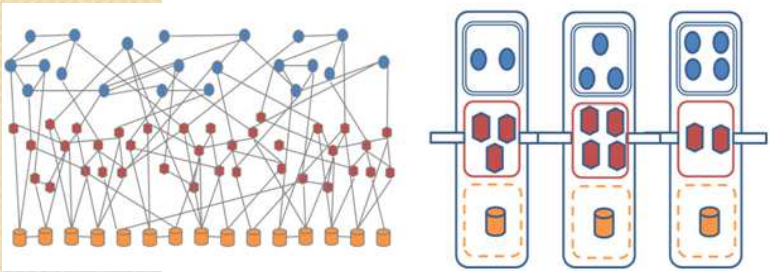
Architectures Logicielles Distribuées basées sur les Micro-Services



Exemple d'implémentation avec:

- Spring Boot
- AngulatrJS
- BootStrap

https://www.youtube.com/watch?v=zBLXWIhrg7U

Mohamed Youssfi

Laboratoire Signaux Systèmes Distribués et Intelligence Artificielle (SSDIA)

ENSET, Université Hassan II Casablanca, Maroc

Email : med@youssfi.net

Supports de cours : http://fr.slideshare.net/mohamedyoussfi9

Chaîne vidéo : http://youtube.com/mohamedYoussfi

Plan

- Evolution des approches de programmation
- Industrialisation du génie logiciel
- Exigences des systèmes logiciels
- Principe de de l'inversion de contrôle
- Exemples d'architectures logicielles modernes
- Problème des applications monolithiques
- Architectures logicielles basées sur les Micro-services
- Mise en œuvre des architectures basées sur les Micro-services : Spring Boot

Evolution des techniques de programmation

Langage Machine — Chaque famille de CPU possède son porophore jeu d'instructions

Assembleur — Langage bas niveau

Programmation Procédurale — Sous programmes: Procédures, fonctions (Basic, Pascal, C, Fortran,..)

Programmation Orientée Objet — Objet = Etat+ Comportement + Identité
Concepts fondamentaux : Objet, classe, Héritage, polymorphisme, encapsulation (C++, JAVA, C#, ..)

Programmation Orientée Objet Distribués — Objets distribués sur plusieurs machines
Middlewares : (RMI, CORBA, JMS, …)

Programmation Orientée composants — Objets distribués, réutilisables, configurables, Interchangeables, évolutifs, mobiles, surveillable à chaud : Conteneur (EJB, Spring) : AOP

Programmation Orientée Services — Composant disponibles à d'autres applications distantes hétérogènes via des protocoles (http) transportant des données: XML, JSON => SOAP et REST

Programmation Orientée Agents — ? Service + Intelligence + Apprentissage+ …

Industrialisation du génie logiciel

- Le processus du développement logiciel est, aujourd'hui complètement industrialisé.
- Un logiciel est construit à base de composants
  - Réutilisables
  - Interchangeable
  - Évolutifs
  - Reconfigurables
  - Mobiles
  - Surveillables à chaud
  - …

Avènement des technologies Open Source

- En dix ans le développement logiciel a évolué en grande partie grâce aux technologies Open Sources.
- Celles ci permettent aux développeurs
  - de ne pas réinventer la roue
  - et de se concentrer plus sur les aspects métiers des applications
  - et utiliser les Framework pour résoudre les aspects techniques.
- Exemples :
  - Spring, Struts, Hibernate (Coté Serveur)
  - AngularJS, BootStrap (Coté Client)
  - RMI, CORBA, JMS, JAXWS, JAXRS (Systèmes Distribués)

Avènement des méthodologies Agile

- Les méthodes Agile de gestion projet et de développement comme Scrum ou l'eXtreme Programming (XP) partent du constat que le cycle de développement en cascade est un échec.
- Développement en cascade:
  - spécifier pendant X mois,
  - puis ensuite coder Y mois,
  - tester Z mois,
  - pour livrer au client un projet qui ne correspond plus tout à fait à ses attentes.

Avènement des méthodologies Agile

- Les méthodes Agile prônent
  - Les tests avant le développement (Test Driven Development : TDD)
  - Des cycles de développement itératifs,
  - L'implication du client final tout au long du projet,
  - Des spécifications réduites en début de projet,
  - etc.
- Les méthodologies Agile sont pragmatiques,
- Ont fait leurs preuves et sont prisées par de grands industriels de logiciels.

Industrialiser le cycle de vie : améliorations, maintenance, corrections

- Pour industrialiser les composants logiciels, il est nécessaire d'utiliser des outils qui permettent d'automatiser le processus de fabrication des logiciels
  - **Frameworks de tests** : JUnit
    - Automatiser les Tests Unitaires
  - **Outils d'intégration continue** : Maven
    - Ils jouent le rôle de chef d'orchestre pour piloter et automatiser le processus de développement logiciel :
      - Gérer les dépendances
      - Lancer la compilation des sources
      - Lancer les Test Unitaires
      - Générer les packages (jar, war,ear)
      - Installer les packages dans le repository
      - Déployer l'application dans le serveur
      - Générer la documentation du projet
      - …

Exigences d'un projet informatique

- Exigences fonctionnelles:
  - Une application est créée pour répondre , tout d'abord, aux besoins fonctionnels des entreprises.
- Exigences Techniques :
  - Les performances:
    - Montée en charge
    - Haute disponibilité et tolérance aux pannes
  - La maintenance:
    - **Fermée** à la modification et **Ouverte** à l'extension
  - Sécurité
  - Portabilité
  - Distribution
  - Capacité de communiquer avec d'autres applications distantes.
  - Capacité de fournir le service à différents type de clients (Desk TOP, Mobile, SMS, http…)
  - …..
- Exigences financières : Coût du logiciel

Comment faire surmonter toutes contraintes

- Il est très difficile de développer un système logiciel qui respecte ces exigences sans utiliser l'expérience des autres :
  - Bâtir l'application sur une architecture d'entreprise: (JEE,. Net)
  - **Framework pour l'Inversion de contrôle:**
    - Permettre au développeur de se concentrer sur le code métier (**Exigences fonctionnelles**)
    - Le Framework s'occupe du code Technique (**Exigence Technique**)
    - Grâce à la **Programmation Orientée Aspect (AOP)**
    - Exemple : Spring, EJB pour l'architecture JEE
  - Utiliser des modèles de conceptions confirmés : Design Patterns
  - Frameworks :
    - Mapping objet relationnel (ORM) : JPA, Hibernate, Toplink, …
    - Applications Web coté serveur : Struts, JSF, SpringMVC
    - Applications web coté client : AngylarJS, BootStrap
    - …
  - Middlewares pour les applications Distribuées :
    - RMI, CORBA : Applications distribuées
    - JAXWS pour Web services basés SOAP (HTTP+XML)
    - JAXRS pour les Web services RESTful (HTTP+(JSON/XML,…))
    - JMS : Communication asynchrone entre les application
    - …

Exemple : sans utiliser d'inversion de contrôle

```
public void virement(int c1, int c2, double mt) {
   /* Création d'une transaction */
   EntityTransaction transaction=entityManager.getTransaction();
   /* Démarrer la transaction */
   transaction.begin();
   try {
```

Code Technique

```
        /* Code métier */
        retirer(c1,mt);
        verser(c2,mt);
```

Code Métier

```
        /* Valider la transaction  */
        transaction.commit();
   } catch (Exception e) {
      /* Annuler la transaction en cas d'exception  */
        transaction.rollback();
        e.printStackTrace();
   }
}
```

Code Technique

Exemple : en utilisant l'inversion de contrôle

```
@Transactional
public void virement(int c1, int c2, double mt) {
        retirer(c1,mt);
        verser(c2,mt);
}
```

Code Métier

Ici, avec l'annotation @Transactional, nous avons délégué la gestion des transaction au conteneur Spring IOC

Evolution des techniques de programmation

Langage Machine — Chaque famille de CPU possède son porophore jeu d'instructions

Assembleur — Langage bas niveau

Programmation Procédurale — Sous programmes: Procédures, fonctions (Basic, Pascal, C, Fortran,..)

Programmation Orientée Objet — Objet = Etat+ Comportement + Identité
Concepts fondamentaux : Objet, classe, Héritage, polymorphisme, encapsulation (C++, JAVA, C#, ..)

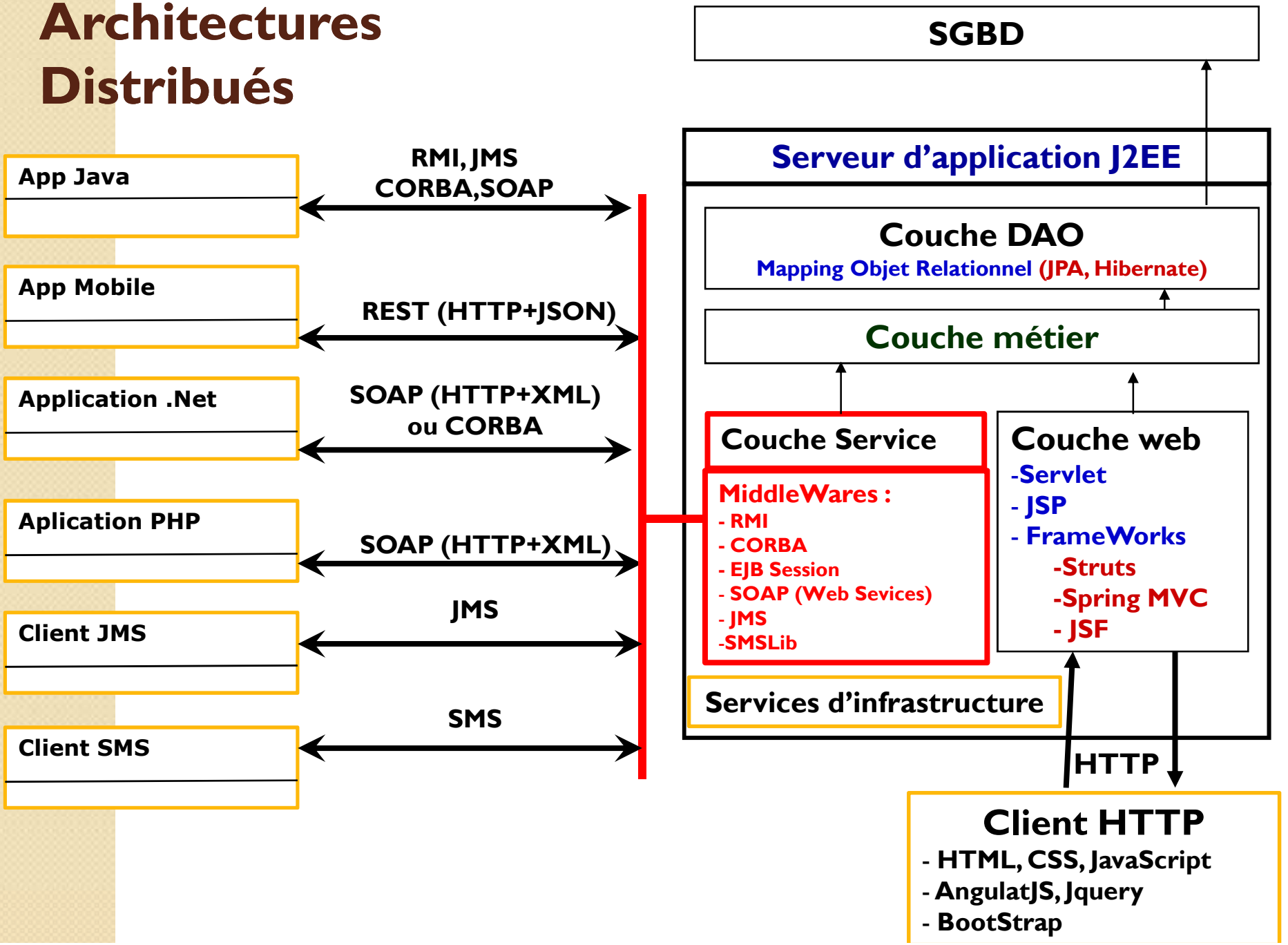
Programmation Orientée Objet Distribués — Objets distribués sur plusieurs machines
Middlewares : (RMI, CORBA, JMS, …)

Programmation Orientée composants — Objets distribués, réutilisables, configurables, Interchangeables, évolutifs, mobiles, surveillable à chaud : Conteneur (EJB, Spring) : AOP

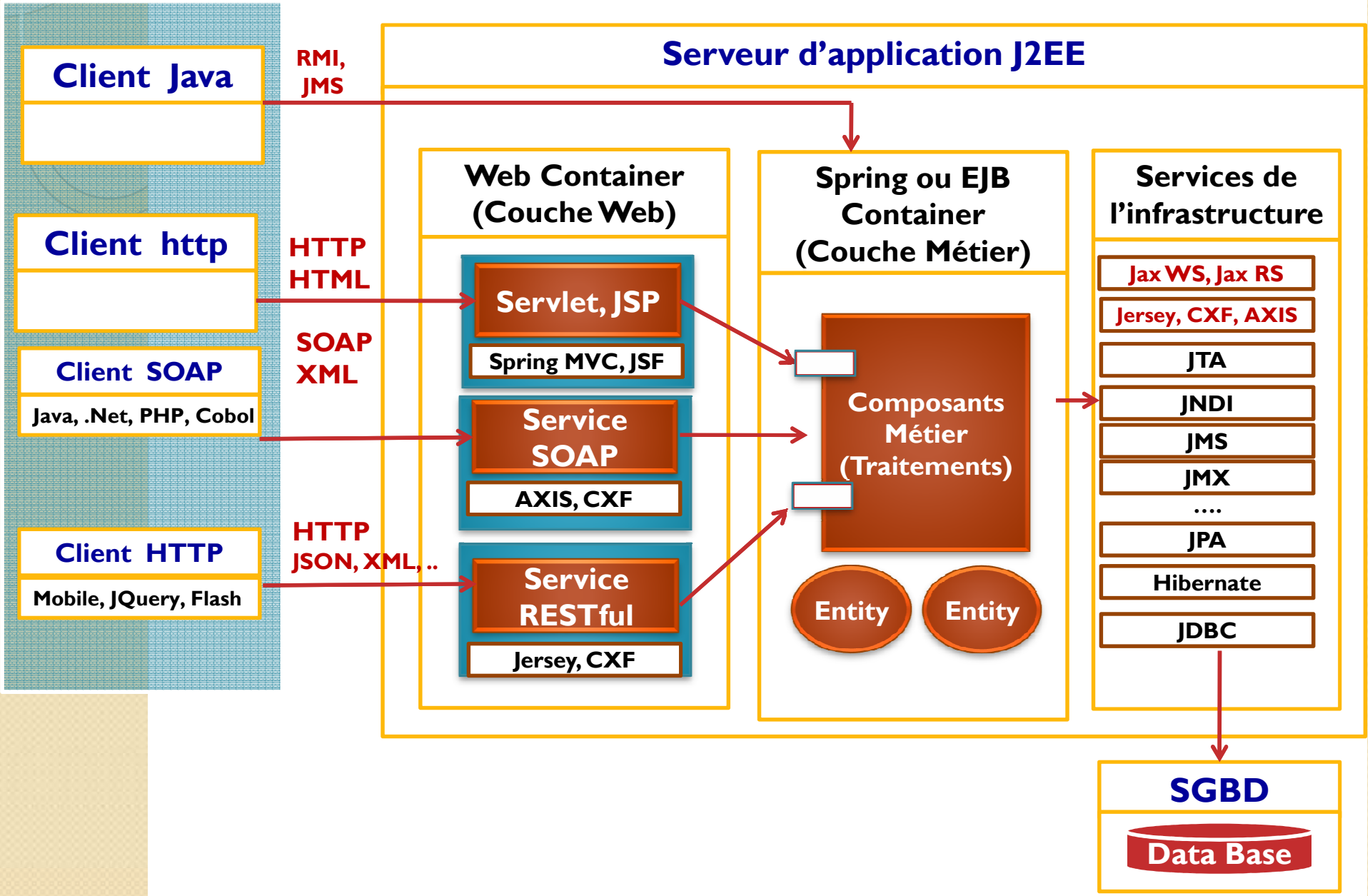
Programmation Orientée Services — Composant disponibles à d'autres applications distantes hétérogènes via des protocoles (http) transportant des données: XML, JSON => SOAP et REST

Programmation Orientée Agents — ? Service + Intelligence + Apprentissage+ …

Architectures Distribués

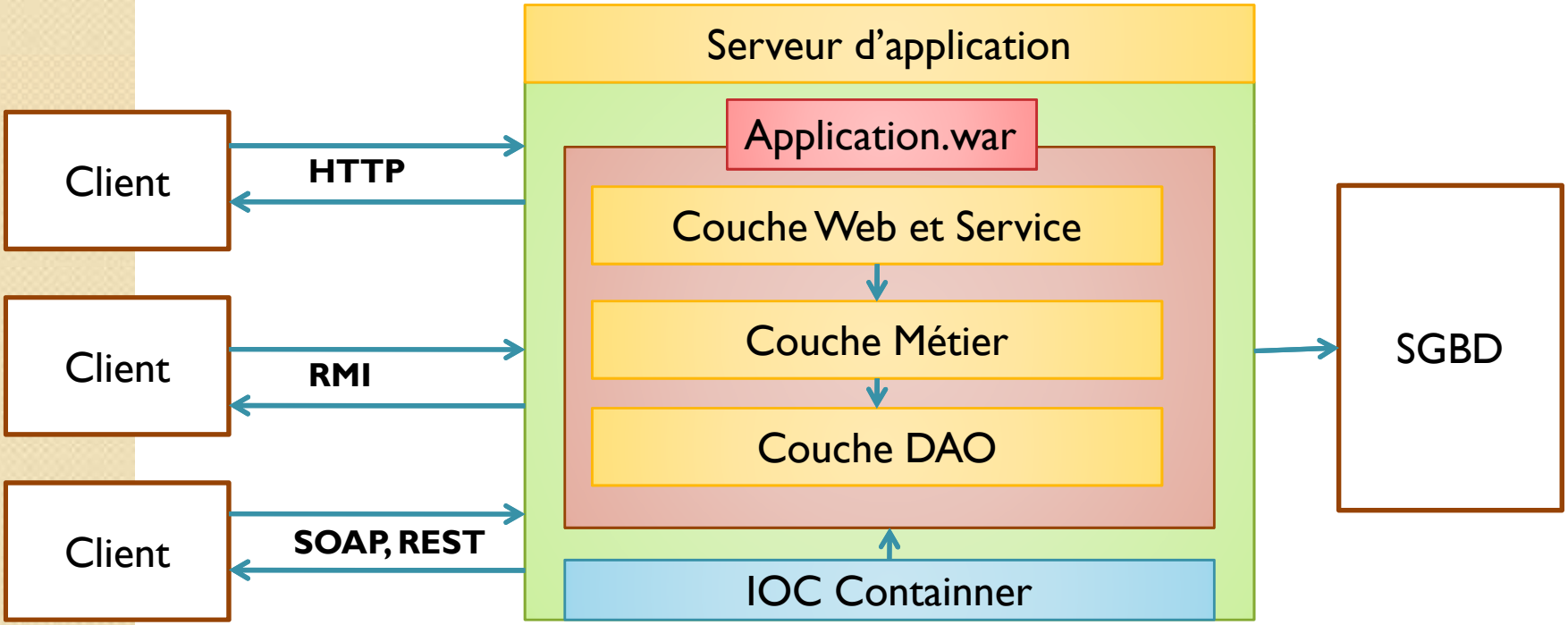


Architecture J2EE



De l'application monolithique aux architectures micro services

- Une application monolithique est une application qui est développée en un seul bloc (war, jar, Ear) et déployée d'une manière unitaire dans un serveur d'application



Problèmes des applications monolithiques

- Les principaux problème des applications monolithiques sont :
  - Elles centralisent tous les besoins fonctionnels
  - Elles sont réalisées dans une seule technologie.
  - Chaque modification nécessite de :
    - Tester les régressions
    - Redéployer toute l'application
  - Difficile à faire évoluer au niveau fonctionnel
  - Livraison en bloc (Le client attend beaucoup de temps pour commencer à voir les premières versions)

Les Micro services

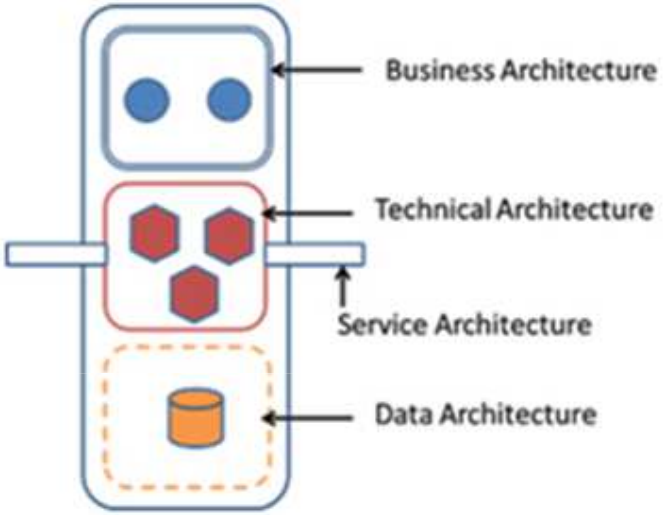
- Les micro services sont une approche d'architecture et de développement d'une **application composées de petits services**.
- L'idée étant de découper un grand problème en petites unités implémentée sous forme de micro-services
- Chaque service est responsable d'une fonctionnalité,
- Chaque micro-service est **développé**, **testé** et **déployé séparément** des autres.
- Chaque service **tourne dans un processus séparé**.
- La seule relation entre les différents micro services est l'échange de données effectué à travers les différentes APIs qu'ils exposent. (SOAP, REST, RMI, CORBA, JMS,…)
- Lorsqu'on **les combinent**, ces micro services peuvent **réaliser des opérations très complexes**.

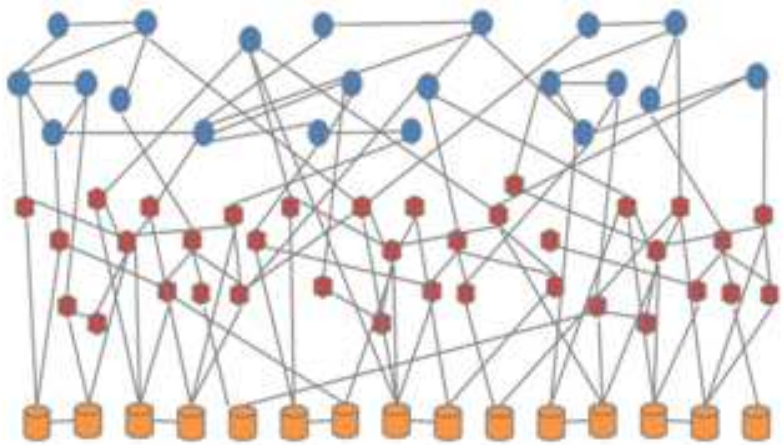
Aspects clef des micro services

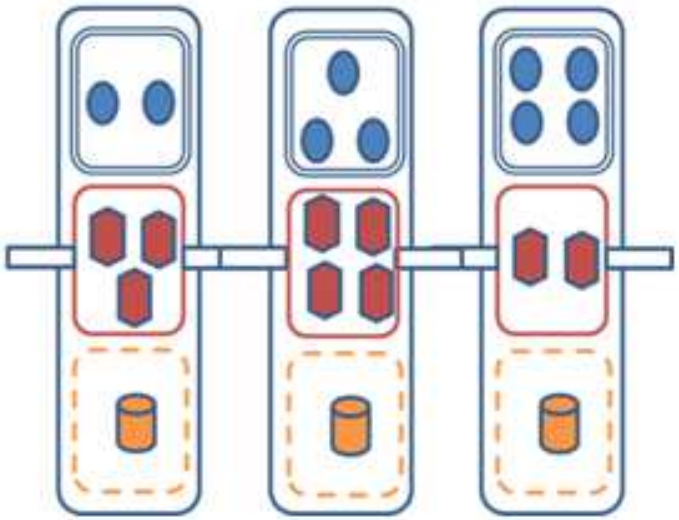
- Chaque micro service peut être conçu à l'aide de n'importe quel outil et développé avec n'importe quel langage et technologie.
- Ils sont faiblement couplés puisque chaque micro service est physiquement séparé des autres,
- Indépendance relative entre les différentes équipes qui développement les différents micro services (en partant du principe que les APIs qu'ils exposent sont définis à l'avance).
- Facilité des tests et du déploiement ou de la livraison continue.

Micro service

- Un micro service combine les trois couches « métier, technique et données » en une seule, accessible via des API.







MISE EN ŒUVRE DES ARCHITECTURES BASÉES SUR LES MICRO-SERVICES

Spring Boot

- Spring Boot est un Framework qui permet de créer des applications basées sur des micro services.
- Atouts de Spring Boot :
  - Faciliter le développement d'applications complexes.
  - Faciliter à l'extrême l'injection des dépendances
  - Réduire à l'extrême les fichier de configurations
  - Faciliter la gestion des dépendances Maven.
  - Auto Configuration : la plupart des beans sont créés si le ou les jar(s) adéquats sont dans le classpath.
  - Fournir un conteneur de servlet embarqué (Tomcat, Jetty)
  - Créer une application autonome (jar ou war)

EXEMPLE D'APPLICATION SPRING BOOT

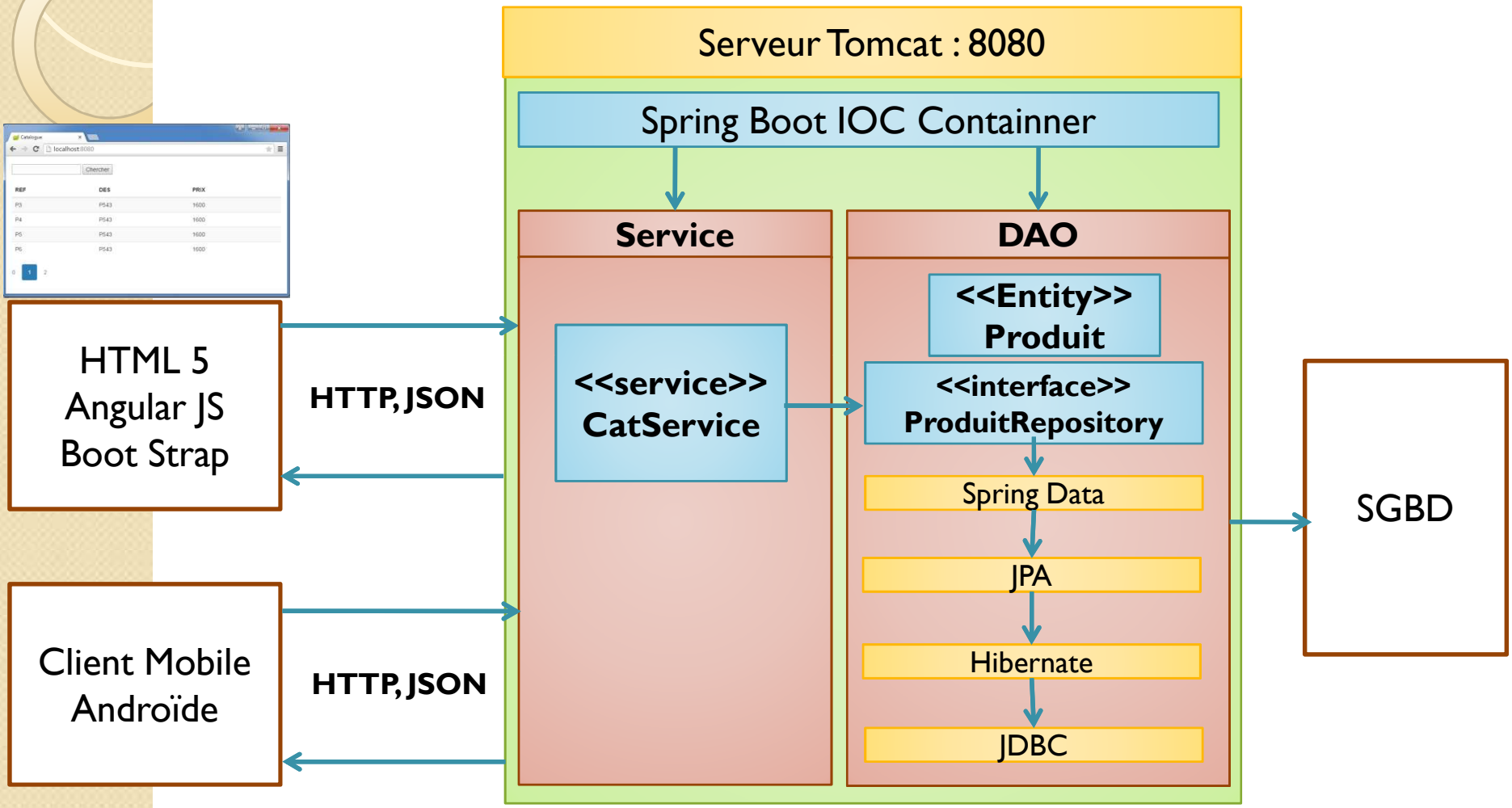
https://www.youtube.com/watch?v=zBLXWIhrg7U

med@youssfi.net | ENSET Université Hassan II de Casablanca

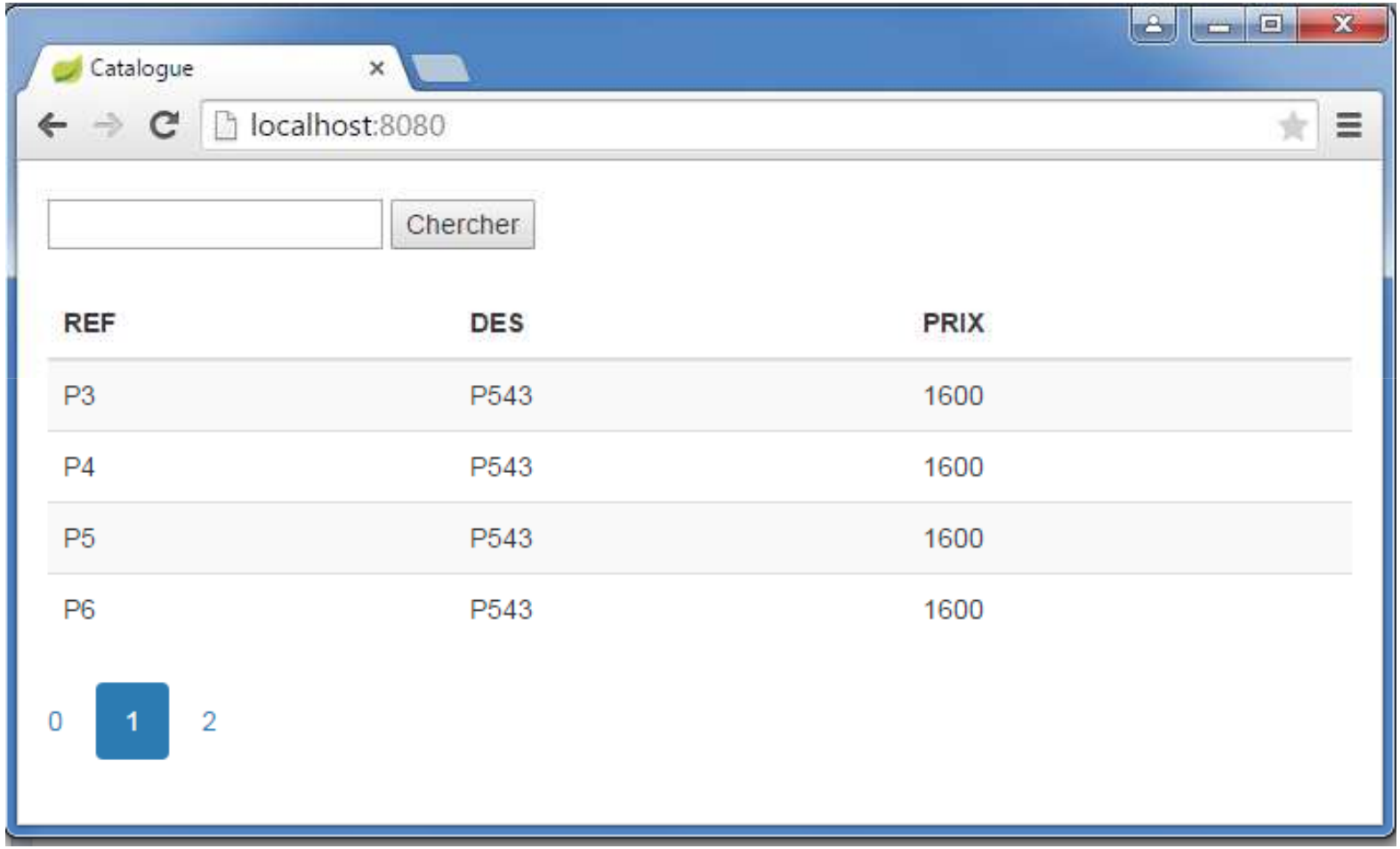
Premier Exemple d'application

- On souhaite créer une application qui permet de gérer des produits.
- Chaque produit est défini par :
  - Sa référence de type String
  - Sa désignation de type String
  - Son prix
- L'applications de permettre de :
  - Ajouter de nouveaux produits
  - Consulter les produits
  - Chercher les produits par mot clé
  - Consulter un produit
  - Mettre à jour un produit
  - Supprimer un produit
- Les données sont stockées dans une base de données MySQL
- L'application est un service Restful basée sur Spring Boot
- La couche web respecte MVC coté Client et basée sur :
  - HTML5
  - CSS, Bootstrap
  - Angular JS

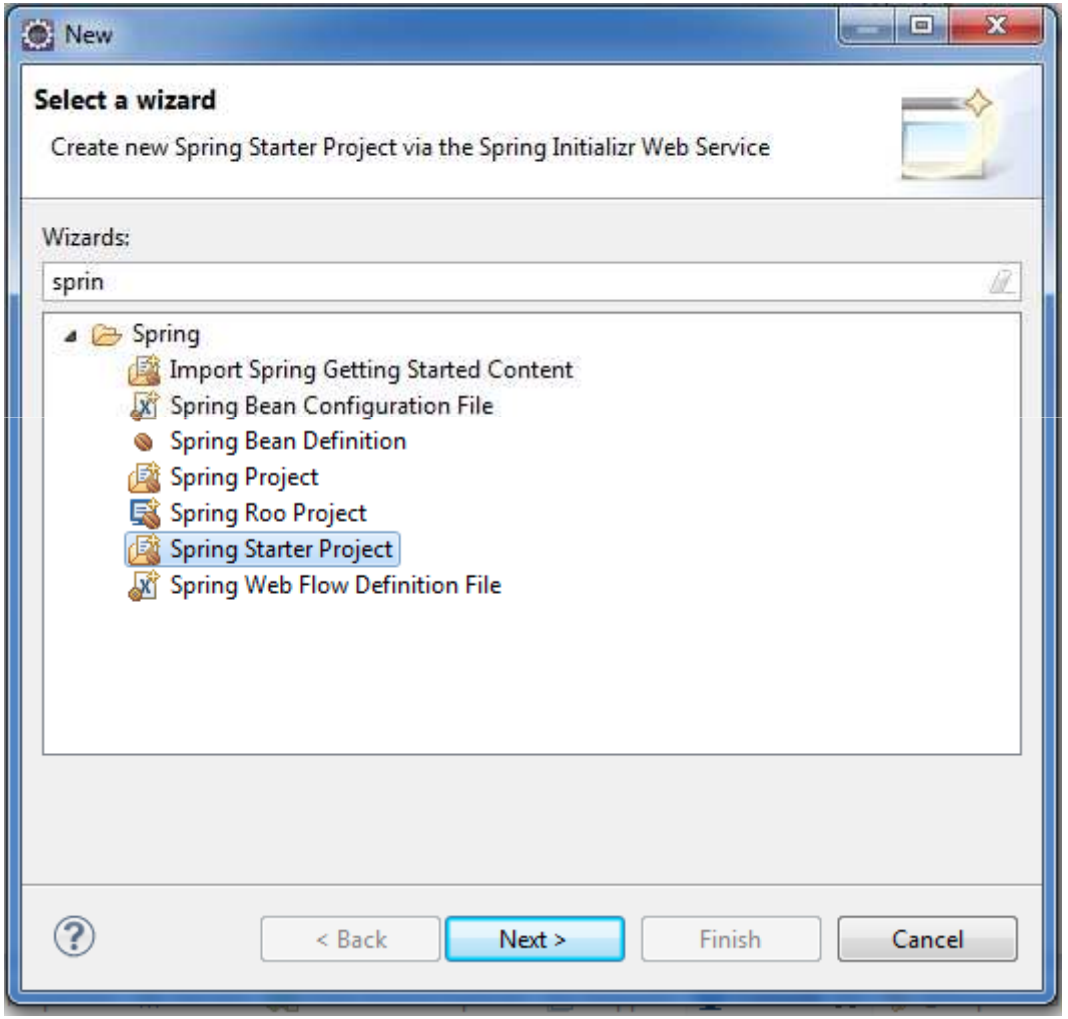
Architecture



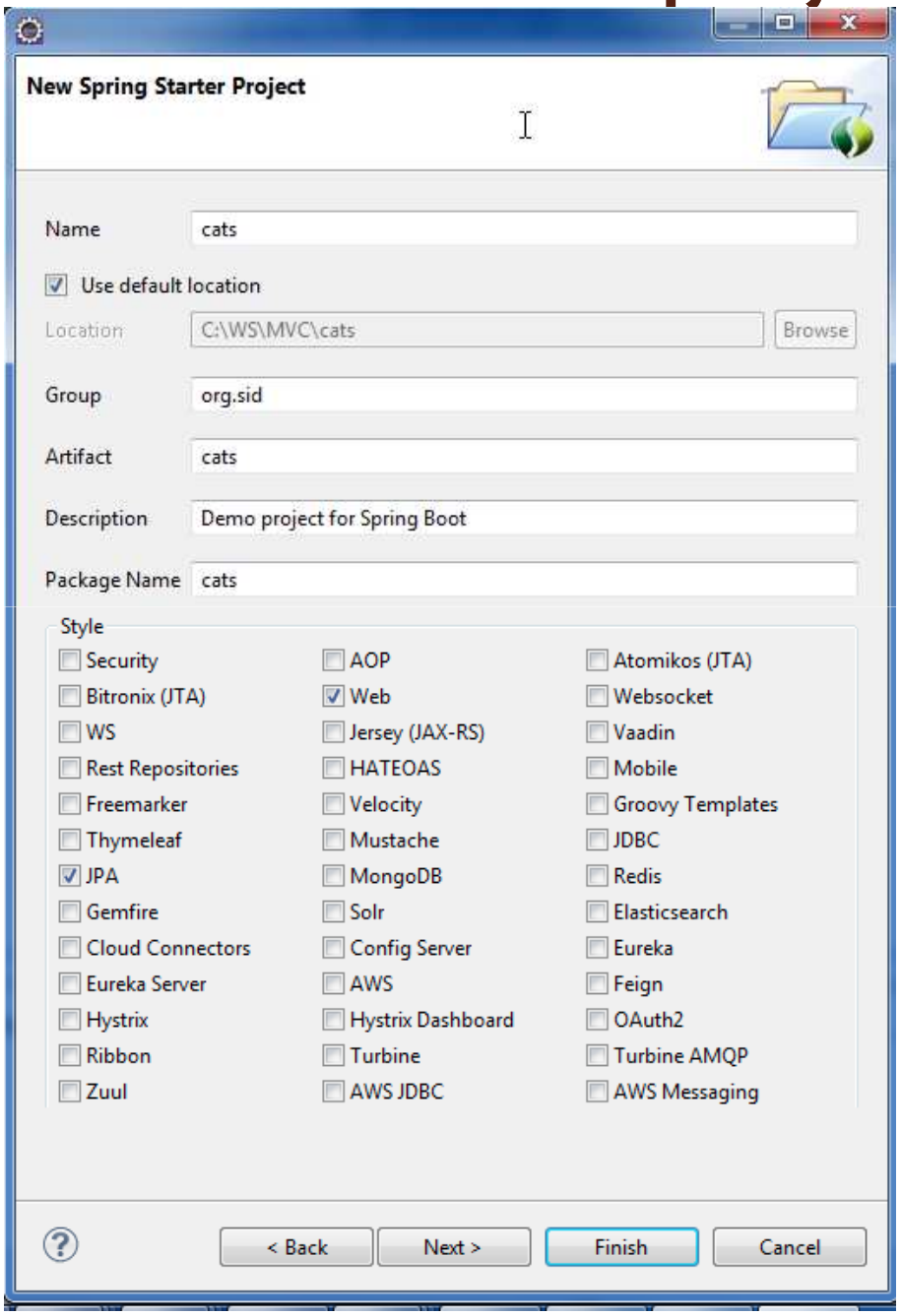
Exemple de Vue à implémenter



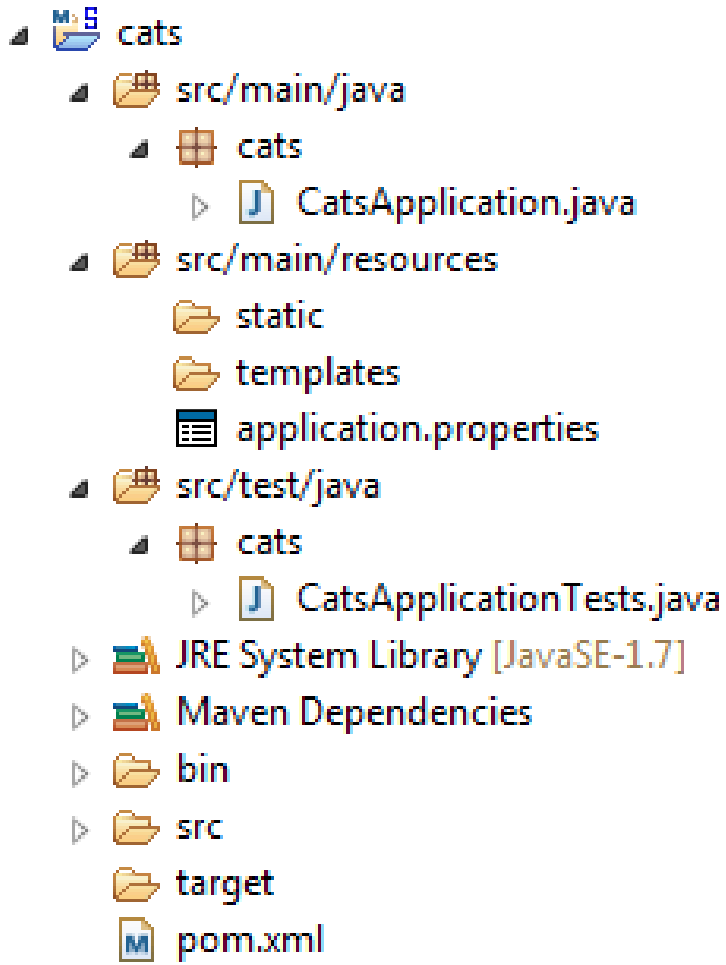
Création d'un projet Spring Boot



Création d'un projet Spring Boot



Structure du projet

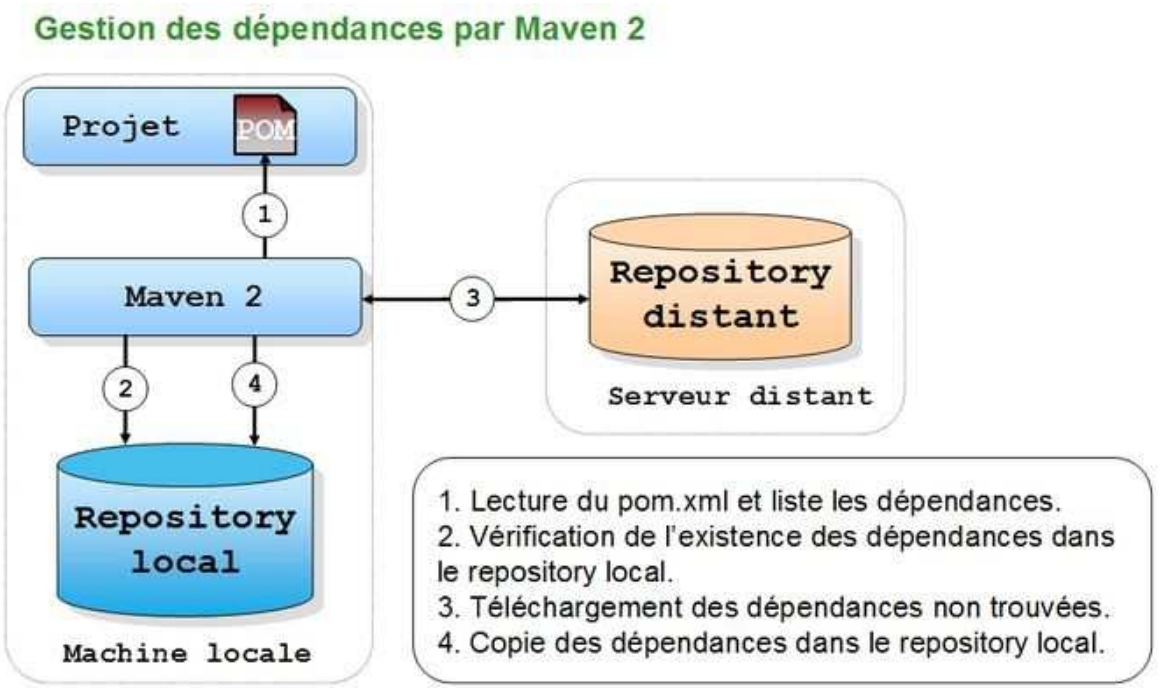


Maven

- **Maven,** géré par l'organisation *Apache Software Foundation*. (*Jakarta Project),* est un **outil pour la gestion et l'automatisation de production des projets logiciels Java en général et Java EE en particulier**.
- L'objectif recherché est de
  - produire un logiciel à partir de ses sources,
  - en optimisant les tâches réalisées à cette fin
  - et en garantissant le bon ordre de fabrication.
    - **Compiler, Tester, Contrôler, produire les packages livrables**
    - **Publier la documentation et les rapports sur la qualité**
- Apports :
  - Simplification du processus de construction d'une application
  - Fournit les bonnes pratique de développement
  - Tend à uniformiser le processus de construction logiciel
  - Vérifier la qualité du code
  - Faciliter la maintenance d'un projet

Maven : POM

- *Maven* utilise un paradigme connu sous le nom de ***Project Object Model* (POM)** afin de :
  - Décrire un projet logiciel,
  - Ses dépendances avec des modules externes
  - et l'ordre à suivre pour sa production.
- Il est livré avec un grand nombre de tâches (**GOLS**) prédéfinies, comme la compilation du code Java ou encore sa modularisation.



pom.xml

```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.sid</groupId>
<artifactId>cats</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>jar</packaging>
<name>cats</name>
<description>Demo project for Spring Boot</description>
<parent>
   <groupId>org.springframework.boot</groupId>
   <artifactId>spring-boot-starter-parent</artifactId>
   <version>1.2.3.RELEASE</version>
   <relativePath/> <!-- lookup parent from repository -->
</parent>
<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <start-class>cats.CatsApplication</start-class>
  <java.version>1.7</java.version>
</properties>
```

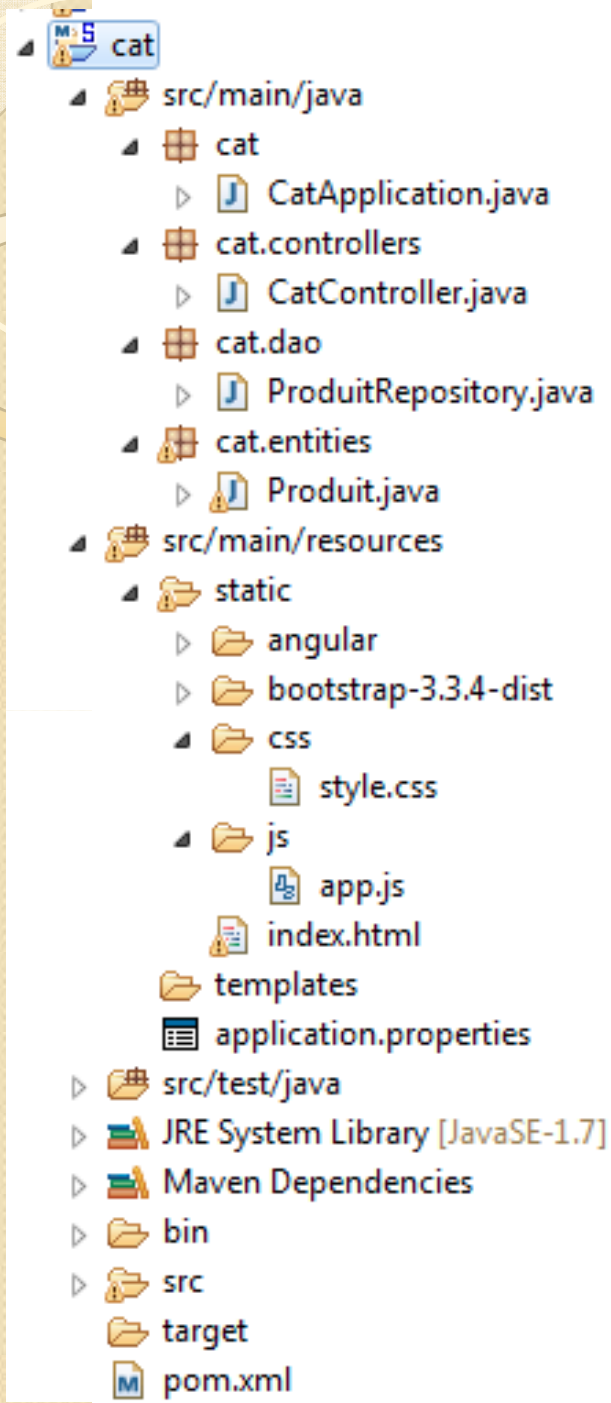
pom.xml

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

pom.xml

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

</project>
```

Structure du projet



application.properties

```
# DataSource settings:
spring.datasource.url = jdbc:mysql://localhost:3306/db_boot_cat5
spring.datasource.username = root
spring.datasource.password =
spring.datasource.driverClassName = com.mysql.jdbc.Driver

# Specify the DBMS
spring.jpa.database = MYSQL
# Show or not log for each sql query
spring.jpa.show-sql = true
# Hibernate ddl auto (create, create-drop, update)
spring.jpa.hibernate.ddl-auto = update
# Naming strategy
spring.jpa.hibernate.naming-strategy = org.hibernate.cfg.ImprovedNamingStrategy

spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect
```

Spring Boot Application

```
package cat;
import org.springframework.boot.SpringApplication;
import
  org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class CatApplication {
    public static void main(String[] args) {
        SpringApplication.run(CatApplication.class, args);
    }
}
```

ENTITÉS

med@youssfi.net | ENSET Université
Hassan II de Casablanca

Entité Produit

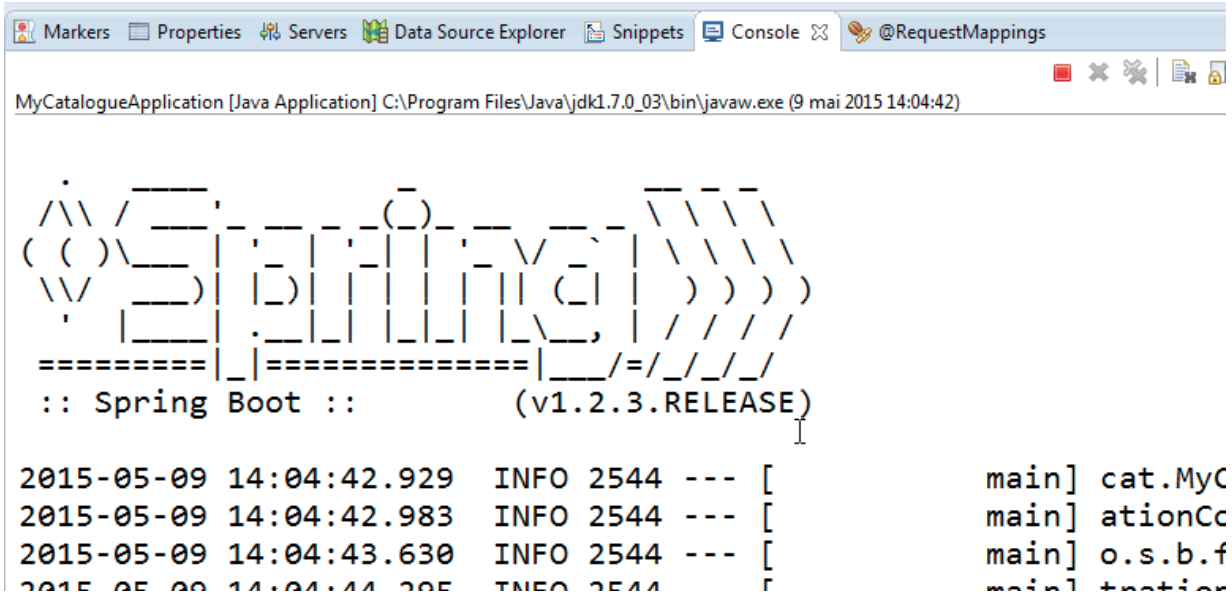
```
package cat.entities;
   @Entity
   public class Produit implements Serializable {
        @Id
        @GeneratedValue(strategy=GenerationType.IDENTITY)
        private Long reference;
        private String designation;
        private double prix;
// getters et Setters
// Constructeurs
}
```

COUCHE DAO

med@youssfi.net | ENSET Université
Hassan II de Casablanca

Tester les entités

- Démarrer l'application et vérifier si la tables produits a été bien créée.



Nom	Type	Interclassement	Attributs	Null	Défaut	Extra
reference	bigint(20)			Non	*Aucune*	AUTO_INCREMENT
designation	varchar(255)	latin1_swedish_ci		Oui	*NULL*	
prix	double			Non	*Aucune*	

Couche DAO

```
package cat.dao;

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);
}
```

COUCHE WEB

med@youssfi.net | ENSET Université
Hassan II de Casablanca

Le contrôleur Rest

```
package cat.controllers;
@RestController
public class CatalogueController {
  @Autowired
  private IProduitRepository produitRepository;

  @RequestMapping("/save")
  public Produit saveProduit(Produit  p){
      produitRepository.save(p);
      return p;
   }
  @RequestMapping("/all")
  public List<Produit> getProduits(){
      return produitRepository.findAll();
  }
```

Le contrôleur Rest

```
@RequestMapping("/produits")
public Page<Produit> getProduits(int page){
  return produitRepository.findAll(new PageRequest(page, 5));
}
@RequestMapping("/produitsParMC")
public Page<Produit> getProduits(String mc,int page){
  return produitRepository.produitParMC("%"+mc+"%", new
  PageRequest(page, 5));
}
@RequestMapping("/get")
public Produit getProduit(Long ref){
  return produitRepository.findOne(ref);
}
```

Le contrôleur Rest

```
@RequestMapping("/delete")
public boolean delete(Long ref){
  produitRepository.delete(ref);
  return true;
}
@RequestMapping("/update")
public Produit update(Produit p){
  produitRepository.saveAndFlush(p);
  return p;
}
}
```

Ajouter un produit

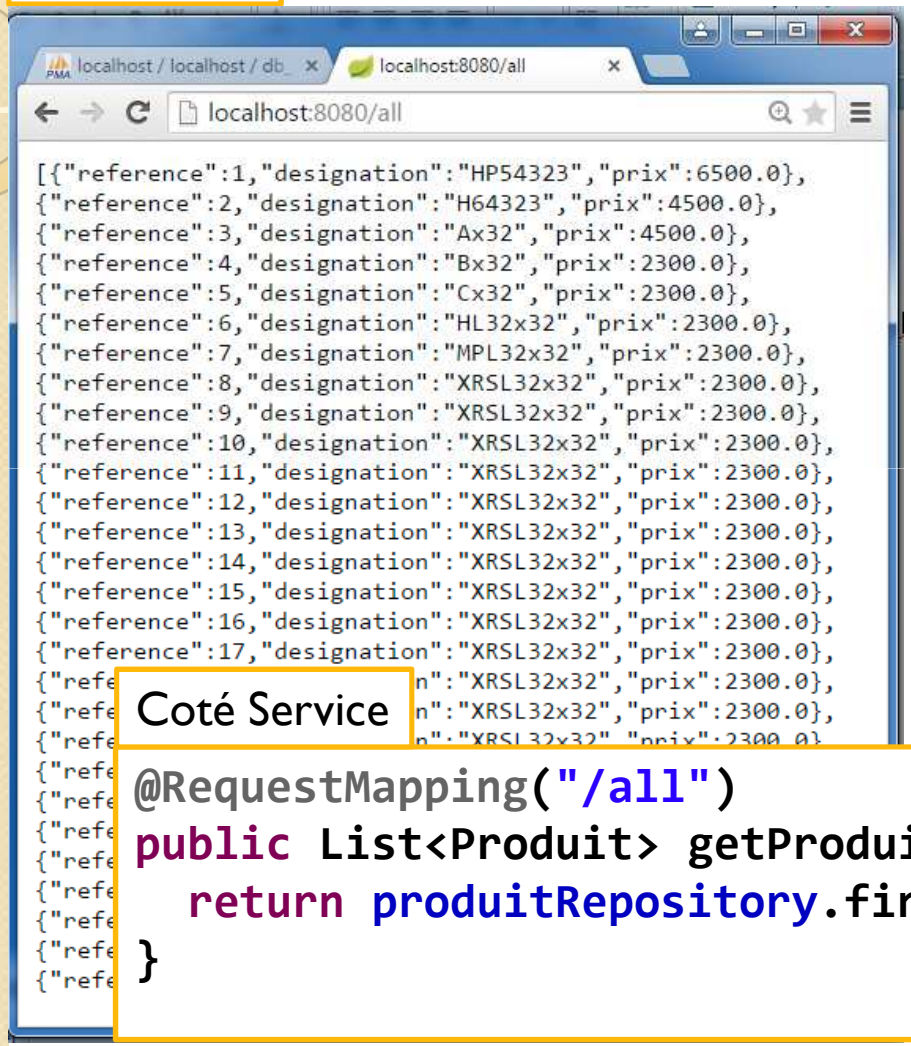
Coté Client

Coté Service

```
@RequestMapping("/save")
public Produit saveProduit(Produit  p){
  produitRepository.save(p);
  return p;
}
```

Consulter tous les produits

Coté Client



Coté Service

```
@RequestMapping("/all")
public List<Produit> getProduits(){
   return produitRepository.findAll();
}
```

Consulter une page de produits

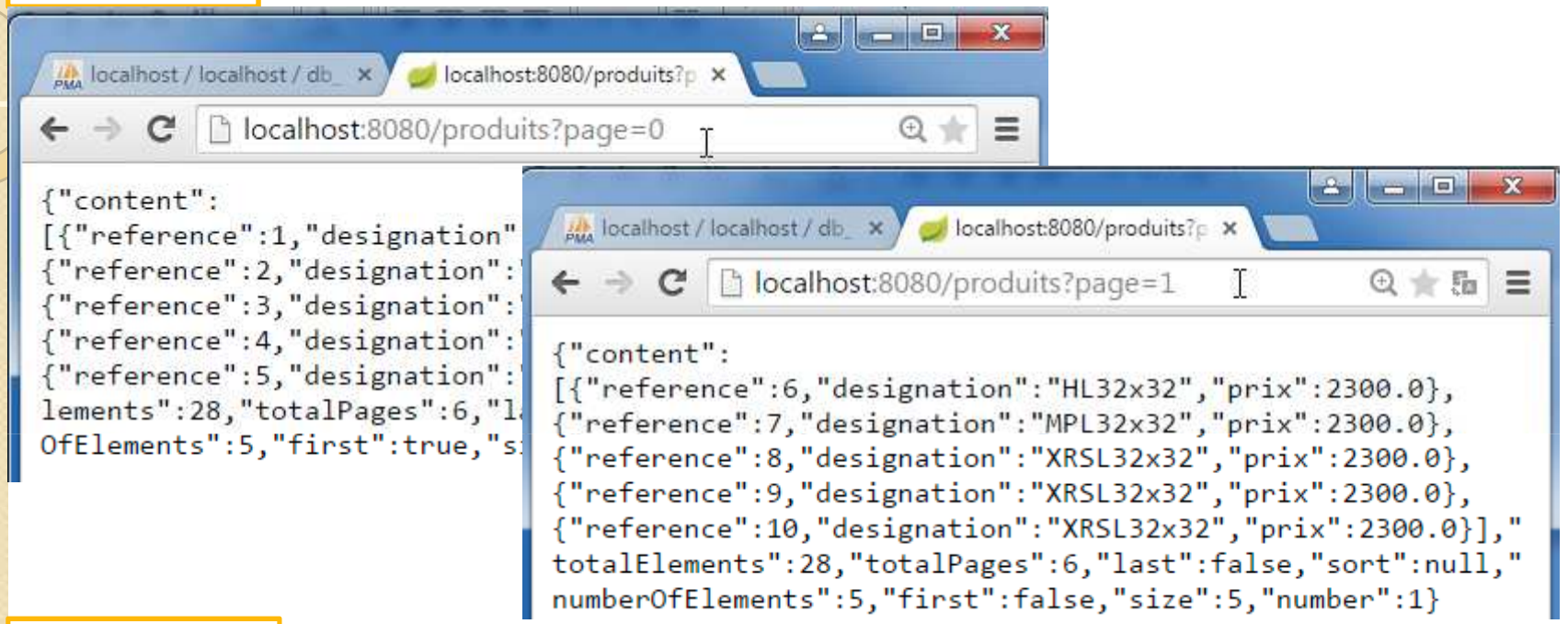
Coté Client

Coté Service

```
@RequestMapping("/produits")
public Page<Produit> getProduits(int page){
  return produitRepository.findAll(new PageRequest(page, 5));
}
```

Consulter une page de produits

Coté Client



Coté Service

```
@RequestMapping("/produits")
public Page<Produit> getProduits(int page){
  return produitRepository.findAll(new PageRequest(page, 5));
}
```

Chercher les produits par mot clé

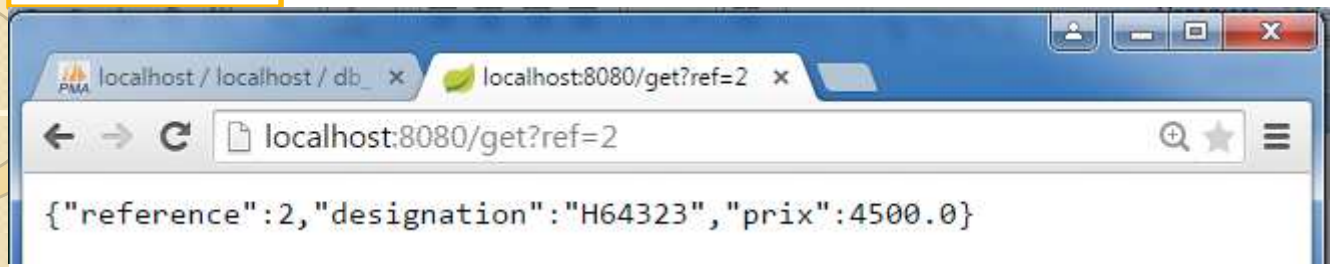
Coté Client

Coté Service

```
@RequestMapping("/produitsParMC")
public Page<Produit> getProduits(String mc,int page){
  return produitRepository.produitParMC("%"+mc+"%", new PageRequest(page, 5));
}
```

Consulter un produit

Coté Client



Coté Service

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

Mettre à jour à produit

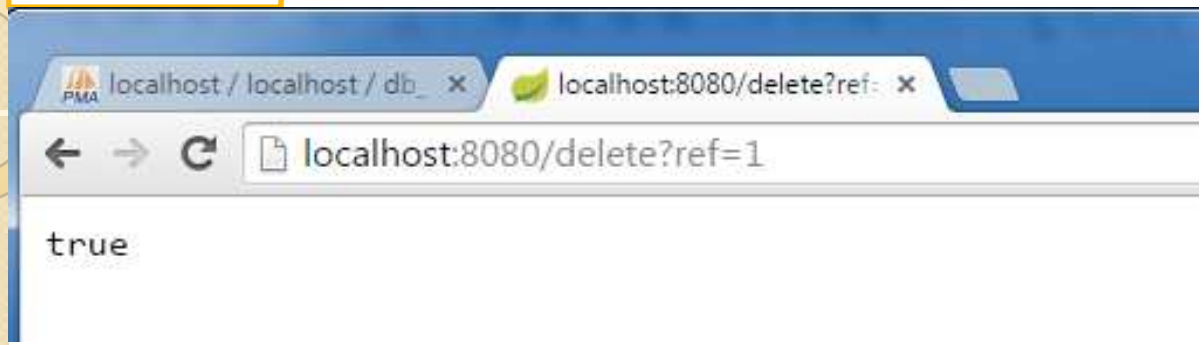
Coté Client

Coté Service

```
@RequestMapping("/update")
public Produit update(Produit p){
  produitRepository.saveAndFlush(p);
  return p;
}
```

Supprimer un produit

Coté Client

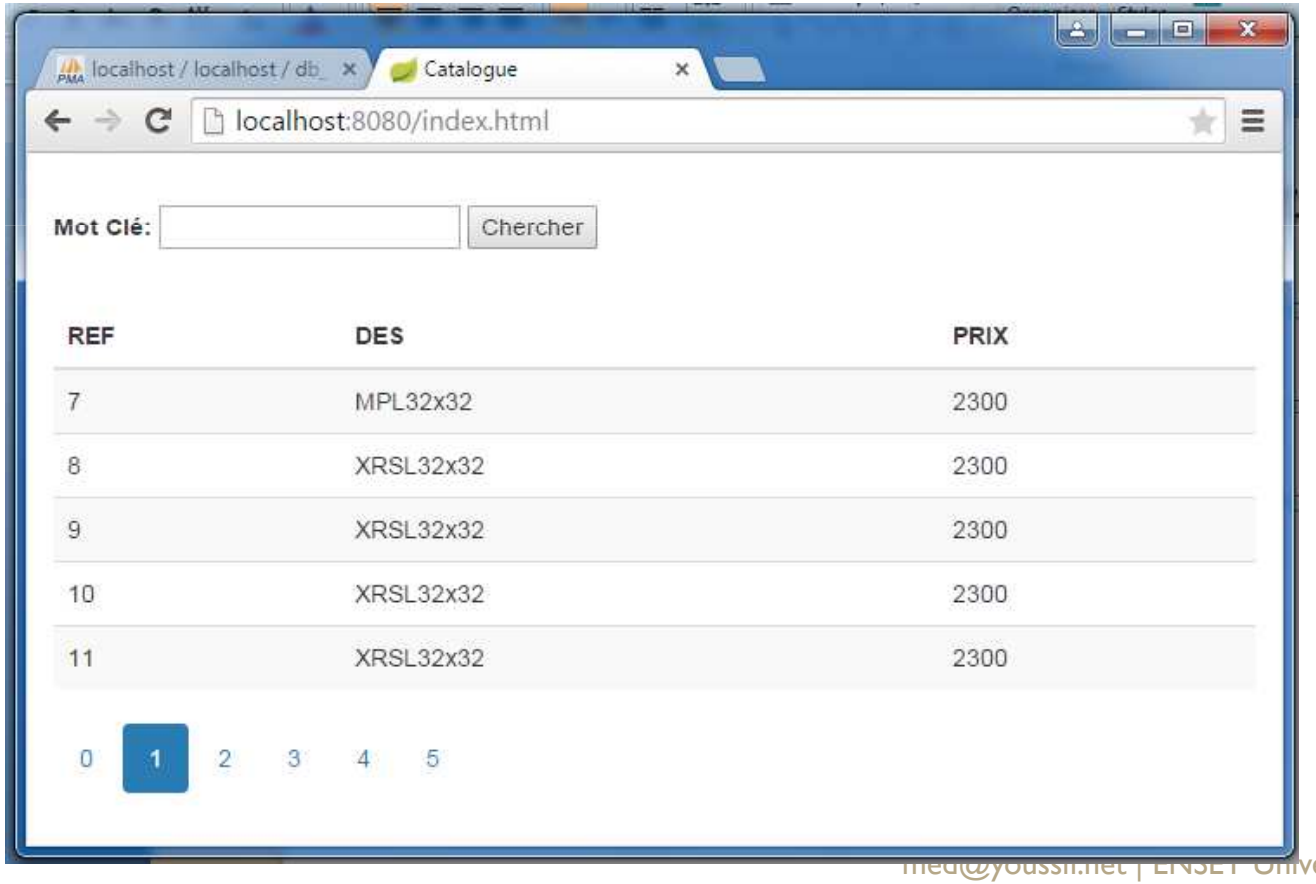


Coté Service

```
@RequestMapping("/delete")
public boolean delete(Long ref){
  produitRepository.delete(ref);
  return true;
}
```

Couche Présentation Web Coté Client

- HTML5
- Java Script avec le framework Angular JS
- CSS avec le framework BootStrap



index.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Catalogue</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>
```

index.html

```
<div class="container spacer">
 <table class="table table-striped">
   <thead>
     <tr>
      <th>REF</th><th>DES</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>
```

index.html

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

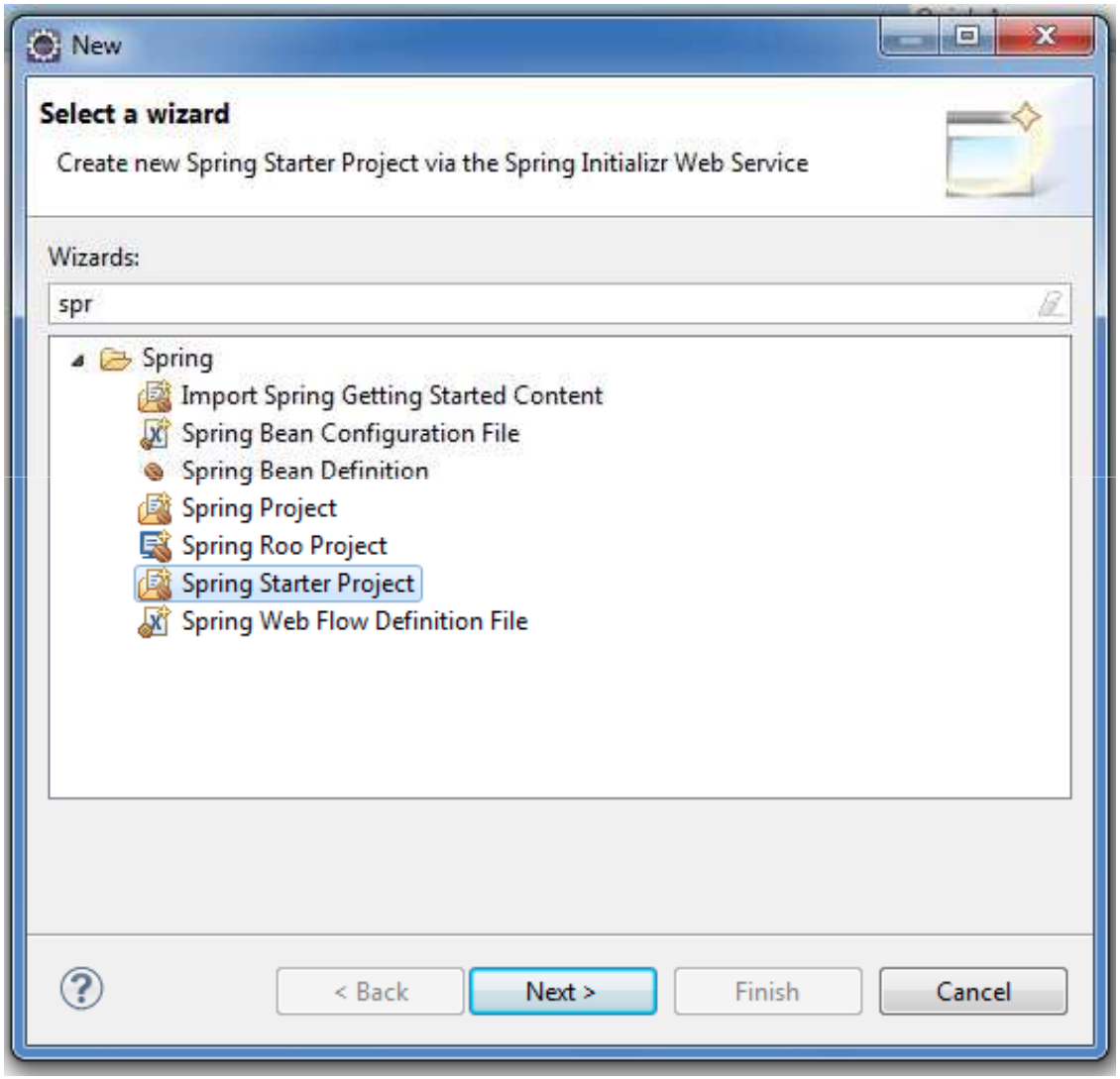
style.css

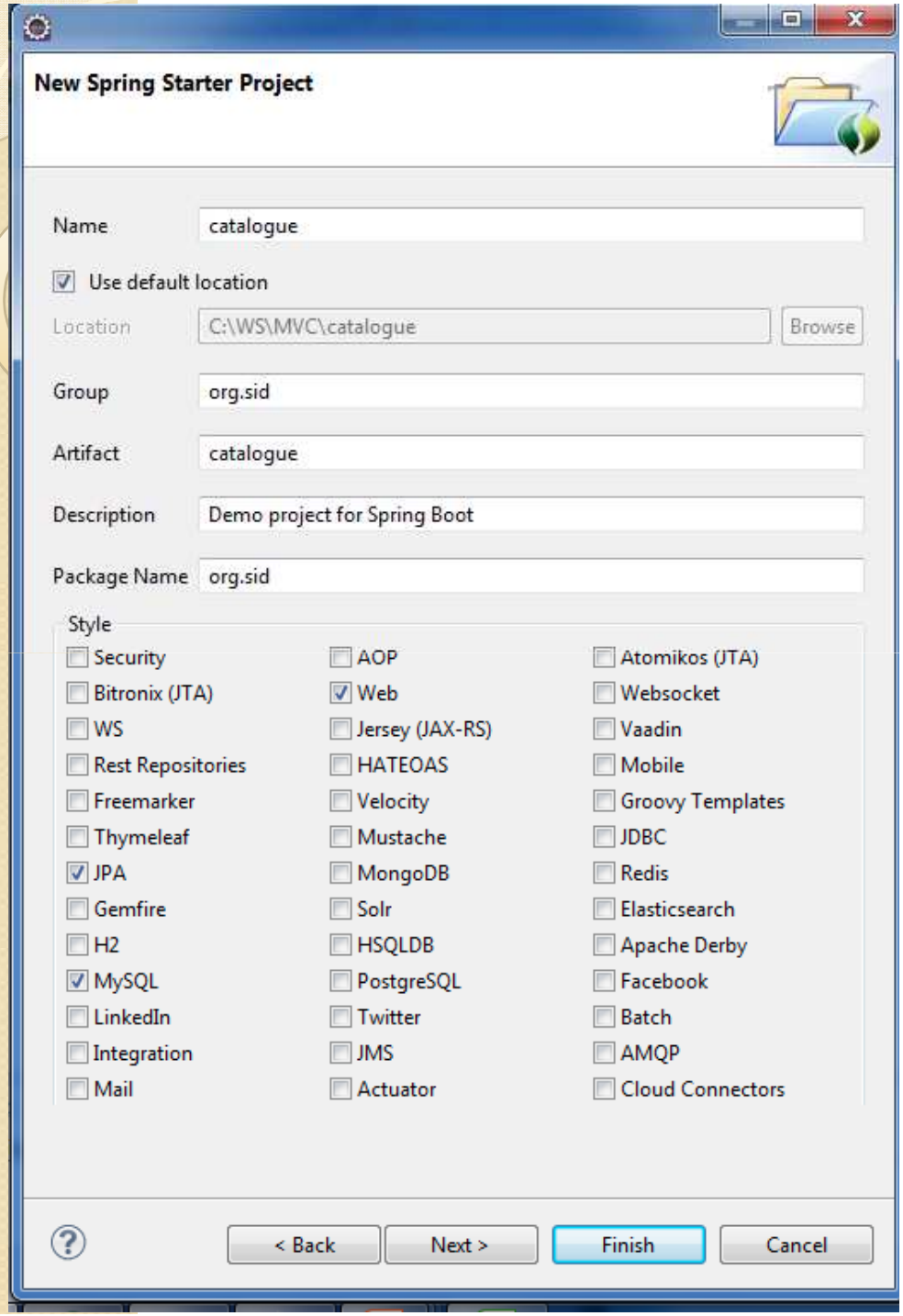
```
.spacer{
   margin-top: 30px;
}
.clickable{
   cursor: pointer;
}
```

Deuxième Application

- Supposant que l'on souhaite créer créer une application qui permet de gérer le catalogue des produits appartenant à des catégories.
- Chaque produit est définit par :
  - Sa référence de type String
  - Sa désignation de type String
  - Son prix de type double
  - Sa quantité de type int
  - Sa disponibilité de type boolean
  - Sa photo
- Une catégorie est définie par :
  - Son code de type Long (Auto Increment)
  - Son nom de type String
- L'application doit permettre
  - D'ajouter une nouvelle catégorie
  - Ajouter un produit appartenant à une catégorie
  - Consulter toutes les catégories
  - Consulter les produits dont le nom contient un mot clé
  - Consulter les produits d'une catégorie
  - Consulter un produit
  - Mettre à jour un produit
  - Supprimer une catégorie
- L'injection des dépendances sera assurée par Spring IOC

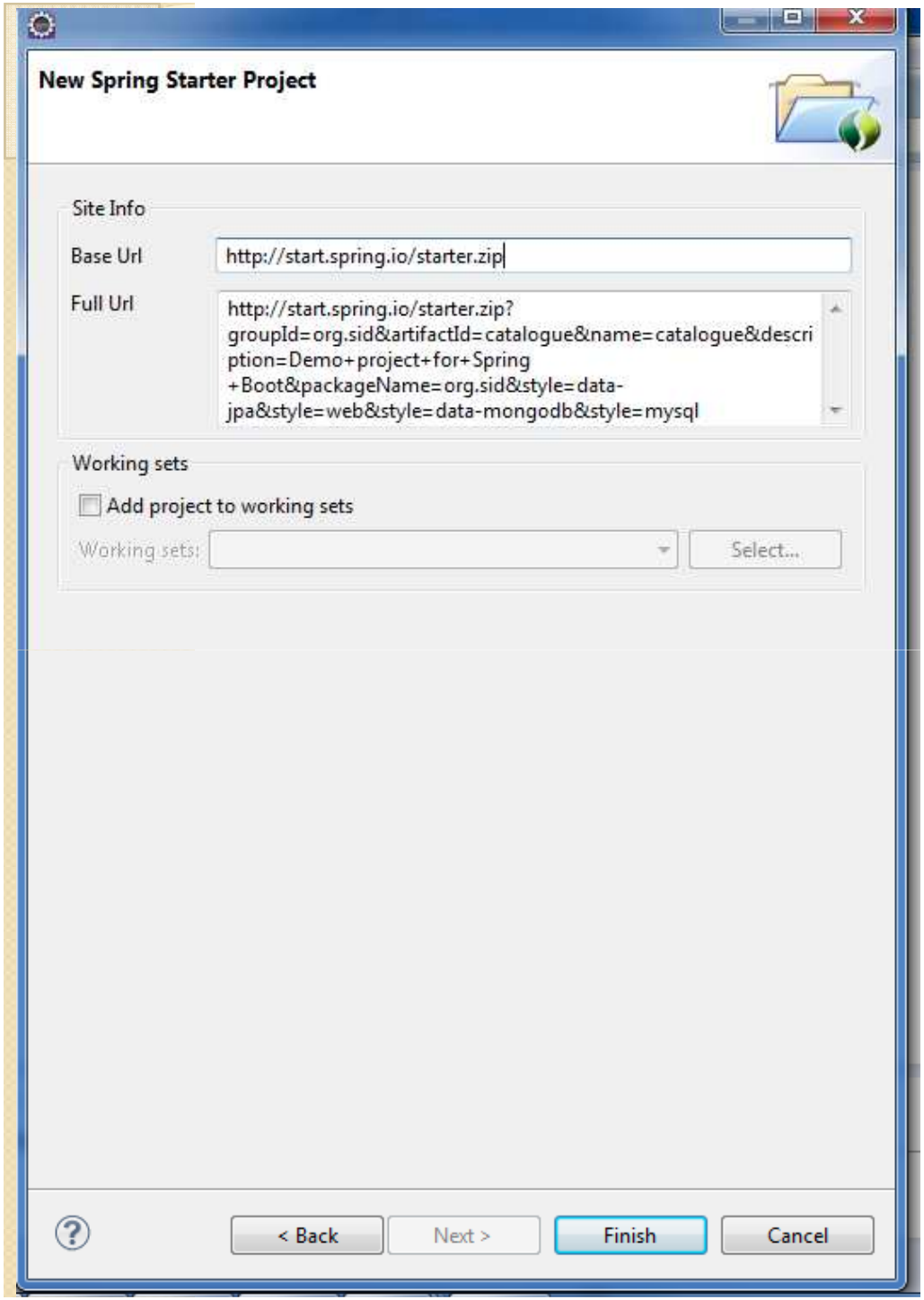
Création d'un projet Spring Starter





Création d'un projet Spring Starter

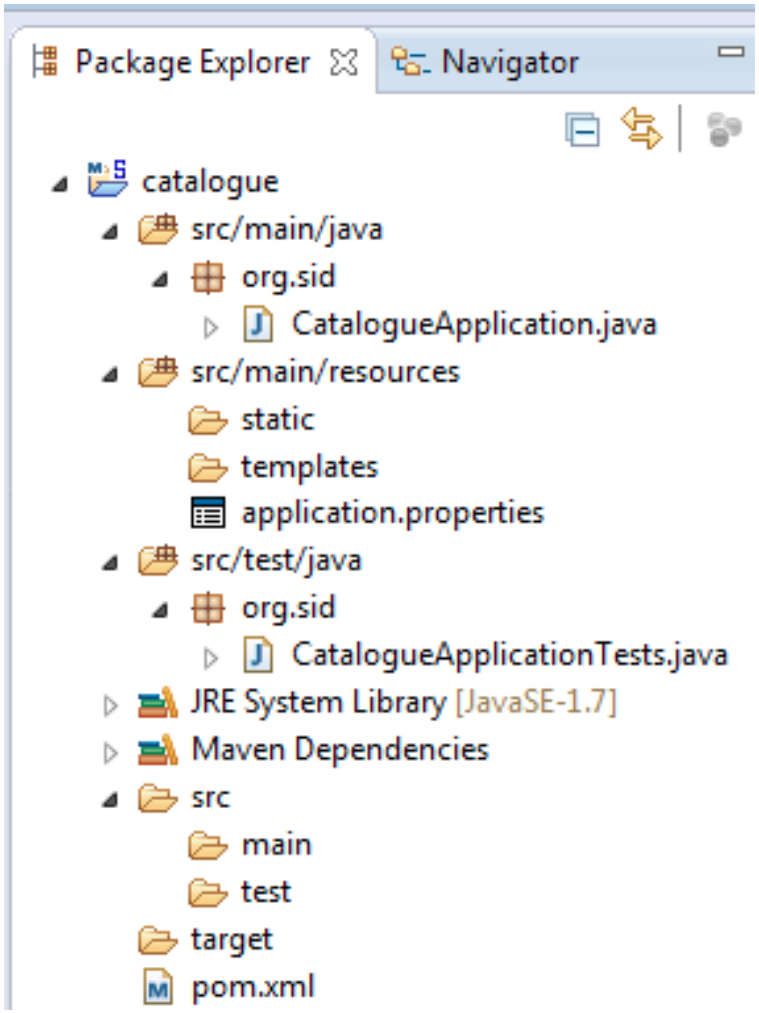
med@youssfi.net | ENSET Université Hassan II de Casablanca



Création d'un projet Spring Starter

med@youssfi.net | ENSET Université
Hassan II de Casablanca

Structure du projet



Pom.xml

```
<modelVersion>4.0.0</modelVersion>
<groupId>org.sid</groupId>
<artifactId>catalogue</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>jar</packaging>

<name>catalogue</name>
<description>Demo project for Spring Boot</description>

<parent>
 <groupId>org.springframework.boot</groupId>
 <artifactId>spring-boot-starter-parent</artifactId>
 <version>1.2.3.RELEASE</version>
 <relativePath/> <!-- lookup parent from repository -->
</parent>
```

Pom.xml

```
<properties>
  <project.build.sourceEncoding>UTF-
  8</project.build.sourceEncoding>
  <start-class>org.sid.CatalogueApplication</start-class>
  <java.version>1.7</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>
```

Pom.xml

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

application.properties

```
# DataSource settings:
spring.datasource.url = jdbc:mysql://localhost:3306/db_boot
spring.datasource.username = root
spring.datasource.password =
spring.datasource.driverClassName = com.mysql.jdbc.Driver

# Specify the DBMS
spring.jpa.database = MYSQL
# Show or not log for each sql query
spring.jpa.show-sql = true
# Hibernate ddl auto (create, create-drop, update)
spring.jpa.hibernate.ddl-auto = update
# Naming strategy
spring.jpa.hibernate.naming-strategy = org.hibernate.cfg.ImprovedNamingStrategy

spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL5Dialect
# View Resolver
spring.view.prefix: /WEB-INF/views/
spring.view.suffix: .jsp
```

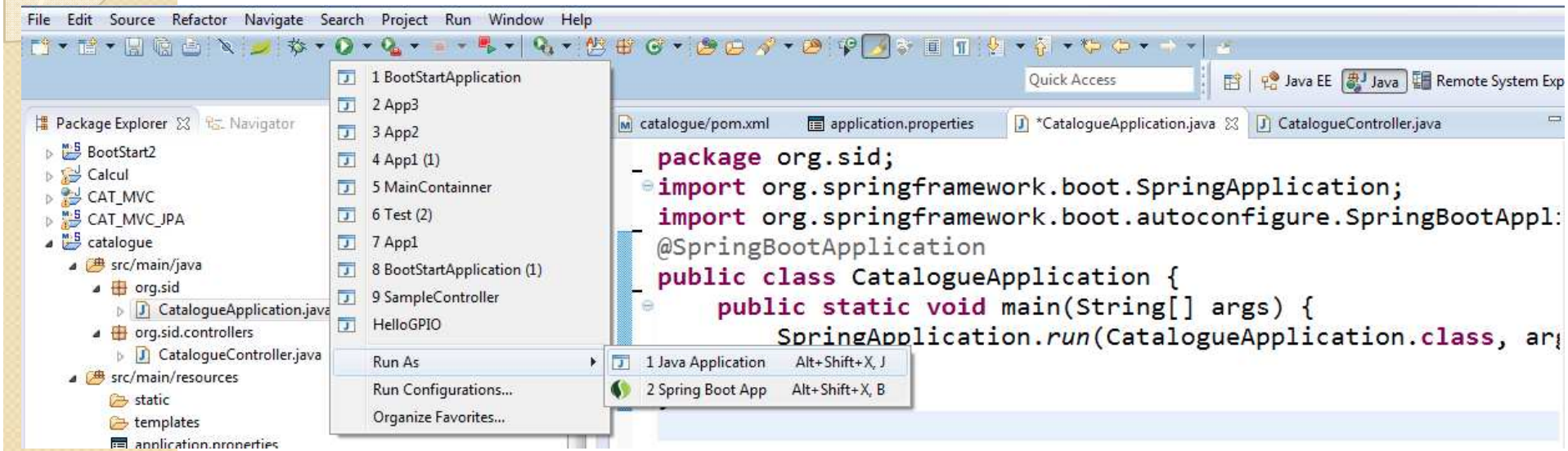
SpringBootApplication

```
package org.sid;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class CatalogueApplication {
    public static void main(String[] args) {
        SpringApplication.run(CatalogueApplication.class, args);
    }
}
```

Premier Controleur Spring

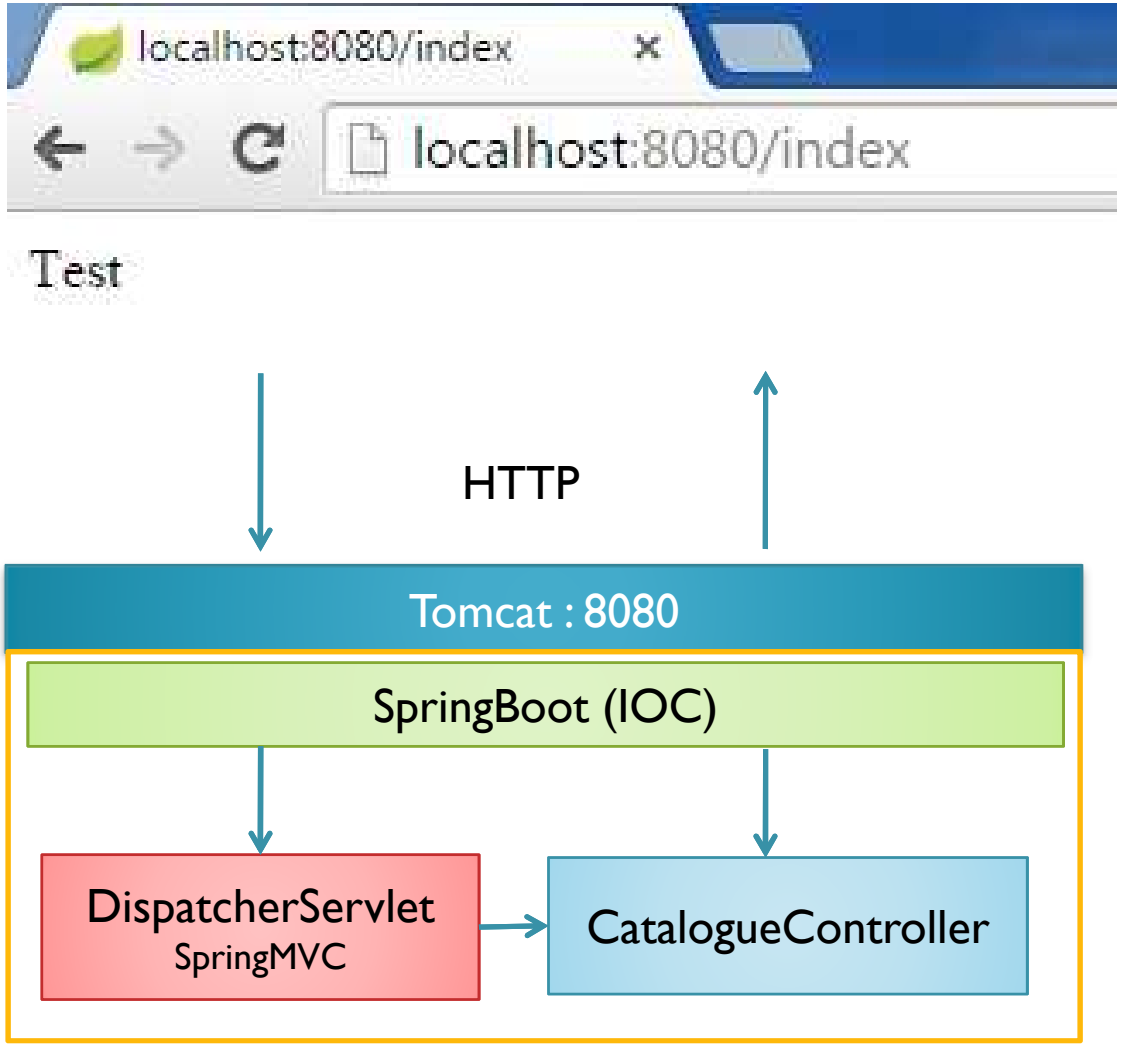
```
package org.sid.controllers;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;
@Controller
public class CatalogueController {
    @RequestMapping(value="/index")
    @ResponseBody
    public String index(){
      return "Test";
    }
    @RequestMapping(value="/test")
    public String test(){
       return "Test";
    }
}
```

Exécuter l'application SpringBoot



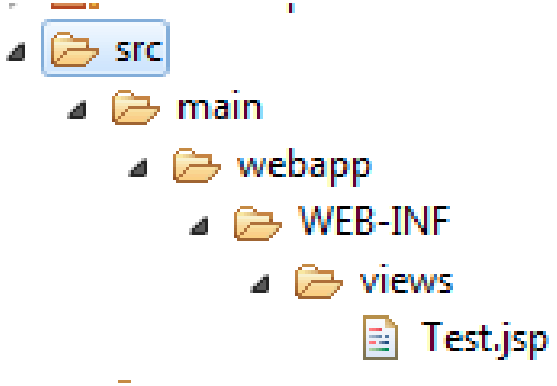
- Run As > Java Application

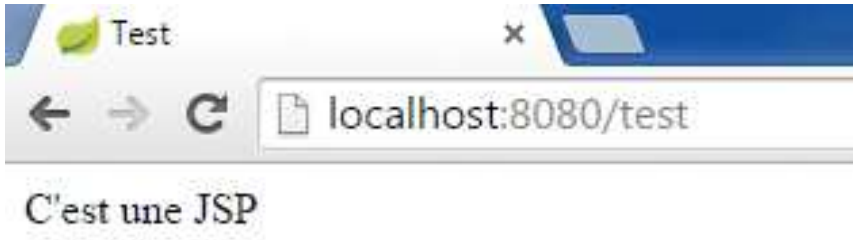
Tester l'application



Travailler Coté serveur avec JSP et JSTL

```
<dependency>
    <groupId>org.apache.tomcat.embed</groupId>
    <artifactId>tomcat-embed-jasper</artifactId>
    <scope>provided</scope>
</dependency>

<dependency>
    <groupId>javax.servlet</groupId>
    <artifactId>jstl</artifactId>
</dependency>
```





Exemple d'application

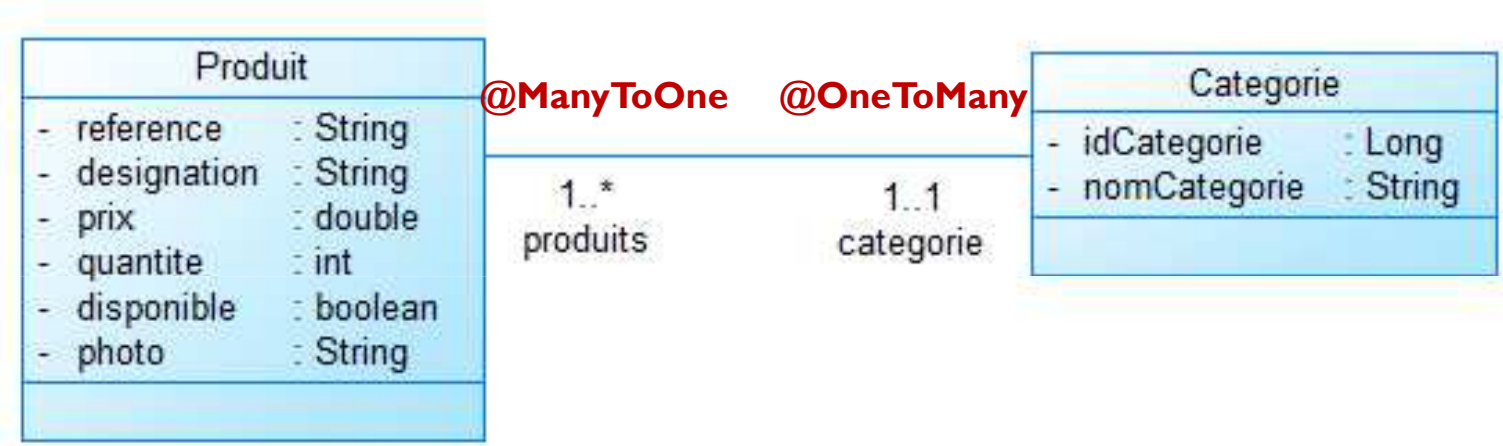
- Supposant que l'on souhaite créer créer une application qui permet de gérer le catalogue des produits appartenant à des catégories.
- Chaque produit est définit par :
  - Sa référence de type String
  - Sa désignation de type String
  - Son prix de type double
  - Sa quantité de type int
  - Sa disponibilité de type boolean
  - Sa photo
- Une catégorie est définie par :
  - Son code de type Long (Auto Increment)
  - Son nom de type String
- L'application doit permettre
  - D'ajouter une nouvelle catégorie
  - Ajouter un produit appartenant à une catégorie
  - Consulter toutes les catégories
  - Consulter les produits dont le nom contient un mot clé
  - Consulter les produits d'une catégorie
  - Consulter un produit
  - Mettre à jour un produit
  - Supprimer une catégorie
- L'injection des dépendances sera assurée par Spring IOC

ENTITÉS

med@youssfi.net | ENSET Université
Hassan II de Casablanca

Entités : Produit et Categorie

Diagramme de classes : Modèle Objet



MLDR : Modèle Relationnel

- CATEGORIES (ID_CAT, NOM_CAT)
- PRODUITS (REF, DES, PRIX, QTE, DISPO, PHOTO, #ID_CAT)

Entité : Produit

```
package dao;
import java.io.Serializable; import javax.persistence.*;
@Entity
public class Produit implements Serializable {
   @Id
  private String reference;
  private String designation; private double prix;   private int quantite;
   private boolean disponible;  private String photo;
  @ManyToOne
  @JoinColumn(name="ID_CAT")
  private Categorie categorie;
   public Produit() { }
   public Produit(String reference, String designation, double prix, int
   quantite, boolean disponible, String photo) {
     this.reference = reference; this.designation = designation;
     this.prix = prix; this.quantite = quantite; this.disponible = disponible;
     this.photo = photo;
    }
  // Getters et Setters
}
```

Entité : Categorie

```
package dao;
import java.io.Serializable; import java.util.Collection;
import javax.persistence.*;
@Entity
public class Categorie implements Serializable {
   @Id
   @GeneratedValue(strategy=GenerationType.IDENTITY)
   private Long idCategorie;
   private String nomCategorie;
   @OneToMany(mappedBy="categorie",fetch=FetchType.LAZY)
   private Collection<Produit> produits;
   public Categorie() { }
   public Categorie(String nomCategorie) {
         this.nomCategorie = nomCategorie;
   }
   // Getters et Setters
   @JsonIgnore
   @XmlTransient
   public Collection<Produit> getProduits() {
         return produits;
   }
}
```

Redémarrer l'application Spring Boot

- Les tables devraient être générées

Table Categories :

Nom	Type	Interclassement	Attributs	Null	Défaut	Extra
id_categorie	bigint(20)			Non	*Aucune*	AUTO_INCREMENT
nom_categorie	varchar(255)	latin1_swedish_ci		Oui	*NULL*	

Table Produits :

#	Nom	Type	Interclassement	Attributs	Null	Défaut	Extra
1	reference	varchar(255)	latin1_swedish_ci		Non	*Aucune*	
2	designation	varchar(255)	latin1_swedish_ci		Oui	*NULL*	
3	disponible	bit(1)			Non	*Aucune*	
4	photo	varchar(255)	latin1_swedish_ci		Oui	*NULL*	
5	prix	double			Non	*Aucune*	
6	quantite	int(11)			Non	*Aucune*	
7	id_cat	bigint(20)			Oui	*NULL*	

Couche DAO

```
package org.sid.dao;
import org.sid.entities.Categorie;
import org.springframework.data.jpa.repository.JpaRepository;


public interface CategorieRepository extends JpaRepository<Categorie, Long> {
}
```

```
package org.sid.dao;
import java.util.List;
import org.sid.entities.*;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.repository.query.Param;

public interface ProduitRepository extends JpaRepository<Produit, String> {
  public List<Produit> findByDesignation(@Param("mc")String mc);
  public Page<Produit> findByCategorie(Categorie categorie,Pageable pageable);
}
```

MVC CLIENT SIDE
- HTML, CSS, JAVA SCRIPT
- ANDROÏDE CLIENT
- IOS CLIENT

med@youssfi.net | ENSET Université
Hassan II de Casablanca

Contrôleur : Gestion des produits et des catégories

```
package org.sid.controllers;
import java.util.List;
import org.sid.dao.CategorieRepository; import org.sid.dao.ProduitRepository;
import org.sid.entities.Categorie; import org.sid.entities.Produit;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.PageRequest;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseBody;
@Controller
public class MyCatalogueController {
  @Autowired
  private CategorieRepository categorieRepository;
  @Autowired
  private ProduitRepository produitRepository;

@RequestMapping(value="/saveCat")
@ResponseBody
 public Categorie saveCategorie(Categorie c){
    return categorieRepository.save(c);
}
```

Contrôleur : Gestion des produits et des catégories

```
@RequestMapping(value="/allCat")
@ResponseBody
public List<Categorie> allCategories(){
  return categorieRepository.findAll();
}
@RequestMapping(value="/saveProduit")
@ResponseBody
public Produit saveProduit(Produit p){
 return produitRepository.save(p);
}
@RequestMapping(value="/allProduits")
@ResponseBody
public List<Produit> allProduits(){
  return produitRepository.findAll();
}
```

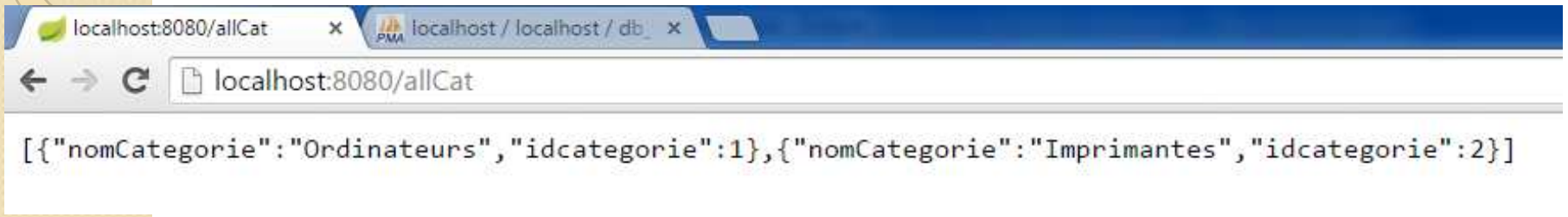
Contrôleur : Gestion des produits et des catégories

```
@RequestMapping(value="/pageProduits")
@ResponseBody
public Page<Produit> pageProduits(int page){
  return produitRepository.findAll(new PageRequest(page, 3));
}

@RequestMapping(value="/produitsParCat")
@ResponseBody
public Page<Produit> produitsParCat(Categorie c,int page){
  return produitRepository.findByCategorie(c, new
  PageRequest(page, 3));
}
@RequestMapping(value="/produitsParMC")
@ResponseBody
public List<Produit> produitsParCat(String mc){
return produitRepository.findByDesignation(mc);
}
}
```

Ajouter une catégorie

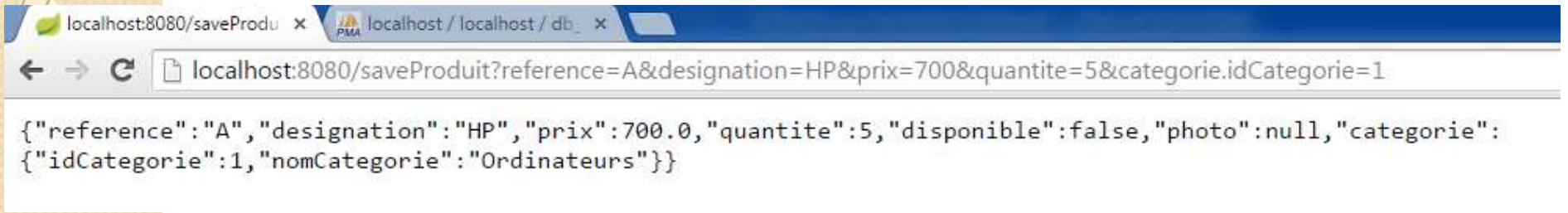

```
@RequestMapping(value="/saveCat")
  @ResponseBody
  public Categorie saveCategorie(Categorie c){
    return categorieRepository.save(c);
 }
```

Consulter toutes les catégories



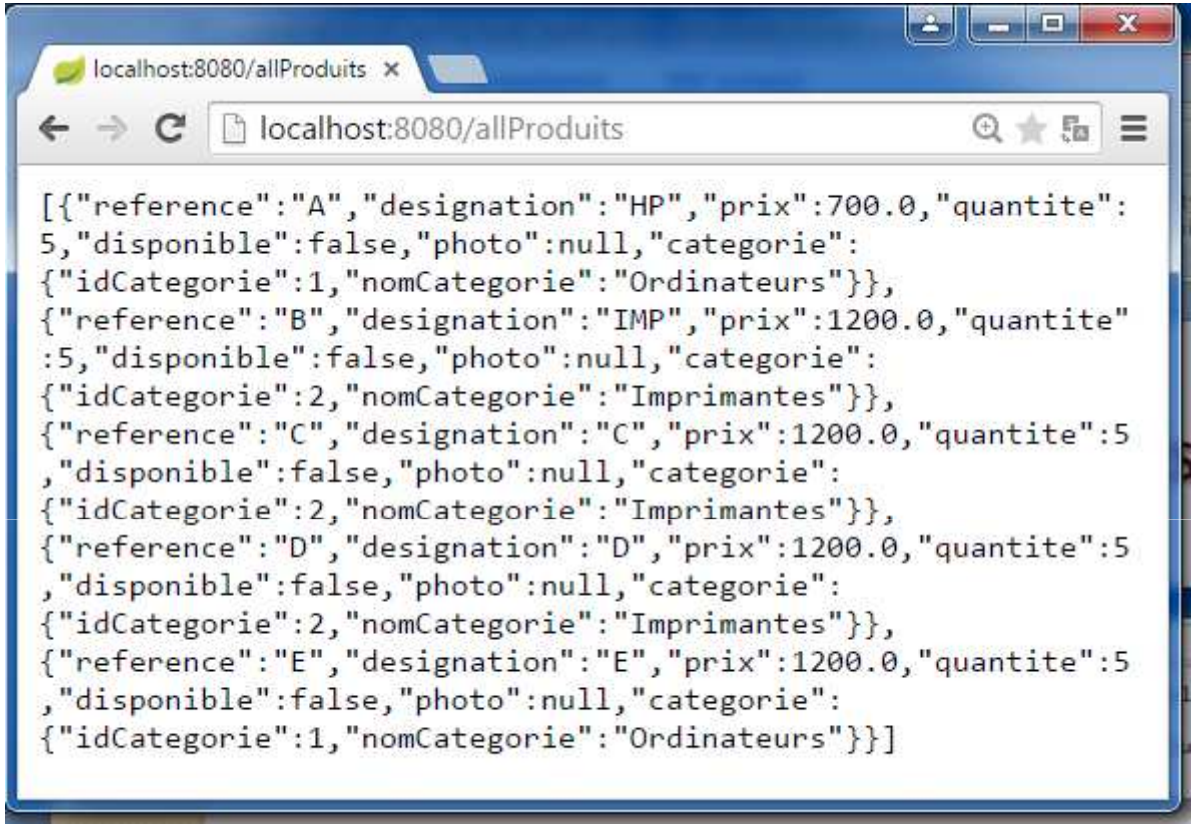

```
@RequestMapping(value="/allCat")
@ResponseBody
public List<Categorie> allCategories(){
  return categorieRepository.findAll();
}
```

Ajouter un produit




```
@RequestMapping(value="/saveProduit")
@ResponseBody
public Produit saveProduit(Produit p){
  return produitRepository.save(p);
}
```

Consulter tous les produits

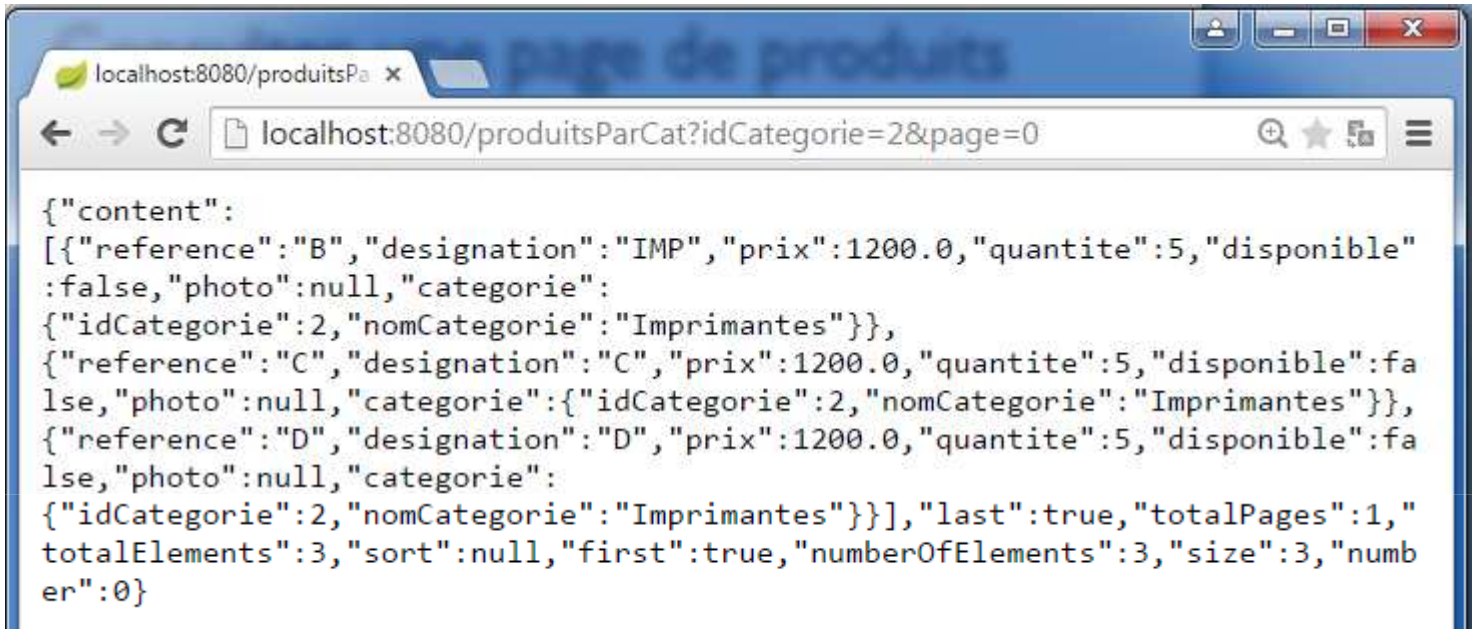



```
@RequestMapping(value="/allProduits")
@ResponseBody
public List<Produit> allProduits(){
  return produitRepository.findAll();
}
```

Consulter une page de produits

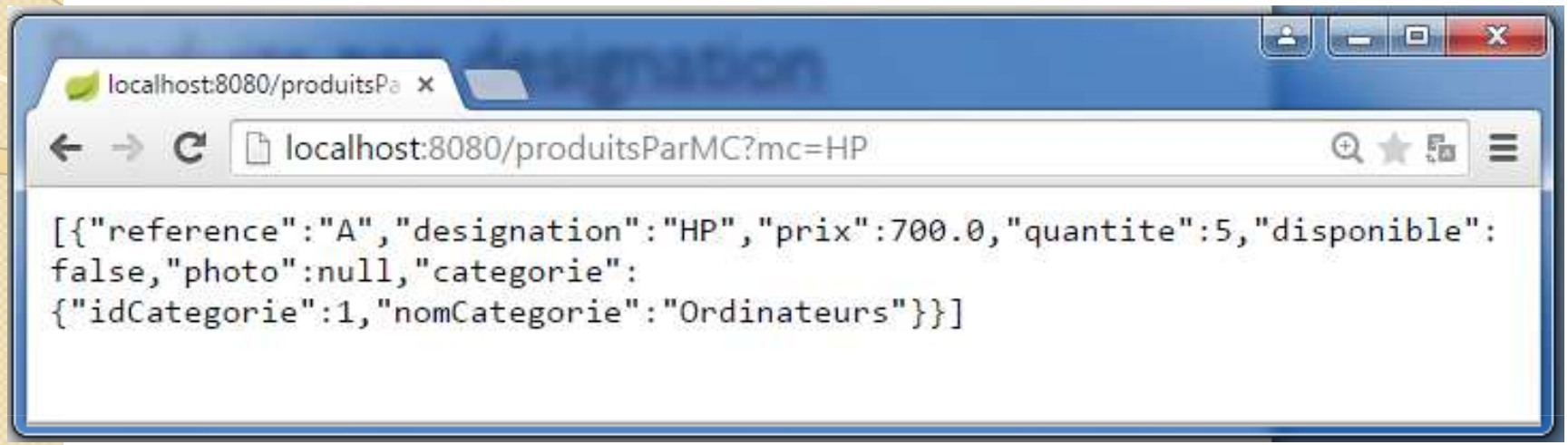

```
@RequestMapping(value="/pageProduits")
@ResponseBody
public Page<Produit> pageProduits(int page){
  return produitRepository.findAll(new PageRequest(page, 3));
}
```

Consulter une page de produits



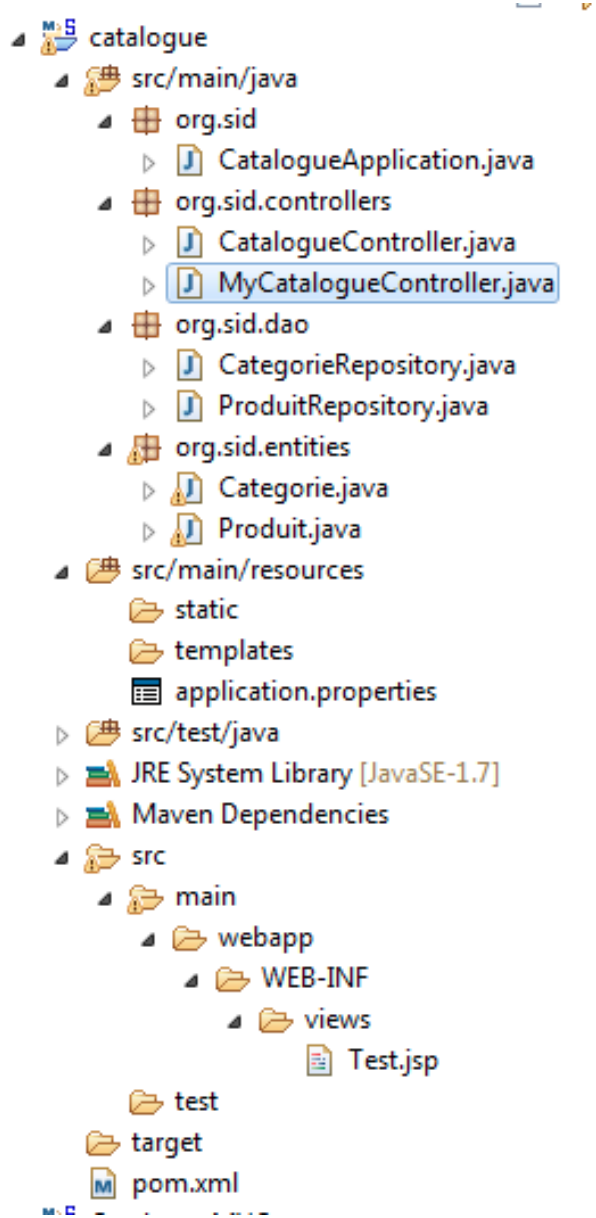

```
@RequestMapping(value="/produitsParCat")
@ResponseBody
public Page<Produit> produitsParCat(Categorie c,int page){
return produitRepository.findByCategorie(c, new PageRequest(page, 3));
}
```

Produits par designation

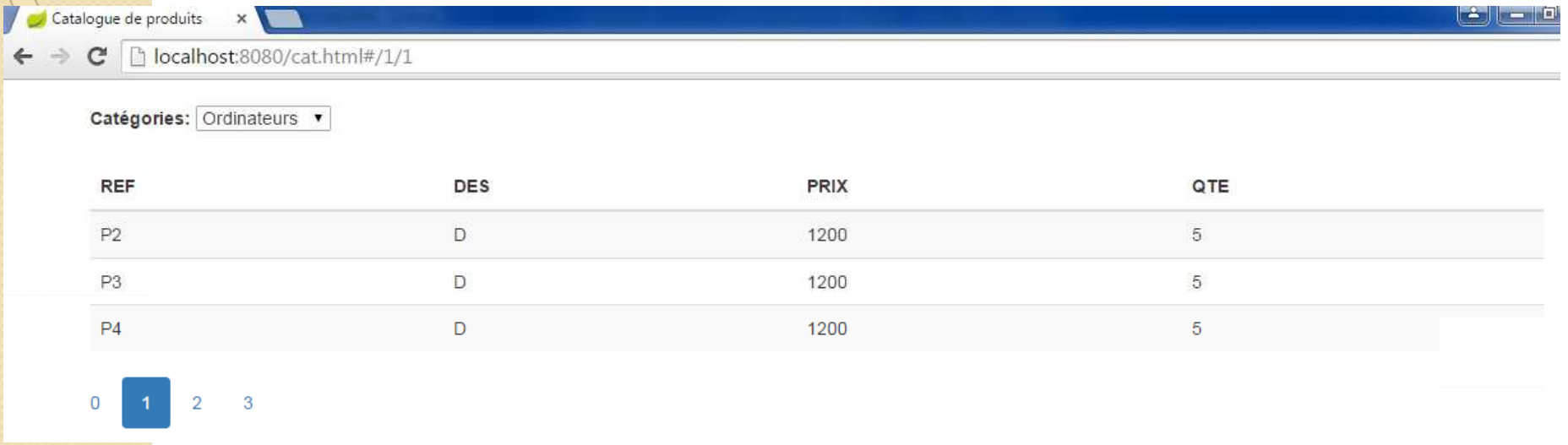



```
@RequestMapping(value="/produitsParMC")
@ResponseBody
public List<Produit> produitsParCat(String mc){
  return produitRepository.findByDesignation(mc);
}
```

Structure du projet

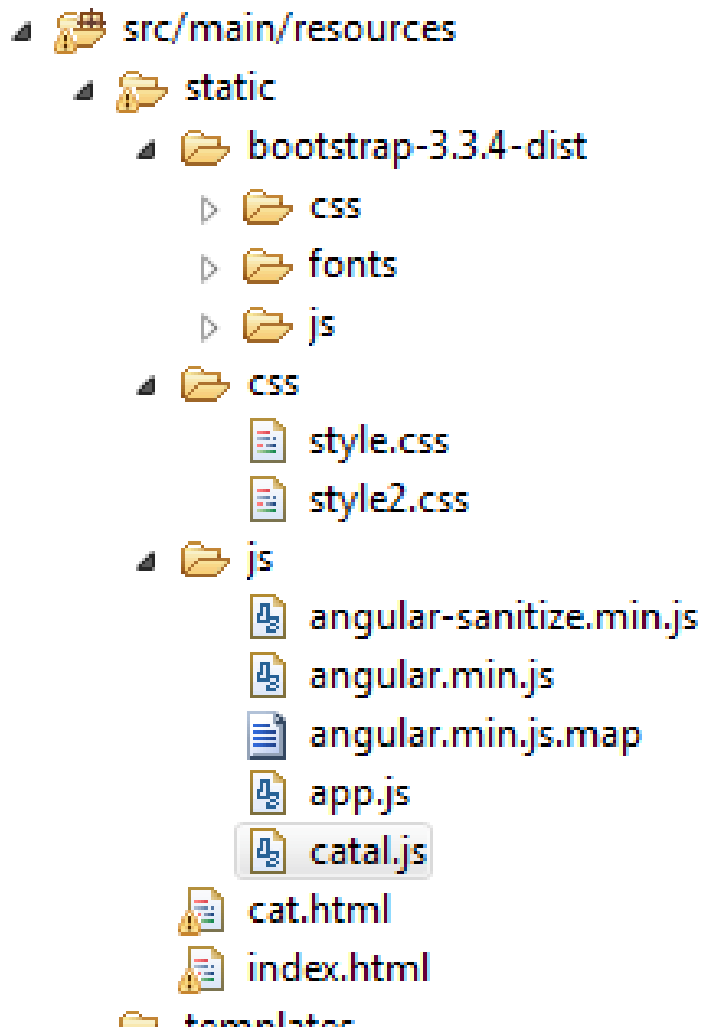


Application web avec HTML, AngularJS et BootStrap coté client



med@youssfi.net | ENSET Université Hassan II de Casablanca

Structure du projet



Catal.js

```
angular.module("catalogue",[])
 .controller("Catcontroller",function($scope,$http,$location){
 $scope.categories=[];
 $scope.produits=[];
 $scope.selectedCategorie=null;
 $scope.pages=new Array();
 $scope.pageCourante=0;
 $scope.chargerCategories=function(){
    $http.get("/allCat")
     .success(function(data){
     $scope.categories=data;
     });
 };
```

Catal.js

```
 $scope.chargerProduits=function(){
$http.get("/produitsParCat?page="+$scope.pageCourante+"&idCategorie
  ="+$scope.selectedCategorie)
     .success(function(data){
     $scope.produits=data;
     $scope.pages=new Array(data.totalPages);
     });
 };
$scope.chargerCategories();
$scope.gotoURL=function(){
$scope.pageCourante=0;
$location.path("/"+$scope.selectedCategorie);
};
$scope.gotoPage=function(page){
$scope.pageCourante=page;
$location.path("/"+$scope.selectedCategorie+"/"+page);
};
```

Catal.js

```
$scope.$watch(
function(){return $location.path();},
function(newPath){
//console.log(newPath);
  var tabPath=newPath.split("/");
  $scope.pageCourante=0;
if(tabPath.length==2){
  $scope.selectedCategorie=tabPath[1];
  $scope.chargerProduits();
}
else if(tabPath.length==3){
  $scope.selectedCategorie=tabPath[1];
  $scope.pageCourante=tabPath[2];
  $scope.chargerProduits();
};
});
 });
```

Cat.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Catalogue de produits</title>
<link rel="stylesheet" href="bootstrap-3.3.4-dist/css/bootstrap.min.css" />
<link rel="stylesheet"
href="bootstrap-3.3.4-dist/css/bootstrap-theme.min.css" />
<link rel="stylesheet" href="css/style2.css" />
</head>
<body ng-app="catalogue" ng-controller="Catcontroller">
<div class="container spacer">
<form>
<label>Catégories:</label> <select ng-model="selectedCategorie"
ng-change="gotoURL()">
<option ng-repeat="cat in categories" value="{{cat.idCategorie}}">
{{cat.nomCategorie}}</option>
</select>
</form>
</div>
```

Cat.html

```
<div class="container">
<table class="table table-striped table-hover spacer">
<thead>
<tr>
<th>REF</th>
<th>DES</th>
<th>PRIX</th>
<th>QTE</th>
</tr>
</thead>
<tbody>
<tr ng-repeat="p in produits.content">
<td>{{p.reference}}</td>
<td>{{p.designation}}</td>
<td>{{p.prix}}</td>
<td>{{p.quantite}}</td>
</tr>
</tbody>
</table>
</div>
```

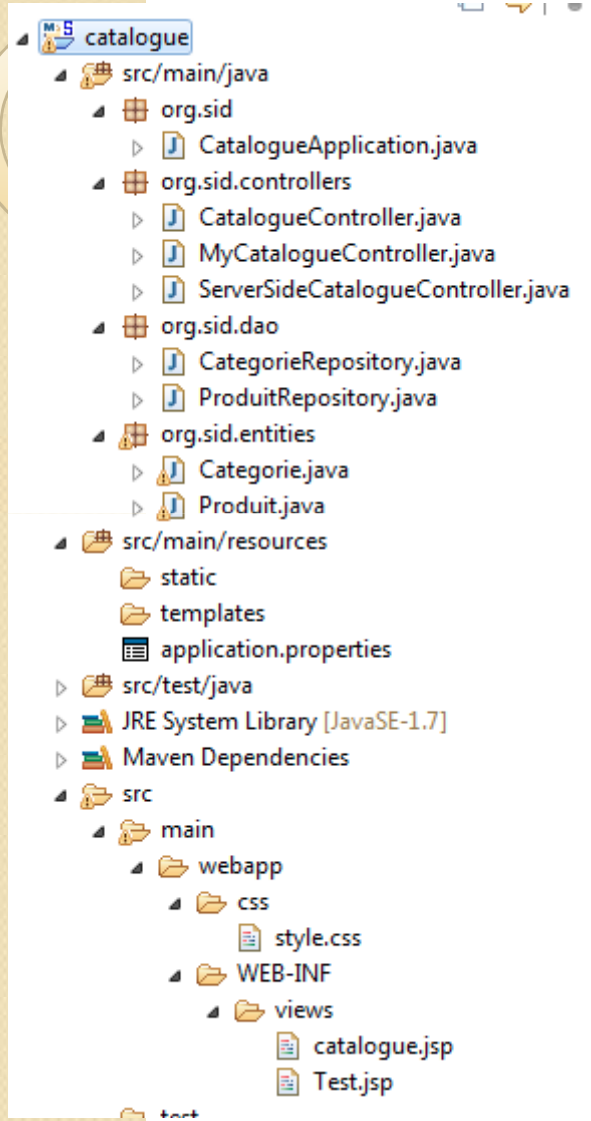
Cat.html

```
<ul ng-show="produits.totalPages>1" class="container nav nav-pills">
  <li class="clickable" ng-repeat="p in pages track by $index" ng-
  class="{active:$index==pageCourante}" >
    <a ng-click="gotoPage($index)">{{$index}}</a>
  </li>
</ul>
<script type="text/javascript" src="js/angular.min.js"></script>
<script type="text/javascript" src="js/catal.js"></script>
</body>
</html>
```

MVC SERVER SIDE
- JSP
- JSTL

med@youssfi.net | ENSET Université
Hassan II de Casablanca

Structure du projet



Catalogue.jsp

Le contrôleur

```
package org.sid.controllers;
import org.sid.dao.CategorieRepository;import org.sid.dao.ProduitRepository;
import org.sid.entities.Categorie; import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.data.domain.PageRequest; import org.springframework.stereotype.Controller;
import org.springframework.ui.Model; import org.springframework.web.bind.annotation.ModelAttribute;
import org.springframework.web.bind.annotation.RequestMapping;
@Controller
public class ServerSideCatalogueController {
@Autowired
private CategorieRepository categorieRepository;
@Autowired
private ProduitRepository produitRepository;
```

Le contrôleur

```
@RequestMapping("/catalogue")
public  String catalogue(Model model){
model.addAttribute("categories",categorieRepository.findAll());
model.addAttribute("categorie", new Categorie());
return "catalogue";
}
@RequestMapping("/produits")
public  String produits(@ModelAttribute Categorie categorie,Model
  model,int page){
model.addAttribute("categories",categorieRepository.findAll());
model.addAttribute("pageProduits",produitRepository.findByCategorie
  (categorie,new PageRequest(page, 3)));
return "catalogue";
}


}
```

catalogue.jsp

```
<%@taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@taglib uri="http://www.springframework.org/tags/form" prefix="f" %>
<!DOCTYPE html>
<html>
<head>
<title>Catalogue de produits</title>
 <link rel="stylesheet" type="text/css"
   href="<%=request.getContextPath()%>/css/style.css"/>
</head>
<body>
  <div>
    <f:form modelAttribute="categorie" action="produits?page=0">
      <table>
        <tr>
          <td>Catérogies:</td>
          <td><f:select path="idCategorie" items="${categories}"
  itemLabel="nomCategorie" itemValue="idCategorie"></f:select></td>
          <td><input  type="submit" value="OK"/></td>
        </tr>
      </table>
    </f:form>
  </div>
```

catalogue.jsp

```
<div>
    <div>
    <table class="table1">
      <tr>
        <th>REF</th><th>DES</th><th>PRIX</th><th>QTE</th> <th>DISPO</th>
  <th>PHOTO</th>
      </tr>
      <c:forEach  items="${pageProduits.getContent()}" var="p">
        <tr>
          <td>${p.getReference() }</td>
          <td>${p.designation }</td>
          <td>${p.prix }</td>
          <td>${p.quantite }</td>
          <td>${p.disponible }</td>
          <td>${p.photo }</td>
        </tr>
      </c:forEach>
    </table>
    </div>
```

catalogue.jsp

```
<div>
  <c:if test="${ not empty pageProduits and
pageProduits.getTotalPages()>=2}">
    <c:forEach begin="0" end="${pageProduits.getTotalPages() -1}"
var="p">
      <span class="autrePage">
        <c:if test="${pageProduits.getNumber()!=p }">
          <a
href="produits?page=${p}&idCategorie=${categorie.idCategorie}" >Page ${p
}</a>
        </c:if>
        <c:if test="${pageProduits.getNumber()==p }">
          Page ${p }
        </c:if>
      </span>
    </c:forEach>
  </c:if>
  </div>
 </div>
</body>
</html>
```

style.css

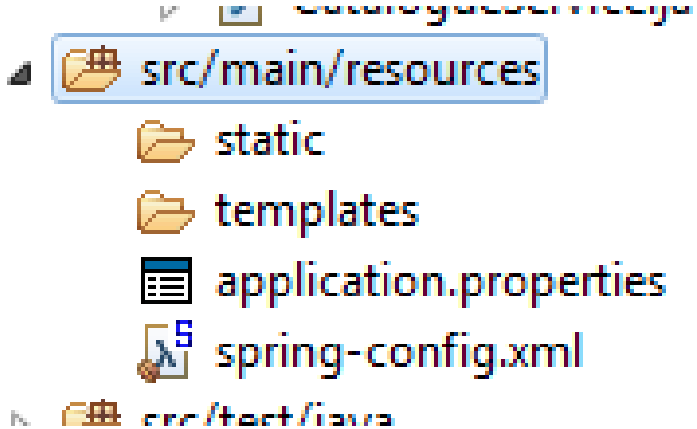
```
body{
font-family: cursive;
font-size: 12px;
}
div{
border: 1px dotted gray;
padding: 5px;
margin: 5px;
}

.table1 th{
border: 1px dotted gray;
padding: 5px;
margin: 5px;
background: pink;
}
.table1 td{
border: 1px dotted gray;
padding: 5px;
margin: 5px;
background: white;
}
```

Web Service SOAP

```
package org.sid.services;
import java.util.List; import javax.jws.WebMethod; import javax.jws.WebParam;
import javax.jws.WebService; import org.sid.dao.CategorieRepository;
import org.sid.dao.ProduitRepository; import org.sid.entities.Categorie;
import org.sid.entities.Produit; import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
@Component
@WebService
public class CatalogueService {
@Autowired
    private ProduitRepository produitRepository;
@Autowired
private CategorieRepository categorieRepository;
@WebMethod
  public List<Categorie> allCategories(){
  return categorieRepository.findAll();
  }
@WebMethod
  public List<Produit> produits(@WebParam(name="idCat")Long idCat){
Categorie c=new Categorie(); c.setIdCategorie(idCat);
  return produitRepository.findByCategorie(c, null).getContent();
  }
}
```

Déployer le web service avec spring

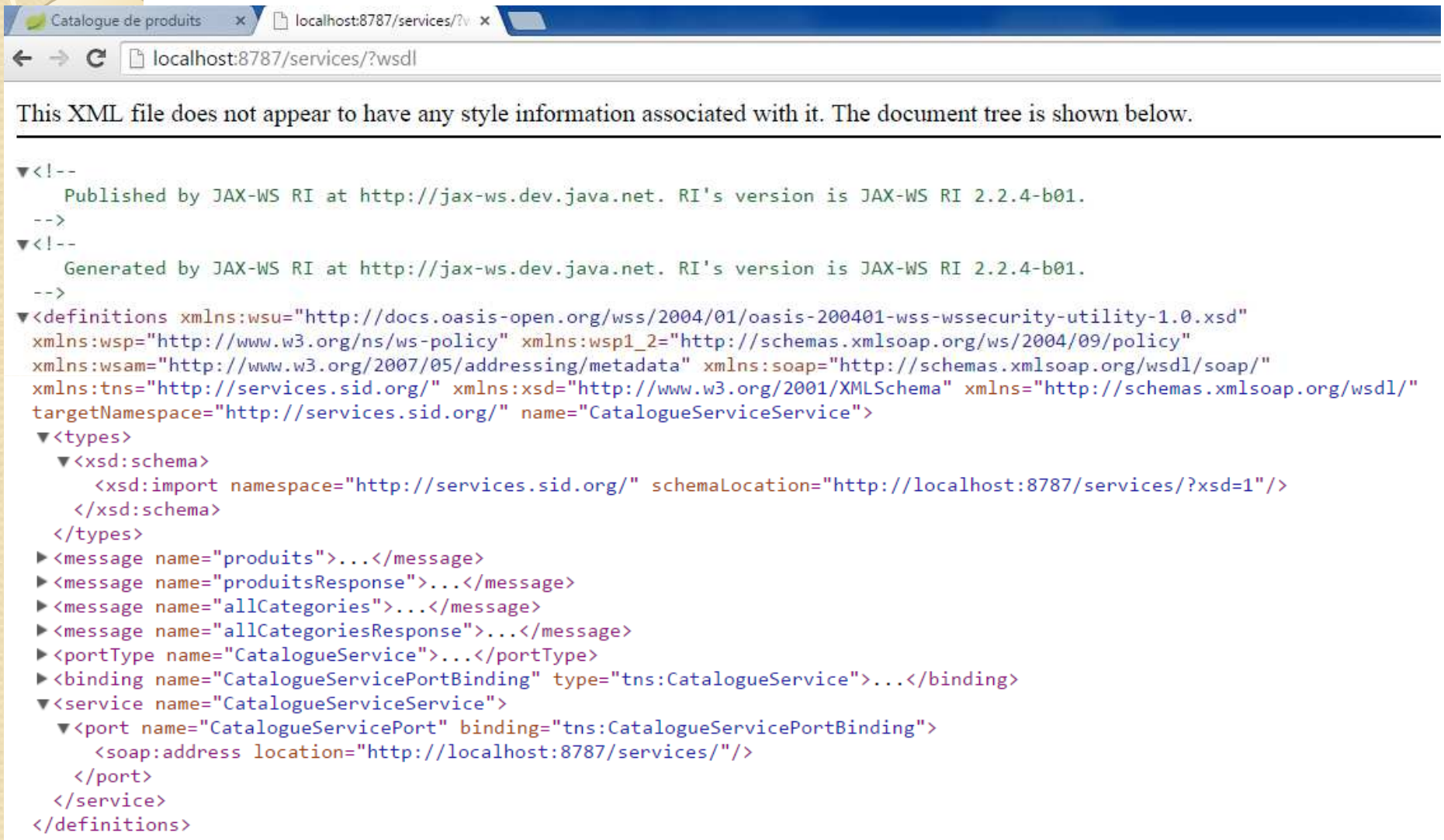
```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans.xsd">
  <bean class="org.springframework.remoting.jaxws.SimpleJaxWsServiceExporter">
     <property name="baseAddress" value="http://0.0.0.0:8787/services/"></property>
   </bean>
</beans>
```



L'application Spring Boot

```
package org.sid;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.annotation.ComponentScan;
import org.springframework.context.annotation.ImportResource;
@SpringBootApplication
@ComponentScan
@ImportResource("classpath:spring-config.xml")
@EnableAutoConfiguration
public class CatalogueApplication {
    public static void main(String[] args) {
        SpringApplication.run(CatalogueApplication.class, args);
    }
}
```

Tester le web service



Consulter les catégories

```
<SOAP-ENV:Envelope xmlns:SOAP-
    ENV="http://schemas.xmlsoap.org/soap/envelope/">
      <SOAP-ENV:Header/>
      <SOAP-ENV:Body>
        <allCategories xmlns="http://services.sid.org/"/>
      </SOAP-ENV:Body>
    </SOAP-ENV:Envelope>
```

```
<?xml version="1.0" ?>
    <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
      <S:Body>
        <ns2:allCategoriesResponse xmlns:ns2="http://services.sid.org/">
          <return>
            <idCategorie>1</idCategorie>
            <nomCategorie>Ordinateurs</nomCategorie>
          </return>
          <return>
            <idCategorie>2</idCategorie>
            <nomCategorie>Imprimantes</nomCategorie>
          </return>
        </ns2:allCategoriesResponse>
      </S:Body>
    </S:Envelope>
```

Consulter les produits d'une catégorie

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
        <SOAP-ENV:Header/>
        <SOAP-ENV:Body>
           <produits xmlns="http://services.sid.org/">
              <idCat xmlns="">2</idCat>
           </produits>
        </SOAP-ENV:Body>
     </SOAP-ENV:Envelope>
```

```
<?xml version="1.0" ?>
     <S:Envelope xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
        <S:Body>
           <ns2:produitsResponse xmlns:ns2="http://services.sid.org/">
              <return>
                 <categorie>
                    <idCategorie>2</idCategorie>  <nomCategorie>Imprimantes</nomCategorie>
                 </categorie>
                 <designation>IMP</designation>   <disponible>false</disponible>
                 <prix>1200.0</prix>  <quantite>5</quantite>
                 <reference>B</reference>
              </return>
              <return>
                 <categorie>
                    <idCategorie>2</idCategorie>   <nomCategorie>Imprimantes</nomCategorie>
                 </categorie>
                 <designation>C</designation>   <disponible>false</disponible>
                 <prix>1200.0</prix>  <quantite>5</quantite>
                 <reference>C</reference>
              </return>
           </ns2:produitsResponse>
        </S:Body>
     </S:Envelope>
```

SERVICE RMI

med@youssfi.net | ENSET Université
Hassan II de Casablanca

Interface Remote

```
package org.sid.rmi;
import java.rmi.Remote;
import java.rmi.RemoteException;
import java.util.List;
import org.sid.entities.Categorie;
import org.sid.entities.Produit;

public interface CatalogueRemote extends Remote {
  public List<Categorie> listCategories()throws RemoteException;
  public List<Produit> produitsParCategories(Long idCat)throws RemoteException;
}
```

Implémentation du service RMI

```
package org.sid.rmi;
import java.rmi.RemoteException; import java.util.List;
import org.sid.dao.CategorieRepository; import org.sid.dao.ProduitRepository;
import org.sid.entities.Categorie; import org.sid.entities.Produit;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
@Component
public class CatalogueRmiService implements CatalogueRemote {
@Autowired private ProduitRepository produitRepository;
@Autowired private CategorieRepository categorieRepository;
@Override
public List<Categorie> listCategories() throws RemoteException {
  return categorieRepository.findAll();
}
@Override
public List<Produit> produitsParCategories(Long idCat)
throws RemoteException { return produitRepository.findByCategorie(new
   Categorie(idCat), null).getContent();
}
}
```

Déploiement du service RMI

```
package org.sid;
import org.sid.rmi.CatalogueRemote; import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.remoting.jaxws.SimpleJaxWsServiceExporter;
import org.springframework.remoting.rmi.RmiServiceExporter;
@Configuration
public class MyConfiguration {
    @Autowired
    CatalogueRemote catalogueRemote;
    @Bean
    public SimpleJaxWsServiceExporter jaxWSExporter(){
           return new SimpleJaxWsServiceExporter();
    }
    @Bean
    public RmiServiceExporter rmiExporter(){
           RmiServiceExporter rmiExporter=new RmiServiceExporter();
           rmiExporter.setService(catalogueRemote);
           rmiExporter.setServiceInterface(CatalogueRemote.class);
           rmiExporter.setServiceName("CAT");
           return rmiExporter;
  }
}
```

Application Spring Boot

```
package org.sid;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.EnableAutoConfiguration;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.ComponentScan;
@SpringBootApplication
@ComponentScan
//@ImportResource("classpath:spring-config.xml")
@EnableAutoConfiguration
public class CatalogueApplication {
public static void main(String[] args) {
        SpringApplication.run(CatalogueApplication.class, args);
    }
}
```

Client RMI

```
package org.sid.rmi;
import java.rmi.Naming; import java.util.List;
import org.sid.entities.Categorie;
public class ClientRMI {
 public static void main(String[] args) {
try {
    CatalogueRemote stub=(CatalogueRemote) Naming.lookup("rmi://localhost:1099/CAT");
    List<Categorie> cats=stub.listCategories();
    for(Categorie c:cats){
      System.out.println(c.getNomCategorie());
    }
} catch (Exception e) {
  e.printStackTrace();
}
}
}
```

Structure finale du projet