**Лабораторна робота № 32 «Створення додатку в Spring Boot з CRUD Restful Web Service»**

В роботі треба розробити додаток **Restful Web Service** за допомогою **Spring Boot**, а також має 4 функції **Create**, **Read**, **Update**, **Delete** (CRUD).

**Метод GET**

Ми побудуємо **URI**, якому призначено (designated) повернути список користувачів (співробітників). Визначте інший **URI**, щоб вернути користувачеві інформацію визначеного співробітника. Дані, які отримають користувач, будуть у форматі **XML** або **JSON**. Ці **URI** приймають тільки запит методом **GET**.

* ОТРИМАТИ http://localhost:8080/employees
* ОТРИМАТИ http://localhost:8080/employee/E01

**Оновлення (метод PUT).**

Побудувати **URI** для обробки запиту (запиту) редагування інформації співробітника (співробітника). Даний **URI** приймає тільки запит методом **PUT**. Дані, прикріплені разом із запитом нової інформації співробітника, мають формат **XML** або **JSON**.

* ВСТАВТЕ http://localhost:8080/employee

Створити (метод POST)

Побудувати **URI** для обробки запиту (запиту) створити нового співробітника (співробітника). Даний **URI** приймає тільки запит методом **POST**. Дані, надіслані з прикріпленим запитом, є інформацією співробітника, яка буде створена, має формат **XML** або **JSON**.

* ОПУБЛІКУЙТЕ http://localhost:8080/employee

**Видалити (метод DELETE).**

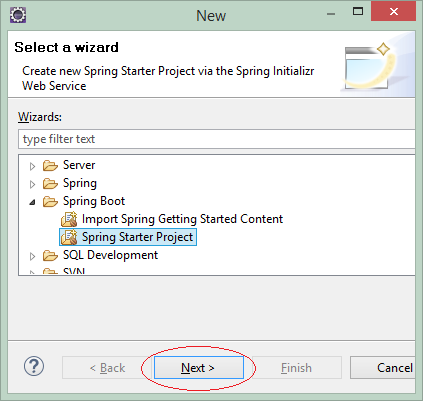
Побудувати **URI** для обробки запиту (запиту) видалити співробітника (співробітника). Даний **URI** приймає тільки запит методом **DELETE**.

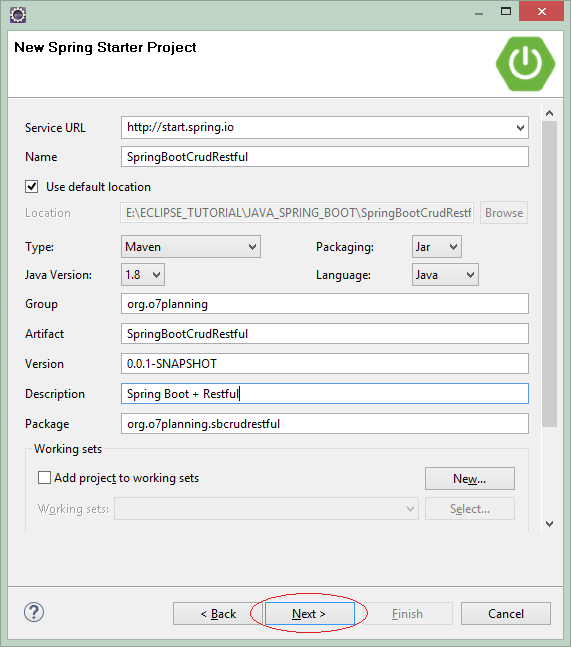
* ВИДАЛИТИ [http://localhost:8080/employee/{empNo}](http://localhost:8080/employee/%7bempNo%7d)

## 2- Створити проект Spring Boot

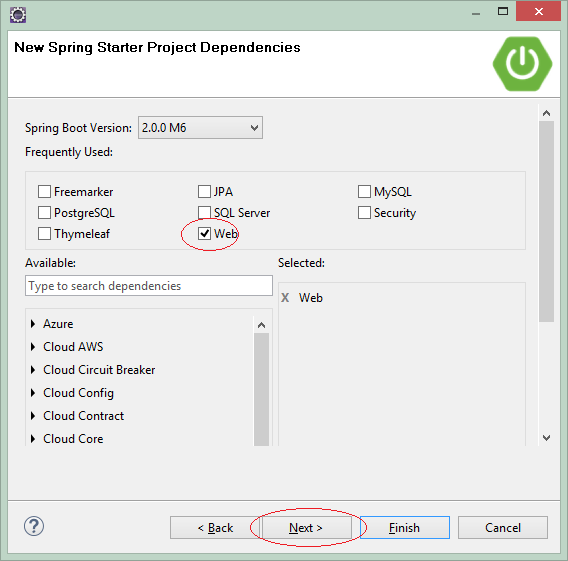
В **Eclipse** вибрати:

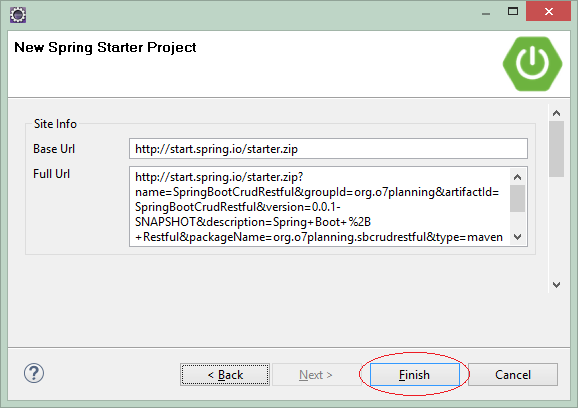
* **Файл/Новий/Інше..**

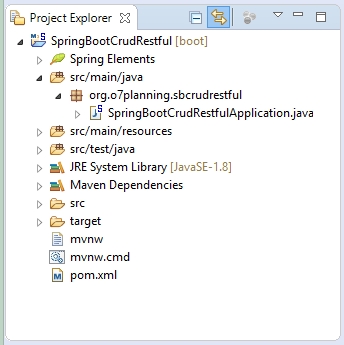




Наступний крок, вам потрібно вибрати технологію використання.







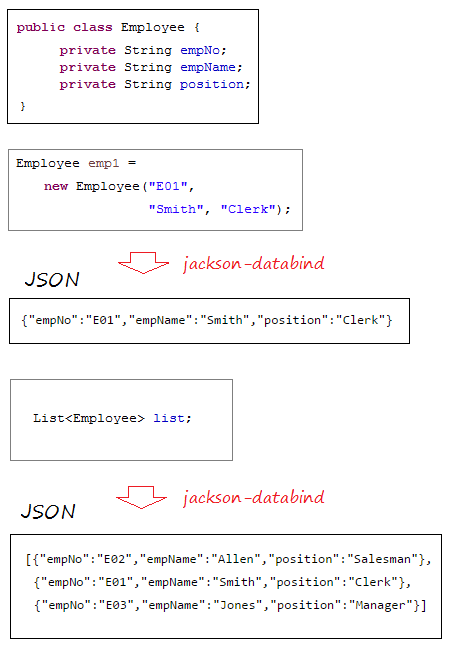
## 

## 3- Конфігурація pom.xml

У цьому прикладі нам потрібна бібліотека для конвертації (перетворення) **XML** в об’єкт **Java** і навпаки. І друга бібліотека для конвертації **JSON** в **Java** і навпаки.

JSON <==> Java

**spring-boot-starter-web** має готову бібліотеку **jackson-databind**, дана бібліотека допомагає конвертувати **JSON** в об’єкт **Java** і навпаки.



XML <==> Java

**Spring Boot** використовує **JAXB** (Готовий у **JDK** ) як бібліотеку за замовчуванням для конвертації **XML** і **Java**. Ваші класи Java повинні бути анотовані (анотовані) за допомогою **@XmlRootElement**,... Тому, моя порада, вам варто використовувати **jackson-dataformat-xml** як бібліотеку для конвертації  **XML** і **Java**. Щоб використовувати **jackson-dataformat-xml,** вам потрібно опублікувати його у файлі **pom.xml** :

\*\* pom.xml \*\*

...

<dependencies>

...

<dependency>

<groupId>com.fasterxml.jackson.dataformat</groupId>

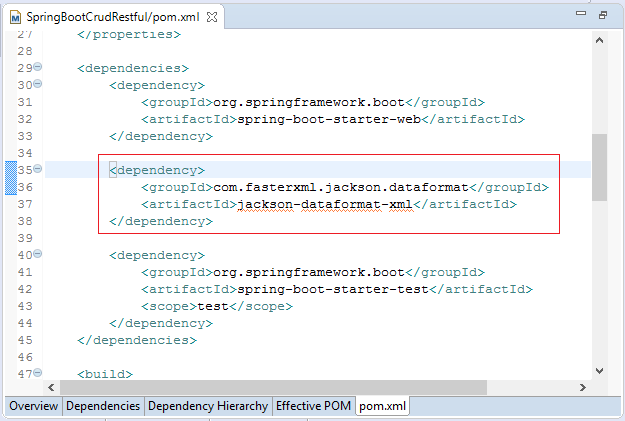
<artifactId>jackson-dataformat-xml</artifactId>

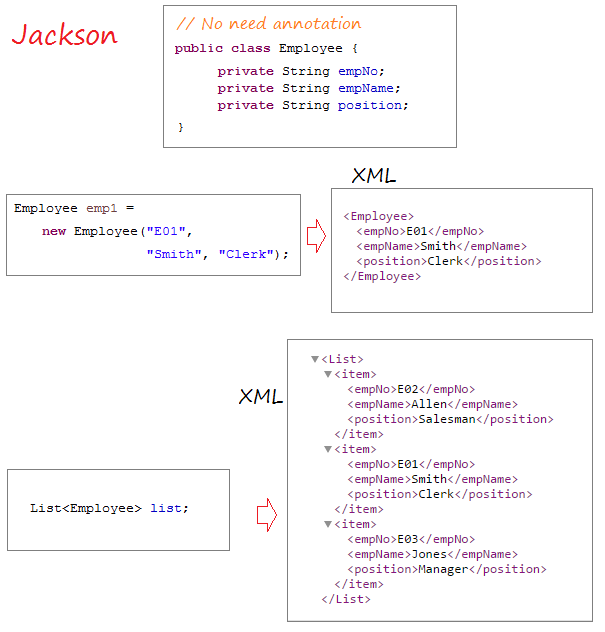
</dependency>

...

</dependencies>

...





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>

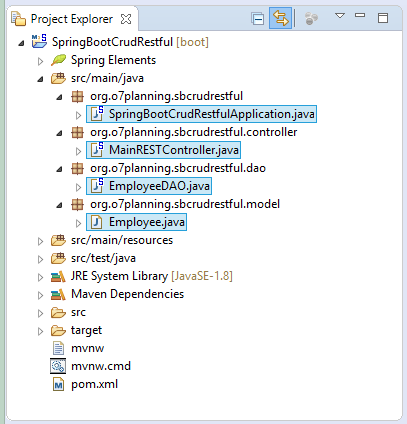
</plugin>

</plugins>

</build>

</project>

## 4- Код додатка



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);

}

}

Класс **Employee** представляє співробітника.

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;

}

}

Клас **EmployeeDAO** анотований (аннотувати) за допомогою **@Repository**, щоб оповістити **Spring**, що він є **Spring BEAN**. Цей клас містить методи, які допомагають дати запит на список співробітників (співробітників), створити співробітника, змінити інформацію про співробітника та видалити співробітника.

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;

}

}

Клас **MainRESTController** анотований (анотувати) за допомогою **@RestController,** щоб сповістити **Spring,** що він є **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);

}

}

Пояснення:

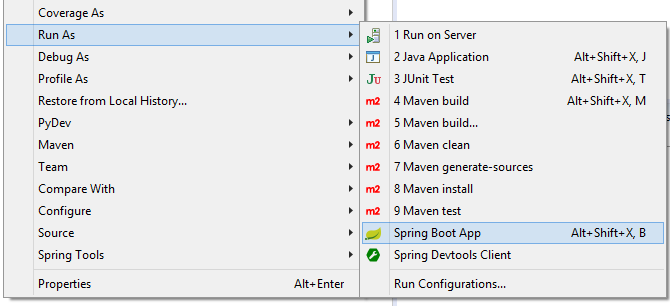
* **створює = { MediaType.APPLICATION\_JSON\_VALUE, MediaType.APPLICATION\_XML\_VALUE }**
* **produces = { "application/json", "application/xml" }**

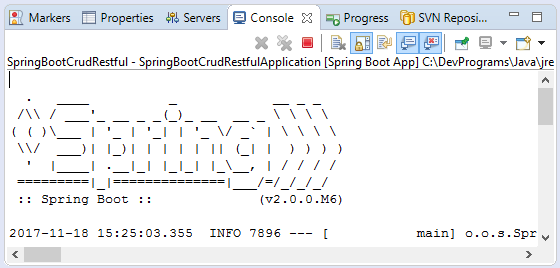
Атрибут **створює** використовується для визначення **URL-адреси**, яка буде створювати тільки (возвращать користувача) дані з такими форматами. Наприклад **"application/json"**, **"application/xml"**.

## 5- Запуск додатків

Щоб запустити додаток, натисніть праву кнопку миші на Project і виберіть:

* **Запустіть додаток As/Spring Boot**





Після запуску програми ви можете протестувати його функції.

**Тест GET:**

Якщо **ПОМИЛКА** ==> Тимчасово використовуйте **Spring Boot 2.0.0.M5**. (І зачекайте на офіційну версію **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>

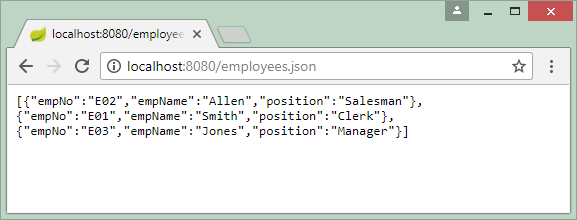
Щоб отримати список співробітників (співробітників), користувачеві потрібно відправити запит (запит) методом **GET**. Ви можете легко протестувати дану функцію за допомогою браузера.

* <http://localhost:8080/employees>

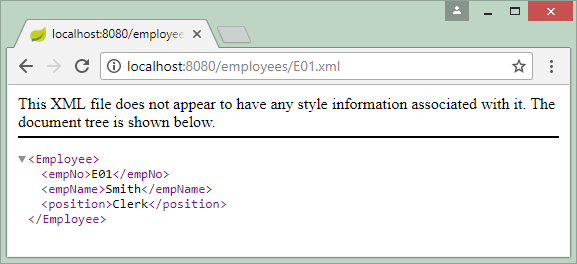


або:

* <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>



Як створити запит за допомогою методів **POST, PUT** або **DELETE** ?

Щоб створити запит (запит) за допомогою методів **POST**, **PUT** або **DELETE** , вам потрібно скористатися інструментом, наприклад **RestClient**, **cURL**,.. або написати ваше власне додаток **Rest Client**.

**Тест POST**

Щоб створити співробітника (співробітника), вам необхідно створити запит за допомогою методу **POST**, і підкріпити інформацію про співробітника, який буде створений, відправлені дані мають формат **JSON** або **XML** :

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"}