JAVA JEE

Table des matières

DocPapier

documentation word: C:\Users\hb-asus\Documents\HelpNDoc\Projects\JAVA JEE.docx

documentation pdf: C:\Users\hb-asus\Documents\HelpNDoc\Projects\JAVA JEE.pdf

Toutes les installations

Nom	Modifié le	Type	Taille
apache-maven-3.3.9	10/11/2015 11:44	Dossier de fichiers	
apache-tomcat-9.0.0.M13	05/12/2016 09:58	Dossier de fichiers	
bootstrap-3.3.7-dist	16/12/2016 15:11	Dossier de fichiers	
eclipse	26/12/2016 18:45	Dossier de fichiers	
Exercices	20/11/2016 22:03	Dossier de fichiers	
installation mysql	25/11/2016 14:04	Dossier de fichiers	
mysql-connector-java-5.1.40	24/09/2016 21:35	Dossier de fichiers	
jdk-8u111-windows-x64.exe	17/11/2016 11:49	Application	199 308 Ko
jstl.jar	06/12/2016 11:39	Executable Jar File	33 Ko
jstl-1.2.jar	07/12/2016 13:50	Executable Jar File	405 Ko
standard.jar	06/12/2016 11:39	Executable Jar File	33 Ko
01-Java_v2.0.2.pdf	03/04/2014 19:12	Fichier PDF	3 454 Ko
Programmation_Java-fr.pdf	10/05/2016 11:22	Fichier PDF	741 Ko
eclipse-jee-neon-1a-win32-x86_64.zip	17/11/2016 11:20	zip Archive	308 878 Ko

NOTIONS GENERALES

A connaitre / Tutoriel

Pour debbuger il faut mettre les points d'arret sur *.java

JSP JavaServer Page.....: https://www.tutorialspoint.com/jsp/jsp_overview.htm
Maven repository.....:

Raccourcis

ctrl /T --> pour vérifier les arborescences
alt F5 --> pour mettre à jour les dépendances
ctrl 1 --> pour assigner une variable
ctrl 2 L --> locale
ctrl 2 F --> field

Syntaxe dans un java

Ajouter les annotations de spring @ :
Instancier un objet ModelAndView en final
Lui ajouter les objets avec addObject("nomUtiliséDansLaJSP", VariableLocale)
Ajouter le nom de la vue avec setViewName

```
@Controller
public class ProduitController {

	@RequestMapping("ListeProduit")
	public ModelAndView getList() {

		final ModelAndView mav = new ModelAndView();

		Produit produit1 = new Produit("art1", 21, "rouge");
		Produit produit2 = new Produit("art2", 21, "bleu");
		Produit produit3 = new Produit("art3", 21, "vert");
		Produit produit4 = new Produit("art4", 21, "noir");

		ArrayList<Produit> tableau = new ArrayList<>();

		tableau.add(produit1);
		tableau.add(produit2);
		tableau.add(produit3);
		tableau.add(produit4);

		mav.addObject("liste", tableau);

		mav.setViewName("ListeProduit");
		return mav;
	}
```

Syntaxe jstl

Dans un fichier jsp, il faut ajouter :

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

<head><meta http-equiv="Content-Type" content="text/html; charset=UTF-8"></head>
```

Utilisation de la librairie core :

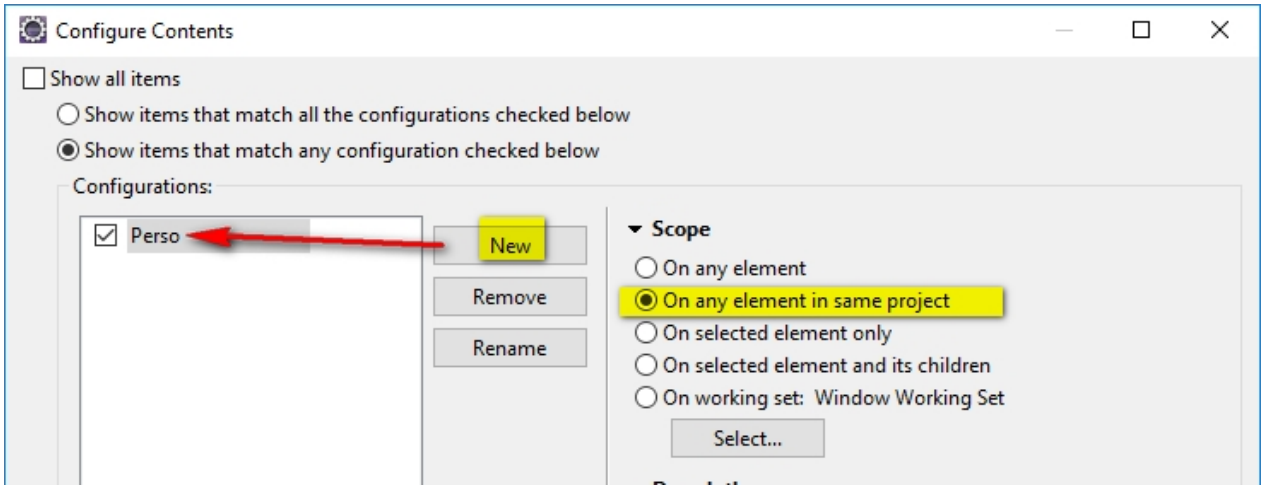
```
<c:forEach items="" var="">       </c:forEach>

<c:if test="" >             </c:if>

<c:out value=""></c:out>
```

Paramétrage Eclipse

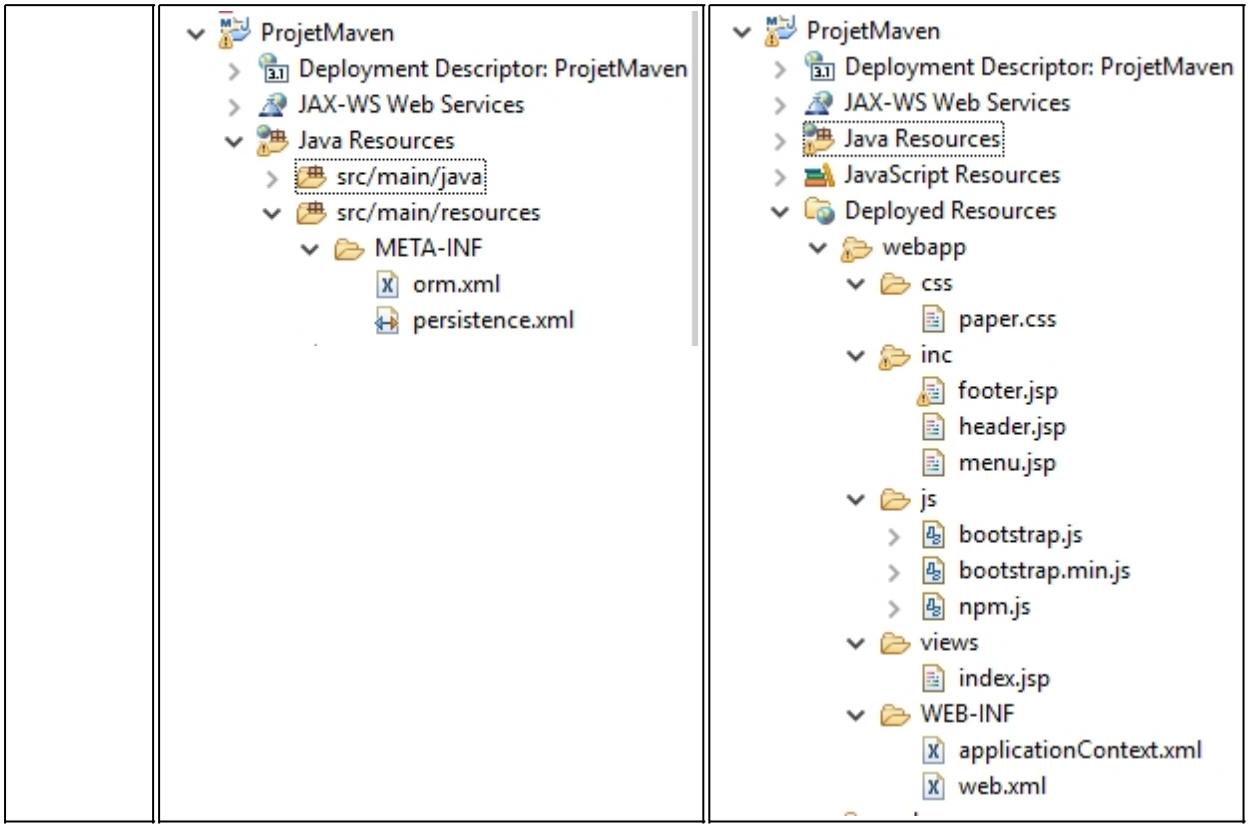
Créé avec HelpNDoc Personal Edition: Générateur facile de livres électroniques et documentation

Config Markers

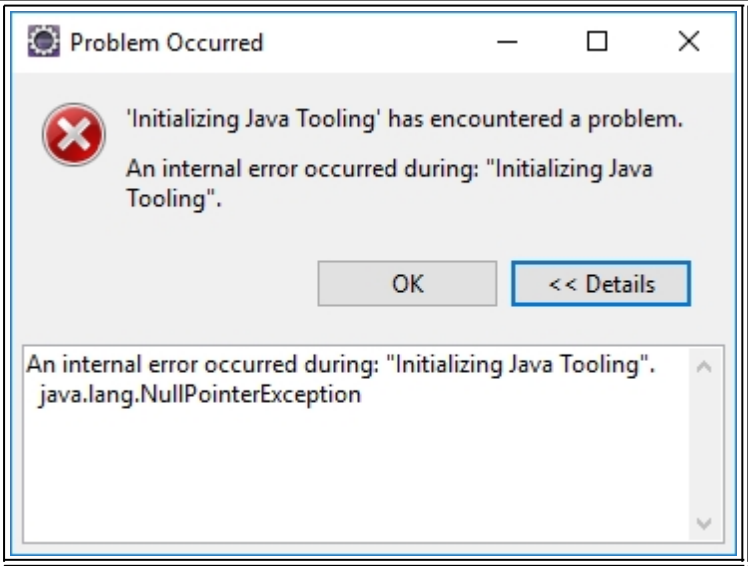
Pour ne voir dans la console **Markers** que les erreurs du projet en cour : (à partir de la petite flèche)



Cartographie



Solutions erreurs



org.springframework.beans.factory.NoSuchBeanDefinitionException: **No qualifying bean** of type	Le problème était que je n'avais pas déclarer mes packages dao et services dans mon fichier servlet-dispatch.xml>

'fr.formation.SERVICE.IngredientService' available: expected at least 1 bean which qualifies as autowire candidate. Dependency annotations: {@org.springframework.beans.factory.annotation.Autowired(required=true)}	Du coup j'en ai déduit que chaque package qui contient des classes utilisant des annotations du framework Spring doit être déclaré dans ce fichier XML !

MAVEN

Quelques commandes utilisées lors du projet :

- mvn package
- mvn dependency:purge-local-repository
- **mvn clean package**

Installation

MAVEN : Outil de gestion de projet

POM : Project Object Model

1) Déclaration des variables d'environnement dans windows

Après avoir téléchargé Maven
Ajouter JAVA_HOME lien vers le répertoire Java/jdk1.8.0_11
Ajouter dans Path le répertoire bin de Maven

Créer l'arborescence des dossier sous le nx projet sous workspace :

Structure du dossier
C:\Users\hb-asus\workspace\CocktailBar

```
C:.
└───src
    ├───main
    │   ├───java
    │   ├───resources
    │   └───webapp
    │       └───WEB-INF
    └───test
        ├───java
```

```
└──resources
```

Par la suite avec la BDD

```
C:.
│   .classpath
│   .project
│   pom.xml
│
├──src
│   ├──main
│       ├──java
│       │   └──fr
│       │       └──formation
│       │           ├──CONTROLLER
│       │           │       IngredientController.java
│       │           │       MainController.java
│       │           ├──DAO
│       │           │       IngredientDAO.java
│       │           ├──entity
│       │           │       Ingredient.java
│       │           ├──MODEL
│       │           │       Menu.java
│       │           └──service
│       │                   IngredientService.java
│       │
│       ├──resources
│       │   │       menu.properties
│       │   └──META-INF
│       │           orm.xml
│       │           persistence.xml
│       │
│       └──webapp
│           ├──css
│           │        paper.css
│           ├──inc
│           │       footer.jsp
│           │       header.jsp
│           ├──views
│           │       addIngredient.jsp
│           │       index.jsp
│           │       ingredients.jsp
│           └──WEB-INF
│                   applicationContext.xml
│                   web.xml
```

liste settings

```
.jsdtscope
org.eclipse.core.resources.prefs
org.eclipse.jdt.core.prefs
org.eclipse.jpt.core.prefs
org.eclipse.m2e.core.prefs
org.eclipse.wst.common.component
org.eclipse.wst.common.project.facet.core.prefs.xml
org.eclipse.wst.common.project.facet.core.xml
org.eclipse.wst.jsdt.ui.superType.container
```

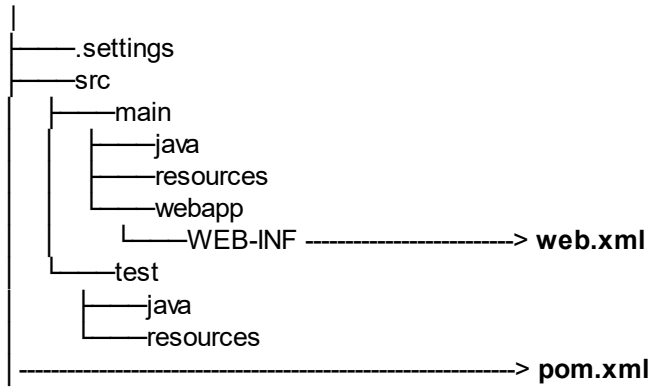
org.eclipse.wst.jsdt.ui.superType.name
org.eclipse.wst.validation.prefs

2) Dans éclipse

Dans éclipse il faut créer deux fichiers :

pom.xml et web.xml

C:.workspace/projet



3) Définition du pom.xml

```
Aller sous mvn repository pour copier les repository
Faire alt+F5 pour mettre à jour les libraries dans le projet
```

```
<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>fr.formation</groupId>
	<artifactId>cocktailBar</artifactId>
	<version>0.0.1</version>
	<packaging>war</packaging>

	<properties>
		<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
	</properties>

	<dependencies>
		<!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
		<dependency>
			<groupId>javax.servlet</groupId>
			<artifactId>javax.servlet-api</artifactId>
			<version>3.1.0</version>
		</dependency>

		<!-- https://mvnrepository.com/artifact/javax.servlet.jsp.jstl/jstl -->
		<dependency>
			<groupId>javax.servlet</groupId>
			<artifactId>jstl</artifactId>
			<version>1.2</version>
```

```
		</dependency>

		<!-- https://mvnrepository.com/artifact/org.springframework/spring-context
-->
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-context</artifactId>
			<version>4.3.4.RELEASE</version>
		</dependency>

		<!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc
-->
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-webmvc</artifactId>
			<version>4.3.4.RELEASE</version>
		</dependency>



	</dependencies>

	<!-- Configuration des plugins -->
	<build>
		<plugins>
			<plugin>
				<groupId>org.apache.maven.plugins</groupId>
				<artifactId>maven-compiler-plugin</artifactId>
				<version>2.5.1</version>
				<configuration>
					<source>1.8</source>
					<target>1.8</target>
				</configuration>
			</plugin>
		</plugins>

	</build>

</project>
```


4) Les liens

https://mvnrepository.com/artifact/org.springframework/spring-webmvc/4.3.4.RELEASE

Nouveau chapitre

Pour utiliser Jquery, ajouter les dépendances suivantes dans pom.xml

JQUERY-UI 1.12.1
JQUERY-UI datatables 1.12.1

JQUERY 3.11.1

JQUERY datatables colreorder 1.2.0

Le code Jquery peut être copié depuis datatables.net puis
-> Examples -->Advanced initialisation
-> DOM/JQuery events

SPRING

Installation

Obtenir de l'aide

Fonctionnement du déploiement

Désolé cette prise de note est en relation directe avec ma compréhension sur le moment... vous comprendrez en la lisant....
Merci Jéremy d'être aller si vite ;-(

Projet/src/main/WEB-INF/applicationContext.xml

Ajouter un rép source/main/**views**

1) Fonctionnement du déploiement

Tomcat/bin > startup.bat
lib
log

firstservlet.war sous webapp

invalidelockHeader -> vider le repository local de maven

tmp wtpwebapps
temp sous tomcat
work fichier temp mais peut être supprimer et se régénère cache de tomcat cote appli
conf

2) Commande mvn :
mvn clean package

3) properies cocktail
web project settings ==> nom du projet en minuscule

Menu Projet > Clean

4) Création file général : Java resources/src/main/resources/menu.properties

@Annotation

Les annotations

```
	@Autowired : permet de voir qu'on surcharge la méthode

final classe et methode empèche la surcharge ou extends
```


Projet Cocktail avant BDD

Cocktail
--> Nom
--> Ingrédients
--> Prix
--> Alcoolisé

Ingrédient
--> nom
--> Quantité
--> Etat

Menu :
Liste des cocktails
Liste des ingrédients
Ajouter un cocktail
Ajouter un ingrédient
Rechercher

pom.xml

```
<!--  ____________________________________________________________ -->
<!-- Aller sous mvn repository pour copier les repository          -->
<!-- Faire alt+F5 pour mettre à jour les libraries dans le projet  -->
<!-- Posé à la racine du projet                                 --  >
<!--  ____________________________________________________________ -->


<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>fr.formation</groupId>
	<artifactId>cocktailBar</artifactId>
	<version>0.0.1</version>
	<packaging>war</packaging>

	<properties>
```

```
		<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
	</properties>

	<dependencies>
		<!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
		<dependency>
			<groupId>javax.servlet</groupId>
			<artifactId>javax.servlet-api</artifactId>
			<version>3.1.0</version>
		</dependency>

		<!-- https://mvnrepository.com/artifact/javax.servlet.jsp.jstl/jstl -->
		<dependency>
			<groupId>javax.servlet</groupId>
			<artifactId>jstl</artifactId>
			<version>1.2</version>
		</dependency>

		<!-- https://mvnrepository.com/artifact/org.springframework/spring-context
-->
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-context</artifactId>
			<version>4.3.4.RELEASE</version>
		</dependency>

		<!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc
-->
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-webmvc</artifactId>
			<version>4.3.4.RELEASE</version>
		</dependency>



	</dependencies>

	<!-- Configuration des plugins -->
	<build>
		<plugins>
			<plugin>
				<groupId>org.apache.maven.plugins</groupId>
				<artifactId>maven-compiler-plugin</artifactId>
				<version>2.5.1</version>
				<configuration>
					<source>1.8</source>
					<target>1.8</target>
				</configuration>
			</plugin>
		</plugins>

	</build>

</project>
```


web.xml

```
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
```

```
          xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
          xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
          version="3.1">
       <display-name>Appli Cocktail</display-name>

<servlet>
       <servlet-name>SpringServlet</servlet-name>
       <servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
       <init-param>
              <param-name>contextConfigLocation</param-name>
              <param-value>/WEB-INF/applicationContext.xml</param-value>
       </init-param>
</servlet>

<servlet-mapping>
       <servlet-name>SpringServlet</servlet-name>
       <url-pattern>*.html</url-pattern>
</servlet-mapping>

</web-app>
```


applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:context="http://www.springframework.org/schema/context"
       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
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd">
       <context:component-scan base-
package
="fr.formation.CONTROLLER,fr.formation.DAO,fr.formation.MODEL,fr.formation.service"></
context:component-scan>

       <!-- nom de la propriété -->
       <!-- view classe -->
       <bean id="viewResolver"
              class="org.springframework.web.servlet.view.UrlBasedViewResolver">
              <property name="viewClass"
                     value="org.springframework.web.servlet.view.JstlView" />
              <property name="prefix" value="/views/" />
              <property name="suffix" value=".jsp" />
       </bean>

       <bean id="messageSource"

class="org.springframework.context.support.ReloadableResourceBundleMessageSource">
              <property name="basename" value="classpath:/menu" />
       </bean>
</beans>
```


menu.properties

```
# Liste des menus de l'application
```

```
menu.list=cocktailList, ingredientList, addCocktail, addIngredient, search

menu.cocktailList.title=Liste des cocktails
menu.cocktailList.url=/cocktails

menu.ingredientList.title=Liste des ingredients
menu.ingredientList.url=/ingredients

menu.addCocktail.title=Ajouter un cocktail
menu.addCocktail.url=/cocktail/add

menu.addIngredient.title=Ajouter un ingredient
menu.addIngredient.url=/ingredient/add

menu.search.title=Recherche
menu.search.url=/search
```

persistant	Business	Presentation
menu.properties -->	---> MessageSource | List<Menu> | ModelAndView -->	-->index.jsp

IngredientController.java

```java
package fr.formation.CONTROLLER;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

import fr.formation.service.IngredientService;

@Controller
@RequestMapping("/ingredients")

public class IngredientController {

	@Autowired
	private IngredientService service;

	@RequestMapping
	public ModelAndView list() {
		final ModelAndView mav = new ModelAndView();
		mav.setViewName("ingredients");

		mav.addObject("ingredients", this.service.getAll());


		return mav;
	}

}
```

Menu.java

```
package fr.formation.MODEL;

public class Menu {

	final private String title;
	final private String url;

	/**
	 *
	 * @param title titre du menu
	 * @param url url de lien d'ouverture du menu
	 */
	public Menu(String title, String url){
		this.title = title;
		this.url = url;
	}

	public String getTitle() {
		return title;
	}

	public String getUrl() {
		return url;
	}


}
```


IngredientService.java

```
package fr.formation.service;

import java.util.List;

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

import fr.formation.DAO.IngredientDAO;
import fr.formation.entity.Ingredient;

@Service
public class IngredientService {

	@Autowired
	private IngredientDAO dao;

	public List<Ingredient> getAll(){
		return this.dao.readAll();

	}
}
```

IngredientDAO.java

```
package fr.formation.DAO;

import java.util.Arrays;
import java.util.List;

import org.springframework.stereotype.Component;

import fr.formation.entity.Ingredient;

@Component
public class IngredientDAO  {



	public List<Ingredient> readAll(){
		return Arrays.asList(new Ingredient(0, "Rhum"),
							new Ingredient(0,"Whiskey"),
							new Ingredient(0,"Tequila"),
							new Ingredient(1, "Ice cubes"),
							new Ingredient(1, "Sugar"),
							new Ingredient(2, "CO2"));

	}
}
```


Ingredient.java

```
package fr.formation.entity;

import java.io.Serializable;

import org.springframework.web.servlet.mvc.method.annotation.ResponseBodyEmitterReturnValueHandler;

public class Ingredient implements Serializable {

	private static final long serialVersionUID = 1L;

	private int etat;
	private String name;

	public Ingredient(){
	}

	public Ingredient(int etat, String name) {
		this.etat = etat;
		this.name = name;
	}

	/**
	 * @return the etat
	 */
	public int getEtat() {
```

```
		return etat;
	}

	public void setEtat(int etat) {
		this.etat = etat;
	}

	public String getName() {
		return name;
	}

	public void setName(String name) {
		this.name = name;
	}

}
```


MainController.java

```
package fr.formation.CONTROLLER;

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

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.MessageSource;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

import fr.formation.MODEL.Menu;

@Controller
public class MainController {

	@Autowired
	private MessageSource messages;

	// Object model and view, pas d'importance sur le nom de la méthode
	@RequestMapping("/index")
	public ModelAndView index() {
		final ModelAndView  mav = new ModelAndView() ;
		mav.setViewName("index");
		// recup list de menu
		final List<String> menuKeys = Arrays.asList(getMessage("menu.list").split(","));
		final List<Menu> menus = new ArrayList<>();

		for (final String menuKey : menuKeys) {
			final String prefix = "menu."+ menuKey.trim();
			final String title =  getMessage(prefix+ ".title");
			final String url = getMessage(prefix+ ".url");
			menus.add(new Menu(title, url));
		}

		mav.getModel().put("menus", menus);
		return mav;
```

```
	}

	private String getMessage(final String key) {
		return this.messages.getMessage(key, null,null);
	}

}
```


Placer correctement les fichiers

```
C:.
├───.settings
├───src
│   ├───main
│   │   ├───java
│   │   │   └───fr
│   │   │       └───formation
│   │   │           ├───CONTROLLER
│   │   │           ├───DAO
│   │   │           ├───entity
│   │   │           ├───MODEL
│   │   │           └───service
│   │   ├───resources
│   │   └───webapp
│   │       ├───views
│   │       └───WEB-INF
│   └───test
│       ├───java
│       └───resources
└───target
    ├───classes
    │   └───fr
    │       └───formation
    │           ├───CONTROLLER
    │           ├───DAO
    │           ├───entity
    │           ├───MODEL
    │           └───service
    ├───cocktailBar-0.0.1
    │   ├───META-INF
    │   ├───views
    │   └───WEB-INF
    │       ├───classes
    │       │   └───fr
    │       │       └───formation
    │       │           └───CONTROLLER
    │       └───lib
    ├───generated-sources
    │   └───annotations
    ├───m2e-wtp
    │   └───web-resources
    │       └───META-INF
    │           └───maven
    │               └───fr.formation
    │                   └───cocktailBar
    ├───maven-archiver
    └───test-classes
```

ingredient.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
	pageEncoding="UTF-8"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/
html4/loose.dtd">

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Liste des ingrédients</title>
</head>
<body>

	<h1>Liste des ingrédients</h1>
	<table>
		<thead>
			<tr>
				<th>Nom</th>
				<th>Etat</th>
			</tr>
		</thead>

		<tbody>
			<c:forEach items="${ingredients}" var="ingredient">
				<tr>
				<td>${ingredient.name}</td>
				<td>${ingredient.etat}</td>
				</tr>
			</c:forEach>

		</tbody>
	</table>


</body>
</html>
```


index.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
	pageEncoding="UTF-8"%>

<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/
html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Gestion des cocktails</title>
<META HTTP-EQUIV="Refresh" CONTENT="5">
</head>
<body>
	<h1>Bienvenue dans l'applicationCocktail</h1>

	<div>
		Menu :
```

```
            <ul>
                  <c:forEach items="${menus}" var="menu">
                        <c:url value="${menu.url}.html" var="menuUrl"></c:url>
                        <li><a href="${menuUrl}">${menu.title}</a></li>
                  </c:forEach>
            </ul>
      </div>
</body>
</html>
```


pom.properties

menu.properties

```
# Liste des menus de l'application

menu.list=cocktailList, ingredientList, addCocktail, addIngredient, search

menu.cocktailList.title=Liste des cocktails
menu.cocktailList.url=/cocktails

menu.ingredientList.title=Liste des ingredients
menu.ingredientList.url=/ingredients

menu.addCocktail.title=Ajouter un cocktail
menu.addCocktail.url=/cocktail/add

menu.addIngredient.title=Ajouter un ingredient
menu.addIngredient.url=/ingredient/add

menu.search.title=Recherche
menu.search.url=/search
```

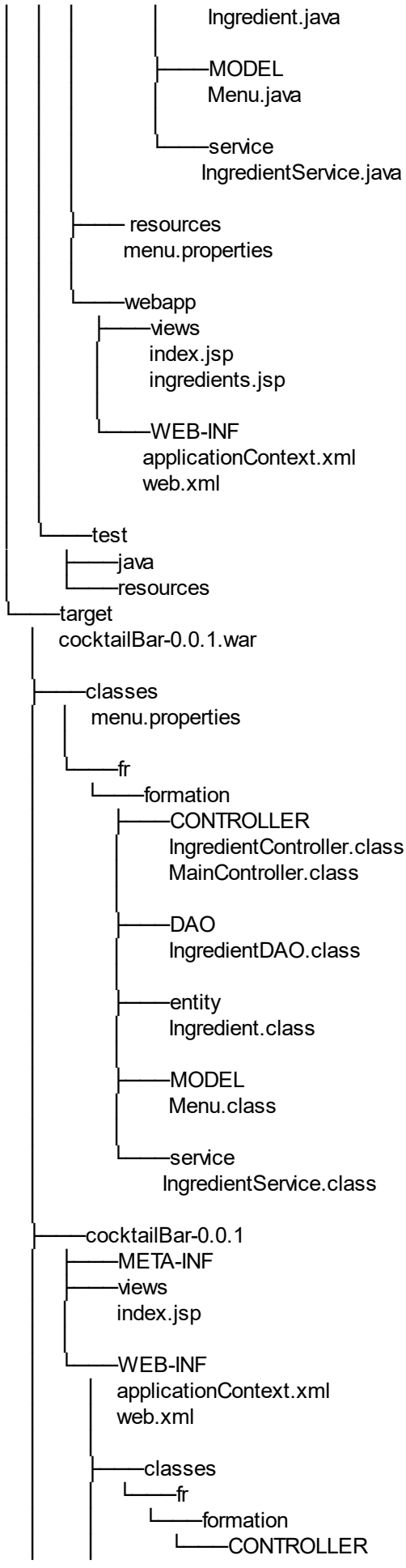

Structure

```
C:.
    .classpath
    .project
    pom.xml

├───src
    ├───main
        ├───java
            └───fr
                └───formation
                    ├───CONTROLLER
                        IngredientController.java
                        MainController.java

                    ├───DAO
                        IngredientDAO.java

                    ├───entity
```


Ingredient.java
MODEL
Menu.java
service
IngredientService.java
resources
menu.properties
webapp
views
index.jsp
ingredients.jsp
WEB-INF
applicationContext.xml
web.xml
test
java
resources
target
cocktailBar-0.0.1.war
classes
menu.properties
fr
formation
CONTROLLER
IngredientController.class
MainController.class
DAO
IngredientDAO.class
entity
Ingredient.class
MODEL
Menu.class
service
IngredientService.class
cocktailBar-0.0.1
META-INF
views
index.jsp
WEB-INF
applicationContext.xml
web.xml
classes
fr
formation
CONTROLLER

```
        MainController.class
    └───lib
        commons-logging-1.2.jar
        javax.servlet-api-3.1.0.jar
        jstl-1.2.jar
        spring-aop-4.3.4.RELEASE.jar
        spring-beans-4.3.4.RELEASE.jar
        spring-context-4.3.4.RELEASE.jar
        spring-core-4.3.4.RELEASE.jar
        spring-expression-4.3.4.RELEASE.jar
        spring-web-4.3.4.RELEASE.jar
        spring-webmvc-4.3.4.RELEASE.jar
├───generated-sources
│   └───annotations
├───m2e-wtp
│   └───web-resources
│       └───META-INF
│           MANIFEST.MF
│
│       └───maven
│           └───fr.formation
│               └───cocktailBar
│                       pom.properties
│                       pom.xml
```


Projet Cocktail avec BDD

orm.xml	Déclaration de la structure des tables mysql
	
	`<entity class="fr.formation.entity.Ingredient">`
	`<table name="ingredient"></table>`
	`<attributes>`
	`<id name="id">`
	`<column name="id_ing" />..................-> nom du champ`
	`<generated-value strategy="IDENTITY" />...-> indique que c'est une clé`
	`</id>`
	
	`<basic name="etat">.....................-> nom dans Java`
	`<column name="state" />.............-> nom dans mysql`
	`</basic>`
	`</attributes>`
	`</entity>`
	
	`<entity class="chemin complet jusqu'à la classe java">`
	`<table name="nom_table"></table>`
	
	`<basic name="idJava">`
	`<column name="id_mysql" />`
	`</basic>`
	
	`</attributes>`
	`</entity>`

<table>
<tr><td></td><td></td></tr>
<tr><td>persistence.xml</td><td>Définit le "DriverManager"</td></tr>
<tr><td>pom.xml</td><td>Ajout des dépendences utiles pour la base :

1) "mysql connector"
2) "spring data jpa
3) "hibernate core" dernière version
4) Copie "hibernate core" changer core en hibernate-entitymanager</td></tr>
<tr><td>applicationContext.xml</td><td>Spring attent entityManagerEntity Name pris dans persistance.xml -->

1. Ajout de 2 beans :
○ entityManagerFactory
○ transactionManager

2. Ajout <jpa:repositories>

base-package="fr.formation.DAO"

<bean id="entityManagerFactory"

class
="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
<property name="persistenceUnitName" value="bar"></property>

</bean>

<bean id="transactionManager"
class="org.springframework.orm.jpa.JpaTransactionManager">
<property name="entityManagerFactory"
ref="entityManagerFactory" />
</bean></td></tr>
<tr><td>IngredientController.java</td><td>Ajout de ModelAndView avec un appel à la jsp qui fait action</td></tr>
<tr><td></td><td>

```
package fr.formation.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import fr.formation.DAO.IngredientDAO;
import fr.formation.entity.Ingredient;

@Service
public class IngredientService {

	@Autowired
	private IngredientDAO dao;

	public List<Ingredient> getAll(){
		return this.dao.findAll();

	}
	@Transactional
	public Ingredient create(final Ingredient ingredient){
		return this.dao.save(ingredient);
	}
}
```

</td></tr>
</table>

orm.xml

```
<?xml version="1.0"?>
<entity-mappings
	xmlns="http://www.eclipse.org/eclipselink/xsds/persistence/orm"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://www.eclipse.org/eclipselink/xsds/persistence/orm
http://www.eclipse.org/eclipselink/xsds/eclipselink_orm_2_1.xsd"
	version="2.1">

	<entity class="fr.formation.entity.Ingredient">
		<table name="ingredient"></table>
		<attributes>
			<id name="id">
				<column name="id_ing" />
				<generated-value strategy="IDENTITY" />
			</id>

			<basic name="etat">
				<column name="state" />
			</basic>

			<basic name="nom">
				<column name="name" />
			</basic>
		</attributes>
	</entity>
</entity-mappings>
```


persistence.xml

```
<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"
		xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
		xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
		http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd"
		version="2.1">

<!-- On met ce que l'on veut -->
<persistence-unit name="bar" >
	 <!-- une classe d'hibernate qui va nous servir -->
	<provider>org.hibernate.jpa.HibernatePersistenceProvider</provider>
	<mapping-file>META-INF/orm.xml</mapping-file>
	<properties>
		<property name="hibernate.connection.driver_class"
value="com.mysql.jdbc.Driver"/>
		<property name="hibernate.connection.url" value="jdbc:mysql://
localhost:3306/cocktail"/>
		<property name="hibernate.connection.user" value="root"/>
		<property name="hibernate.connection.password" value=""/>
		<property name="hibernate.dialect"
value="org.hibernate.dialect.MySQL5Dialect"/>
	</properties>
</persistence-unit>
```

```
</persistence>
```


pom.xml

```
<!-- ____________________________________________________________ -->
<!-- Aller sous mvn repository pour copier les repository -->
<!-- Faire alt+F5 pour mettre à jour les libraries dans le projet -->
<!-- Posé à la racine du projet -->
<!-- ____________________________________________________________ -->


<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>fr.formation</groupId>
	<artifactId>cocktailBar</artifactId>
	<version>0.0.1</version>
	<packaging>war</packaging>

	<properties>
		<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
	</properties>

	<dependencies>
		<!-- https://mvnrepository.com/artifact/javax.servlet/javax.servlet-api -->
		<dependency>
			<groupId>javax.servlet</groupId>
			<artifactId>javax.servlet-api</artifactId>
			<version>3.1.0</version>
		</dependency>

		<!-- https://mvnrepository.com/artifact/javax.servlet.jsp.jstl/jstl -->
		<dependency>
			<groupId>javax.servlet</groupId>
			<artifactId>jstl</artifactId>
			<version>1.2</version>
		</dependency>

		<!-- https://mvnrepository.com/artifact/org.springframework/spring-context
-->
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-context</artifactId>
			<version>4.3.4.RELEASE</version>
		</dependency>

		<!-- https://mvnrepository.com/artifact/org.springframework/spring-webmvc
-->
		<dependency>
			<groupId>org.springframework</groupId>
			<artifactId>spring-webmvc</artifactId>
			<version>4.3.4.RELEASE</version>
		</dependency>


<!-- Dépendance à ajouter pour les accés à la base -->

<!-- 1) "mysql connector" Ajout BDD -->
```

```
		<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
		<dependency>
			<groupId>mysql</groupId>
			<artifactId>mysql-connector-java</artifactId>
			<version>5.1.40</version>
		</dependency>

<!-- 2) "spring data jpa"  -->

		<!-- https://mvnrepository.com/artifact/org.springframework.data/spring-data-jpa -->
		<dependency>
			<groupId>org.springframework.data</groupId>
			<artifactId>spring-data-jpa</artifactId>
			<version>1.10.5.RELEASE</version>
		</dependency>

		<!-- 3) "hibernate core" dernière version -->
		<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
		<dependency>
			<groupId>org.hibernate</groupId>
			<artifactId>hibernate-core</artifactId>
			<version>5.2.5.Final</version>
		</dependency>

		<!-- 4) Copie "hibernate core" changer core en hibernate-entitymanager-->

		<!-- https://mvnrepository.com/artifact/org.hibernate/hibernate-core -->
		<dependency>
			<groupId>org.hibernate</groupId>
			<artifactId>hibernate-entitymanager</artifactId>
			<version>5.2.5.Final</version>
		</dependency>

	</dependencies>

	<!-- Configuration des plugins -->
	<build>
		<plugins>
			<plugin>
				<groupId>org.apache.maven.plugins</groupId>
				<artifactId>maven-compiler-plugin</artifactId>
				<version>2.5.1</version>
				<configuration>
					<source>1.8</source>
					<target>1.8</target>
				</configuration>
			</plugin>
		</plugins>

	</build>

</project>
```


applicationContext.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:jpa="http://www.springframework.org/schema/data/jpa"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/data/jpa
        http://www.springframework.org/schema/data/jpa/spring-jpa.xsd">
       <context:component-scan base-
package="fr.formation.CONTROLLER,fr.formation.MODEL,fr.formation.service"></
context:component-scan>

       <!-- nom de la propriété -->
       <!-- view classe -->
       <bean id="viewResolver"
.....
       <bean id="messageSource"
.....
       <!-- Spring attent entityManagerEntity  NAme pris dans persistance.xml -->
       <bean id="entityManagerFactory"
              class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
              <property name="persistenceUnitName" value="bar"></property>
       </bean>

       <bean id="transactionManager"
              class="org.springframework.orm.jpa.JpaTransactionManager">
              <property name="entityManagerFactory" ref="entityManagerFactory" />
       </bean>


       <jpa:repositories base-package="fr.formation.DAO"></jpa:repositories>


</beans>
```


IngredientController.java

```
package fr.formation.CONTROLLER;

import javax.servlet.http.HttpServletRequest;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.ModelAndView;

import fr.formation.entity.Ingredient;
import fr.formation.service.IngredientService;

@Controller
@RequestMapping("/ingredients")

public class IngredientController {

       @Autowired
       private IngredientService service;
```

```
	@RequestMapping
	public ModelAndView list() {
		final ModelAndView mav = new ModelAndView();
		mav.setViewName("ingredients");

		mav.addObject("ingredients", this.service.getAll());
		return mav;
	}

	@RequestMapping("/add")
	public ModelAndView add() {
		final ModelAndView mav = new ModelAndView();
		mav.setViewName("addIngredient");
		mav.addObject("ingredients", this.service.getAll());

		return mav;
	}

	@RequestMapping(value = "/add", method = RequestMethod.POST)
	public String newIngredient(final HttpServletRequest request) {

		final String name = request.getParameter("name");
		final Integer state = Integer.parseInt(request.getParameter("state"));
		// ----> ajout @Transactional dans IngredientService.java
		this.service.create(new Ingredient(state, name));
		return "redirect:/ingredients/add.html";
	}

	@RequestMapping(value = "/add2", method = RequestMethod.POST)
	public String newIngredient(@RequestParam final String name, @RequestParam final
Integer state ) {

		// ----> ajout @Transactional dans IngredientService.java

		this.service.create(new Ingredient(state, name));
		return "redirect:/ingredients/add.html";
	}

}
```


IngredientDAO.java

```
package fr.formation.DAO;

import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.stereotype.Repository;

import fr.formation.entity.Ingredient;


@Repository
public interface IngredientDAO  extends JpaRepository<Ingredient, Integer>{


}
```

IngredientService.java

```
package fr.formation.service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import fr.formation.DAO.IngredientDAO;
import fr.formation.entity.Ingredient;

@Service
public class IngredientService {

	@Autowired
	private IngredientDAO dao;

	public List<Ingredient> getAll(){
		return this.dao.findAll();

	}
	@Transactional
	public Ingredient create(final Ingredient ingredient){
		return this.dao.save(ingredient);
	}
}
```


AddIngredient.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
	pageEncoding="UTF-8"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/
html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Ajout d'un ingrédient</title>
</head>
<body>

<h1>AJOUT D'UN INGREDIENT</h1>
	<c:url value="/ingredients/add.html" var="addUrl" />
	<form action="${addUrl}"  method="POST">
		<label for="name">Nom : </label>
		<input id="name" name="name" class="form-control" />
		<label for="state">Etat : </label>
		<input id="state" name="state" type="number" min="0" max="2" class="form-
control" />
		<button>VALIDER</button>
	</form>

	<div style="position: fixed; bottom: 0; left: 0; padding: 20px; font-size: 18px;">
		<a href="<c:url value='/'  />" >RETOUR</a>
	</div>
```

```
</body>
</html>
```


Ingredient.jsp

Mysql

JPA **J**ava **P**ersistant **A**PI

--> persistance.xml : config qui utilise JPA , nous JPA puis spring par les repositories

ORM
Object Relational Mapping

Hibernate

1) Ajouter 3 dépendances dans pom.xml

```
1) "mysql connector" Ajout BDD -->
            https://mvnrepository.com/artifact/mysql/mysql-connector-java

2) "spring data jpa" Recherche -->

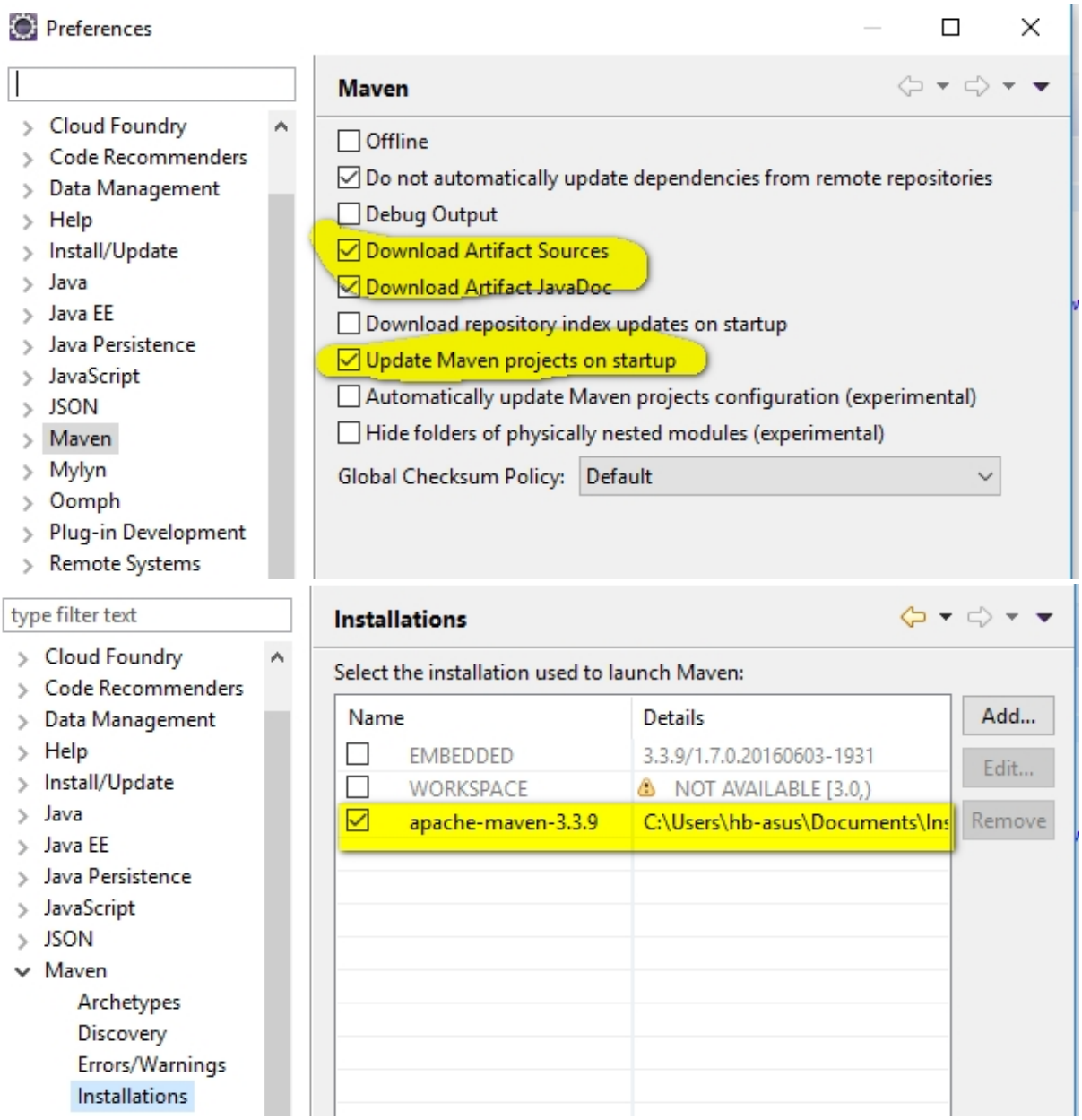
3) "hibernate core" dernière version -->

4) Copier Hibernate core et modifier artifact en hibernate-entitymanager
   <artifactId>hibernate-entitymanager</artifactId>
```

2) Sous /src/java/resources
   1. Créer un dossier META-INF
   2. puis persistence.xml
   3. puis orm.xml

===> Fichier de config JPA

3) Dans windows / preferences/ MAVEN



Pour chaque entité il faut un **id.**

Config de départ

Config tjs JPA

JPA	JAVA	SQL
PersistenceUnit "Cocktail" (*)	IngredientJava |--> new Ingredient("",0)	Table ingredient

PersistenceContext (**) --> contient les informations Unit	**EntityManagerFactory** : on lui configure le nom de notre entité = "Cocktail" (*) (**) en JPA on a un persistence objet **EntityManager** : gère les objetsJava qui sont des entités (0,"Whiskey"), (0,"Tequila"), (1, "Ice cubes"), (1, "Sugar") (2, "CO2")); EntityManaget soccupe de créer les requêtes sql **TransactionManager**	Ligne dans la table
	Requête JPQL = équivalent de requêtage SQL mais en java	Passe les requêtes en SQL via le JPQL

Vue/Controller/

VUE	CONTROLLER		
ingredients.jsp ----------	IngredientController RequestMapping ModelAndView view name model List<Ingredient> <----------	**@Transactional** IngredientService DAO<----------	IngredientDAO JPARepository Spring EntityManager BDD Requêtes JPQL

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:jpa="http://www.springframework.org/schema/data/jpa"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/spring-beans.xsd
        http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context.xsd
        http://www.springframework.org/schema/data/jpa/spring-jpa.xsd">
```

GIT

<table>
<tr><th>GIT</th><th>REMOTES</th></tr>
<tr><td rowspan="2">Gestionnaire de source
--> repository distant
--> Histoire de toutes les modifications
--> Commit --> nouvelle révision / Version</td><td>Github --> Origin

Branches
○ MASTER
○ tags</td></tr>
<tr><td>Local (clone)</td></tr>
<tr><td></td><td>PC
--> Staging
--> Working Directory</td></tr>
</table>

Différence entre SVN et GIT : plusieurs repository avant l'original
Branche principale :
tag : copie figée des sources

<table>
<tr><th colspan="3">3 couches en local</th><th>Distant</th></tr>
<tr><td>Workspace</td><td>staging</td><td>Repository local</td><td>Repository distant (remote)</td></tr>
<tr><td>src/pom.xml-------
|---------
---------------------></td><td>git add

src/pom.xml
|----------
></td><td>

git commit
|----------
----------></td><td>

git push
révision #1</td></tr>
<tr><td>pom.xml</td><td>git add</td><td>révision #2
révision #1</td><td>révision #1</td></tr>
<tr><td></td><td></td><td></td><td>Si projet #32 ne fonctionne plus on peut revenir sur projet #28</td></tr>
</table>

Liste des commandes

1. Créer un repositiry de son poste sur github

C:/user/workspace > mkdir <MonProjet>

- **git clone https://github/flo1012/nom_du_repository**
- **git add** <fichier>
- **git commit -m "commentiare"**
- **git push origin master** ----------> demande mot de passe

2. Récupérer un repository

Sur github :

Aller sur le repository qui nous interesse , récupérer l'@ https sous clone puis :

- **git clone https://github/flo1012/nom_du_repository**

Pour mettre à jour :

- **git pull**

3. Autres commandes

- **git status**
- **git remote**

- **git rm nom_fichier**
- **git reset**

Pour voir la différence :

git diff

4. Avec plusieurs branches

-
- `git branch`
- `git checkout security`
- `git status`
- `git branch`

Création de mon Git pour la doc

…or create a new repository on the command line

```
echo "# JAVA_JEE" >> README.md
git init
git add *
```

git commit -m "**Documentation chronologique de la formation Java JEE**"

git remote add origin https://github.com/Flo1012/JAVA_JEE.git
git push -u origin master

Projet Cocktail de Jeremy

org.webjars
- jquery-ui 1.12.1
- datatables 1.10.12-1
- datatables-colreorder 1.2.0
- bootstrap 3.1.0

jquery datatables (jquery est un framework de javascript)

src/main/java

CONTROLLER

CocktailController

```
package fr.formation.controller;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.ModelAndView;

import fr.formation.entity.Cocktail;
import fr.formation.entity.CocktailPart;
import fr.formation.service.CocktailService;

@Controller
@RequestMapping("/cocktails")
public class CocktailController {

	@Autowired
	private CocktailService service;

	@RequestMapping("/add")
	public ModelAndView add() {
		final ModelAndView mav = new ModelAndView();
```

```
		mav.setViewName("addCocktail");
		return mav;
	}

	@RequestMapping
	public ModelAndView list() {
		final ModelAndView mav = new ModelAndView();
		mav.setViewName("cocktails");
		mav.addObject("cocktails", this.service.getAll());
		return mav;
	}

	@RequestMapping(value = "/add", method = RequestMethod.POST)
	public String newCocktail(@RequestParam final String name,
			@RequestParam final Float price,
			@RequestParam(required = false) final Boolean withAlcohol) {
		final Cocktail cocktail = new Cocktail();
		cocktail.setName(name);
		System.out.println("Cocktail name : " + name);
		cocktail.setPrice(price);
		cocktail.setWithAlcohol(withAlcohol != null);
		this.service.create(cocktail);
		return "redirect:/cocktails/add.html";
	}

	@RequestMapping("/test")
	public void test() {
		final List<CocktailPart> parts = this.service.getCocktailParts();
		System.out.println("parts size : " + parts.size());
	}

}
```


IngredientController

```
package fr.formation.controller;

import javax.servlet.http.HttpServletRequest;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestMethod;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.servlet.ModelAndView;

import fr.formation.entity.Ingredient;
import fr.formation.service.IngredientService;

@Controller
@RequestMapping("/ingredients")
public class IngredientController {

	@Autowired
	private IngredientService service;

	@RequestMapping("/add")
```

```
	public ModelAndView add() {
		final ModelAndView mav = new ModelAndView();
		mav.setViewName("addIngredient");
		return mav;
	}

	@RequestMapping
	public ModelAndView list() {
		final ModelAndView mav = new ModelAndView();
		mav.setViewName("ingredients");
		mav.addObject("ingredients", this.service.getAll());
		return mav;
	}

	@RequestMapping(value = "/add", method = RequestMethod.POST)
	public String newIngredient(final HttpServletRequest request) {
		final String name = request.getParameter("name");
		final Integer state = Integer.parseInt(request.getParameter("state"));

		this.service.create(new Ingredient(name, state));
		return "redirect:/ingredients/add.html";
	}

	@RequestMapping(value = "/add2", method = RequestMethod.POST)
	public String newIngredient2(@RequestParam final String name,
			@RequestParam final Integer state) {
		this.service.create(new Ingredient(name, state));
		return "redirect:/ingredients/add.html";
	}
}
```


MainController

```
package fr.formation.controller;

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

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.MessageSource;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

import fr.formation.model.Menu;

@Controller
public class MainController {

	@Autowired
	private MessageSource messages;

	private String getMessage(final String key) {
		return this.messages.getMessage(key, null, null);
	}
```

```
	@RequestMapping("/index")
	public ModelAndView index() {
		final ModelAndView mav = new ModelAndView();
		mav.setViewName("index");
		final List<String> menuKeys = Arrays
				.asList(this.getMessage("menu.list").split(","));
		final List<Menu> menus = new ArrayList<>();
		for (final String menuKey : menuKeys) {
			final String prefix = "menu." + menuKey.trim();
			final String title = this.getMessage(prefix + ".title");
			final String url = this.getMessage(prefix + ".url");
			menus.add(new Menu(title, url));
		}
		mav.getModel().put("menus", menus);
		return mav;
	}
}
```


DAO

CocktailDAO

```
package fr.formation.dao;

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

import fr.formation.entity.Cocktail;

public interface CocktailDao extends JpaRepository<Cocktail, Integer> {

}
```


IngredientDao

```
package fr.formation.dao;

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

import fr.formation.entity.Ingredient;

public interface IngredientDao extends JpaRepository<Ingredient, Integer> {

}
```


ENTITY

Cocktail

Ingredient

MODEL

Menu

SERVICE

CocktailService

IngredientService

sr/main/resources

menu.properties

META-INF

orm.xml

persistence.xml

webapp

css

Nouveau chapitre

views

WEB-INF

Outils

STAN

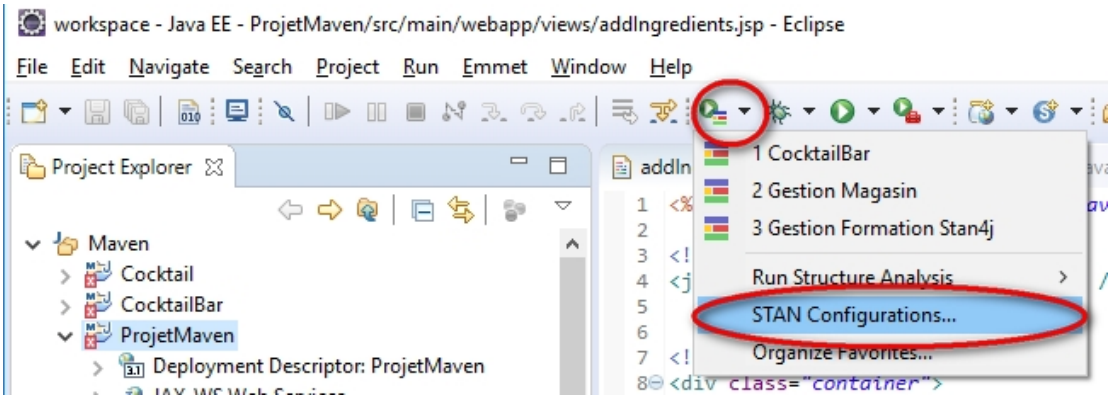
Télécharger Plugins pour Eclipse

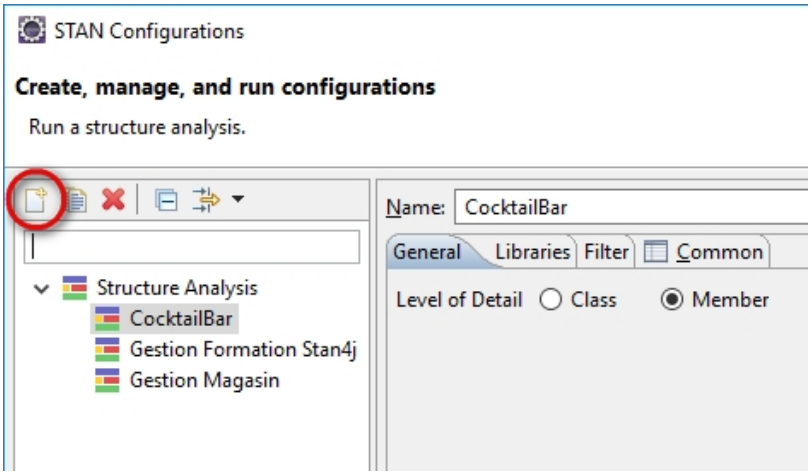
Installation de la fonction Eclipse

STAN est disponible via notre site de mise à jour Eclipse. Veuillez suivre les instructions ci-dessous.

- Depuis Eclipse, allez à Aide - Installer un nouveau logiciel ...
- Ajouter un nouveau site pour STAN avec l'URL http://update.stan4j.com/ide
- Sélectionnez la fonction STAN IDE et appuyez sur "Suivant"
- Suivez l'assistant, lisez et acceptez les termes de la licence
- Redémarrer Eclipse

Générer un projet STAN à partir d'un projet éclipse :





==>

Organigramme choix Objet Collection

GlassFish

Install

Télécharger https://glassfish.java.net/download.html
Sous unix : **jar xvf glassfish-4.1.zip**

Lire le fichier readme.txt :

Lancer la console Glassfish

La commande "asadmin" est utilisé pour controler et manager GlassFish

- start,
- stop,
- configure,
- deploy applications,
- etc...

Lancer les commandes GlassFish sous **glassfish4\glassfish\bin** !

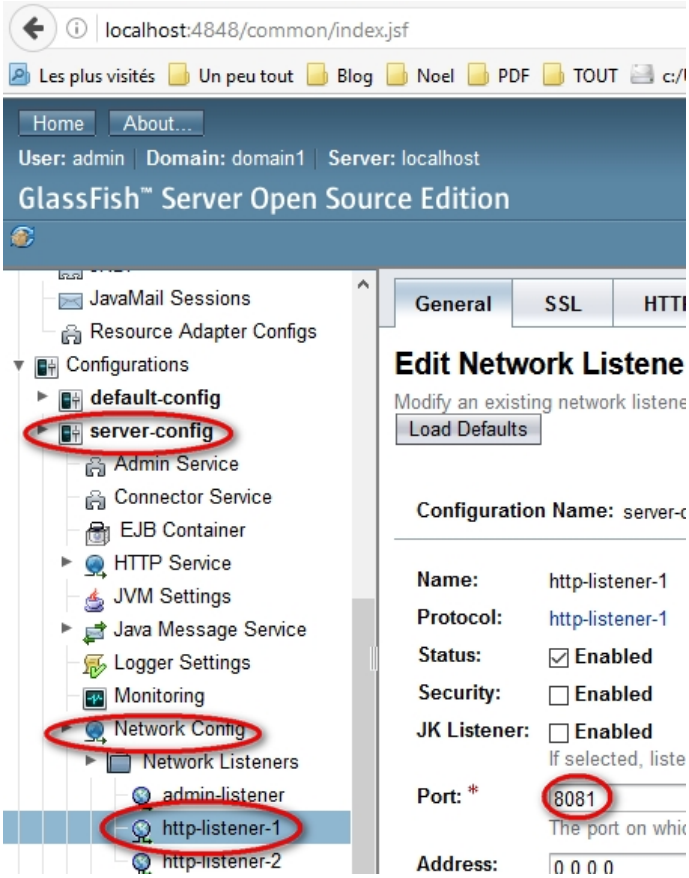
Pour démarrer : asadmin start-domain

Pour arrêter : asadmin stop-domain

Ouvrir GlassFish: **http://localhost:4848**

Changement port 8080

Aller sous la console Glassfish
changer



ajouter la variable d'environnement `GLASSFISH_AUTODEPLOY` qui pointe vers

```
C:\Users\hb-asus>echo %GLASSFISH_AUTODEPLOY%
```

```
C:\Users\hb-asus\Documents\Install\glassfish4\glassfish\domains\domain1\autodeploy
```


Paramétrage des logs

Plusieurs niveaux de logs :

1. fatal
2. error
3. warning
4. debug
5. info

1) Historique des API

API de Login JUL

Pour gérer les logs on peut utiliser l'API de Login JUL : Java Util Login
Celle-ci utilise la sortie standard
Un peu vieille

API Log4J

Nouvelle librairy Log4J puis Log4J2 et SLF4J qui centralise toutes les logs en seul point pour créer un pont entre toutes les api java.

Implémentation LogBack

Puis LogBack (implementation) plus pratique de SLF4J

Dans les prog, on utilise SLF4J.

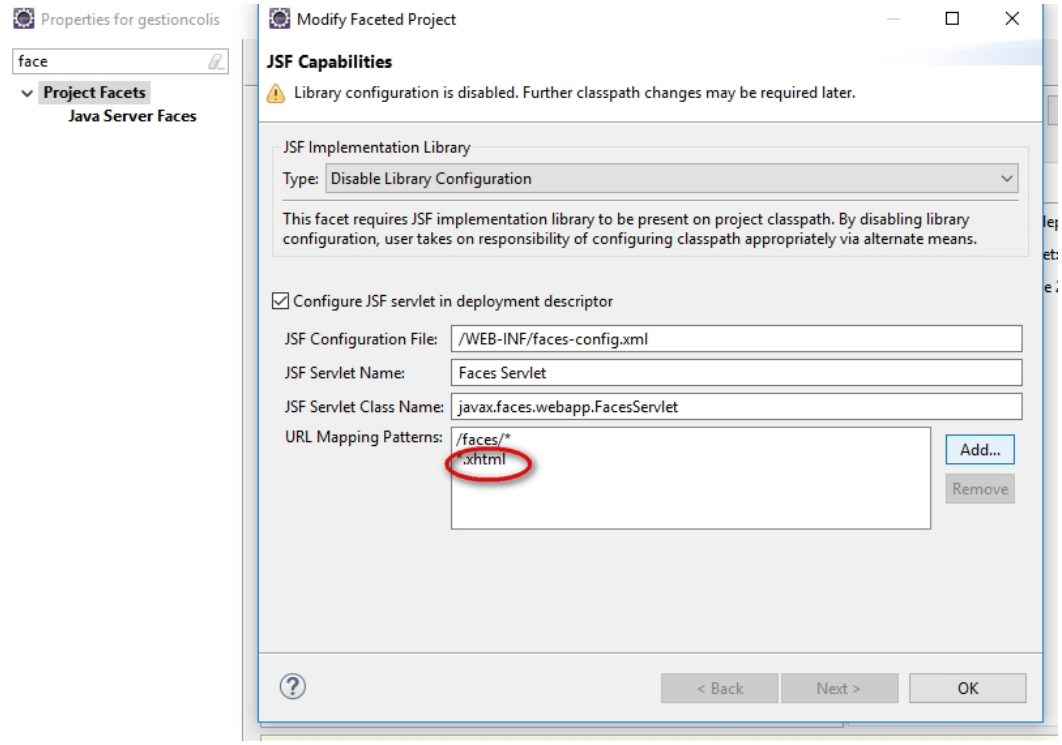
```
L'utilisation de <scope> :
<scope>test</scope> ---> lors de la compilation du WAR (Web Archive) n'importe pas les librairies
```

MyFaces

L'implementation Oracles JSF : Mojara

Le JAR des Drivers doit être posé dans Glassfish

Config altEntrée

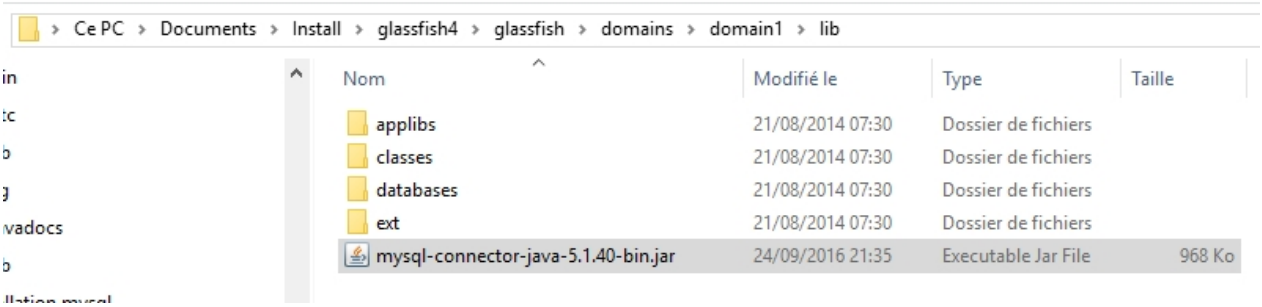


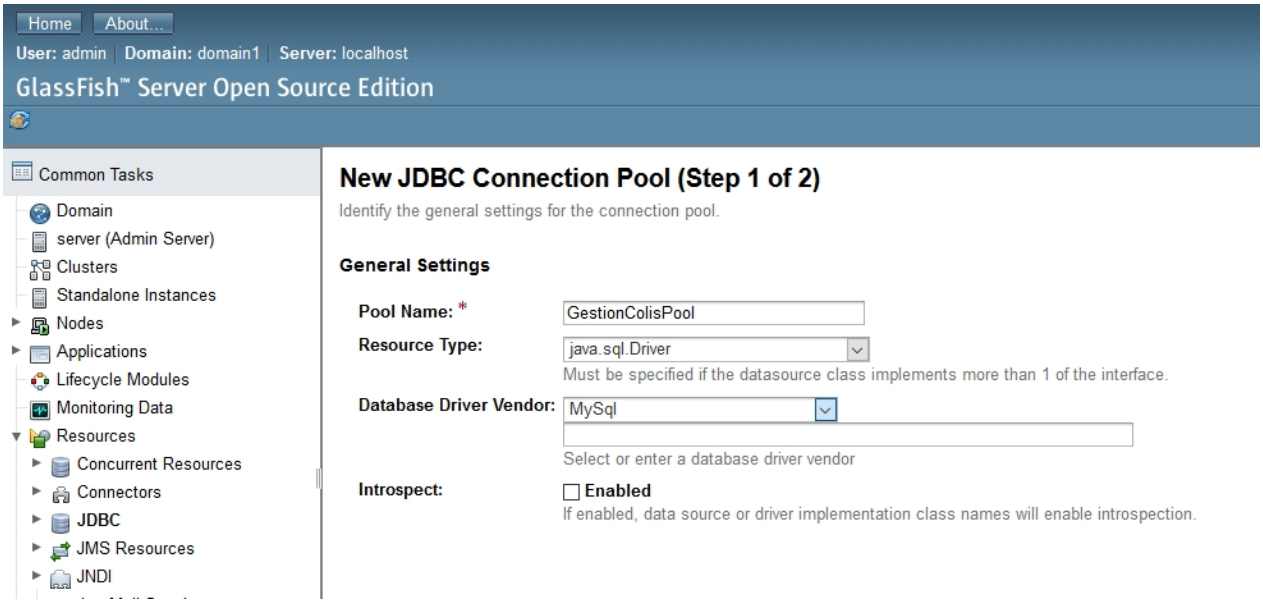
Ajout de

Mysql pool

Copier le jar de mysql dans Glassfish :

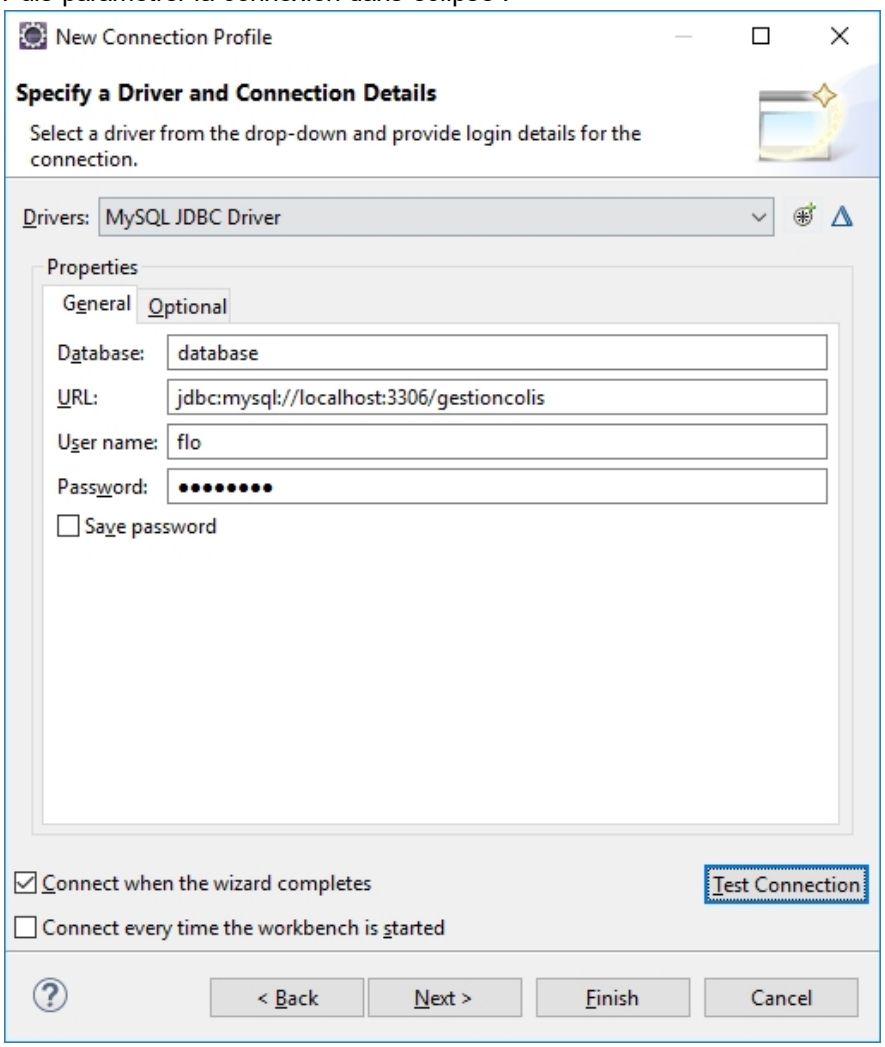
C:\Users\hb-asus\Documents\Install\glassfish4\glassfish\domains\domain1\lib





Créer un user dans la base de données avec un mot de passe.

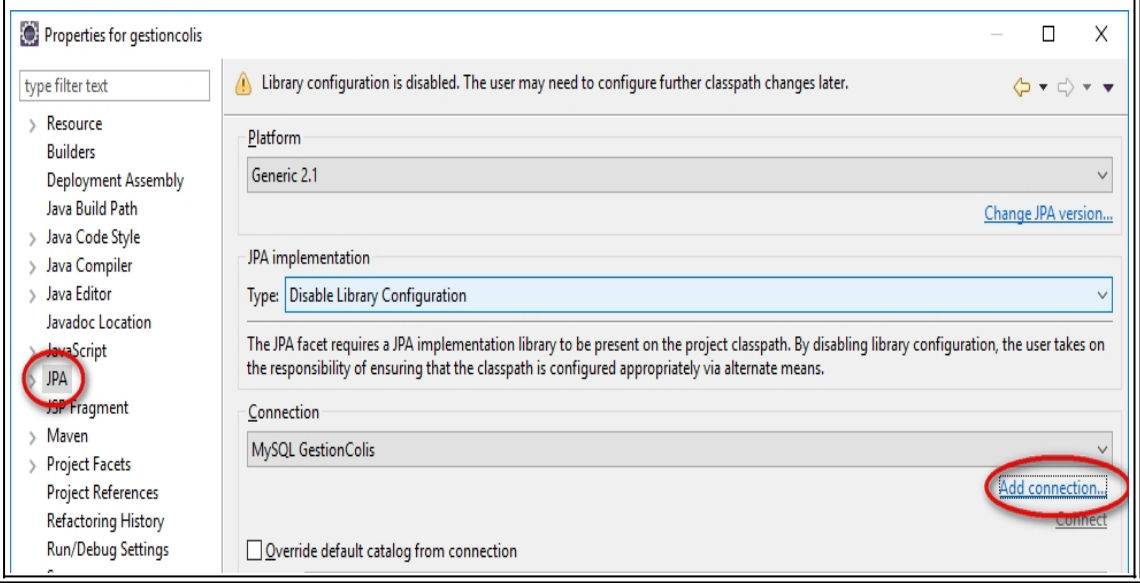
Puis paramétrer la connexion dans eclipse :



Dans eclipse

Création de la connection à la base de données

Aller dans les propriétés du projet :



Penser à sélectionner le drivers de la BDD

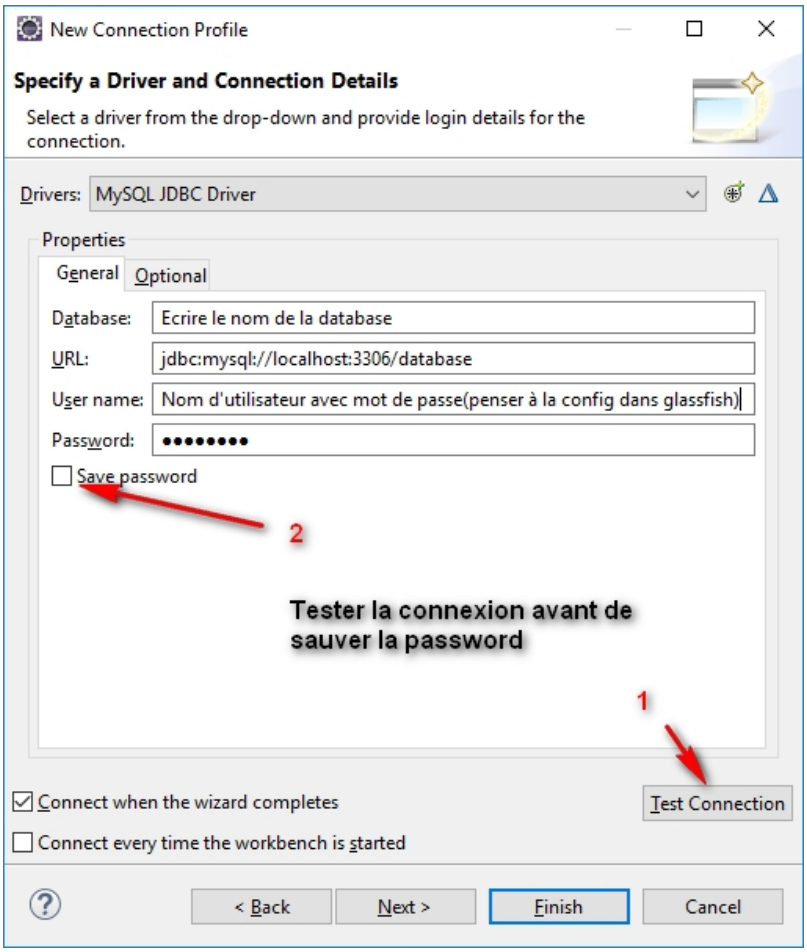


Schéma Appli

APPLICATION			***Serveur GlassFish***	***BDD Mysql***
Présentation	***Métier***	***Persistence***		
HTML	Commande Id=1	Hibernate	Ressource JBBC	Table Commande
Render Response	Commande Id=null	JPA	1 connexion	
XHTML	Commande Id=2	Entity Manager	Pool de connexion	
Autres étape du cycle		Persistence Context		

JSF	Managed Beans	DAO		

Pas de couche SERVICES =objet passe plat idée de passage dans la couche service

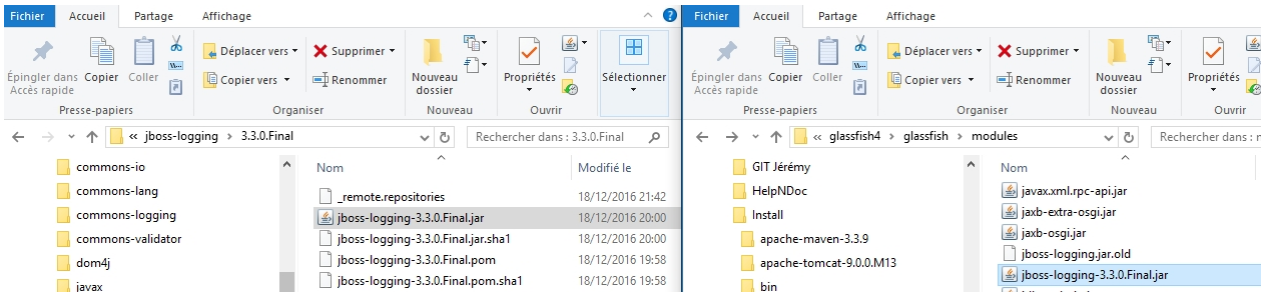
MAJ dans GlassFish

Copier jboss-logging-3.3.0.Final.jar :

Qui se trouve sous C:\Users\hb-asus\.m2\repository\org\jboss\logging\jboss-logging\3.3.0.Final

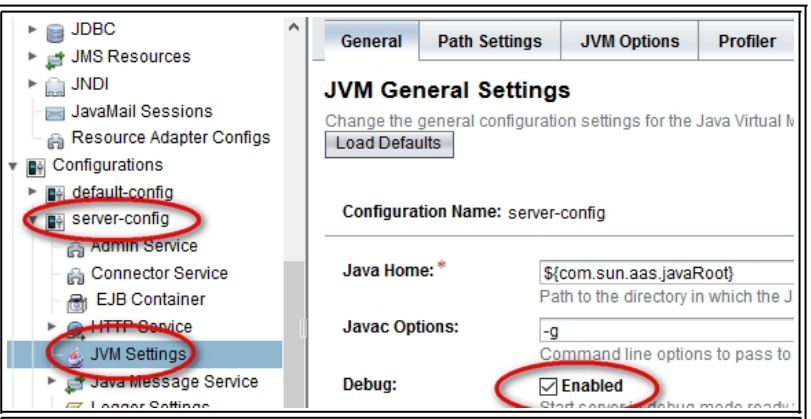
Dans répertoire de GlassFish :

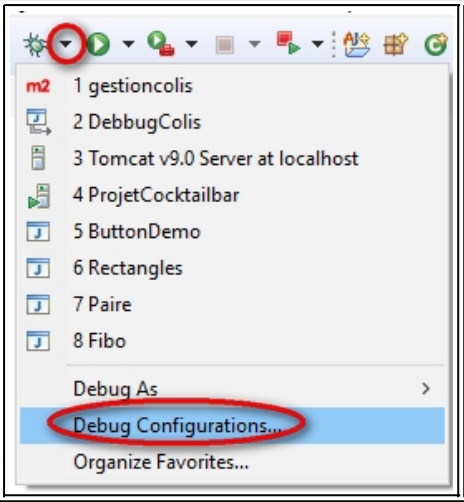
1. Renommer jboss-logging.jar en jboss-logging.jar.old
2. C:\Users\hb-asus\Documents\Install\glassfish4\glassfish\modules



Remote debugging

Configuration Glassfish et éclipse

Sous Glassfish	
Sous eclipse	Sous Debug config remote Java Appli new nom lier appli localhost

	port 9009

Déclaration des Logs en java

Commande Maven pour connaître les dépendances créer dans le pom.xml

- mvn dependency:tree

Qd on déclarer un log :

On ne doit déclarer qu'un seul log par classe en static et final

- ```
  private static final Logger LOGGER =
  LoggerFactory.getLogger(ProductController.class);
  ```

postConstruct méthode

entité détachée c'est à dire entité non manager.
(à voir)

Astuce:

Normalement, pour le readAll, il faut une transaction.
On ne va pas rajouter une transaction mais appeler le read() de la Dao et utiliser la même transaction.

Lors de l'écriture de la classe qui gère les erreurs, `pas de constructeur par défaut cela évite de créer une erreur sans rien`

Classe EntityManager et UserTransaction

EntityManager :

C'est une classe qui est chargée de mettre en musique les correspondances définies dans les entités, et qui réalise donc toutes les opérations CRUD (Create, Read, Update, Delete) sur la base de données.
  ```

UserTransaction

C'est une classe

Schéma de vie d'une exception

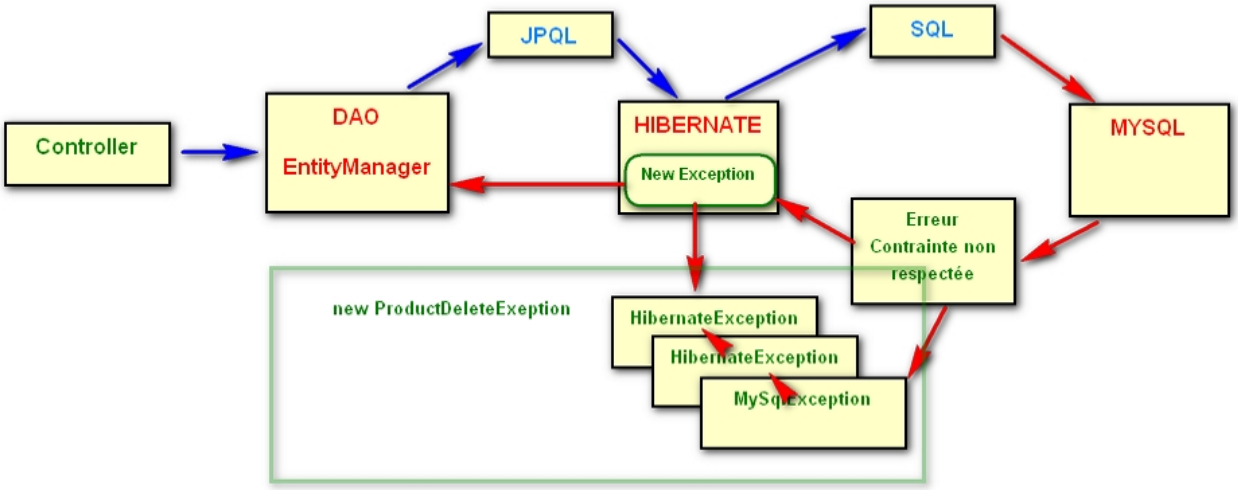
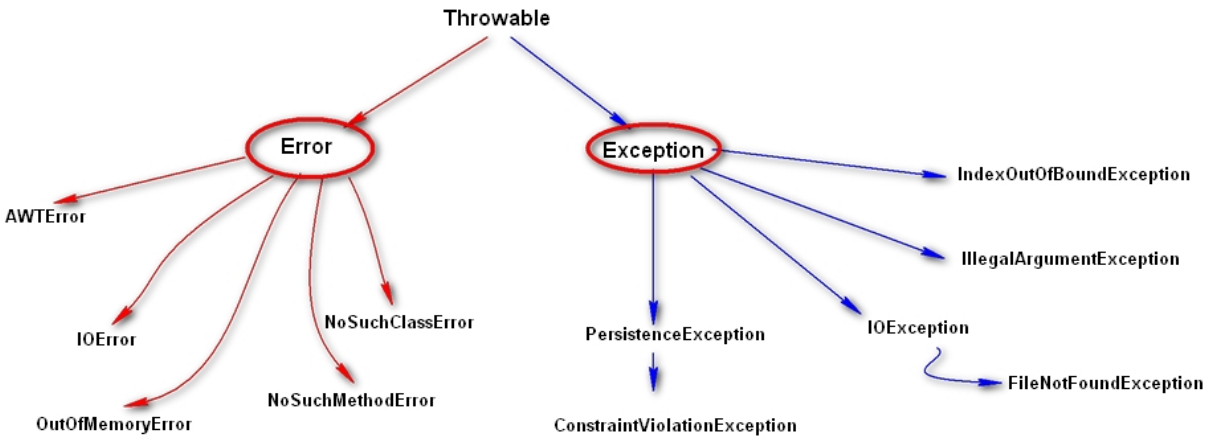


Schéma des ERROR / EXCEPTION



Liste des tutos donnés par Jérémy

1. JSF (liste) : http://www.mkyong.com/tutorials/jsf-2-0-tutorials/

1. Navigation implicite :
   http://www.mkyong.com/jsf2/implicit-navigation-in-jsf-2-0/
   - Vous donnera un aperçu de la configuration de navigation dans faces-config.xml

2. Navigation par action dans un form et méthode Java :
   http://www.mkyong.com/jsf2/jsf-form-action-navigation-rule-example

3. Composant de liste déroulante :
   http://www.mkyong.com/jsf2/jsf-2-dropdown-box-example/

4. Le tag 'f:param' pour envoyer des paramètres (sera utilisé pour l'édition) :
   http://www.mkyong.com/jsf2/jsf-2-param-example/

5. Utilisation simple d'un bean managé (explique la configuration nécessaire pour les anciennes version de JSF) :
   http://www.mkyong.com/jsf2/configure-managed-beans-in-jsf-2-0/

2. Spring

   1. Hello World Spring dans un projet Java (pas de web) :
      http://www.mkyong.com/spring3/spring-3-hello-world-example/

   2. Exemple d'injection de dépendance (DI) sans l'annotation Autowired, avec de la configuration XML et le setter :
      http://www.mkyong.com/spring/spring-di-via-setter-method/

   3. Exemple d'injection de dépendances avec annotation (dans la 2eme partie) :
      http://www.mkyong.com/spring/spring-auto-scanning-components/

3. Hibernate

   1. Association one-to-many (XML) :
      http://www.mkyong.com/hibernate/hibernate-one-to-many-relationship-example/

   2. Association one-to-many (Annotations) :
      http://www.mkyong.com/hibernate/hibernate-one-to-many-relationship-example-annotation/

4. Jackson (conversion Java<->JSON)

   1. http://www.mkyong.com/java/how-to-convert-java-object-to-from-json-jackson/

JFS

Annotations

Synthèse

	bean		controller	dao
	entity	login		

implements	Serializable	Serializable	Serializable	
extends				AbstractDao<Product>
Classe	@ManagedBean @ViewScoped	@ManagedBean @ApplicationScoped	@ManagedBean @ViewScoped	rien
Attributs				@PersistanceContext protected EntityManager em; @Resource private UserTransaction transaction;
Méthode				

	entity	exception
implements	Serializable	
extends		
Classe	@Entity @NamedQuery(name="Bordereau.findAll", query="SELECT b FROM Bordereau b") public class Bordereau {	
Attributs	@Id @GeneratedValue(strategy=GenerationType.*IDENTITY*) private int id; @Temporal(TemporalType.*TIMESTAMP*) @Column(name="DATE_SIGNATURE") private Date dateSignature; private String detail; //bi-directional many-to-one association to Commande @ManyToOne @JoinColumn(name="COMMANDE") private Commande commandeBean;	
Méthode		

Annotations BEAN

Annotations nécessaires pour BEAN

1) Au dessus des classes en connexion avec la BDD

@ManagedBean
@ViewScoped

```
public class ProductBean implements Serializable {
```

OU

1) Au dessus de la classe de login

```
@ManagedBean
@SessionScoped
public class LoginBean implements Serializable {
```


Annotations CONTROLLER

Annotations nécessaires pour CONTROLLER

1) Au dessus de la classe

```
@ManagedBean
@ViewScoped
	public class ProductController implements Serializable {

		private static final Logger LOGGER =
LoggerFactory.getLogger(ProductController.class);
```

2) Au dessus de l'attribut

```
@ManagedProperty("#{productBean}")
	private ProductBean productBean;
```


Annotations DAO

Annotations nécessaires pour DAO

1) Au dessus de la classe

```
Rien
```

2) Attribut

```
	@PersistenceContext
	protected EntityManager em;

	@Resource
	private UserTransaction transaction;
```

Annotations ENTITY

Annotations nécessaires pour ENTITY

1) Au dessus de la classe

@ManageBean

Cette classe sera instancié :
au démarrage de l'application
par défaut le nom est basé sur le nom de la classe avec une minuscule sur la première lettre

Voir l'option lazy

Ajax

En ajax, la page n'est pas rafraîchit, il faut indiquer ce qui doit être mis à jour
le render("le rendu") appelle l'attribut à mettre à jour.

Selon l'API utilisant ajax, <h:ajax> <f:ajax> : L'écriture sera particulière à chaque API, les balises n'auront pas les même arguments.

Pour <f:ajax >

Si on souhaite rafraîchir un élément d'un autre formulaire, il faut faire précéder le champ par le nom du formulaire
ex : @form updateModeleValue voir tout ce qui est cablé sur le formulaire.

D'autres attributs sont disponible pour ajax :

- **Attribut disable**
  Dès que l'on commence ajax, on est en ajax donc si l'on souhaite récupérer autre chose, il faut utilisé la balise **disable.**

- **Attribut event**
  celui-ci déclenche une requête ajax dans un événement inputtext onchange

Lorsque l'on teste un élément à null il faut que se soit un objet. Donc pour les int il faut convertir en Integer!

Qq tags

<f:metadata>	Tag permettant de gérer des données contextuelles à la vue

`<f:viewParam name="productId" value="#{productController.productId}" />`	Ce tag permet de lier une propriété d'un bean sans en faire un champ dans la page. C'est simplement contextuel à la vue.
`<f:event type="preRenderView" listener="#{productController.prepareEdit}" />` `</f:metadata>`	Ce tag permet de déclencher une méthode Java lors d'un "process event" du cycle de vie JSF. Le preRenderView est l'événement déclenché avant la dernière étape du cycle 'RenderView'.

`<ui:fragment rendered="#{empty productController.productId}">`
`<h2>Créer un nouveau produit :</h2>`
`</ui:fragment>`
	Tag permettant d'encapsuler un autre tag sans générer d'HTML supplémentaire. La balise fragment disparaît lors de l'étape RenderView qui génère le code HTML.

`<ui:fragment rendered="#{!empty productController.productId}">`
`<h2>Modifier un produit :</h2>`
`</ui:fragment>`
	L'attribut rendered permet de conditionner (true/false) la présence d'un tag dans le code HTML.
	Cet attribut permet donc de conditionner l'affichage, mais il faut bien se rappeler qu'il gère
	cette présence lors de l'étape RenderView, pas dans la page HTML. Ce n'est donc quelque chose
	qu'on ne peut altérer côté client en CSS ou JS, car un élément dont le rendered est à false ne
	sera pas du tout dans la page.

`<b:form>`	

`<b:inputText label="Id" value="#{productController.productId}"`
`rendered="#{!empty productController.productId}"`
`readonly="true" />`
	Ce champ n'est visible qu'en édition est n'est pas modifiable (readonly).
	Il existe en CSS un sélecteur ':read-only' qui permet de modifier facilement le style.

`<b:inputText label="Intitulé" value="#{productBean.intitule}">`	

`<f:validateRequired />`
`</b:inputText>`
`<b:inputText type="number" label="Poids" value="#{productBean.poids}">`
	Rend la saisie du champ obligatoire.

`<f:validateRequired />`
`<f:validateDoubleRange />`
`</b:inputText>`
`<b:inputText label="Référence (unique)" value="#{productBean.reference}">`
`<f:validateRequired />`
`<f:validateLength minimum="3" maximum="32" />`
`</b:inputText>`
`<b:commandButton value="Valider"`
	La saisie doit pouvoir être transformée en float Java.

```
action="#{productController.save}" />
</b:form>
```


Web Services

Spring security

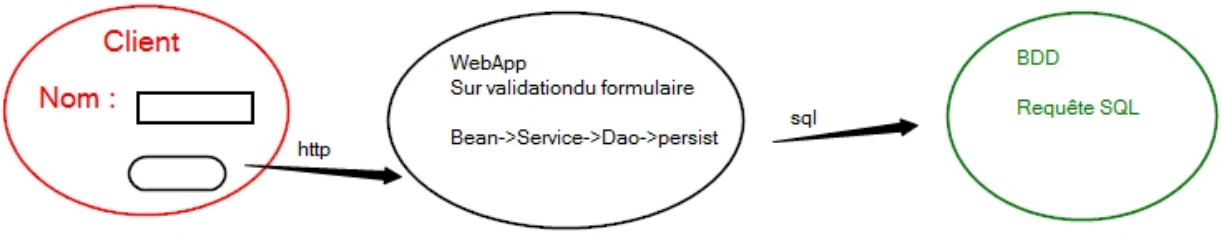
La sécurité est souvent posée au niveau réseau.
code pin généré par des calculettes, permet de se connecter àun environnement sécurisé machine virtuelle ou ssh.

Niveau applicatif rare besoin de sécurité car géré par les parfeu

Authentification : savoir reconnaître un user de confiance
Authorisation : trier sur les droits de ce que l'on peut faire

Vérification sur le serveur

Hackage possible



en sql -- : met en commentaire
notre code et le remplacepour
eviter pn fait de empèchement de
caractères spécifique au SGBD

Deux attaques Web possible :

Le navigateur n'envoie plus de requête http strict http : HSTS

kick jacking : fenêtre extérieure à l'appli qui renvoie sur une action

CSRF : client envoie une 1er requête le serveur renvoie un header de type CSRF et une clé qui ne peut pas être devinée.

Spring security

Voir la doc sur https://projects.spring.io/spring-security/

Spring Security est un framework qui met l'accent sur l'authentification et l'autorisation des applications Java.

Comme tous les projets Spring, le véritable pouvoir de Spring Security réside dans la facilité avec laquelle il peut être étendu pour répondre aux exigences personnalisées

Caractéristiques

- Prise en charge complète et extensible de l'authentification et de l'autorisation
- Protection contre les attaques comme la fixation de session, le clickjacking, la falsification de requêtes sur site, etc.
- Intégration de l'API Servlet
- Intégration facultative avec Spring Web MVC
- Beaucoup plus…

La mise en place de Spring security :
il supporte un 15ne de standard comme SSO avec NTLM (windows) les sessions Windows sont enregistrées : carberos et

OAUTH
XSS
openid

Pour JSF

Spring Security	
Client requete HTTP	Server Filter->Filter->Filter->Filter-> les filtres choisissent cequ'il faut faire

Test unitaire

Service rest

Quelques explications

Comme on a besoin d'une requête vers la BBD il faut ingredientService
Le but de REST est de faire une API
depuis internet envoi/reception depuis un serveur
les annotations de REST s'accorde avec Spring Security

Respect de http standard mais coté présentation

- ajouter (POST),
- modifier (PUT),
- Lire (GET)
- Supprimer (DELETE)

ex : dataIngredient on peut faire les 4 actions par leur identifiant

Un service REST est un service WEB qui fait les mm actions que le CRUD

NIVEAU 1 : Bien définir API REST avec les noms logiques
NIVEAU 2 : Standardisé pour être utilisable avec une documentation
NIVEAU 3 : Pas de doc l'api fournit tte la doc (http option)

Dans DataController :

```
@Controller
@RequestMapping("/data")
public class Datacontroller {

	@Autowired
	private IngredientService ingredientService;
```

```
//            APPLICATION_JSON_UTF8_VALUE    ==> tjs prendre _value
      @GetMapping(value = "/ingredients" , produces =
MediaType.APPLICATION_JSON_UTF8_VALUE)

            public List<Ingredient> listIngredients() {
                  return this.ingredientService.getAll();
            }
}
```