

---

## **TP N°3 : Développement d'une application web avec Spring Boot, Spring MVC, RestController, Spring Data JPA et MySQL.**

---

## SOMMAIRE

I- Objectifs : .....	3
II- Outils utilisés : .....	3
III- Installation de MySQL 5.0 et configuration de la perspective « Database Development » au niveau d'Eclipse .....	Erreur ! Signet non défini.
IV- Développement de l'application .....	3
1. Génération du squelette de l'application moyennant Spring Initializr .....	3
2. pom.xml .....	7
3. Le fichier application.properties .....	11
4. Le modèle .....	11
5. Le Value Object .....	12
6. La couche DAO .....	14
7. La couche service .....	14
8. Les contrôleurs .....	16
9. Les pages JSP .....	21
10. Les tests .....	23

### I- Objectifs :

- ✓ Développer les 03 couches :
  - Couche présentation avec Spring MVC ;
  - Couche métier avec RestController ;
  - Couche DAO avec Spring Data JPA.
- ✓ Comprendre le Design Pattern : Value Object. En effet, au niveau du RestController, l'objet qui sera converti en XML/JSON ne doit pas être une classe persistance mais un Java Bean simple.
- ✓ Utiliser Spring Data JPA pour simplifier le développement de la couche DAO.
- ✓ Utiliser Spring Boot pour créer et configurer facilement l'application.

### II- Outils utilisés :

Dans ce TP, nous allons utiliser les outils suivants :

- ✓ Eclipse Neon avec le plugin Maven 3.x ;
- ✓ JDK 1.8 ;
- ✓ Connexion à Internet pour permettre à Maven de télécharger les dépendances nécessaires (Spring Boot 2.2.0, ...) ;
- ✓ POSTMAN ou un autre outil pour tester les méthodes POST, PUT et DELETE ;
- ✓ La base de donnée Mysql 5.0. Pour la base de données, l'exécutable à installer est « MySQL 5.0/Setup.exe ».
- ✓ MySQL 8.0 (le TP n°3 montre comment installer MySQL 8.0).

### III- Développement de l'application

#### **1. Génération du squelette de l'application moyennant Spring Initializr**

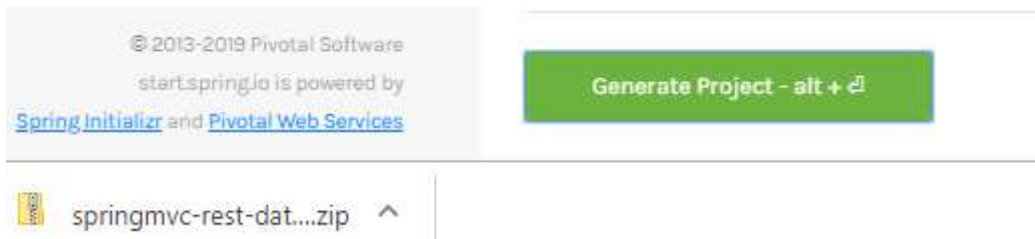
- Aller au site : <https://start.spring.io/> :

The image shows the Spring Initializr web interface for creating a new project. The interface is divided into several sections: Project, Language, Spring Boot, Project Metadata, Packaging, Java Version, Dependencies, and a final Generate Project button. Red boxes and numbers 1 through 10 highlight the steps to follow:

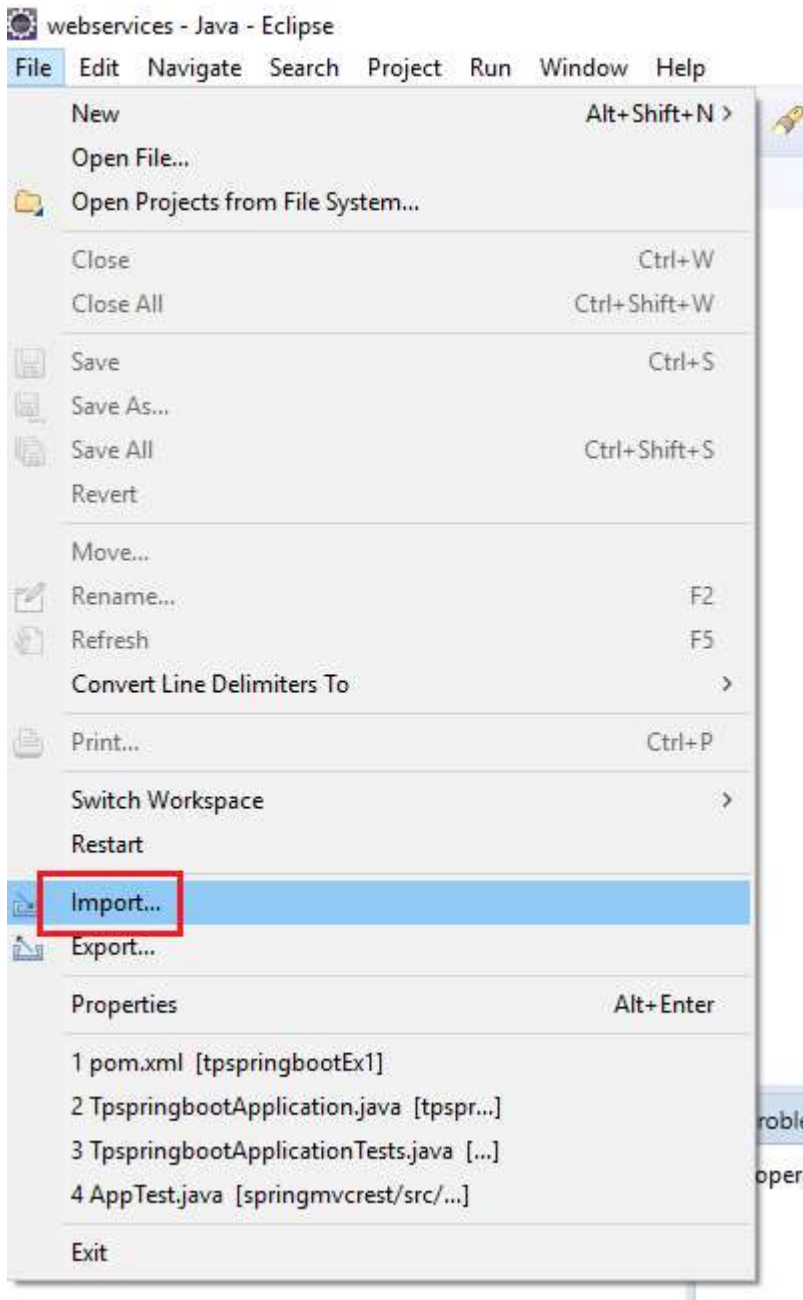
- 1**: Select **Maven Project** under the Project section.
- 2**: Select **Java** under the Language section.
- 3**: Select **2.2.0 (SNAPSHOT)** under the Spring Boot section.
- 4**: Enter **ma.cigma** in the Group field.
- 5**: Enter **springmvc-rest-data-jpa** in the Artifact field.
- 6**: Enter **springmvc-rest-data-jpa** in the Name field.
- 7**: Enter **Spring MVC +RestController +Spring Data JPA + MySQL** in the Description field.
- 8**: Enter **ma.cigma.springmvcrestdatajpa** in the Package Name field.
- 9**: Select **War** under the Packaging section.
- 10**: Select **8** under the Java Version section.
- 10**: In the Dependencies section, select **Web [Web]**, **JPA [SQL]**, and **MySQL [SQL]** from the list of selected dependencies.

At the bottom, there is a green button labeled **Generate Project - alt + ⌘**.

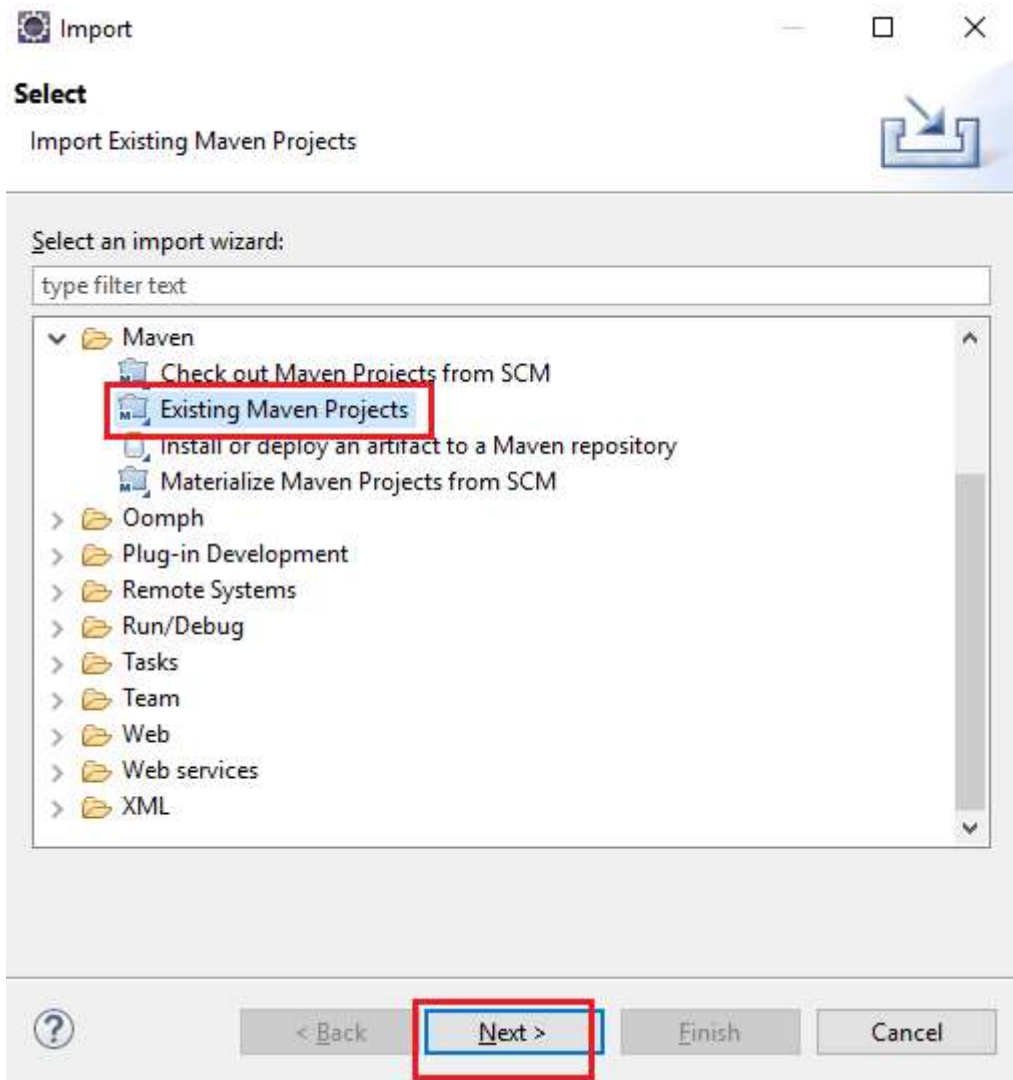
- 1 : Choisir « Maven Project » et Java dans Language.
- 2 : Choisir la version 2.2.0 de Spring Boot.
- 3 : Entrer le group (ma.cigma).
- 4 : Entrer l'artifact (springmvc-rest-data-jpa)
- 5 : Entrer le nom (springmvc-rest-data-jpa).
- 6 : Entrer la description de votre projet (Spring MVC + RestController +Spring Data JPA + MySQL).
- 7 : Entrer le nom du package racine (ma.cigma.springmvcrestdatajpa).
- 8 : Choisir War.
- 9 : Choisir 8 dans la version de Java.
- 10 : Ajouter les dépendances : web, JPA et MySQL.
- Enfin, cliquer sur le bouton « Generate Project ». ». Le fichier ZIP suivant sera généré automatiquement :



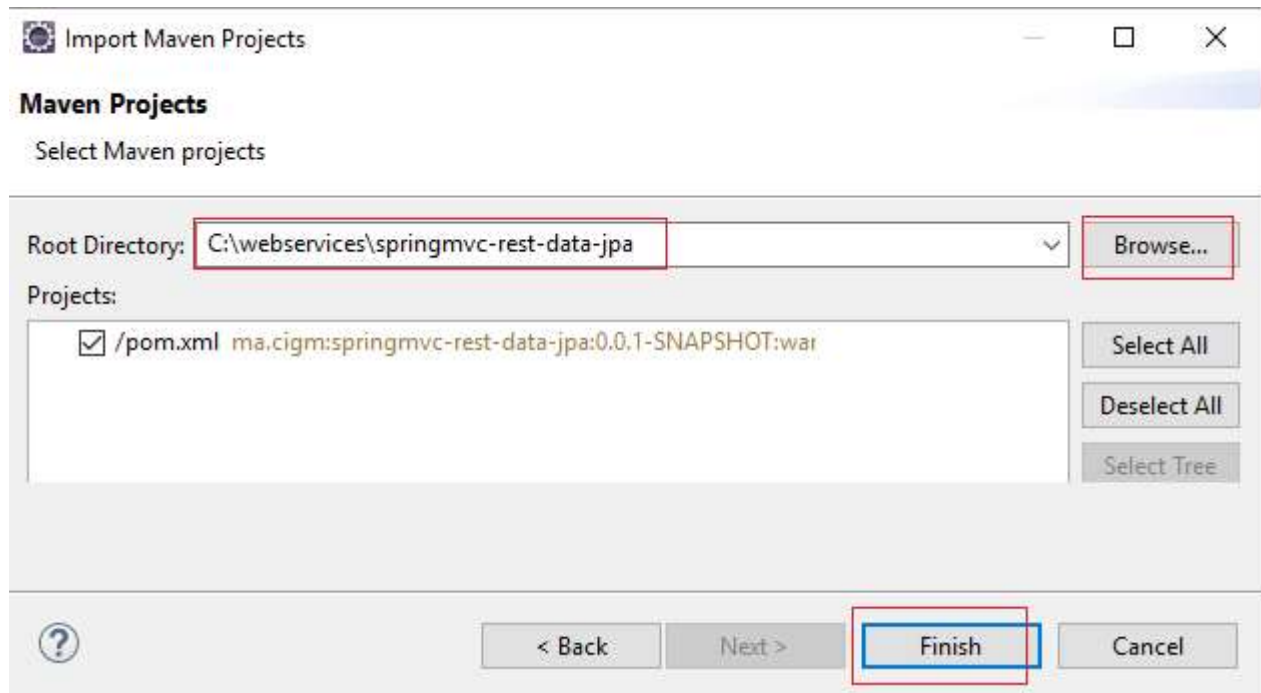
- Décompresser le fichier springmvc-rest-data-jpa.zip dans le dossier c:\webservices par exemple, ensuite importer le projet Maven au niveau d'éclipse comme illustré ci-après :



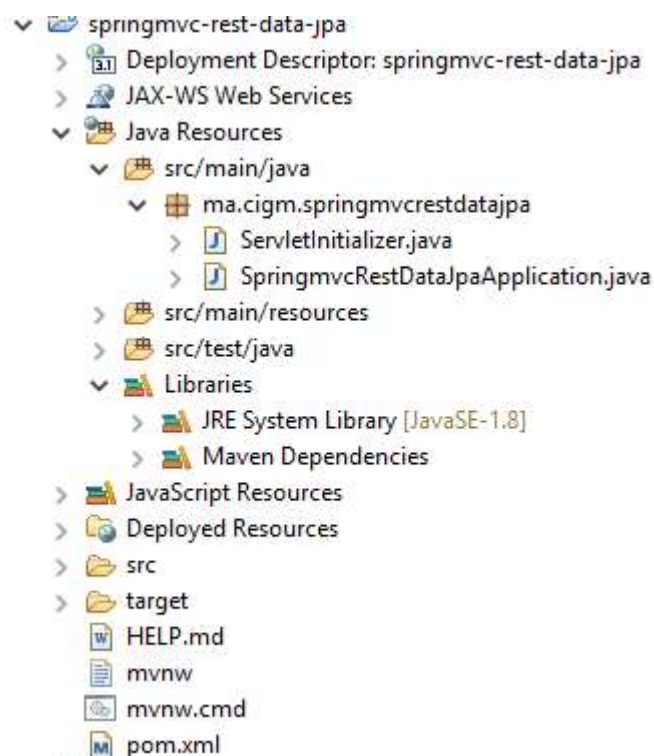
- Cliquer sur le menu « Import... » :



- Choisir « Existing Maven Projects » et cliquer sur Next> :



- Cliquer sur « Browse... » et choisir le répertoire dans lequel existe votre projet Maven (ici : c:\webservices\ springmvc-rest-data-jpa) et cliquer sur Finish. L'arborescence du projet « springmvc-rest-data-jpa » est comme suit :



## 2. pom.xml

- Editer le fichier pom.xml et ajouter les dépendances suivantes :

```

<!-- Pour pouvoir utiliser JSP, les dépendances suivantes sont nécessaires -->
<dependency>
  <groupId>javax.servlet.jsp.jstl</groupId>
  <artifactId>javax.servlet.jsp.jstl-api</artifactId>
  <version>1.2.1</version>
</dependency>

<dependency>
  <groupId>taglibs</groupId>
  <artifactId>standard</artifactId>
  <version>1.1.2</version>
</dependency>

<dependency>
  <groupId>org.apache.tomcat.embed</groupId>
  <artifactId>tomcat-embed-jasper</artifactId>
  <version>9.0.2</version>
</dependency>

<!-- Pour que les RestController puissent produire le format XML, la
dépendance suivante est nécessaire -->
<dependency>
  <groupId>com.fasterxml.jackson.dataformat</groupId>
  <artifactId>jackson-dataformat-xml</artifactId>
</dependency>

```

- Pour le driver MySQL :

```

<dependency>
  <groupId>mysql</groupId>
  <artifactId>mysql-connector-java</artifactId>
  <scope>runtime</scope>
</dependency>

```

Le fichier pom.xml, une fois les modifications opérées, est le suivant :

```

<?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>
  <parent>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-parent</artifactId>
    <version>2.2.0.BUILD-SNAPSHOT</version>
    <relativePath /> <!-- lookup parent from repository -->
  </parent>

```



```

<groupId>ma.cigm</groupId>
<artifactId>springmvc-rest-data-jpa</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>war</packaging>
<name>springmvc-rest-data-jpa</name>
<description>Spring MVC + RestController +Spring Data JPA + MySQL</description>

<properties>
    <java.version>1.8</java.version>
</properties>

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

    <!-- Préciser la version de MySQL que vous avez pour que Maven puisse télécharger le
    driver.
    Si vous ne précisez par la version, Spring Boot utilisera la dernière version de
    MySQL, soit la version 8. -->
    <dependency>
        <groupId>mysql</groupId>
        <artifactId>mysql-connector-java</artifactId>
        <scope>runtime</scope>
    </dependency>

    <!-- Pour pouvoir utiliser JSP, les dépendances suivantes sont nécessaires -->
    <dependency>
        <groupId>javax.servlet.jsp.jstl</groupId>
        <artifactId>javax.servlet.jsp.jstl-api</artifactId>
        <version>1.2.1</version>
    </dependency>

    <dependency>
        <groupId>taglibs</groupId>
        <artifactId>standard</artifactId>
        <version>1.1.2</version>
    </dependency>

    <dependency>
        <groupId>org.apache.tomcat.embed</groupId>
        <artifactId>tomcat-embed-jasper</artifactId>
        <version>9.0.2</version>

```

```

        </dependency>

<!-- Pour que les RestController puissent produire le format XML, la dépendance
      suivante est nécessaire -->
        <dependency>
            <groupId>com.fasterxml.jackson.dataformat</groupId>
            <artifactId>jackson-dataformat-xml</artifactId>
        </dependency>
    </dependencies>

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

    <repositories>
        <repository>
            <id>spring-snapshots</id>
            <name>Spring Snapshots</name>
            <url>https://repo.spring.io/snapshot</url>
            <snapshots>
                <enabled>true</enabled>
            </snapshots>
        </repository>
        <repository>
            <id>spring-milestones</id>
            <name>Spring Milestones</name>
            <url>https://repo.spring.io/milestone</url>
        </repository>
    </repositories>
    <pluginRepositories>
        <pluginRepository>
            <id>spring-snapshots</id>
            <name>Spring Snapshots</name>
            <url>https://repo.spring.io/snapshot</url>
            <snapshots>
                <enabled>true</enabled>
            </snapshots>
        </pluginRepository>
        <pluginRepository>
            <id>spring-milestones</id>
            <name>Spring Milestones</name>
            <url>https://repo.spring.io/milestone</url>
        </pluginRepository>
    </pluginRepositories>
</project>

```

### 3. Le fichier application.properties

Le contenu du fichier application.properties est :

```
#Pour Spring MVC :
spring.mvc.view.prefix=/vues/
spring.mvc.view.suffix=.jsp

#Pour MySQL :
spring.datasource.driver-class-name=com.mysql.jdbc.Driver
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5InnoDB
Dialect
spring.jpa.show-sql=true
spring.datasource.url=jdbc:mysql://localhost:3306/cigmadb
spring.datasource.username=root
spring.datasource.password=root

# Pour la génération automatique des tables.
# Attention, il faut commenter cette ligne sur l'environnement
# de production.
spring.jpa.hibernate.ddl-auto=update
```

Ce fichier permet de configurer Spring MVC (le chemin des pages \*.jsp), le driver, le dialect, l'url, le nom de l'utilisateur, le mot de passe et également la configuration de JPA pour créer automatiquement les tables.

### 4. Le modèle

- Créer le package `ma.cigm.springmvcrestdatajpa.service.modele`
- Ensuite, créer la classe **Emp** suivante :

```
package ma.cigm.springmvcrestdatajpa.service.modele;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.Id;

@Entity
public class Emp {
    @Id
    @GeneratedValue
    private Long id;
    private String name;
    private Double salary;
    private String fonction;

    @Override
    public String toString() {
        return "Emp [id=" + id + ", name=" + name + ", salary=" + salary + ", fonction=" +
fonction + "]";
    }

    public Emp() {
```

```

        super();
    }

    public Emp(String name, Double salary, String fonction) {
        super();
        this.name = name;
        this.salary = salary;
        this.fonction = fonction;
    }

    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 Double getSalary() {
        return salary;
    }

    public void setSalary(Double salary) {
        this.salary = salary;
    }

    public String getFonction() {
        return fonction;
    }

    public void setFonction(String fonction) {
        this.fonction = fonction;
    }
}

```

## 5. Le Value Object

- Créer le package `ma.cigm.springmvcrestdatajpa.domaine`.
- Ensuite, créer la classe ***EmpVo*** suivante :

```

package ma.cigm.springmvcrestdatajpa.domaine;

public class EmpVo {
    private Long id;
    private String name;
    private Double salary;
    private String fonction;
    public EmpVo() {

```

```

        super();
    }
    public EmpVo(Long id, String name, Double salary, String fonction) {
        super();
        this.id = id;
        this.name = name;
        this.salary = salary;
        this.fonction = fonction;
    }
    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 Double getSalary() {
        return salary;
    }
    public void setSalary(Double salary) {
        this.salary = salary;
    }
    public String getFonction() {
        return fonction;
    }
    public void setFonction(String fonction) {
        this.fonction = fonction;
    }
}

```

- Créer la classe **EmpConverter** suivante :

```

package ma.cigm.springmvcrestdatajpa.domaine;

import java.util.ArrayList;
import java.util.List;
import ma.cigm.springmvcrestdatajpa.service.modele.Emp;

public class EmpConverter {
    public static EmpVo toVo(Emp bo) {
        if (bo == null || bo.getId() == null)
            return null;
        EmpVo vo = new EmpVo();
        vo.setId(bo.getId());
        vo.setName(bo.getName());
        vo.setSalary(bo.getSalary());
        vo.setFonction(bo.getFonction());
        return vo;
    }
    public static Emp toBo(EmpVo vo) {
        Emp bo = new Emp();
        bo.setId(vo.getId());
        bo.setName(vo.getName());
        bo.setSalary(vo.getSalary());
    }
}

```

```

        bo.setFonction(vo.getFonction());
        return bo;
    }
    public static List<EmpVo> toListVo(List<Emp> listBo) {
        List<EmpVo> listVo = new ArrayList<>();
        for (Emp emp : listBo) {
            listVo.add(toVo(emp));
        }
        return listVo;
    }
}

```

## 6. La couche DAO

- Créer le package `ma.cigm.springmvcrestdatajpa.dao`.
- Ensuite, créer l'interface ***EmpRepository*** suivante :

```

package ma.cigm.springmvcrestdatajpa.dao;

import java.util.List;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;

import ma.cigm.springmvcrestdatajpa.service.modele.Emp;

/**
 *
 * Ici, l'interface EmpRepository hérite de l'interface JpaRepository de Spring
 * DATA. Il faut juste préciser la classe "Modele" et le type de la classe qui
 * représente la clé primaire.
 *
 * Spring Data prendra en charge l'implémentation des 04 méthode ci-dessous à
 * condition de respecter la nomenclature supportée par Spring Data.
 *
 * @Query offre la possibilité d'exécuter des requêtes plus complexes.
 *
 */
public interface EmpRepository extends JpaRepository<Emp, Long> {
    List<Emp> findBySalary(Double salary);
    List<Emp> findByFonction(String designation);
    List<Emp> findBySalaryAndFonction(Double salary, String fonction);
    @Query(" SELECT e from Emp e where e.salary=(select MAX(salary) as salary FROM Emp)")
    Emp getEmpHavaingMaxSalary();
}

```

## 7. La couche service

- Créer le package `ma.cigm.springmvcrestdatajpa.service`
- Créer en suite l'interface ***IService*** suivante :

```

package ma.cigm.springmvcrestdatajpa.service;
import java.util.List;
import ma.cigm.springmvcrestdatajpa.domaine.EmpVo;
public interface IService {
    List<EmpVo> getEmployees();
}

```

```

    void save(EmpVo emp);
    EmpVo getEmpById(Long id);
    void delete(Long id);
    List<EmpVo> findBySalary(Double salary);
    List<EmpVo> findByFonction(String designation);
    List<EmpVo> findBySalaryAndFonction(Double salary, String fonction);
    EmpVo getEmpHavaingMaxSalary();
    //Pour la pagination
    List<EmpVo> findAll(int pageId, int size);
    //pour le tri
    List<EmpVo> sortBy(String fieldName);
}

```

- Créer ensuite la classe **ServiceImpl** suivante :

```

package ma.cigm.springmvcresdatajpa.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.CommandLineRunner;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.data.domain.Sort;
import org.springframework.data.domain.Sort.Direction;
import org.springframework.stereotype.Service;
import ma.cigm.springmvcresdatajpa.dao.EmpRepository;
import ma.cigm.springmvcresdatajpa.domaine.EmpConverter;
import ma.cigm.springmvcresdatajpa.domaine.EmpVo;
import ma.cigm.springmvcresdatajpa.service.modele.Emp;

@Service
public class ServiceImpl implements IService, CommandLineRunner {
    @Autowired
    private EmpRepository empRepository;
    @Override
    public List<EmpVo> getEmployees() {
        List<Emp> list = empRepository.findAll();
        return EmpConverter.toListVo(list);
    }
    @Override
    public void save(EmpVo emp) {
        empRepository.save(EmpConverter.toBo(emp));
    }
    @Override
    public EmpVo getEmpById(Long id) {
        boolean trouve = empRepository.existsById(id);
        if (!trouve)
            return null;
        return EmpConverter.toVo(empRepository.getOne(id));
    }
    @Override
    public void delete(Long id) {
        empRepository.deleteById(id);
    }
    @Override
    public List<EmpVo> findBySalary(Double salary) {

```

```

        List<Emp> list = empRepository.findBySalary(salaty);
        return EmpConverter.toListVo(list);
    }
    @Override
    public List<EmpVo> findByFonction(String fonction) {
        List<Emp> list = empRepository.findByFonction(fonction);
        return EmpConverter.toListVo(list);
    }
    @Override
    public List<EmpVo> findBySalaryAndFonction(Double salary, String fonction) {
        List<Emp> list = empRepository.findBySalaryAndFonction(salary, fonction);
        return EmpConverter.toListVo(list);
    }
    @Override
    public EmpVo getEmpHavaingMaxSalary() {
        return EmpConverter.toVo(empRepository.getEmpHavaingMaxSalary());
    }
    @Override
    public List<EmpVo> findAll(int pageId, int size) {
        Page<Emp> result = empRepository.findAll(PageRequest.of(pageId, size,
Direction.ASC, "name"));
        return EmpConverter.toListVo(result.getContent());
    }
    @Override
    public List<EmpVo> sortBy(String fieldName) {
        return EmpConverter.toListVo(empRepository.findAll(Sort.by(fieldName)));
    }
    /**
     * Spring Boot lance cette méthode une fois l'application est démarré.
     */
    @Override
    public void run(String... args) throws Exception {
        empRepository.deleteAll();
        empRepository.save(new Emp("name1", 8500d, "Technicien"));
        empRepository.save(new Emp("name2", 8500d, "Technicien"));
        empRepository.save(new Emp("name3", 8500d, "Chauffeur"));
        empRepository.save(new Emp("name4", 8500d, "Comptable"));
        empRepository.save(new Emp("name5", 10000d, "Comptable"));
        empRepository.save(new Emp("name6", 15000d, "Chef de projet"));
        empRepository.save(new Emp("name7", 17500d, "Responsable du service"));
        empRepository.save(new Emp("name8", 10000d, "Comptable"));
    }
}

```

## 8. Les contrôleurs

- Créer le package `ma.cigm.springmvcrestdatajpa.controller`
- Ensuite, créer la classe ***EmpController*** suivante :



```

package ma.cigm.springmvcrestdatajpa.controller;

import java.util.ArrayList;
import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;

import ma.cigm.springmvcrestdatajpa.domaine.EmpVo;
import ma.cigm.springmvcrestdatajpa.service.IService;

@Controller
public class EmpController {

    @Autowired
    private IService service;

    /**
     * Lorsqu'on tape le lien http://localhost:8080, la page
     * /WEB-INF/vues/index.jsp. Aucun objet n'est passé dans le Model.
     */
    @RequestMapping("/")
    public String showWelcomeFile(Model m) {
        return "index";
    }

    /**
     * Permet d'afficher la page /WEB-INF/vues/empform.jsp. L'objet qui est
     * passé dans la requête est "employee" de type la classe EmpVo. Les
     * attributs de l'objet "employee" seront accessibles au niveau de la page
     * moyennant les getters et les setters.
     */
    @RequestMapping("/empform")
    public String showform(Model m) {
        m.addAttribute("empVo", new EmpVo());
        return "empform";
    }

    /**
     * 1°) Au niveau du formulaire "empform.jsp", lorsqu'on clique sur le bouton
     * Submit, l'action "/save" sera exécutée. Les valeurs du formulaires seront
     * passés dans l'objet EmpVo. Ici, il faut préciser que la méthode HTTP est
     * bien POST car la méthode par défaut est GET.
     *
     * 2°) la méthode save() de l'interface IService sera lancée. 3°) Ensuite la
     * réponse sera redirigée vers la page /WEB-INF/vues/viewemp.jsp
     */
    @RequestMapping(value = "/save", method = RequestMethod.POST)
    public String save(@ModelAttribute("empVo") EmpVo emp) {
        service.save(emp);
        return "redirect:/viewemp"; // will redirect to viewemp request mapping
    }

    /**

```

```

    * lorsqu'on tape le lien http://localhost:8080/viewemp, la page
    * /WEB-INF/vues/viewemp.jsp sera affichée. La liste des employés est
    * placée dans le Model.
    */
    @RequestMapping("/viewemp")
    public String viewemp(Model m) {
        List<EmpVo> list = service.getEmployees();
        m.addAttribute("list", list);
        return "viewemp";
    }

    /**
     * lorsqu'on tape le lien http://localhost:8080/editemp/id, la page
     * /WEB-INF/vues/empeditform.jsp sera affichée. L'objet EmpVo est placé dans
     * le Model.
     */
    @RequestMapping(value = "/editemp/{id}")
    public String edit(@PathVariable Long id, Model m) {
        EmpVo emp = service.getEmpById(id);
        m.addAttribute("empVo", emp);
        return "empeditform";
    }

    /**
     * lorsqu'on tape le lien http://localhost:8080/editsave, l'objet EmpVo est
     * passé dans la requête, ensuite on exécute la méthode save(). Ensuite, on
     * redirige la réponse vers la page /WEB-INF/vues/viewemp.jsp. Ici, il faut
     * préciser la méthode POST.
     */
    @RequestMapping(value = "/editsave", method = RequestMethod.POST)
    public String editsave(@ModelAttribute("empVo") EmpVo emp) {
        service.save(emp);
        return "redirect:/viewemp";
    }

    /**
     * lorsqu'on tape le lien http://localhost:8080/deleteemp/id, on récupère la
     * valeur du paramètre id, on exécute save() et après on redirige la réponse
     * vers la page /WEB-INF/vues/viewemp.jsp.
     */
    @RequestMapping(value = "/deleteemp/{id}", method = RequestMethod.GET)
    public String delete(@PathVariable Long id) {
        service.delete(id);
        return "redirect:/viewemp";
    }

    /**
     * Chercher la liste des employés ayant le même salaire
     */
    @RequestMapping("/salary/{salary}")
    public String getBySalary(@PathVariable Double salary, Model m) {
        List<EmpVo> list = service.findBySalary(salary);
        m.addAttribute("list", list);
        return "viewemp";
    }

    /**
     * Chercher la liste des employés ayant la même fonction
     */

```

```

@RequestMapping("/fonction/{fonction}")
public String getByFonction(@PathVariable String fonction, Model m) {
    List<EmpVo> list = service.findByFonction(fonction);
    m.addAttribute("list", list);
    return "viewemp";
}

/**
 * Chercher la liste des employés ayant le même salaire et la même fonction
 */
@RequestMapping("/salary_and_fonction/{salary}/{fonction}")
public String getBySalaryAndFonction(@PathVariable Double salary, @PathVariable
String fonction, Model m) {
    List<EmpVo> list = service.findBySalaryAndFonction(salary, fonction);
    m.addAttribute("list", list);
    return "viewemp";
}

/**
 * Chercher l'employé qui le grand salaire
 */
@RequestMapping("/max_salary")
public String getMaxSalary(Model m) {
    EmpVo empVo = service.getEmpHavaingMaxSalary();
    List<EmpVo> list = new ArrayList<>();
    list.add(empVo);
    m.addAttribute("list", list);
    return "viewemp";
}

/**
 * Afficher la liste des employés en utilisant la pagination
 */
@RequestMapping("/pagination/{pageid}/{size}")
public String pagination(@PathVariable int pageid, @PathVariable int size, Model m)
{
    List<EmpVo> list = service.findAll(pageid, size);
    m.addAttribute("list", list);
    return "viewemp";
}

/**
 * Trier les employés par le nom de champs qu'on passe dans l'URL
 */
@RequestMapping("/sort/{fieldName}")
public String sortBy(@PathVariable String fieldName, Model m) {
    List<EmpVo> list = service.sortBy(fieldName);
    m.addAttribute("list", list);
    return "viewemp";
}
}

```

- Créer le package `ma.cigm.springmvcrestdatajpa.controller.rest`
- Ensuite, créer la classe **EmpRestController** suivante :

```

package ma.cigm.springmvcrestdatajpa.controller.rest;

import java.util.List;

import javax.validation.Valid;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.MediaType;
import org.springframework.http.ResponseEntity;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.DeleteMapping;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.PathVariable;
import org.springframework.web.bind.annotation.PostMapping;
import org.springframework.web.bind.annotation.PutMapping;
import org.springframework.web.bind.annotation.RequestBody;
import org.springframework.web.bind.annotation.RestController;

import ma.cigm.springmvcrestdatajpa.domaine.EmpVo;
import ma.cigm.springmvcrestdatajpa.service.IService;

@RestController
public class EmpRestController {
    /**
     * @Autowired permet d'injecter le bean de type IProdcutService (objet
     * représentant la couche métier). Ici, le Design Pattern qui est
     * appliqué est l'IOC (Inversion Of Control).
     */
    @Autowired
    private IService service;

    /**
     * Pour chercher tous les employés
     */
    @GetMapping(value = "/rest/emp", produces = { MediaType.APPLICATION_XML_VALUE,
    MediaType.APPLICATION_JSON_VALUE })
    public List<EmpVo> getAll() {
        return service.getEmployees();
    }

    /**
     * Pour chercher un employé par son id
     */
    @GetMapping(value = "/rest/emp/{id}")
    public ResponseEntity<Object> getEmpById(@PathVariable(value = "id") Long empVoId)
    {
        EmpVo empVoFound = service.getEmpById(empVoId);
        if (empVoFound == null)
            return new ResponseEntity<>("employee doesn't exist", HttpStatus.OK);
        return new ResponseEntity<>(empVoFound, HttpStatus.OK);
    }

    /**
     * Pour créer un nouveau employé
     */
    @PostMapping(value = "/rest/emp")
    public ResponseEntity<Object> createEmp(@Valid @RequestBody EmpVo empVo) {
        service.save(empVo);
    }
}

```

```

        return new ResponseEntity<>("employee is created successfully",
HttpStatus.CREATED);
    }

    /**
     * Pour modifier un produit par son id
     */
    @PutMapping(value = "/rest/emp/{id}")
    public ResponseEntity<Object> updateEmp(@PathVariable(name = "id") Long empVoId,
@RequestBody EmpVo empVo) {
        EmpVo empVoFound = service.getEmpById(empVoId);
        if (empVoFound == null)
            return new ResponseEntity<>("employee doesn't exist", HttpStatus.OK);
        empVo.setId(empVoId);
        service.save(empVo);
        return new ResponseEntity<>("Employee is updated successssfully",
HttpStatus.OK);
    }

    /**
     * Pour supprimer un employé par son id
     */
    @DeleteMapping(value = "/rest/emp/{id}")
    public ResponseEntity<Object> deleteEmp(@PathVariable(name = "id") Long empVoId) {
        EmpVo empVoFound = service.getEmpById(empVoId);
        if (empVoFound == null)
            return new ResponseEntity<>("employee doesn't exist", HttpStatus.OK);
        service.delete(empVoId);
        return new ResponseEntity<>("Employee is deleted successssfully",
HttpStatus.OK);
    }

    /**
     * Pour chercher tous les employés
     */
    @GetMapping(value = "/rest/sort/{fieldName}", produces = {
MediaType.APPLICATION_XML_VALUE, MediaType.APPLICATION_JSON_VALUE })
    public List<EmpVo> sortBy(@PathVariable String fieldName) {
        return service.sortBy(fieldName);
    }

    /**
     * Afficher la liste des employés en utilisant la pagination
     */
    @GetMapping("/rest/pagination/{pageid}/{size}")
    public List<EmpVo> pagination(@PathVariable int pageid, @PathVariable int size,
Model m) {
        return service.findAll(pageid, size);
    }
}

```

## 9. Les pages JSP

- Créer le dossier « vues » dans src/main/webapp.
- Créer en suite les pages suivantes :

src/main/webapp/vues/index.jsp

<a href="empform">Add Employee</a>

```
<a href="viewemp">View Employees</a>
```

```
src/main/webapp/vues/empform.jsp
```

```
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<h1>Add New Employee</h1>
<form:form method="post" action="save" modelAttribute="empVo">
    <table>
        <tr>
            <td>Name :</td>
            <td><form:input path="name" /></td>
        </tr>
        <tr>
            <td>Salary :</td>
            <td><form:input path="salary" /></td>
        </tr>
        <tr>
            <td>Fonction :</td>
            <td><form:input path="fonction" /></td>
        </tr>
        <tr>
            <td></td>
            <td><input type="submit" value="Save" /></td>
        </tr>
    </table>
</form:form>
```

```
src/main/webapp/vues/viewemp.jsp
```

```
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

<h1>Employees List</h1>
<table border="2" width="70%" cellpadding="2">
    <tr>
        <th>Id</th>
        <th>Name</th>
        <th>Salary</th>
        <th>Fonction</th>
        <th>Edit</th>
        <th>Delete</th>
    </tr>
    <c:forEach var="empVo" items="${list}">
        <tr>
            <td>${empVo.id}</td>
            <td>${empVo.name}</td>
            <td>${empVo.salary}</td>
            <td>${empVo.fonction}</td>
            <td><a href="editemp/${empVo.id}">Edit</a></td>
            <td><a href="deleteemp/${empVo.id}">Delete</a></td>
        </tr>
    </c:forEach>
</table>
<br />
<a href="empform">Add New Employee</a>
```

src/main/webapp/vues/ **empeditform.jsp**

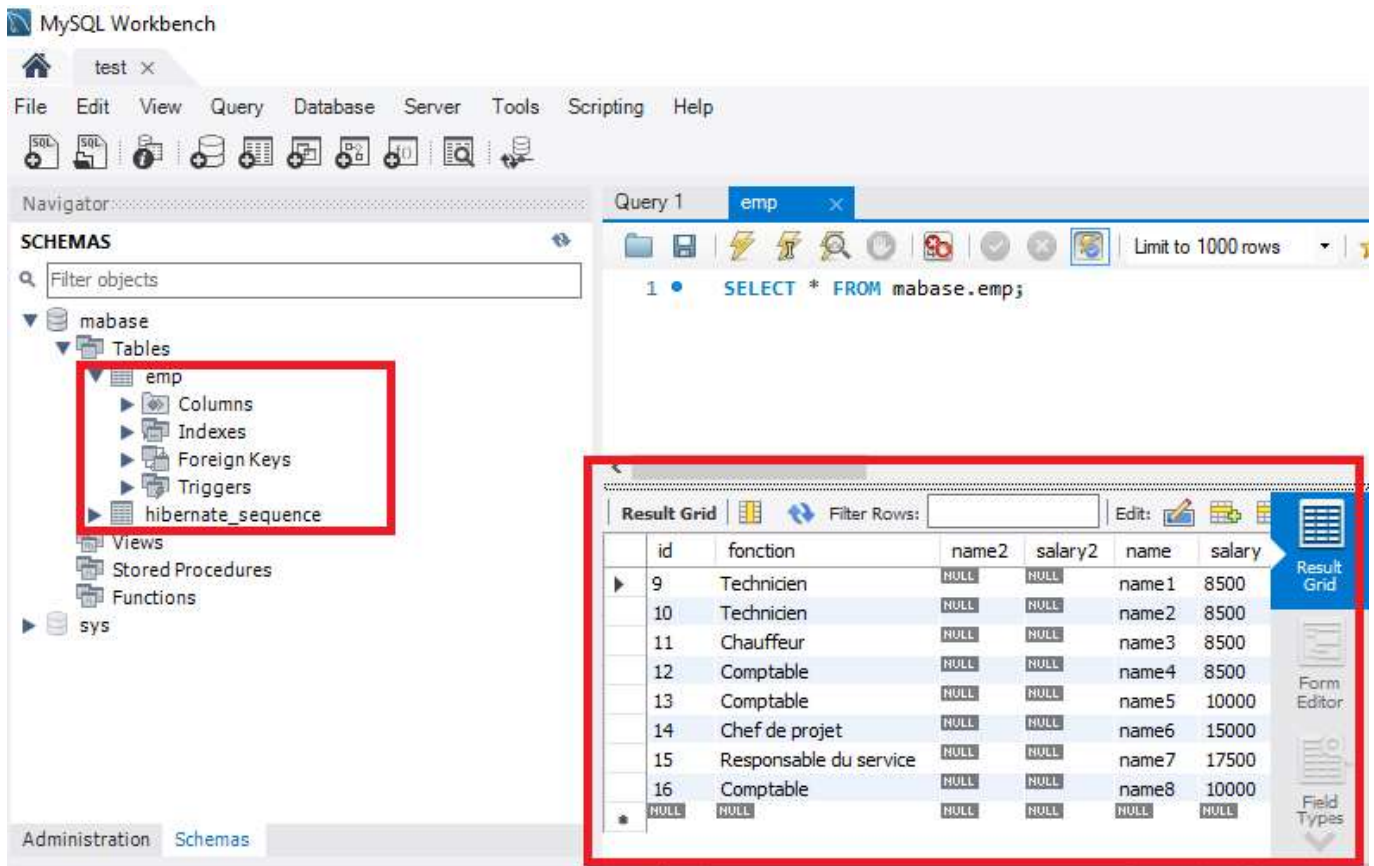
```
<%@ taglib uri="http://www.springframework.org/tags/form" prefix="form"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>

    <h1>Edit Employee</h1>
    <form:form method="POST" action="/editsave" modelAttribute="empVo">
        <table >
            <tr>
                <td></td>
                <td><form:hidden path="id" /></td>
            </tr>
            <tr>
                <td>Name : </td>
                <td><form:input path="name" /></td>
            </tr>
            <tr>
                <td>Salary :</td>
                <td><form:input path="salary" /></td>
            </tr>
            <tr>
                <td>Fonction :</td>
                <td><form:input path="fonction" /></td>
            </tr>

            <tr>
                <td> </td>
                <td><input type="submit" value="Edit Save" /></td>
            </tr>
        </table>
    </form:form>
```

## 10. Les tests

- Lancer la méthode main de la classe SpringmvcRestDataJpaApplication et vérifier que la table Emp a été bien créée. Vérifier également que la méthode run(String... args) de la classe ServiceImpl a été bien exécutée :

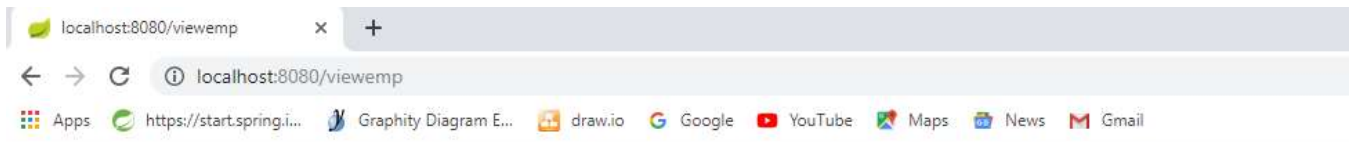


- Dans le navigateur, lancer le lien <http://localhost:8080/> et vérifier que la page index.jsp a été bien exécutée :

[Add Employee](#) [View Employees](#)

- Cliquer sur le lien « View Employees » et vérifier que l'application affiche la liste des employés :



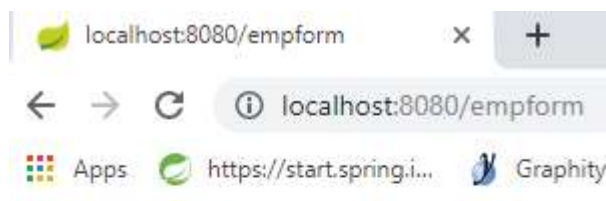


## Employees List

Id	Name	Salary	Fonction	Edit	Delete
1	name1	8500.0	Technicien	<a href="#">Edit</a>	<a href="#">Delete</a>
2	name2	8500.0	Technicien	<a href="#">Edit</a>	<a href="#">Delete</a>
3	name3	8500.0	Chauffeur	<a href="#">Edit</a>	<a href="#">Delete</a>
4	name4	8500.0	Comptable	<a href="#">Edit</a>	<a href="#">Delete</a>
5	name5	10000.0	Comptable	<a href="#">Edit</a>	<a href="#">Delete</a>
6	name6	15000.0	Chef de projet	<a href="#">Edit</a>	<a href="#">Delete</a>
7	name7	17500.0	Responsable du service	<a href="#">Edit</a>	<a href="#">Delete</a>
8	name8	10000.0	Comptable	<a href="#">Edit</a>	<a href="#">Delete</a>

[Add New Employee](#)

- Pour ajouter un nouvel employé, cliquer sur « Add New Employee » :



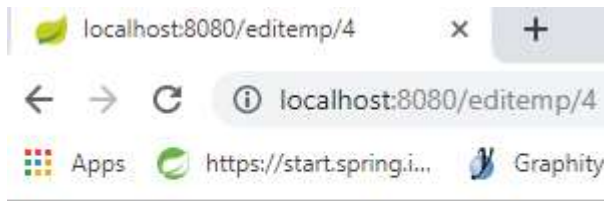
## Add New Employee

Name :

Salary :

Fonction :

- Entrer les données et cliquer sur Save pour créer le nouvel employé.
- Pour modifier un employé, cliquer sur « Edit » :



## Edit Employee

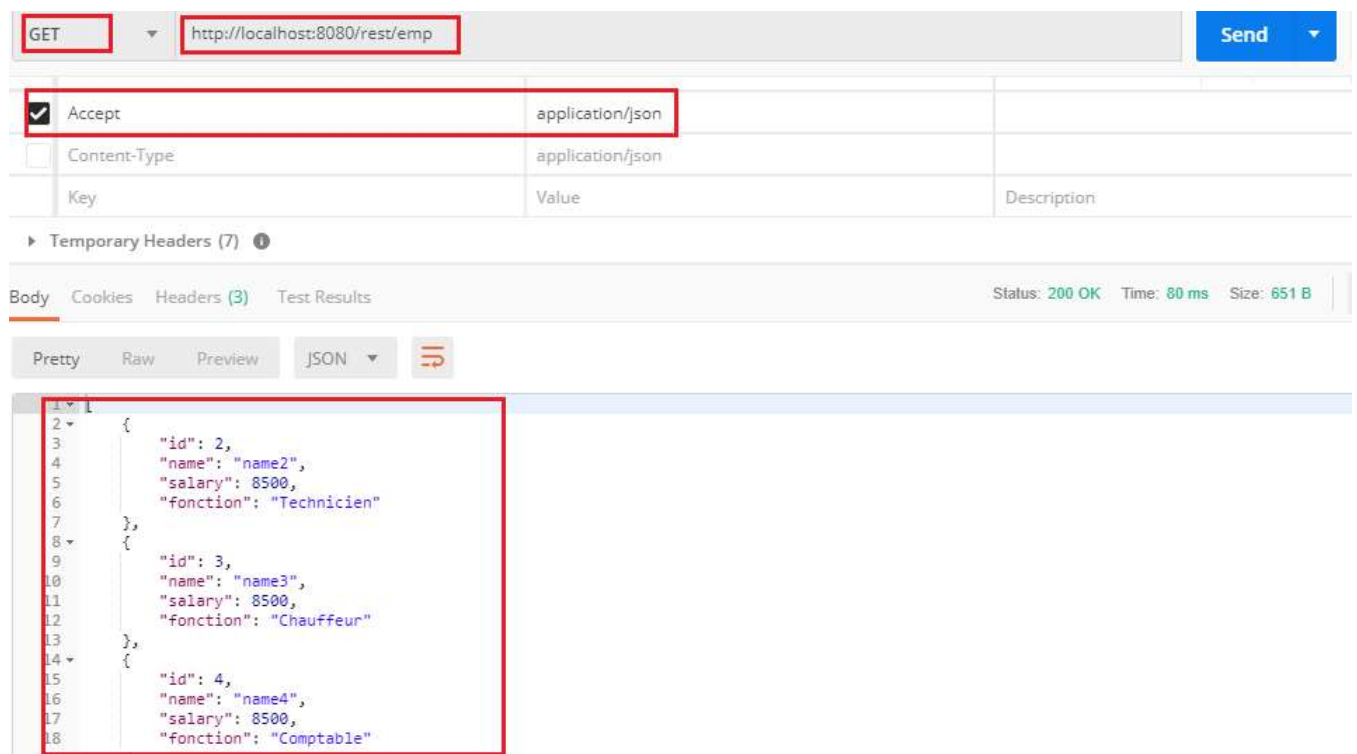
Name :	<input type="text" value="name4"/>
Salary :	<input type="text" value="8500.0"/>
Fonction :	<input type="text" value="Comptable"/>
<input type="button" value="Edit Save"/>	

- Entrer les modifications et ensuite cliquer sur « Edit Save » pour valider les modifications.
- Pour supprimer un employé cliquer sur « Delete ». L'application permettra de supprimer l'employé sélectionné et ensuite redirige la réponse vers la page viewemp.jsp.

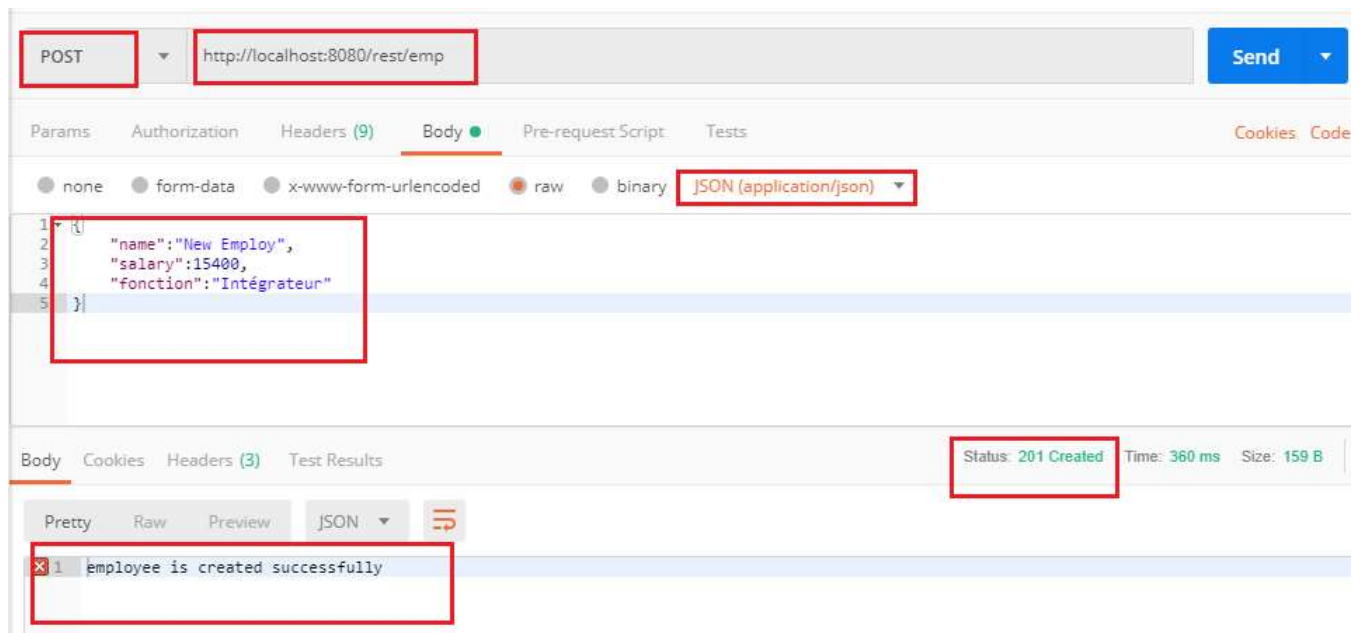
### Les tests de l'API REST :

Lancer POSTMAN et tester les méthodes GET, POST, PUT et DELETE.

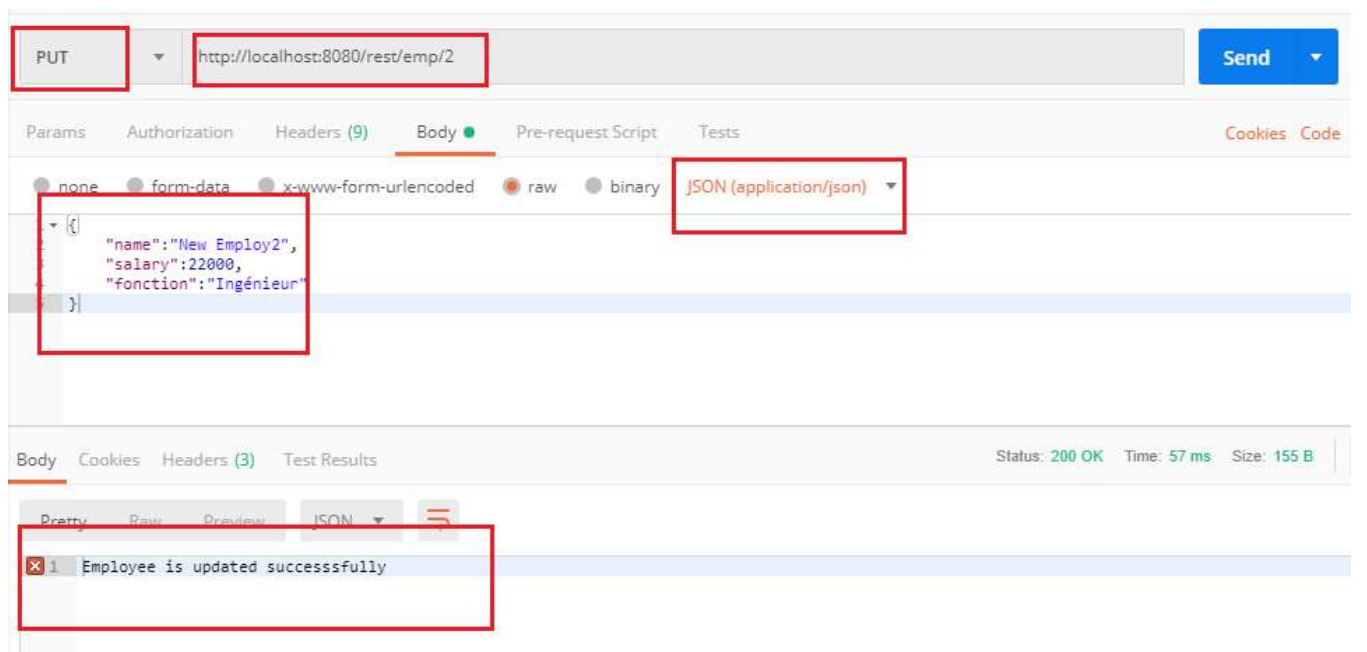
Pour GET :



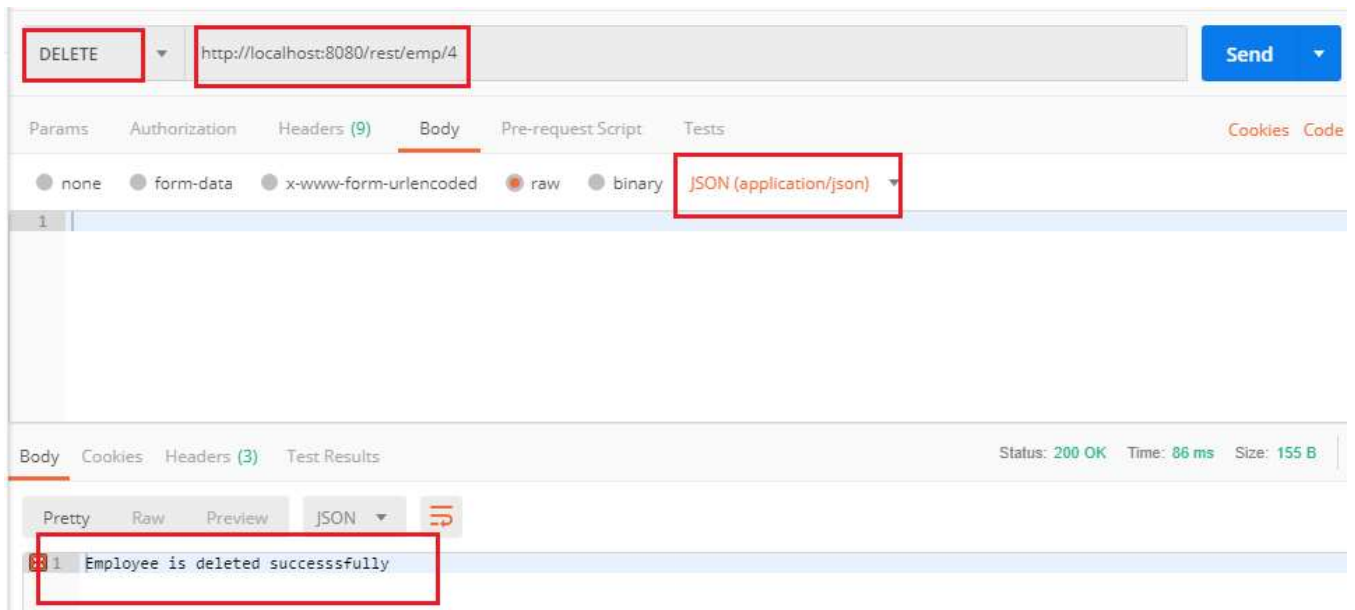
Pour POST :



Pour PUT :

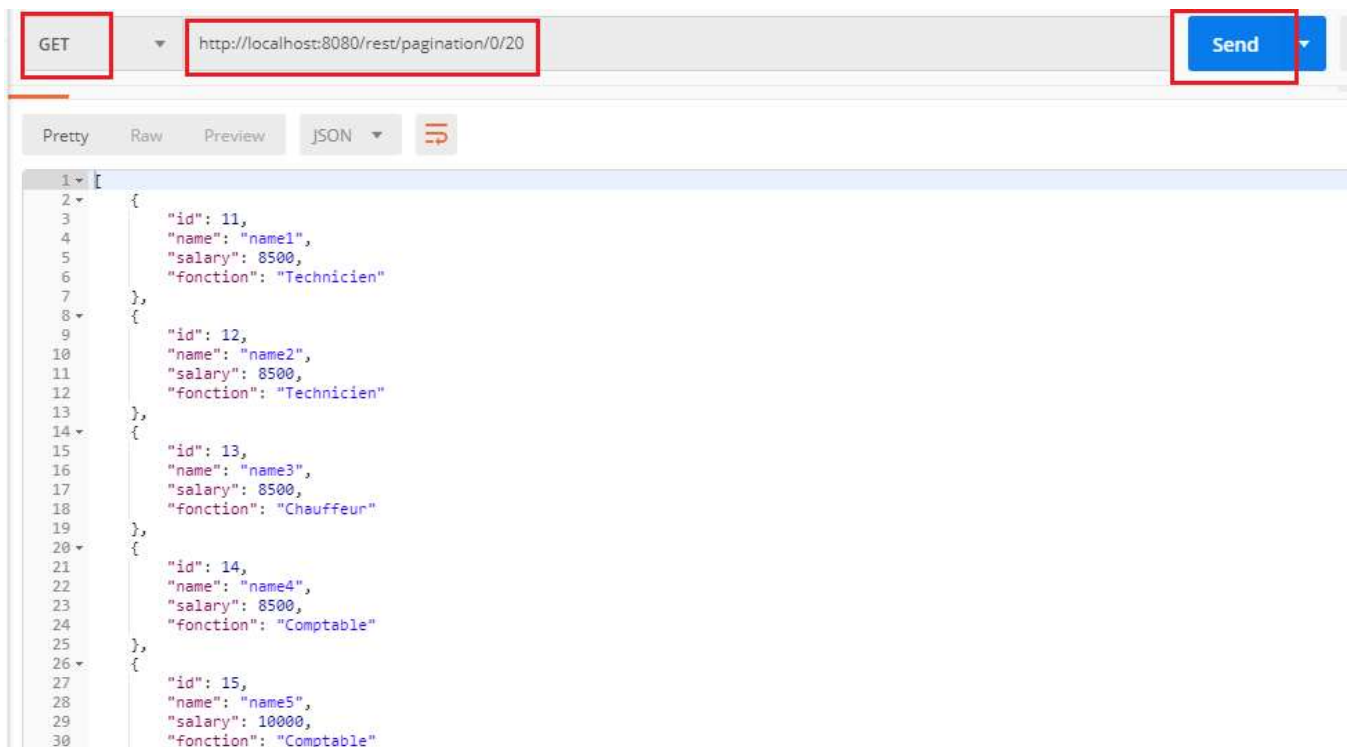


Pour DELETE :

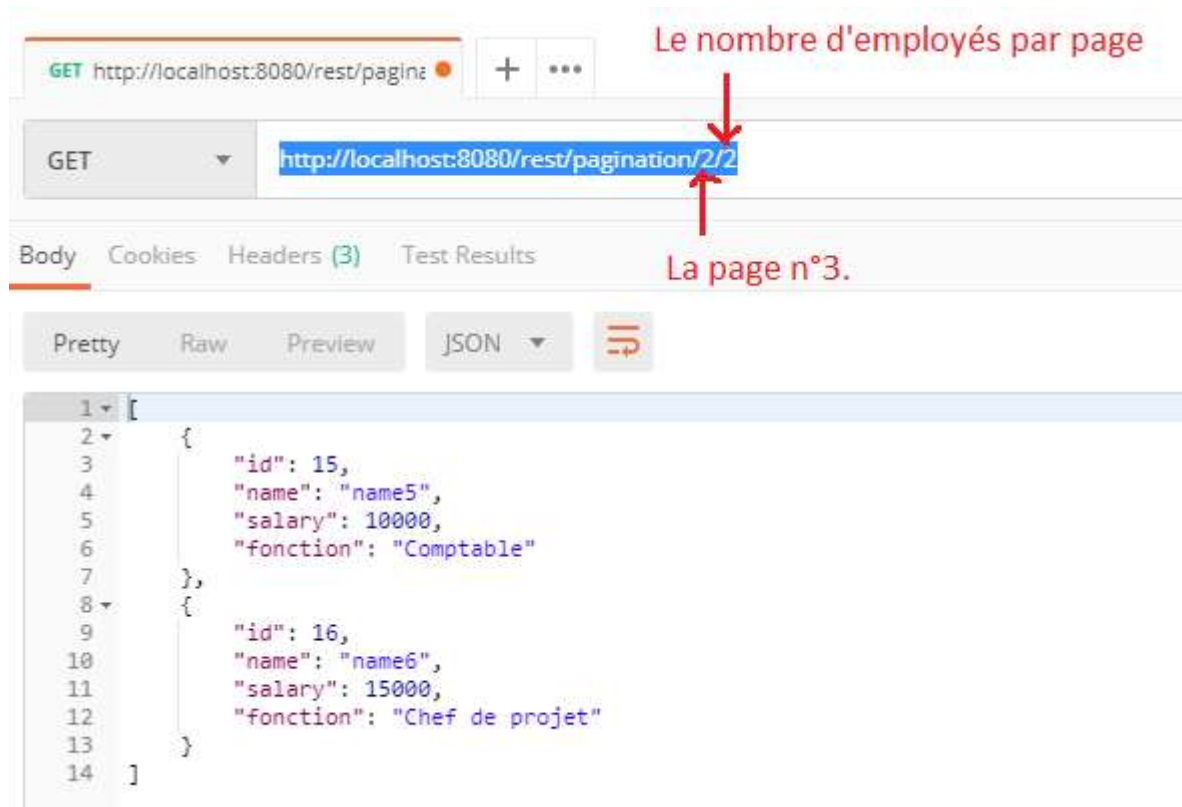


### Les tests de la pagination :

\*Lancer le lien <http://localhost:8080/rest/pagination/0/20>, l'application affichera la première page avec comme nombre d'employé est 20. Le résultat est :



\*Lancer le lien <http://localhost:8080/rest/pagination/2/2>, le résultat est ;



### Les tests concernant le tri :

Lancer le lien <http://localhost:8080/rest/sort/name>, l'application affichera la liste des employés trié par nom :

GET http://localhost:8080/rest/sort/name

Le tri dans cet exemple est effectué par name.

Body Cookies Headers (3) Test Results

Pretty Raw Preview JSON

```
1 [
2   {
3     "id": 11,
4     "name": "alami",
5     "salary": 40000,
6     "fonction": "Directeur"
7   },
8   {
9     "id": 12,
10    "name": "name2",
11    "salary": 8500,
12    "fonction": "Technicien"
13  },
14  {
15    "id": 13,
16    "name": "name3",
17    "salary": 8500,
18    "fonction": "Chauffeur"
19  },
20  {
21    "id": 14,
22    "name": "name4",
23    "salary": 8500,
24    "fonction": "Comptable"
25  }
26 ]
```

- Lancer le lien <http://localhost:8080/rest/sort/salary>, le résultat est :

GET http://localhost:8080/rest/sort/salary

Body Cookies Headers (3) Test Results

Pretty Raw Preview JSON

```
21      "id": 15,  
22      "name": "name5",  
23      "salary": 10000,  
24      "fonction": "Comptable"  
25    },  
26    {  
27      "id": 18,  
28      "name": "name8",  
29      "salary": 10000,  
30      "fonction": "Comptable"  
31    },  
32    {  
33      "id": 16,  
34      "name": "name6",  
35      "salary": 15000,  
36      "fonction": "Chef de projet"  
37    },  
38    {  
39      "id": 17,  
40      "name": "name7",  
41      "salary": 17500,  
42      "fonction": "Responsable du service"  
43    },  
44    {  
45      "id": 11,  
46      "name": "alami",  
47      "salary": 40000,
```



Le tri est effectué par salaire.