ENSET MOHAMMEDIA

GLSID 3

TP: Catalogue Produits

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

Framework **SPRING**

SGBD **MySQL**

Web Services **RestFul - SOAP**

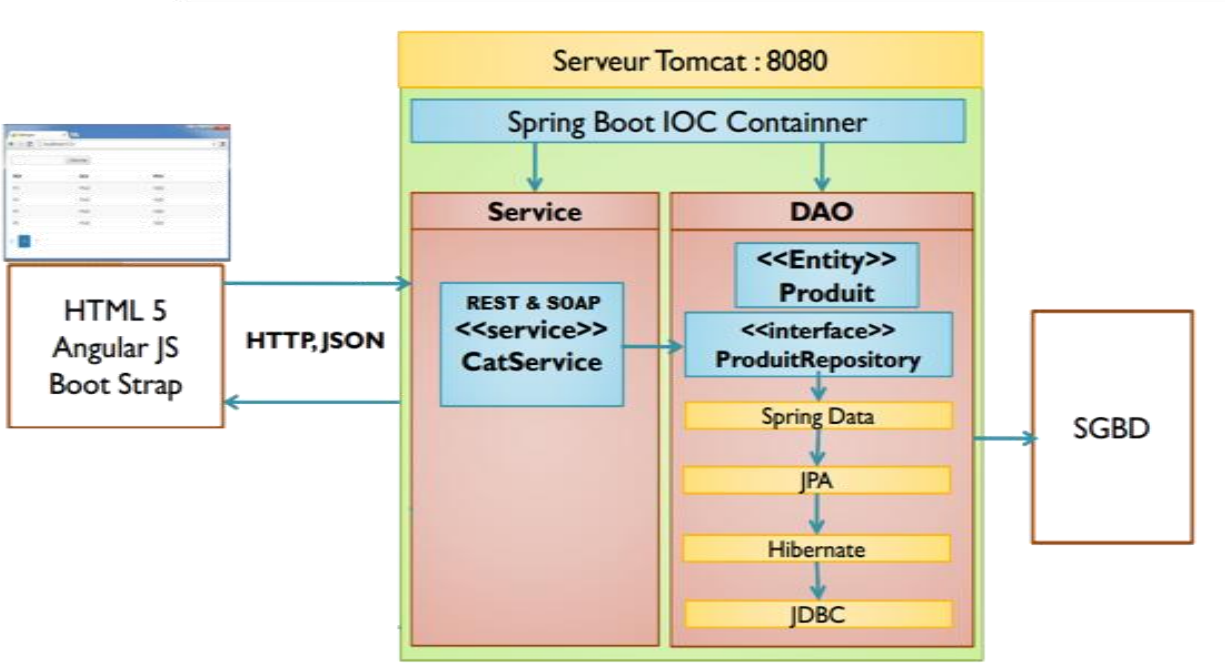
Par : **MOUSTAID Ayoub**

Enseignant : **M. Yousfi**

Année Universitaire **2016/2017**

Structure

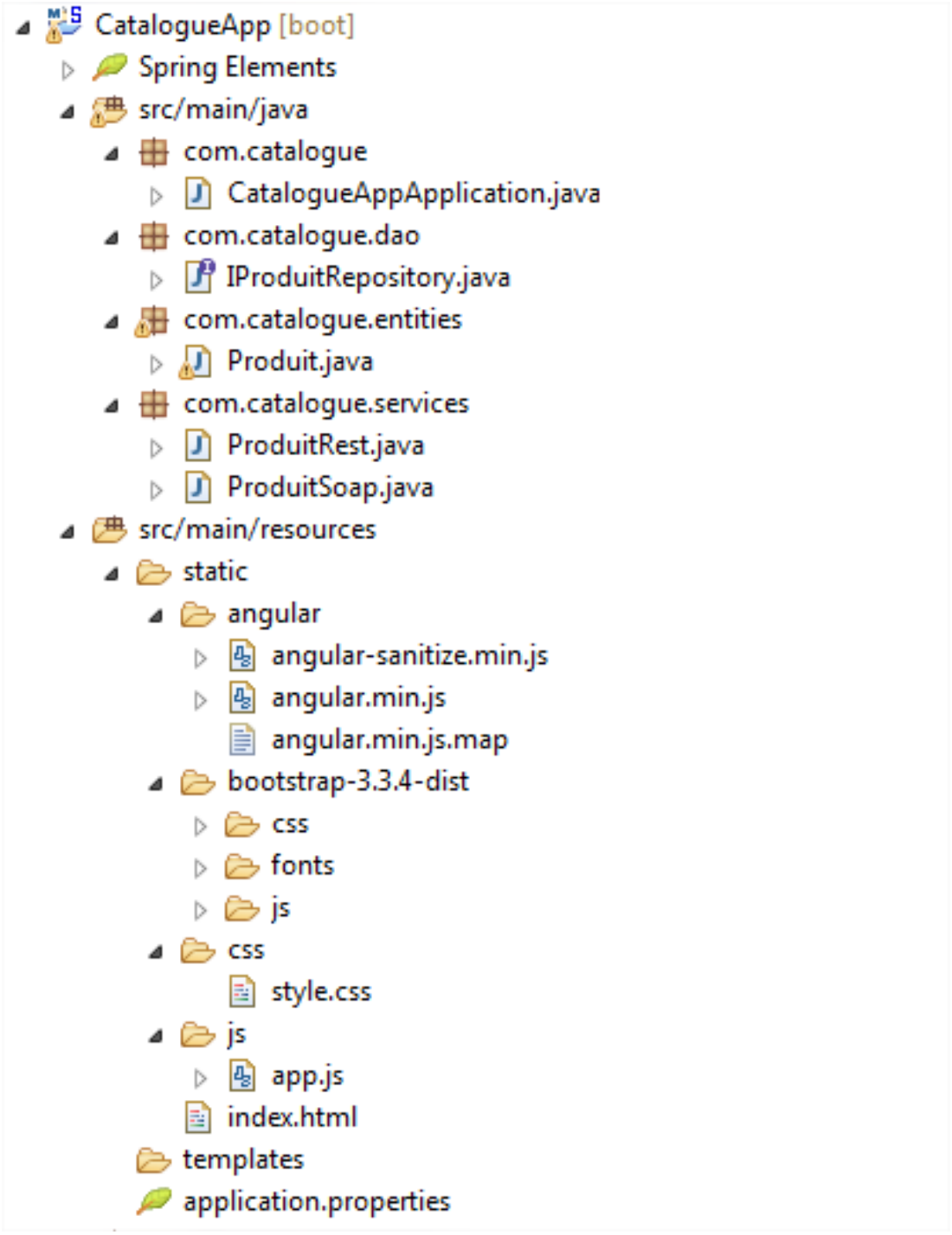
- **Schema TP**



- **Structure Base Données**

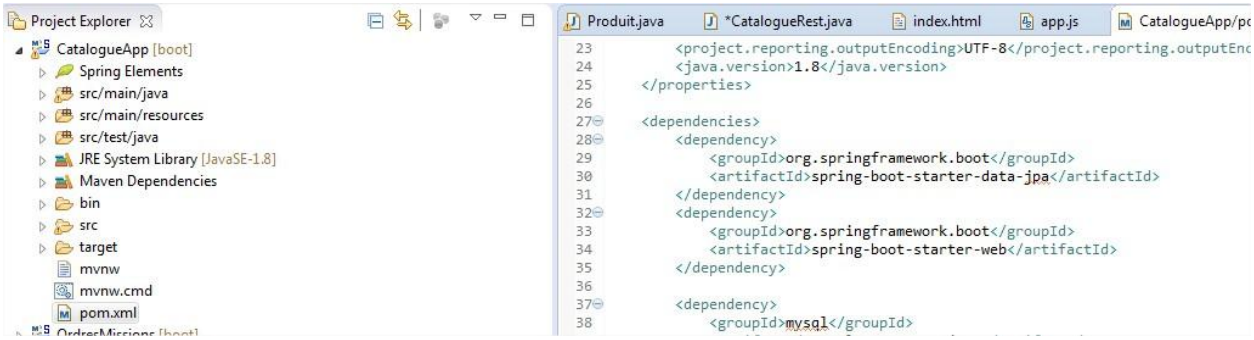
#	Colonne	Type	Interclassement	Attributs	Null	Défaut	Extra	Action
1	reference	bigint(20)			Non	*Aucune*	AUTO_INCREMENT	Modifier Supprimer plus
2	designation	varchar(255)	latin1_swedish_ci		Oui	*NULL*		Modifier Supprimer plus
3	prix	double			Non	*Aucune*		Modifier Supprimer plus

- **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);
}
```

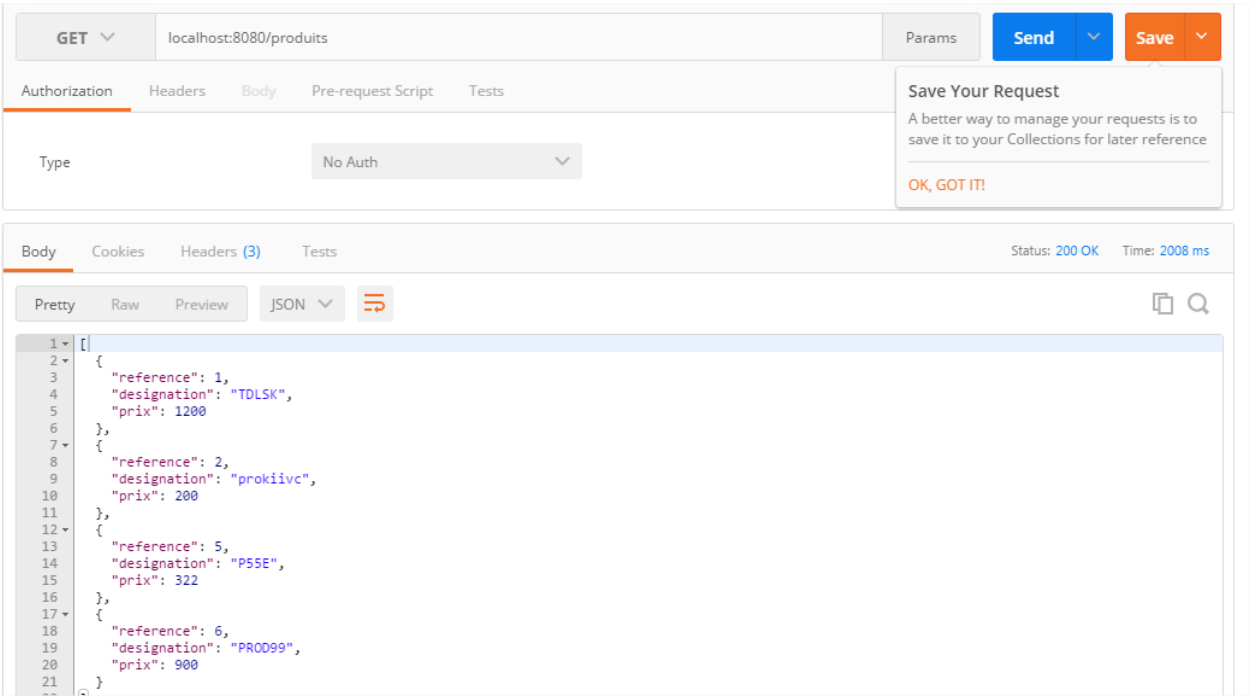
5. Création Service Restful

- **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



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

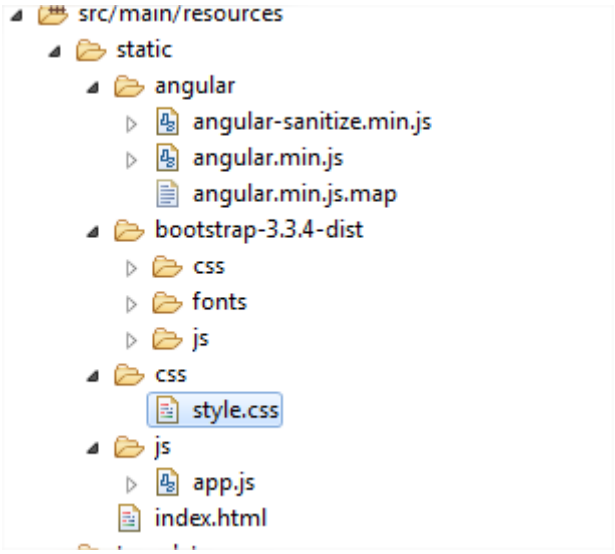
- **Index.html**

```
<!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);

	}


}

}
```

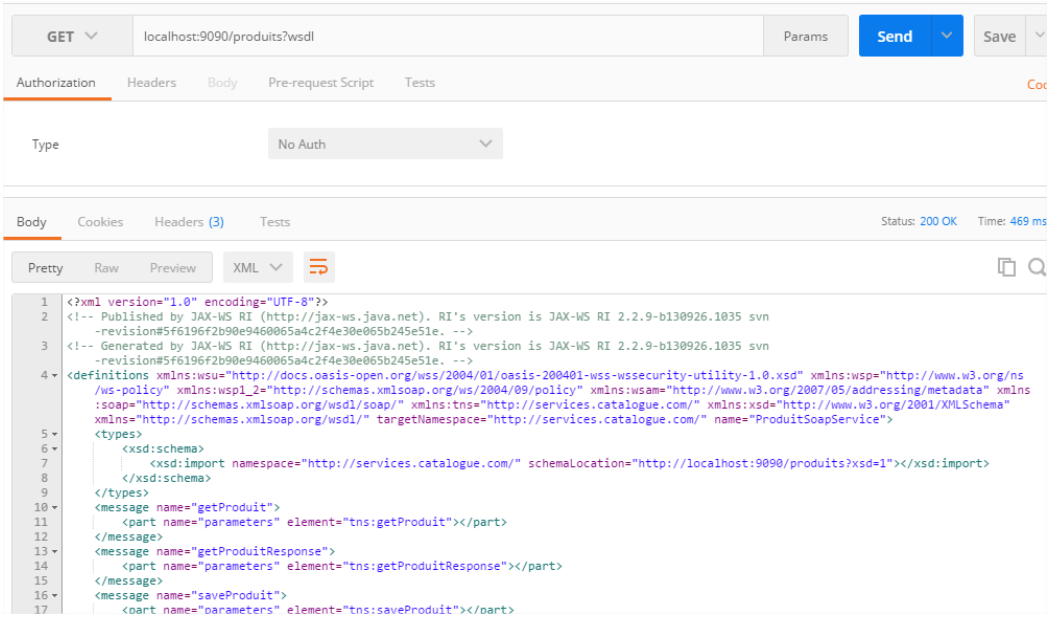
9. Déploiement Service SOAP

- **Configuration.java**

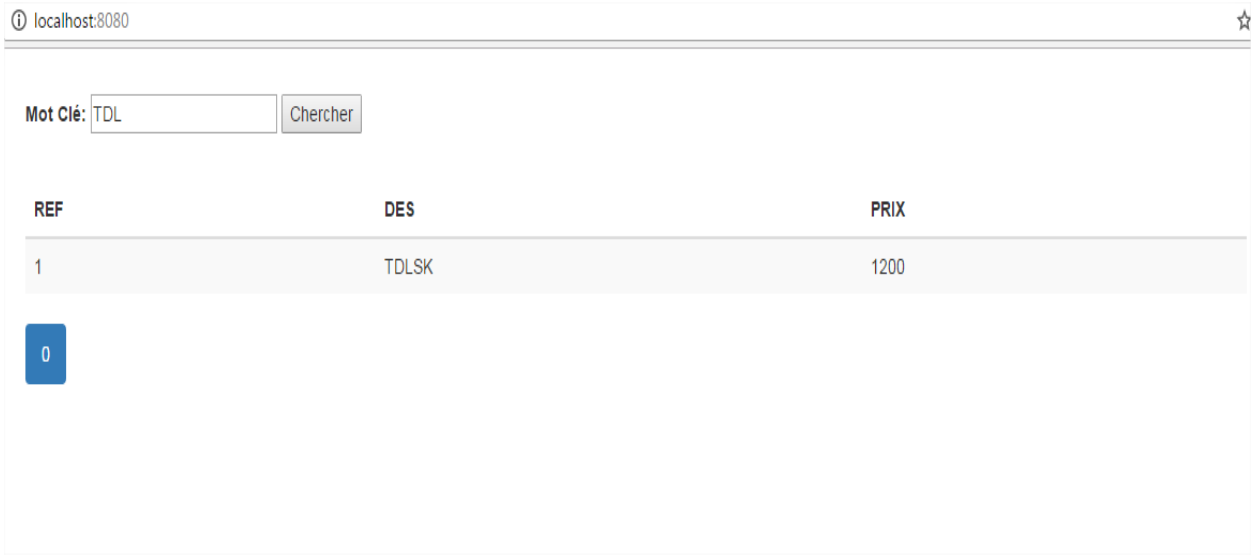
Le déploiement en utilisant une classe de configuration java est une alternative pour le déploiement avec un fichier de beans en xml

```
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



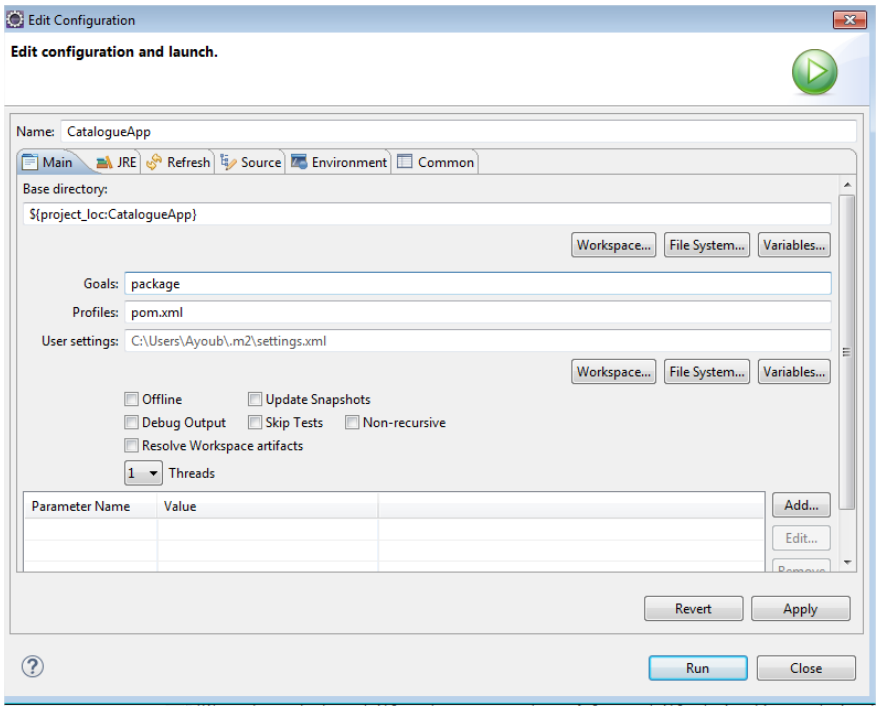
Interface Client web



Déploiement

- **Déploiement en .jar**

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



Pour génerer un fichier .war de l'application on **mentionne** l'extension war au lieu du jar et on supprimer la dépendence Spring starter web depuis le fichier de dépendences **'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);
   }
}
```