

# TP: Catalogue Produits

MVC Coté Client **AngularJS, BootStrap, HTML**

Framework **SPRING**

SGBD **MySQL**

Web Services **RestFul - SOAP**

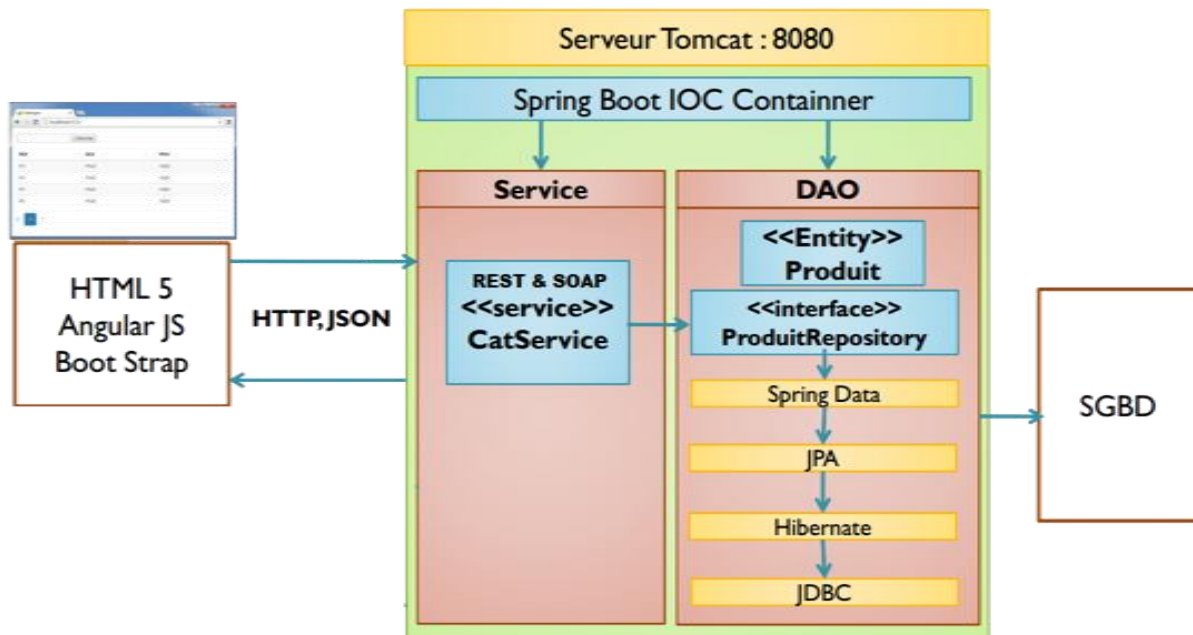


Par : **MOUSTAID Ayoub**

Enseignant : **M. Yousfi**

# Structure

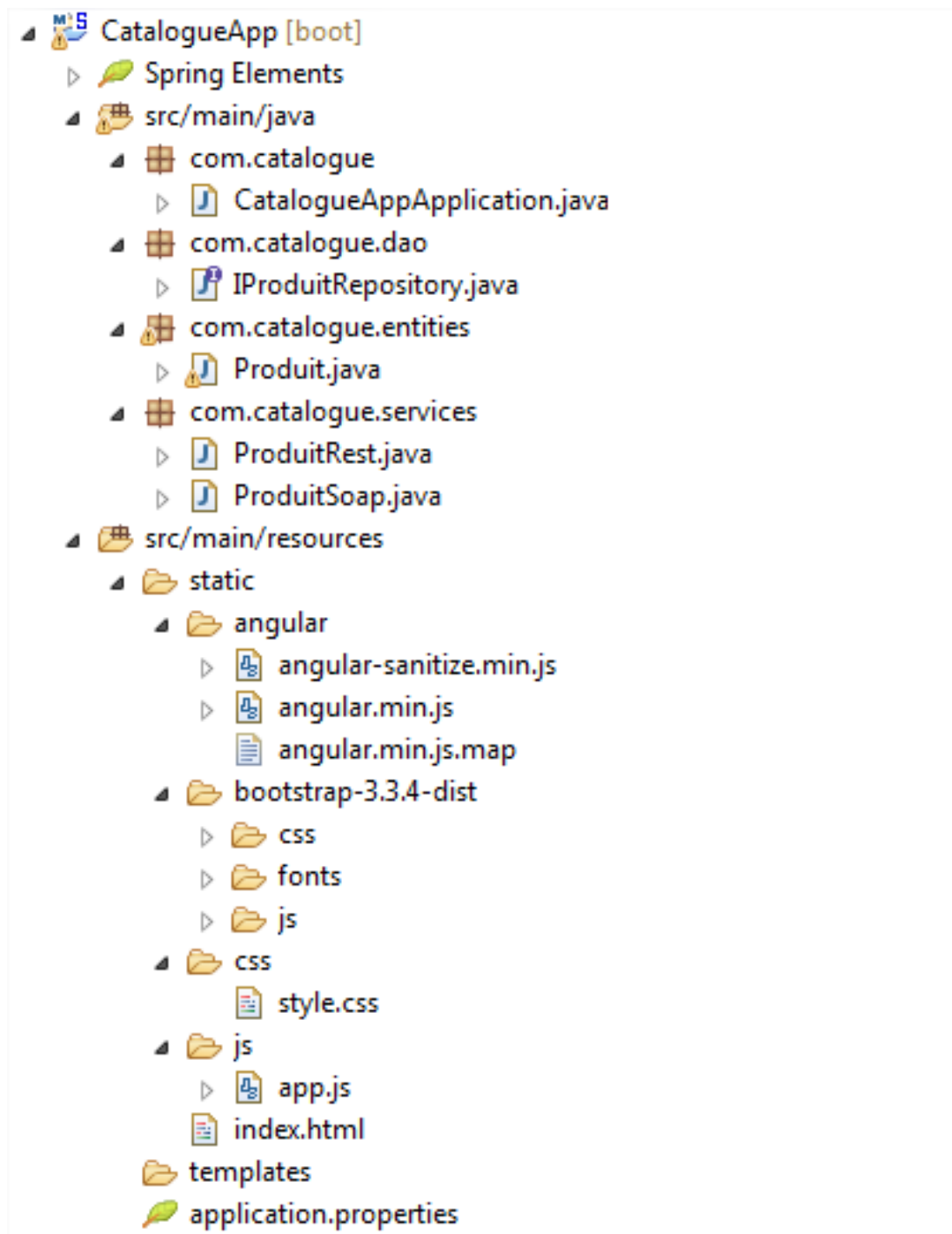
- Schema TP



- Structure Base Données

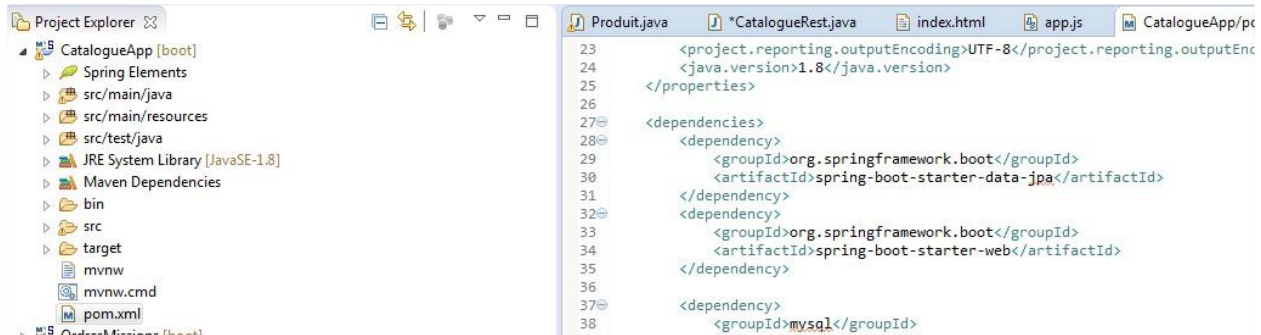
#	Colonne	Type	Interclassement	Attributs	Null	Défaut	Extra	Action
1	<u>reference</u>	bigint(20)			Non	Aucune	AUTO_INCREMENT	Modifier  Supprimer plus ▼
2	designation	varchar(255)	latin1_swedish_ci		Oui	NULL		Modifier  Supprimer plus ▼
3	prix	double			Non	Aucune		Modifier  Supprimer plus ▼

- IDE : Eclipse - Architecture



# Implémentation

## 1. Création projet Spring starter avec les dépendance WEB, MYSQL, JPA



## 2. Création Entité Produit

- **Produit.java**

```
package com.catalogue.entities;
```

```
import java.io.Serializable;
```

```
import javax.persistence.Entity;
```

```
import javax.persistence.GeneratedValue;
```

```
import javax.persistence.GenerationType;
```

```
import javax.persistence.Id;
```

```
@Entity
```

```
public class Produit implements Serializable {
```

```
    @Id
```

```
    @GeneratedValue(strategy = GenerationType.IDENTITY)
```

```
    private Long reference;
```

```
    private String designation;
```

```
    private double prix;
```

```
    public Long getReference() {
```

```
        return reference;
```

```
    }
```

```
    public void setReference(Long reference) {
```

```
        this.reference = reference;
```

```
}

    public String getDesignation() {
        return designation;
    }

    public void setDesignation(String designation) {
        this.designation = designation;
    }

    public double getPrix() {
        return prix;
    }

    public void setPrix(double prix) {
        this.prix = prix;
    }

    public Produit() {
        super();
        // TODO Auto-generated constructor stub
    }

    public Produit(String designation, double prix) {
        super();
        this.designation = designation;
        this.prix = prix;
    }
}
```

### 3. Configuration Data source

- Application.properties

```
spring.datasource.url = jdbc:mysql://localhost:3306/db_cat_ms
spring.datasource.username = root
spring.datasource.password =
spring.datasource.driverClassName = com.mysql.jdbc.Driver
spring.jpa.database = MYSQL
spring.jpa.show-sql = true
spring.jpa.hibernate.ddl-auto = update
spring.jpa.hibernate.naming-strategy = org.hibernate.cfg.ImprovedNamingStrategy
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect
```

### 4. Création interface JPA Repository basé sur Spring Data

- IProduitRepository.java

```
package com.catalogue.dao;

import java.util.List;

import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import org.springframework.data.repository.query.Param;

import com.catalogue.entities.Produit;

public interface IProduitRepository extends JpaRepository<Produit, Long> {
    @Query("select p from Produit p where p.designation like %:x%")
    public Page<Produit> produitParMC(@Param("x")String mc,Pageable p);

    public List<Produit> findByDesignation(String des);
    public Page<Produit> findByDesignation(String des,Pageable p);
}
```

## 5. Création Service Restful

- **ProduitRest.java**

```
package com.catalogue.services;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
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.catalogue.dao.IProduitRepository;
import com.catalogue.entities.Produit;

@RestController
@RequestMapping("/produits")
public class ProduitRest {
    @Autowired
    private IProduitRepository produitRepository;

    @RequestMapping(method = RequestMethod.POST)
    public Produit saveProduit(Produit p) {
        produitRepository.save(p);
        return p;
    }

    @RequestMapping(method = RequestMethod.GET)
    public List<Produit> getProduits() {
        return produitRepository.findAll();
    }

    @RequestMapping(value = "/{page}", method = RequestMethod.GET)
    public Page<Produit> getProduits(@PathVariable int page) {
        return produitRepository.findAll(new PageRequest(page, 5));
    }

    @RequestMapping(value = "/produitsParMC", method = RequestMethod.GET)
    public Page<Produit> getProduits(String mc, int page) {
        return produitRepository.produitParMC(mc, new PageRequest(page, 3));
    }
}
```

```

@RequestMapping("/findone/{ref}")
public Produit getProduit(@PathVariable Long ref) {
    return produitRepository.findOne(ref);
}

@RequestMapping(method = RequestMethod.DELETE)
public boolean delete(Long ref) {
    produitRepository.delete(ref);
    return true;
}
}

```

## 6. Test du MicroService avec POSTMAN

The screenshot shows the Postman interface. At the top, a GET request is configured to `localhost:8080/produits`. The 'Authorization' tab is selected, showing 'No Auth'. A 'Save Your Request' notification is visible on the right. Below the request configuration, the 'Body' tab is active, displaying a JSON array of product data in 'Pretty' format. The status bar at the bottom indicates a successful response with status 200 OK and a time of 2008 ms.

**Request:**

- Method: GET
- URL: localhost:8080/produits
- Authorization: No Auth

**Response (JSON):**

```

[
  {
    "reference": 1,
    "designation": "TDLSK",
    "prix": 1200
  },
  {
    "reference": 2,
    "designation": "prokiivc",
    "prix": 200
  },
  {
    "reference": 5,
    "designation": "P55E",
    "prix": 322
  },
  {
    "reference": 6,
    "designation": "PROD99",
    "prix": 900
  }
]

```



## 7. Création Application web basé sur AngularJS et Bootstrap (MVC Client)

- Index.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Catalogue Produits</title>
  <link rel="stylesheet" type="text/css" href="bootstrap-3.3.4-
dist/css/bootstrap.min.css"/>
  <link rel="stylesheet" type="text/css" href="css/style.css"/>
</head>
<body ng-app="MyCat" ng-controller="CatController" >
  <div class="container spacer">
    <form>
      <label>Mot Clé :</label>
      <input type="text" ng-model="motCle">
      <button ng-click="charger()">Chercher</button>
    </form>
  </div>
  <div class="container spacer">
    <table class="table table-striped">
      <thead>
        <tr>
          <th>REFERENCE</th><th>DESIGNATION</th><th>PRIX</th>
        </tr>
      </thead>
      <tbody>
        <tr ng-repeat="p in produits.content">
          <td>{{p.reference}}</td>
          <td>{{p.designation}}</td>
          <td>{{p.prix}}</td>
        </tr>
      </tbody>
    </table>
  </div>
  <div class="container">
    <ul class="nav nav-pills">
      <li ng-class="{active:$index==pageCourante}" class="clickable" ng-repeat="p in
pages track by $index">
        <a ng-click="gotoPage($index)">{{ $index }}</a>
      </li>
    </ul>
  </div>
  <script type="text/javascript" src="angular/angular.min.js"></script>
  <script type="text/javascript" src="js/app.js"></script>
</body>
</html>
```

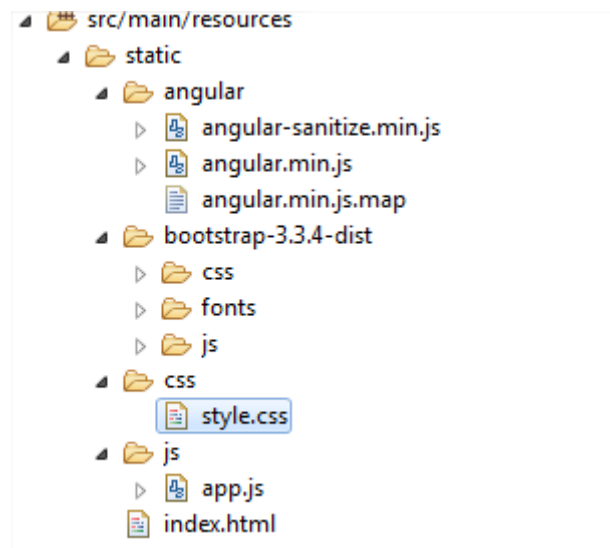
- **app.js**

```
var app=angular.module("MyCat",[]);
app.controller("CatController",function($scope,$http){
    $scope.produits=[];
    $http.get("produits/produits/0/")
    .success(function(data){
        $scope.produits = data;
        $scope.pages=new Array(data.totalPages);
    });

    $scope.motCle=null;
    $scope.pageCourante=0;
    $scope.charger=function(){
        $http.get("produits/produitsParMC?mc="+$scope.motCle+"&page="+$scope.pageCoura
nte)
        .success(function(data){
            $scope.produits=data;
            $scope.pages=new Array(data.totalPages);
        });
    };

    $scope.gotoPage=function(p){
        $scope.pageCourante=p;
        $scope.charger();
    };
});
```

- **Utilisation d'angularJS, bootstrap, css**



## 8. Création Service SOAP

- **ProduitSoap.java**

```
package com.catalogue.services;

import java.util.List;

import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebService;

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

import com.catalogue.dao.IProduitRepository;
import com.catalogue.entities.Produit;

@Component
@WebService
public class ProduitSoap {
    @Autowired
    private IProduitRepository produitRepository;
    @WebMethod(operationName="saveProduit")
    public Produit saveProduit(@WebParam(name="des")String des,
                               @WebParam(name="prix")double prix){
        Produit p=new Produit(des,prix);
        produitRepository.save(p);
        return p;
    }
    @WebMethod
    public List<Produit> listProduits(){
        return produitRepository.findAll();
    }
    @WebMethod
    public Produit getProduit(@WebParam(name="ref")Long ref){
        return produitRepository.findOne(ref);
    }
}
}
```

## 9. Déploiement Service SOAP

- **Configuration.java**

Le déploiement en utilisant une classe de configuration java est une alternative pour le déploiement avec un fichier de beans en xml

```
package com.catalogue.services;

import org.springframework.context.annotation.Bean;
import org.springframework.remoting.jaxws.SimpleJaxWsServiceExporter;

@Configuration
public class Configuration {

    @Bean
    public SimpleJaxWsServiceExporter getExporter(){
        SimpleJaxWsServiceExporter serviceExporter = new SimpleJaxWsServiceExporter();
        serviceExporter.setBaseAddress("http://localhost:9090/produits");
        return serviceExporter;
    }
}
```

## 10. Teste Déploiement Service SOAP

The screenshot shows a web browser window with the address bar set to `localhost:9090/produits?wsdl`. The browser has tabs for 'GET', 'Params', 'Send', and 'Save'. Below the address bar, there are tabs for 'Authorization', 'Headers', 'Body', 'Pre-request Script', and 'Tests'. The 'Body' tab is selected, showing the WSDL content. The status bar at the bottom indicates 'Status: 200 OK' and 'Time: 469 ms'. The WSDL content is displayed in a 'Pretty' view, showing the service definition for 'ProduitSoapService'.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Published by JAX-WS RI (http://jax-ws.java.net). RI's version is JAX-WS RI 2.2.9-b130926.1035 svn
-revision#5f6196f2b90e9460065a4c2f4e30e065b245e51e. -->
<!-- Generated by JAX-WS RI (http://jax-ws.java.net). RI's version is JAX-WS RI 2.2.9-b130926.1035 svn
-revision#5f6196f2b90e9460065a4c2f4e30e065b245e51e. -->
<definitions xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd" xmlns:wsp="http://www.w3.org/ns
/ws-policy" xmlns:wsp1_2="http://schemas.xmlsoap.org/ws/2004/09/policy" xmlns:wsam="http://www.w3.org/2007/05/addressing/metadata" xmlns
:soap="http://schemas.xmlsoap.org/wsdl/soap/" xmlns:tns="http://services.catalogue.com/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns="http://schemas.xmlsoap.org/wsdl/" targetNamespace="http://services.catalogue.com" name="ProduitSoapService">
  <types>
    <xsd:schema>
      <xsd:import namespace="http://services.catalogue.com/" schemaLocation="http://localhost:9090/produits?xsd=1"/><xsd:import>
    </xsd:schema>
  </types>
  <message name="getProduit">
    <part name="parameters" element="tns:getProduit"/></part>
  </message>
  <message name="getProduitResponse">
    <part name="parameters" element="tns:getProduitResponse"/></part>
  </message>
  <message name="saveProduit">
    <part name="parameters" element="tns:saveProduit"/></part>
```

# Interface Client web

localhost:8080

Mot Clé:

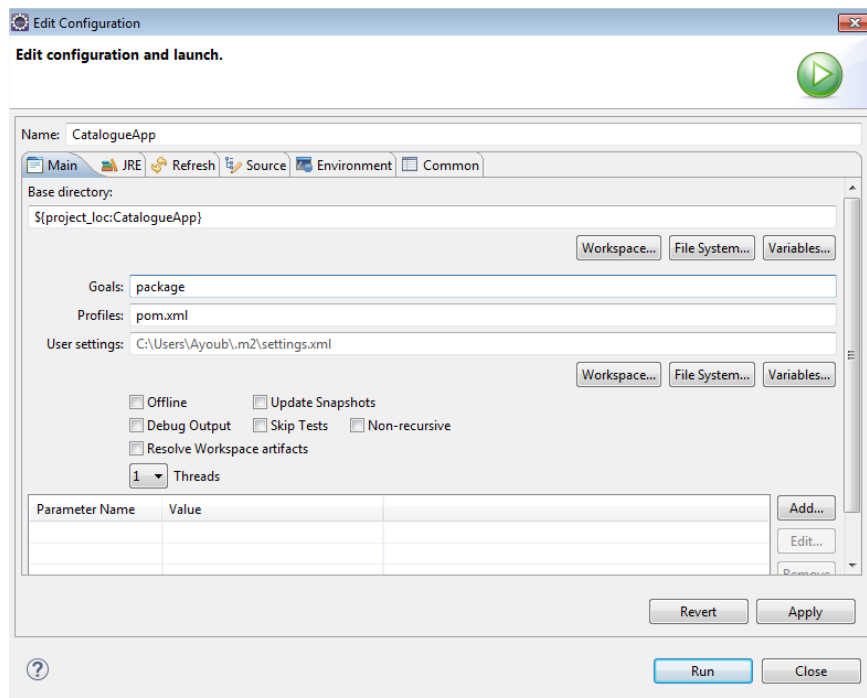
REF	DES	PRIX
1	TDLSK	1200

0

## Déploiement

- **Déploiement en .jar**

On va utiliser la commande 'package' pour que maven compile et effectue s'il y a des testes et génère le fichier .jar dans le dossier target



Pour générer un fichier .war de l'application on **mentionne** l'extension war au lieu du jar et on supprime la dépendance Spring starter web depuis le fichier de dépendances '**pom.xml**'.

L'application aura besoin d'une classe qui va remplacer le fichier web.xml, donc on crée la classe '**ApplicationInitializer**' qui hérite de la classe **SpringBootServletInitializer** et redéfinit la méthode '**Configure**', ensuite il ne reste que l'exécution de l'application sur un serveur, (Tomcat dans notre cas)

- **ApplicationInitialiser.java**

```
public class ApplicationInitializer extends SpringBootServletInitializer {  
    @Override  
    protected SpringApplicationBuilder configure(SpringApplicationBuilder builder)  
    {  
        return builder.sources(CatalogueApplication.class);  
    }  
}
```