {
        empMap.remove(empNo);
    }

    public List<Employee> getAllEmployees() {
        Collection<Employee> c = empMap.values();
        List<Employee> list = new
ArrayList<Employee>();
        list.addAll(c);
        return list;
    }

}
```

The **MainRESTController** class is annotated by **@RestController** to inform to the **Spring** that it is a **Spring Restful Controller**,

MainRESTController.java

?

```
package org.o7planning.sbcrudrestful.controller;

import java.util.List;

import org.o7planning.sbcrudrestful.dao.EmployeeDAO;
import org.o7planning.sbcrudrestful.model.Employee;
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.RequestBody;
import
org.springframework.web.bind.annotation.RequestMapping;
import
org.springframework.web.bind.annotation.RequestMethod;
import
org.springframework.web.bind.annotation.ResponseBody;
import
org.springframework.web.bind.annotation.RestController;

@RestController

public class MainRESTController {

    @Autowired
    private EmployeeDAO employeeDAO;

    @RequestMapping("/")
    @ResponseBody
    public String welcome() {
        return "Welcome to RestTemplate Example.";
    }

    // URL:
    // http://localhost:8080/SomeContextPath/employees
    // http://localhost:8080/SomeContextPath/employees.xml
    // http://localhost:8080/SomeContextPath/employees.json
    @RequestMapping(value = "/employees", //
            method = RequestMethod.GET, //
            produces = { MediaType.APPLICATION_JSON_VALUE,
//
                    MediaType.APPLICATION_XML_VALUE })
    @ResponseBody
    public List<Employee> getEmployees() {
```

```
        List<Employee> list =
employeeDAO.getAllEmployees();
        return list;
    }

    // URL:
    //
http://localhost:8080/SomeContextPath/employee/{empNo}
    //
http://localhost:8080/SomeContextPath/employee/{empNo}.xml
    //
http://localhost:8080/SomeContextPath/employee/{empNo}.json
    @RequestMapping(value = "/employee/{empNo}", //
            method = RequestMethod.GET, //
            produces = { MediaType.APPLICATION_JSON_VALUE,
//
                    MediaType.APPLICATION_XML_VALUE })
    @ResponseBody
    public Employee getEmployee(@PathVariable("empNo")
String empNo) {
        return employeeDAO.getEmployee(empNo);
    }

    // URL:
    // http://localhost:8080/SomeContextPath/employee
    // http://localhost:8080/SomeContextPath/employee.xml
    // http://localhost:8080/SomeContextPath/employee.json

    @RequestMapping(value = "/employee", //
            method = RequestMethod.POST, //
            produces = { MediaType.APPLICATION_JSON_VALUE,
//
                    MediaType.APPLICATION_XML_VALUE })
    @ResponseBody
    public Employee addEmployee(@RequestBody Employee emp) {

        System.out.println("(Service Side) Creating
employee: " + emp.getEmpNo());

        return employeeDAO.addEmployee(emp);
    }

    // URL:
    // http://localhost:8080/SomeContextPath/employee
    // http://localhost:8080/SomeContextPath/employee.xml
    // http://localhost:8080/SomeContextPath/employee.json
    @RequestMapping(value = "/employee", //
```

```
            method = RequestMethod.PUT, //
            produces = { MediaType.APPLICATION_JSON_VALUE,
//
                    MediaType.APPLICATION_XML_VALUE })
    @ResponseBody
    public Employee updateEmployee(@RequestBody Employee
emp) {

        System.out.println("(Service Side) Editing
employee: " + emp.getEmpNo());

        return employeeDAO.updateEmployee(emp);
    }

    // URL:
    //
http://localhost:8080/SomeContextPath/employee/{empNo}
    @RequestMapping(value = "/employee/{empNo}", //
            method = RequestMethod.DELETE, //
            produces = { MediaType.APPLICATION_JSON_VALUE,
MediaType.APPLICATION_XML_VALUE })
    @ResponseBody
    public void deleteEmployee(@PathVariable("empNo") String
empNo) {

        System.out.println("(Service Side) Deleting
employee: " + empNo);

        employeeDAO.deleteEmployee(empNo);
    }

}
```

Explanation:

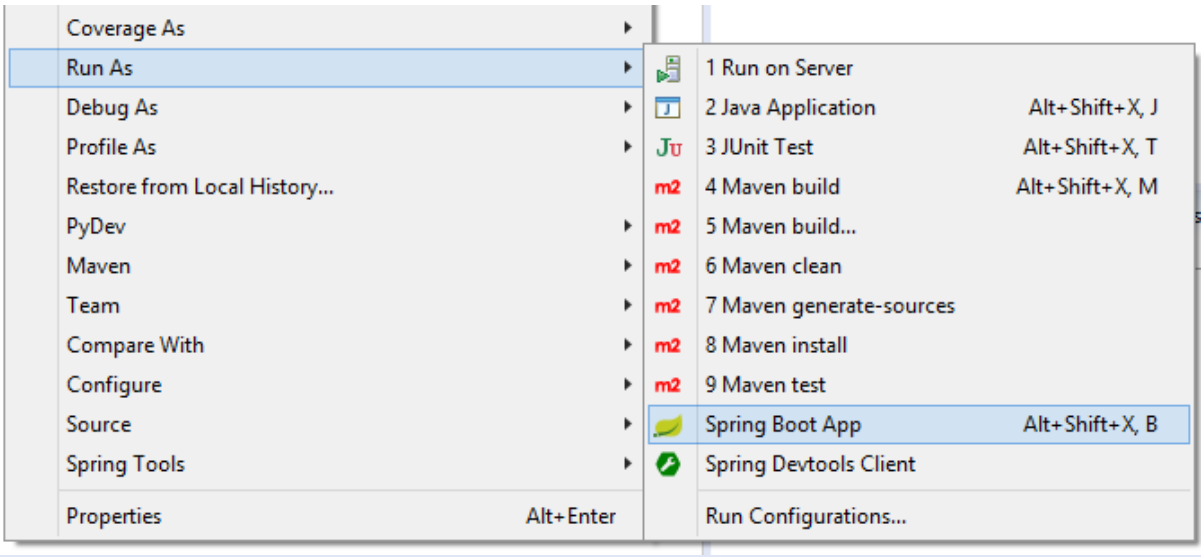
- **produces = { MediaType.APPLICATION_JSON_VALUE, MediaType.APPLICATION_XML_VALUE }**
- **produces = { "application/json" , "application/xml" }**

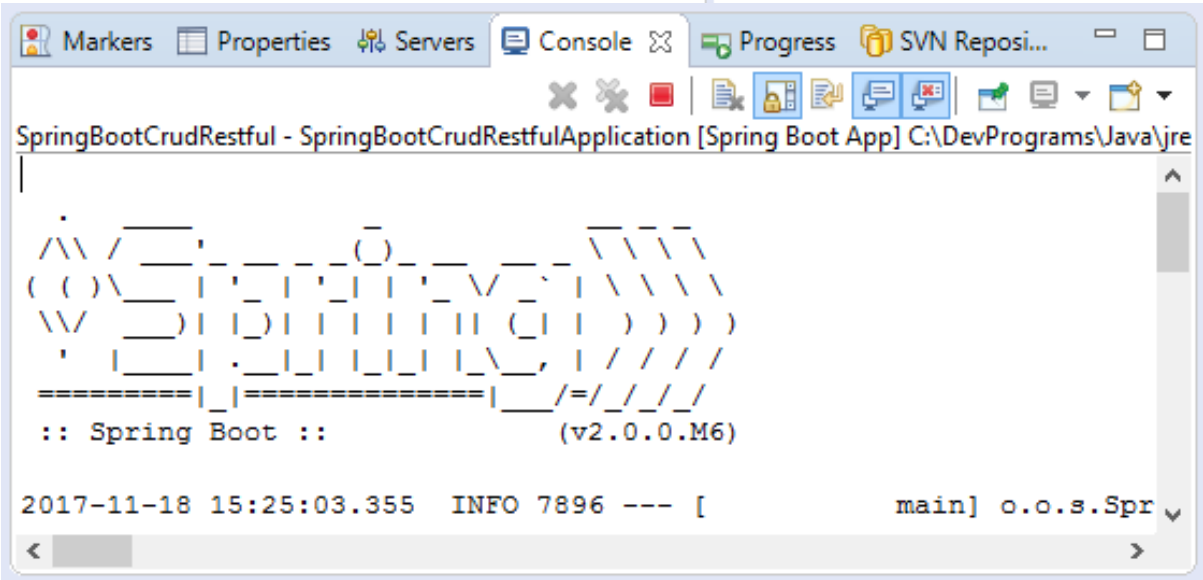
produces attribute is used to specify an **URL** that will only create (return to the user) the data in which format, for example,**"application / json", "application / xml"**.

5- Run the application

To run the application, right click on the Project, select:

- **Run As/Spring Boot App**





After running the application, you can test its functions.

Test GET:

*If **ERROR**==> Please use **Spring Boot 2.0.0.M5** temporarily.(And wait for the official version of **Spring Boot 2**).*

?

```
<parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
```

```
    <version>2.0.0.M5</version>
    <relativePath/> <!-- lookup parent from repository
-->
</parent>
```

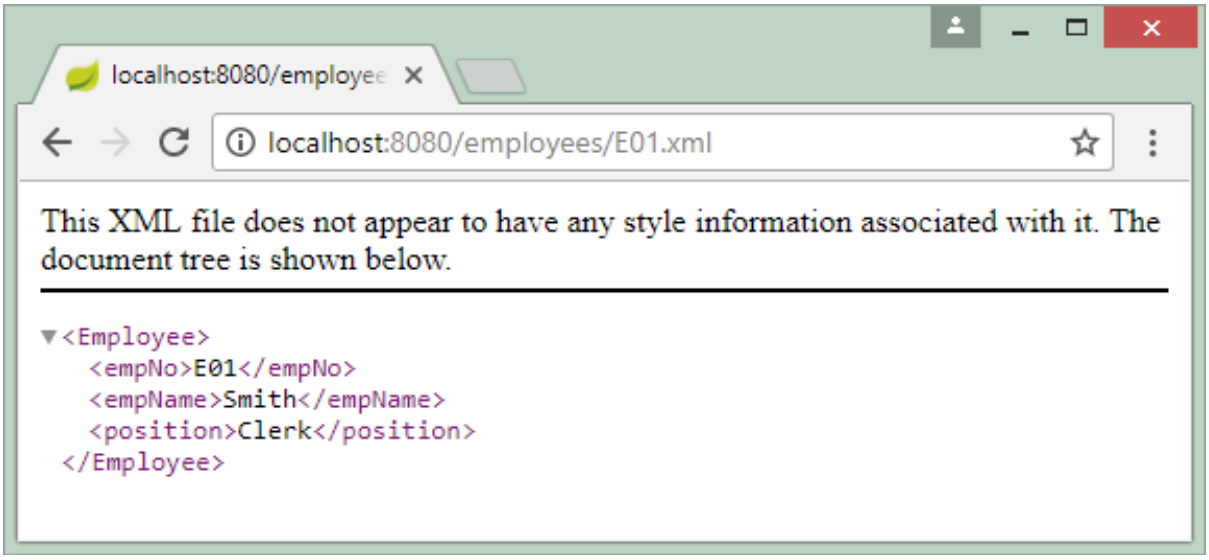
To take the list of employees, the user need to send a request with **GET** method. It can be easy for you to test this function by using a browser.

- http://localhost:8080/employees

Or:

- http://localhost:8080/employees.json
- http://localhost:8080/employees.xml

- http://localhost:8080/employee/E01
- http://localhost:8080/employee/E01.xml
- http://localhost:8080/employee/E01.json



How to create a request using the POST, PUT, or DELETE methods?

*To create a request with the **POST**, **PUT** or **DELETE** methods, you have to use a tool, for example, **RestClient**, **cURL**,.. or write your own **Rest Client** . See more:*

- *RESTClient A Debugger for RESTful Web Services*
- *Spring Boot Restful Client with RestTemplate example*
- *Secure Spring Boot RESTful Service using Basic Authentication*

Test POST

To create an employee, you need to create a request with the **POST** method and attach it to the information of the employee to be created. The data sent will be in **JSON** format or **XML** format:

```
?
POST http://localhost:8080/employee
Acept: application/xml

<Employee>
    <empNo>E11</empNo>
    <empName>New Employee</empName>
    <position>Clerk</position>
</Employee>
```

```
?
POST http://localhost:8080/employee
Acept: application/json

{"empNo":"E11","empName":"New
Employee","position":"Clerk"}
```