

# FORMATION JSF 2.2

# PLAN DE FORMATION

- Introduction
  - Architecture JSF 2 et Pattern MVC 2
  - Cycle de vie de requête JSF 2
  - Vues JSF 2 avec Facelets
  - Beans Managés
  - JSF Expression language
  - Composants de la bibliothèque JSF 2
  - Création des composants visuels personnalisés
  - Création des composants non Visuels
  - Gestion des ressources avec JSF 2
  - Gestion des messages utilisateurs
  - Conversion avec JSF 2
- 
- Validation avec JSF 2
  - Gestion de évènements avec JSF
  - Navigation JSF 2
  - JSF 2 et AJAX
  - JSF 2 & Primefaces/Richefaces/Icefaces
  - Internationaliser une application JSF 2
  - Utilisation des objets implicites
  - JSF 2 et les technologies Java EE (CDI, EJB 3, WebServices Rest, JPA, Spring).
  - JSF 2 et Spring 3

## LES OBJECTIFS DE LA FORMATION

➤ Cette formation vous permettra :

- de renforcer vos compétences techniques sur JSF 2 ,
- de maîtriser les concepts clés de JSF 2 , ,
- de devenir rapidement autonome pour une productivité maximale sur des projets à haute valeur ajoutée
- de découvrir l'impact des bibliothèques additionnelles comme Primefaces, Richfaces.
- de mettre en œuvre l'intégration de JSF 2.1.X avec Spring, les EJB3, et les WebServices
- De mettre en œuvre les nouveautés de JSF 2.2

➤ Notre approche pédagogique vous permettra, d'élaborer une application complète grâce à une alternance de présentation et de travaux pratiques.

# INTRODUCTION

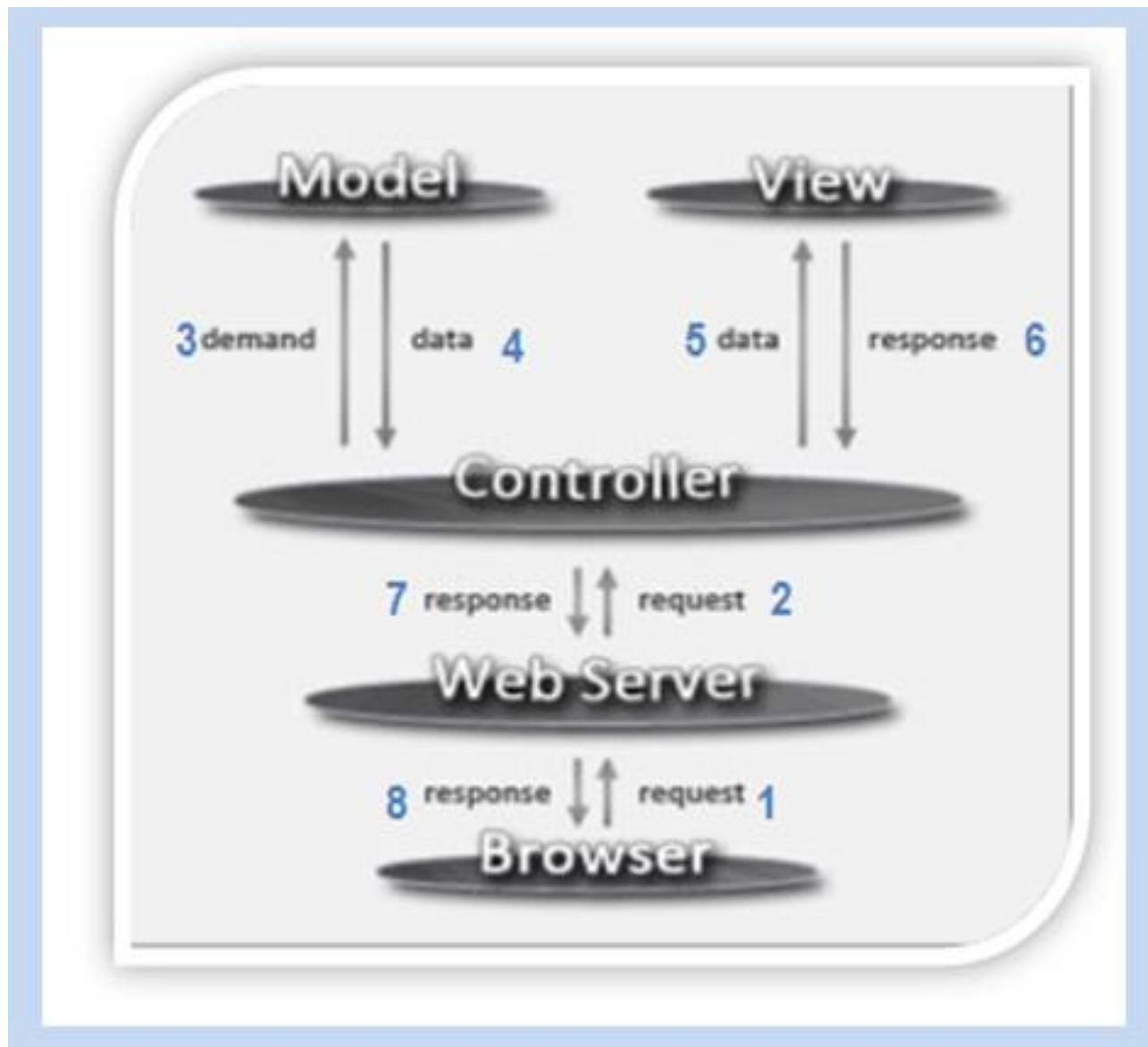
- Rappels sur l'architecture JEE
- Introduction à JSF 2
  - Découvrez la spécification JSR 344 JSF 2.1.X standard JEE
  - Historique JSF1.2 & JSF2.1.X & JSF 2.2 avec Faces flow, MultiTemplating, HTML5
  - Comprenez les 8 Objectifs de JSF 2.1.X dont l'implémentation de référence : Mojarra
  - Les concurrents directs et indirects de JSF 2

# INTRODUCTION

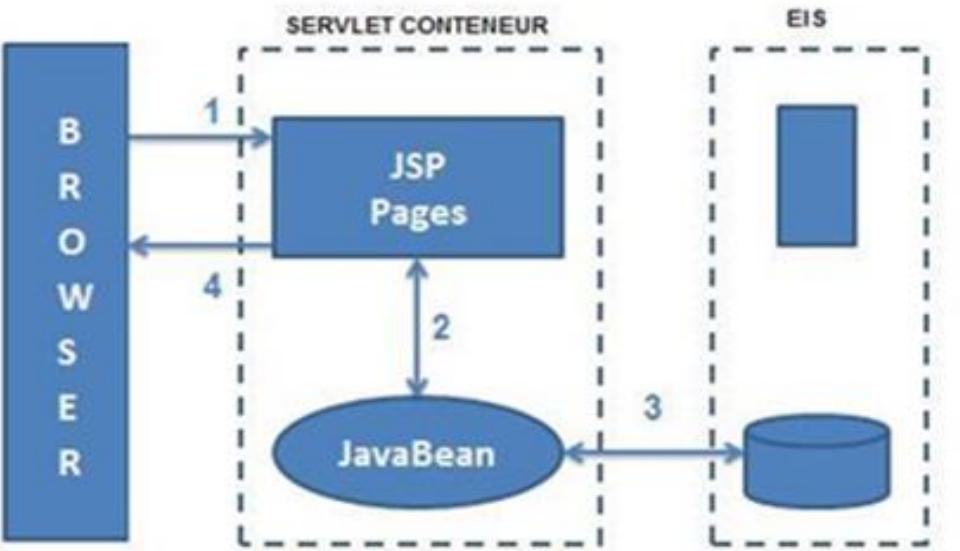
## ➤ Rappels sur l'architecture JEE

### MVC: AVANTAGES

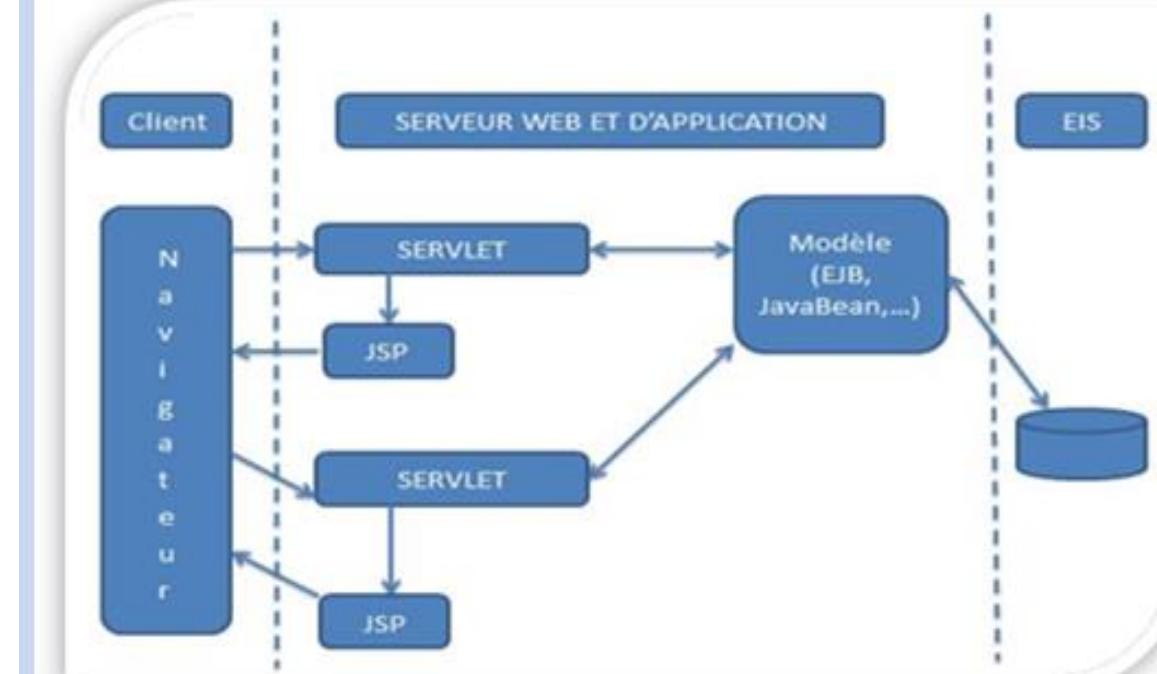
- Plusieurs vues peuvent utiliser le même modèle.
- Gain en coût de développement important.
- Deux équipes peuvent travailler en parallèle.
- Une équipe d'infographistes peut travailler sur les vues.
- Une équipe de développeurs peut travailler sur le modèle et le contrôleur.
- Les trois couches doivent être réellement indépendantes et ne doivent communiquer que par des interfaces.



# INTRODUCTION



ARCHITECTURE CENTRÉE JSP



MVC CENTRÉE SERVLET

# INTRODUCTION

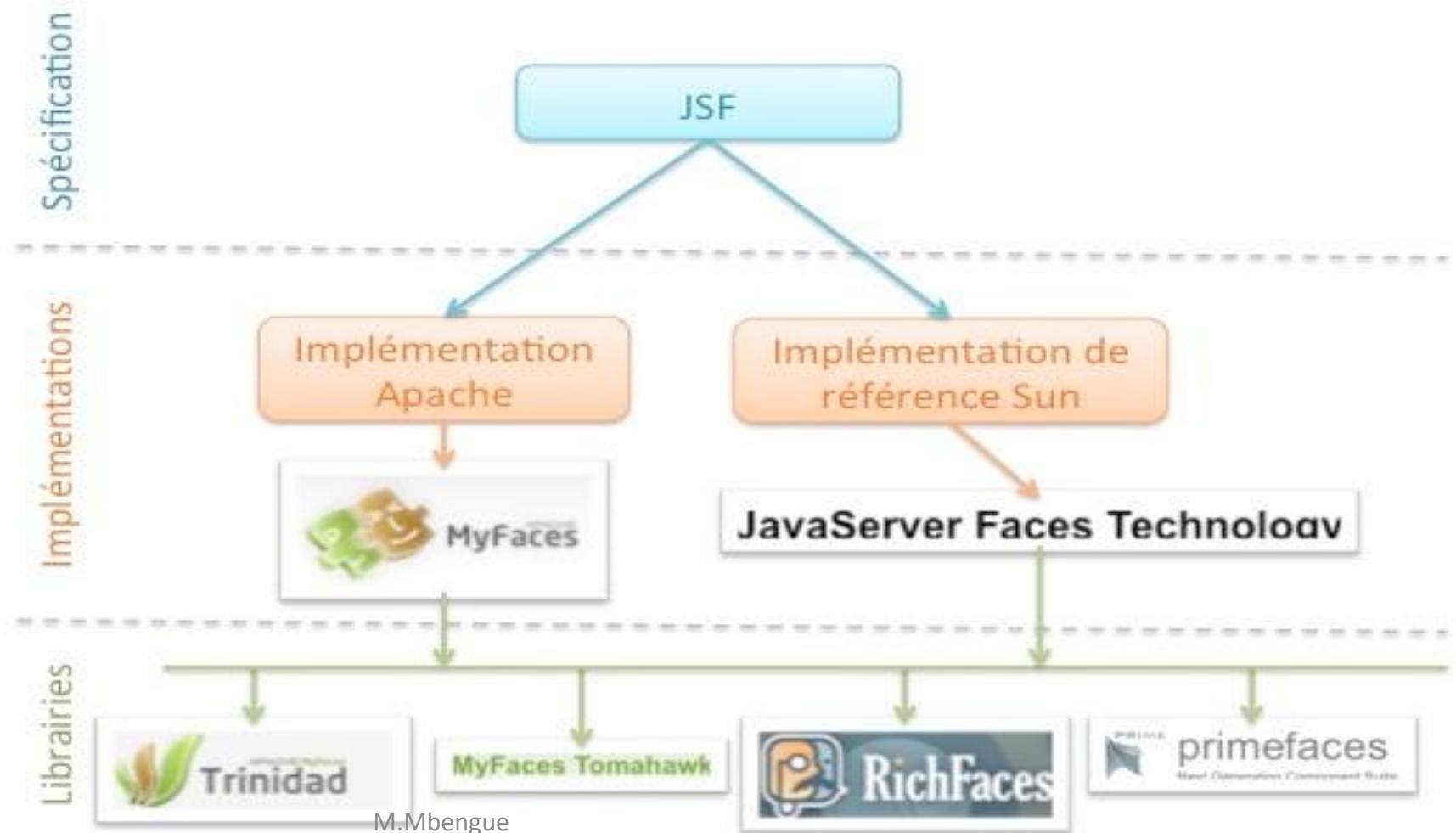
## Limitations du couple Servlet/JSP

- Le couple Servlet/JSP possède plusieurs limitations :
  - La création de balises personnalisées est assez lourde,
  - Il n'existe pas de balises de haut niveau en JSP,
  - Il n'est pas possible de récupérer facilement les entrées utilisateurs,
  - Il n'existe pas de guide qui indique comment développer en Servlet/JSP.
- Ainsi chaque développeur peut organiser son site comme il veut. Ceci rend la maintenance difficile.

# INTRODUCTION

## Introduction à JSF 2

- **JSF est une spécification** pour la création d'applications web en Java et **orienté composants**, il fait parti intégrante de la plate forme JEE depuis sa version 5.



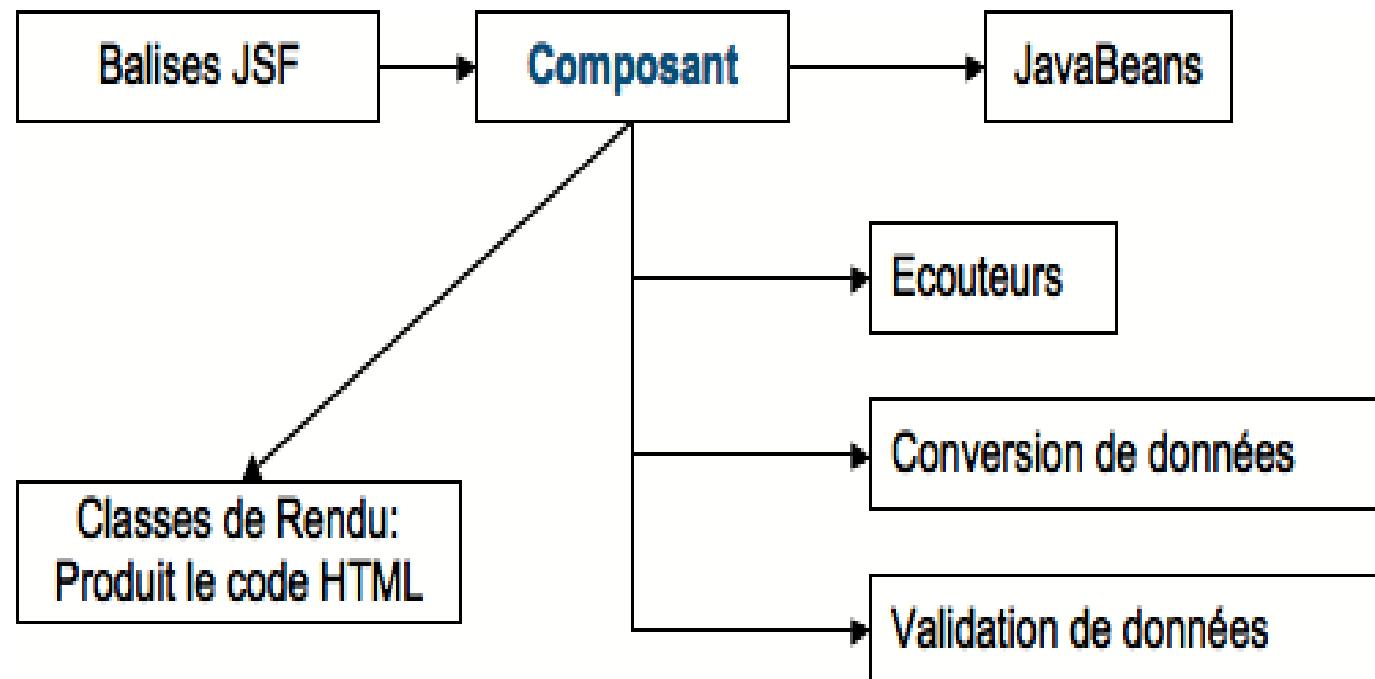
## JSF : un Framework orienté ‘composant’

On peut voir JSF comme un Swing adapté pour le Web.

En effet JSF se base sur un système de composant pour représenter la structure d'une page.

Les composants fournissent des méthodes pour gérer :

- des évènements,
- la validation des entrées utilisateurs
- et la conversion de données
- ....



## JSF : un Framework orienté ‘composant’

Le système de composant de JSF fournit les éléments suivants:

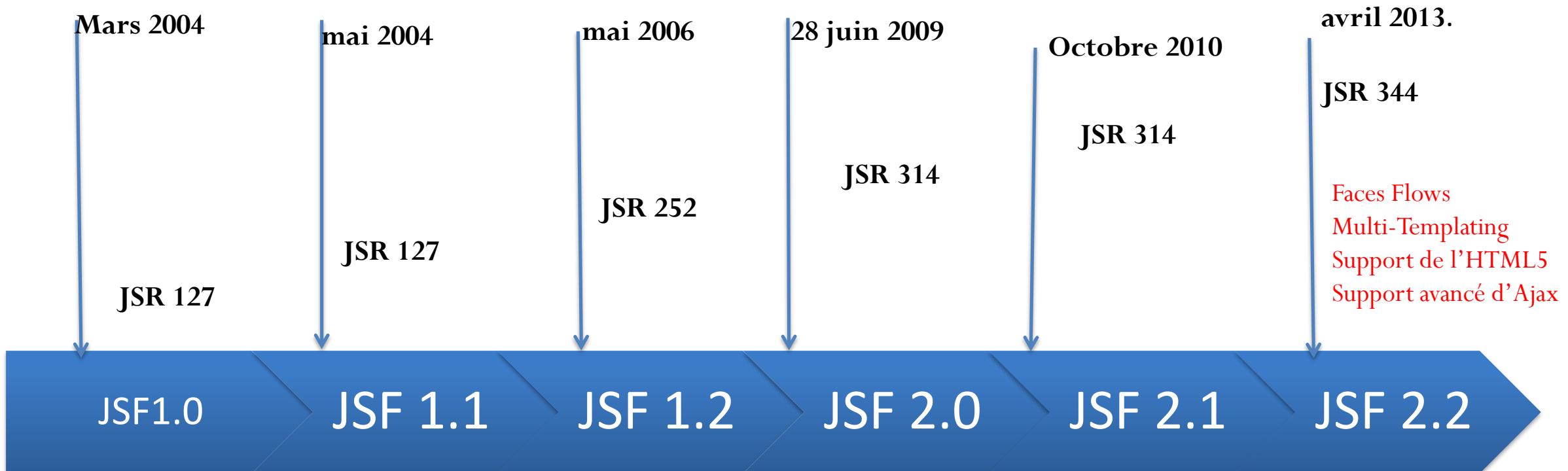
- Interaction avec des JavaBeans.
- Les JavaBeans représentent l’état des éléments à afficher et permettent d’interagir avec les couches plus basses de l’application,
- Un système de rendu (Renderer) qui permet de redéfinir comment sera affichée (traduit en code HTML) une balise JSF,
- Un système d’événements et d’écouteurs pour effectuer des traitements en fonction des actions des utilisateurs,
- Un modèle de conversion qui permet de convertir des types de données,
- Un système de validation qui permet de vérifier les entrées utilisateurs.

## Les Objectifs

- Créer des pages Web dynamiques avec des composants construits sur le serveur.
- Permet de bien séparer l'interface utilisateur, la couche de persistance et les processus métier.
- Conversion des données (tout est texte dans l'interface utilisateur).
- Validation des données saisies par l'utilisateur.
- Automatisation de l'affichage des messages d'erreur en cas de problèmes de conversion ou de validation.
- Internationalisation.
- Support d'Ajax sans programmation JavaScript.
- Fournit des composants standards pour l'interface utilisateur, puissants et faciles à utiliser.
- Possible d'ajouter ses propres composants.

# INTRODUCTION

## HISTORIQUE JSF



# INTRODUCTION

## JSR 314 JSF 2.0

- le fichier faces-config.xml devient optionnel, annotations pour la configuration ;
- les pages Facelets XHTML deviennent le standard pour créer des pages JSF, au lieu d'utiliser des pages JSP ;
- le support natif d'Ajax ;
- la déclaration des Beans managés via des annotations ;
- le support de la JSR 303 Bean Validation ;
- la création de composants rendu plus facile.
- Navigations implicites (selon conventions)

# INTRODUCTION

## JSR 314 JSF 2.0 (Suite)

- Prise en charge du mode "GET" (avec "View Parameters")
- l'injection de dépendance (@ManagedProperty , @Inject, ...)

# INTRODUCTION

## JSR 344 JSF 2.2

- **Faces Flows** : Cette nouvelle fonctionnalité permet de gérer un flow.

L'annotation `@FlowScoped(id="flowName")` permet de définir un “flow scoped Bean”, utilisé pour retourner un flow complet et non plus une seule vue/page.

Ce Bean sera uniquement accessible depuis les vues/pages qui composent ce flow. La définition du flow s'effectue avec un fichier `flowName.xml` et permet de préciser les règles de navigation avec les différentes vues correspondantes.

- **Multi-Templating** : cette fonctionnalité permet de changer le look and feel d'une application JSF à partir de différents Template.
- Chacun de ces Template disposera de ses propres ressources (css, js, images) et sera indépendant de l'application.

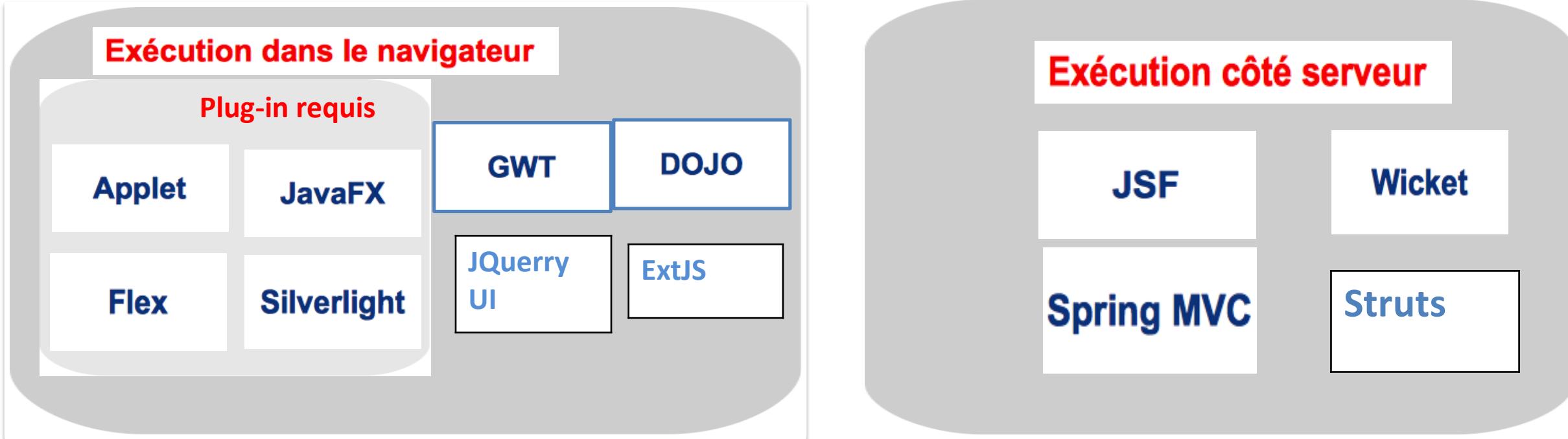
# INTRODUCTION

## JSR 344 JSF 2.2

- **Support de l'HTML5** : JSF 2.2 supporte tous les nouveaux attributs HTML5 (tel que l'attribut placeholder d'une balise input).
- **Support avancé d'Ajax** : JSF 2.2 propose un composant pour l'upload de fichier avec la possibilité de traiter l'envoi en mode synchrone ou en mode asynchrone.
- Une nouvelle fonctionnalité est la gestion d'une file d'attente des requêtes Ajax avec l'utilisation du tag `<f:ajax>`.
  - Son attribut `value` permet de définir un délai pour l'envoi de la requête afin de pouvoir traiter les requêtes dans l'ordre voulu.

Enfin, JSF 2.2 fournit une sécurité renforcée contre les attaques de type CSRF.

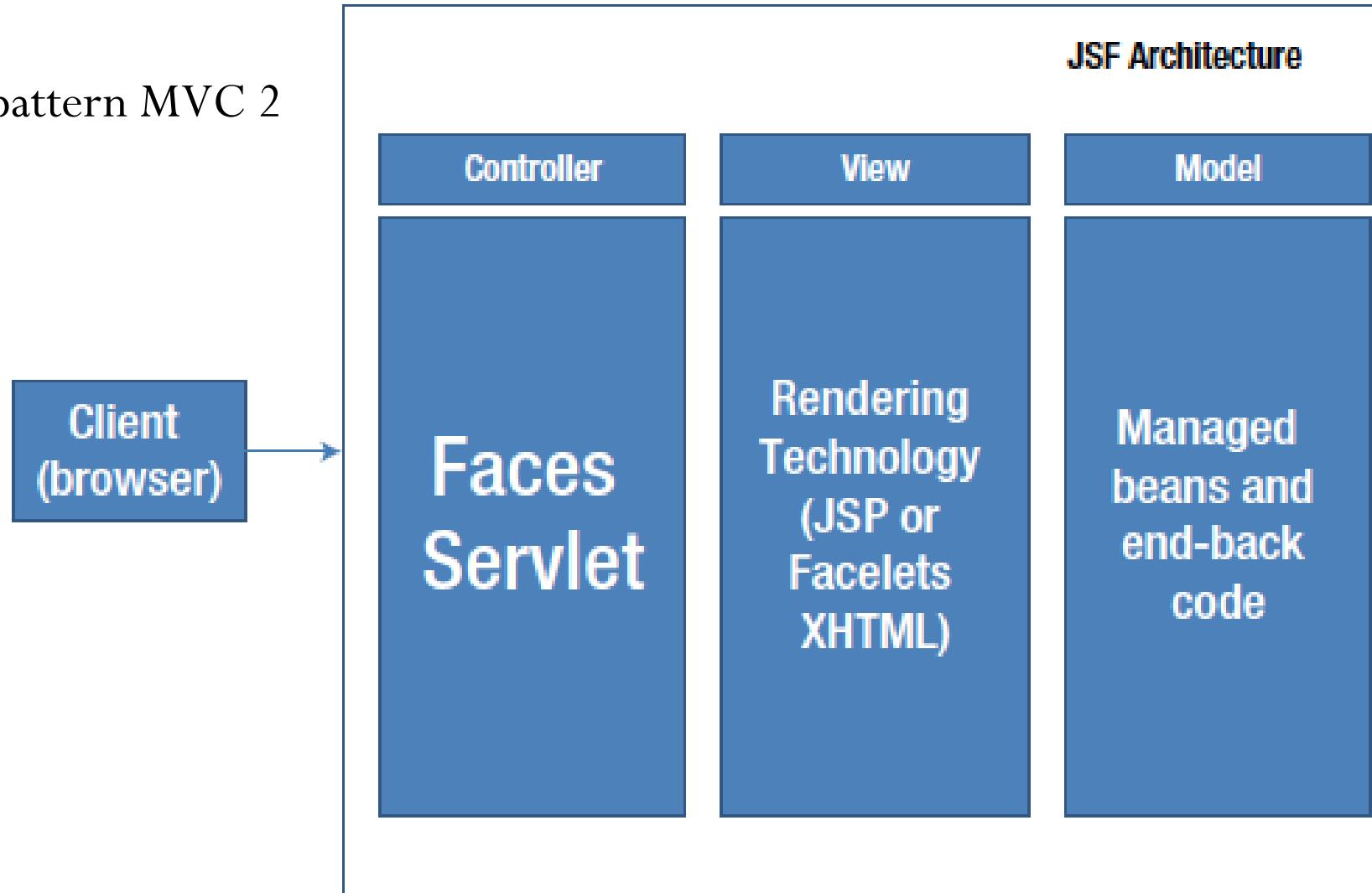
## Les concurrents de JSF



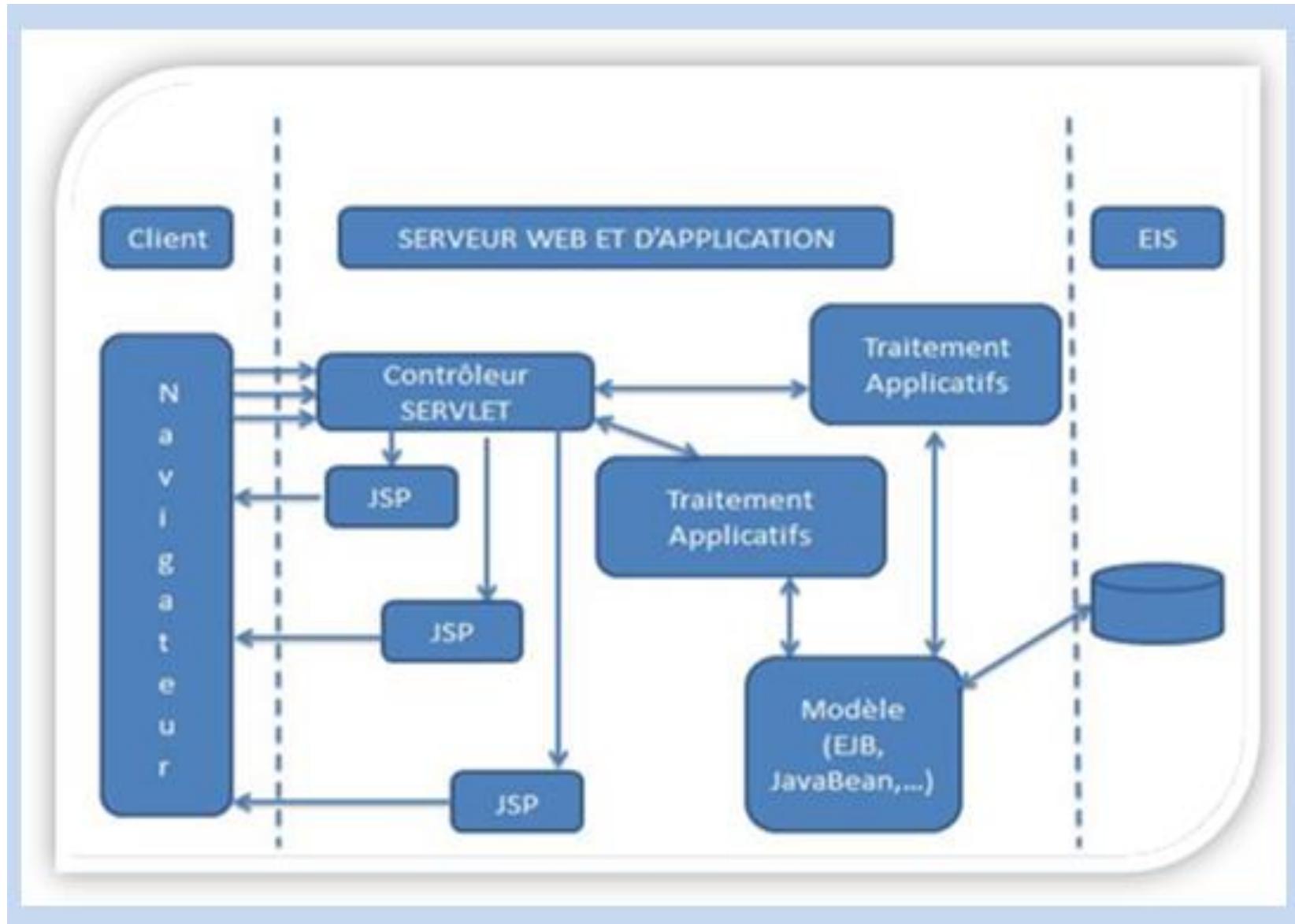
- JSF se positionne comme la plus part des Frameworks qui consistent à répondre aux problématiques du développement des applications Web côté IHM

## ARCHITECTURE JSF

- L'architecture JSF repose sur le pattern MVC 2

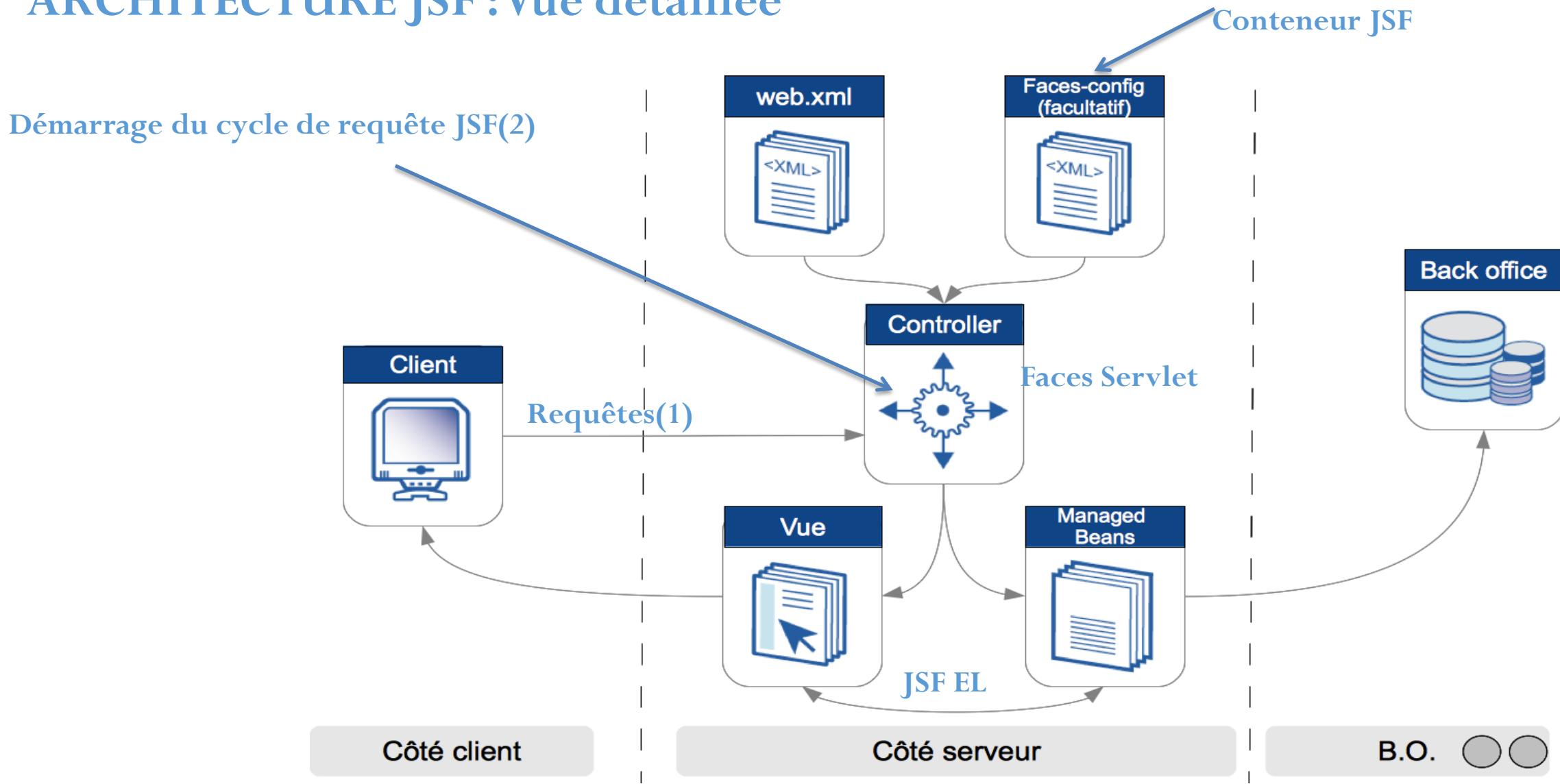


## Le Modèle MVC 2



# ARCHITECTURE JSF

## ARCHITECTURE JSF : Vue détaillée



## Les éléments au cœur de l'architecture JSF

JSF repose principalement sur les trois éléments suivants :

- JSF Expression Language qui permet d'accéder rapidement en:  
Lecture comme en écriture aux propriétés des composants et des Beans du modèle géré.
- Deux bibliothèques de balises spécifiques, étendant le jeu de balises standards JSP.  
Celles-ci sont conçues pour faciliter l'intégration des composants graphiques dans les pages JSP, ainsi que pour permettre leur liaison avec les composants côté serveur.
- Une API spécifique permettant de disposer d'une représentation des composants graphiques côté serveur.

# Cycle de vie de requête JSF 2

## Cycle de vie de requête JSF 2 : Exemple démonstratif

1- Vue développée par le développeur

2- Faces Servlet génère un arbre de composant de la vue .

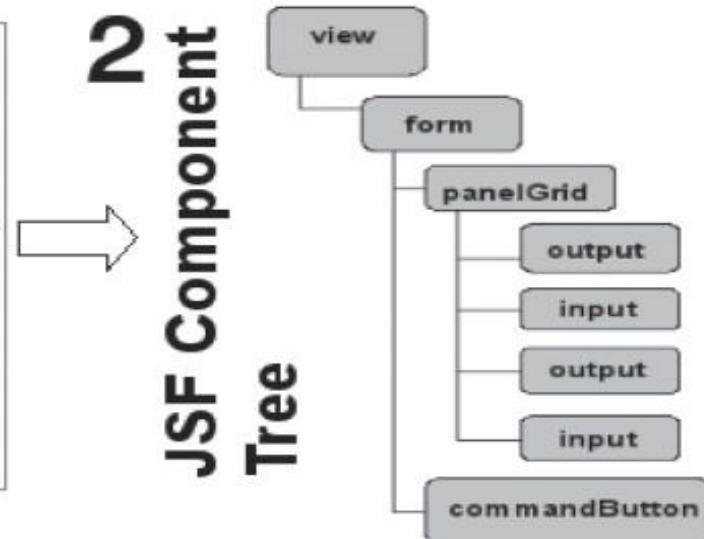
3-Après Exécution sur le serveur , il Génère une page au format HTML.

4- Le navigateur interprète ce code

### 1 Development

```
<h:form>
  <h:panelGrid columns="2">
    <h:outputText value="#{msg.emailLabel}" />
    <h:inputText value="#{login.email}" />
    <h:outputText value="#{msg.passwordLabel}" />
    <h:inputText value="#{login.password}" />
  </h:panelGrid>
  <h:commandButton action="#{login.check}"
                    value="#{msg.btnLabel}" />
</h:form>
```

### 2 JSF Component Tree



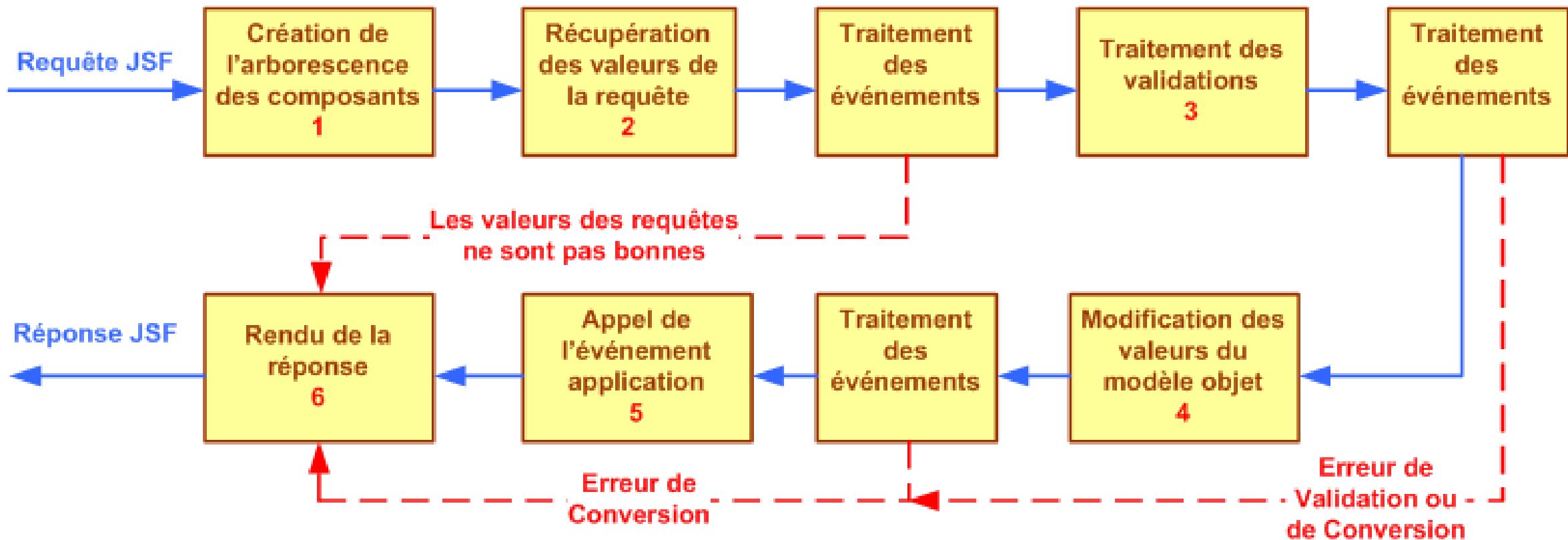
### 3 Output

HTML	Browser
<td>Email:</td> <td><input type="text" name="j_id2:j_id5" value="" /></td> </tr> <tr> <td>Password:</td> <td><input type="text" name="j_id2:j_id7" value="" /></td> </tr> </tbody> </table> <input type="submit" name="j_id2:j_id8" value="Sign In" />	Email: Password: Sign In

# Cycle de vie de requête JSF 2

## Les 6 étapes clés du cycle de vie requête JSF

Ces phases sont gérées par le servlet « Faces Servlet »(activé lorsque qu'une requête /faces/\* ou \*.xhtml).



## Cycle de vie de requête JSF 2

- Les 6 phases du cycle sont ordonnées de manière logique:
  - ❖ ***Restore View***: rétablit ou crée la vue demandée par le client
  - ❖ ***Apply Request Value***: met à jour les objets Java des composants (UIComponent) avec les données envoyées par le client.
  - ❖ ***Process Validations***: effectue les validations/conversions des nouvelles données
  - ❖ ***Update Model Values***: met à jour les propriétés des beans associées aux composants
  - ❖ ***Invoke Application***: exécute l'action
  - ❖ ***Render Response***: sauvegarde l'état de la vue puis renvoie la réponse

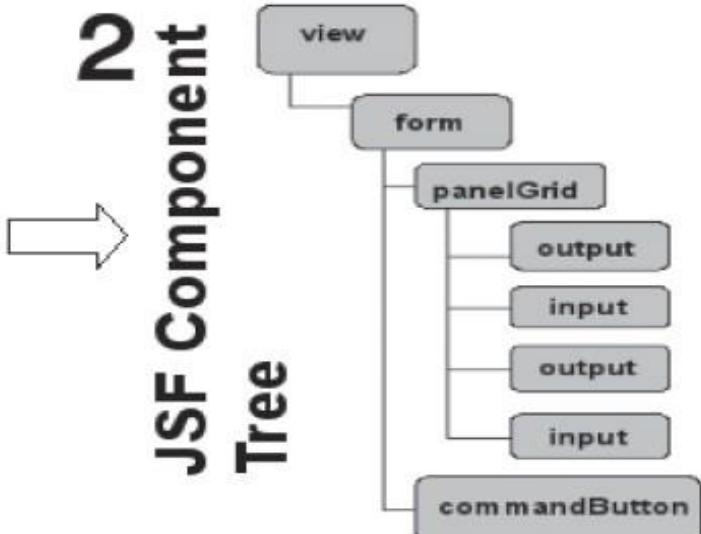
# Cycle de vie de requête JSF 2

## Etape 1 : restauration de la vue

1 Development

```
<h:form>
  <h:panelGrid columns="2">
    <h:outputText value="#{msg.emailLabel}" />
    <h:inputText value="#{login.email}" />
    <h:outputText value="#{msg.passwordLabel}" />
    <h:inputText value="#{login.password}" />
  </h:panelGrid>
  <h:commandButton action="#{login.check}"
                    value="#{msg.btnLabel}" />
</h:form>
```

2 JSF Component Tree



3 Output

HTML	Browser
<td>Email:</td> <td><input type="text" name="j_id2:j_id5" value="" /></td> </tr> <tr> <td>Password:</td> <td><input type="text" name="j_id2:j_id7" value="" /></td> </tr> </tbody> </table> <input type="submit" name="j_id2:j_id8" value="Sign In" />	Email:  Password:  Sign In

## Cycle de vie de requête JSF 2

### Etape 1 : restauration de la vue

- Lors de cette phase, la vue demandée est soit **restaurée**, soit **créée**.
- Elle sera **créée** uniquement dans le cas d'une requête de type “*nonpostback*“.
- *Créée* = Construire en mémoire d'un arbre de composant.
- Une requête “*postback*” serait, par exemple, la validation d'un formulaire précédemment affiché.
- JSF conserve l'état de la vue dans son contexte (**FacesContext**) lors de la dernière phase du cycle de vie (Render Response).

## Cycle de vie de requête JSF 2

### Etape 2 : application des paramètres de la requête

- **Mapping** entre **les valeurs des composants** (UIComponent) de l'arbre hiérarchique et celles contenues dans **les éléments graphiques** du client.
- Les valeurs des inputs de la vue seront reportées dans l'arbre de composants JSF.
- En fait, chaque composant de la vue récupère ses propres paramètres dans la requête http.
- Par exemple, si le composant texte d'un formulaire contient un nom, le composant Java associé conserve ce nom dans une variable

## Cycle de vie de requête JSF 2

### Etape 3 : validation et conversion

- C'est là que sont effectuées les validations et les conversions associées aux données.
- Si une validation échoue, la main est donnée à la phase de rendu de la réponse.
- Si une erreur apparaît, un **FacesMessage** est immédiatement ajouté à la stack afin d'être potentiellement rendu sur la vue prochainement affichée.
- les phases suivantes sont sautées

## Cycle de vie de requête JSF 2

### Etape 4 : mise à jour du model

- Les attributs des beans (modèle) de l'application seront mis à jour avec les valeurs contenues dans les composants avec lesquels ils sont liés.

### Etape 5 : logique métier

- Durant cette phase s'effectue le traitement **concret** des données.
- Les actions associées aux boutons ou aux liens sont exécutées.
- Le plus souvent le lancement des processus métier se fait par ces actions.
- La valeur de retour de ces actions va déterminer la prochaine page à afficher (navigation)

## Cycle de vie de requête JSF 2

### Etape 6 : rendu de la réponse

- La page déterminée par la navigation est encodée en HTML et envoyée vers le client HTTP  
C'est également ici qu'est sauvegardé l'état de la vue, au cas où elle serait restaurée dans le futur, suite à une requête **postback**.

### Fichiers clés : **web.xml** et **faces-config.xml**

- **WEB-INF/web.xml** fichier global de configuration d'une application Web Java
- **WEB-INF/faces-config.xml** : configuration de la partie JSF de l'application, par exemple pour indiquer la navigation entre les pages ou pour déclarer les beans utilisés par les pages ;
- **faces-config.xml** : plus indispensable avec JSF 2.x grâce aux annotations Java

### Configuration du contrôleur FacesServlet dans web.xml

- Toutes les requêtes vers des pages « JSF » sont interceptées par un servlet défini dans le fichier web.xml de l’application Web

```
<servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
```

## Le fichier faces-config.xml

- Le fichier gérant la logique de l'application web s'appelle par défaut faces-config.xml
- Il est placé dans le répertoire WEB-INF au même niveau que web.xml
- Il décrit essentiellement six principaux éléments :
  - les Beans managés <managed-bean>
  - les règles de navigation <navigation-rule>
  - les ressources éventuelles suite à des messages <message-bundle>
  - la configuration de la localisation <resource-bundle>
  - la configuration des Validators et des Converters <validator> , <converter>
  - autres éléments liés à des nouveaux composants JSF <render-kit>

## Configuration JSF 2

**Url gérés : \*.faces, \*.jsf, /faces/\***

Les pages JSF sont traitées par le servlet parce que le fichier web.xml contient une configuration telle que celle-ci :

```
<servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>/faces/*</url-pattern>
</servlet-mapping>
```

Le pattern est souvent aussi de la forme **\*.xhtml** ou **\*.faces** ou **\*.jsf...**

### Paramétrer le mode développement : javax.faces.PROJECT\_STAGE

Pour voir les erreurs pendant le développement

```
<context-param>
    <param-name>javax.faces.PROJECT_STAGE</param-name>
    <param-value>Development</param-value>
</context-param>
```

### Annotations ou xml ?

JSF 2.0 a développé l'emploi des annotations des classes pour configurer son fonctionnement

Le fichier XML l'emporte sur les annotations, ce qui permet :

- de changer la configuration même si on n'a pas le code source
- d'éviter de recompiler si on a le code source.

## Installation(1/3)

➤ **Obtenir les logiciels requis:**

**Java**

- Tomcat 6 avec Java 5
- Tomcat 7 avec Java 6
- JBoss 6 ou JBoss 7
- Glassfish 3

...

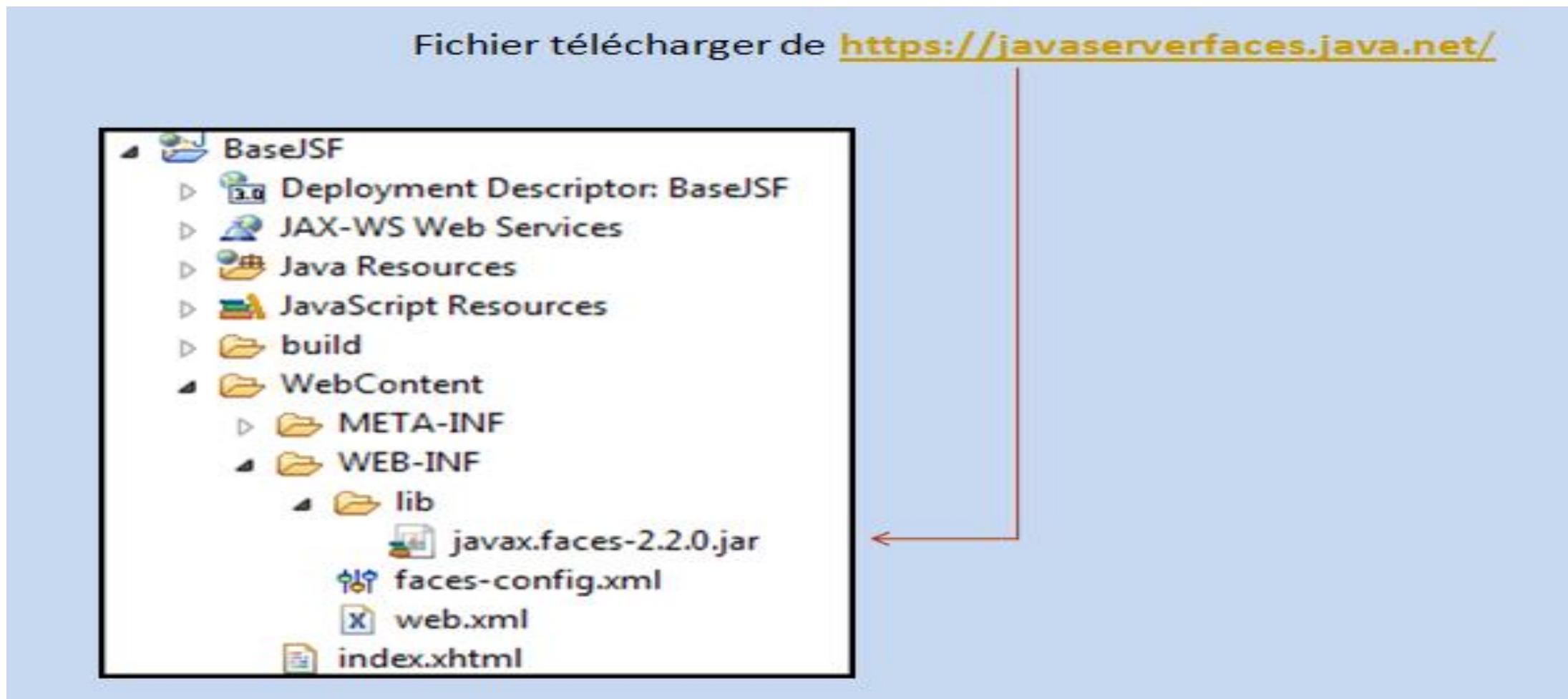
➤ **Librairie à ajouter au Serveur**

Pour les **serveurs** qui supportent **servlet 2.5** (Ex: Tomcat, Jetty...) => Utiliser [*javax.faces-2.1.7*] + [JSTL 1.2 JARs ]

➤ Pour les **serveurs JEE 6** (Ex: JBoss 6, Glassfish 3 ...) **rien à ajouter**

# Configuration JSF 2

## Installation(2/3)



# Configuration JSF 2

## Installation(3/3)

### web.xml

```
<web-app ...>
    <context-param>
        <param-name>javax.faces.PROJECT_STAGE</param-name>
        <param-value>Development</param-value>
    </context-param>

    <servlet>
        <display-name>Faces Servlet</display-name>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>/faces/*</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>Faces Servlet</servlet-name>
        <url-pattern>*.xhtml</url-pattern>
    </servlet-mapping>

    <welcome-file-list>
        <welcome-file>index.xhtml</welcome-file>
    </welcome-file-list>
</web-app>
```

Version 3.0 pour Tomcat 7 ou JEE 6

**Context**  
*Production, la valeur par défaut*

**Faces Servlet**

**URL mapping**

Plusieurs Formes:  
\* .jsf  
\* .faces  
\* .xhtml

---

**TP : Lab\_BaseJSF**

## Navigation JSF 2

- JSF 2, introduit la navigation implicite ([implicit navigation](#)), navigation de l'application sans fichier de configuration XML.
- Il existe deux types de navigation:
  - la navigation directe: appel à la page XHTML sans passer par le Managed Bean
  - navigation indirecte: appel à une méthode du Managed Bean

## Navigation JSF 2

- La navigation entre les pages indique quelle page est affichée quand l'utilisateur clique sur:
  - un bouton pour soumettre un formulaire
  - ou sur un lien
- La navigation peut être définie par des règles :
  - dans le fichier de configuration **faces-config.xml**
  - par des valeurs écrites dans le code Java(implicite/explicite) dans la page JSF

## Navigation statique (1/3)

- Définie « **en dur** » dans la page JSF.

Exemple :

```
<h:commandButton value="Texte du bouton" action="pageSuivante"/>
```

- Lorsque l'utilisateur clique sur le lien HTML correspondant alors **l'outcome pageSuivante est renvoyé au** gestionnaire de navigation de JSF.
- Le gestionnaire cherche une règle de navigation applicable au contexte courant :

# Navigation JSF 2

➤ Par défaut :

- si **pageSuivante.xhtml** existe alors cette page est renvoyée
- si une correspondance est trouvée alors la page suivante est affichée
- sinon la page courante est rechargée

## Navigation statique (2/3)

Avec le fichier de configuration **faces-config.xml**

```
<navigation-rule>
    <from-view-id>/index.xhtml</from-view-id>   ← page source
    <navigation-case>
        <from-outcome>pageSuivante</from-outcome>
        <to-view-id>/succes.xhtml</to-view-id>
    <navigation-case>
        ...
</navigation-rule>
```

M.Mbengue

# Navigation JSF 2

## Navigation statique (3/3)

```
<h:commandButton value="Texte du bouton"  
action= "pageSuivante"/>
```

- si une correspondance est trouvée alors la page est affichée
- sinon la page courante est rechargée

```
<navigation-rule>  
  <from-view-id>/index.xhtml</from-view-id>  
  <navigation-case>  
    <from-outcome>pageSuivante</from-outcome>  
    <to-view-id>/succes.xhtml</to-view-id>  
  <navigation-case>  
</navigation-rule>
```

## Navigation dynamique (1/3)

- C'est une méthode qui retourne un nom de page:

```
<h:commandButton value="Texte du bouton" action="# {nomBean.nomMethode}"/>
```

L'écriture **# {nomBean.nomMethode}** indique à JSF que cette expression est une EL ou Expression Language.

Exemple :

- Paramétrages

```
<navigation-rule>
    <from-view-id>/formulaire.jsp</from-view-id>
    <navigation-case>
        <from-outcome>Ok</from-outcome>
        <to-view-id>/ok.jsp</to-view-id>
    </navigation-case>
    <navigation-case>
        <from-outcome>Erreur</from-outcome>
        <to-view-id>/erreur.jsp</to-view-id>
    </navigation-case>
</navigation-rule>
```

## Navigation JSF 2

- Règle de navigation

Lorsqu'on clique sur ce bouton, le serveur appellera la méthode action() du Bean nomBean.

```
<h:commandButton value="Valider" action="# {nomBean.action}" />
```

L'action qui est appelée est codé comme ci-dessus :

```
public void action() {  
    If (this.rempli) return "Ok";  
    else return "Erreur";  
}
```

Si l'action retourne "Ok", on redirigera le client vers /ok.jsp sinon si elle retourne "Erreur", on le redirige vers /erreur.jsp

## Compléments – Joker

- S'il n'y a pas d'élément <from-view-id>, la règle de navigation s'applique à toutes les pages  
On peut mettre un joker à la fin de la valeur de

```
<from-view-id>/alpha/*</from-view-id>
```

- On peut ajouter <redirect/> à la fin de la règle de navigation pour que l'URL de la page soit modifié (redirection à la place d'un forward)

# Navigation JSF 2

## Compléments – <from-action>

Exemple :

```
<h:form>
    <h:commandButton action="#{pageController.processPage1}" value="Page1"
    />

    <h:commandButton action="#{pageController.processPage2}" value="Page2"
    />

</h:form>
```

- Règle de navigation

```
<faces-config ..... >
    <navigation-rule>
        <from-view-id>start.xhtml</from-view-id>
        <navigation-case>
            <from-action>#{pageController.processPage1}</from-action>
            <from-outcome>success</from-outcome>
            <to-view-id>page1.xhtml</to-view-id>
        </navigation-case>
        <navigation-case>
            <from-action>#{pageController.processPage2}</from-action>
            <from-outcome>success</from-outcome>
            <to-view-id>page2.xhtml</to-view-id>
        </navigation-case>
    </navigation-rule>
</faces-config>
```

## Compléments – Navigation conditionnelle

- La condition doit être remplie pour que le cas de navigation soit pris en compte

### EXEMPLE

```
<navigation-rule>
    <from-view-id>start.xhtml</from-view-id>
    <navigation-case>
        <from-outcome>payment</from-outcome>
        <if> #{paymentController.orderQty < 100}</if>
        <to-view-id>ordermore.xhtml</to-view-id>
    </navigation-case>
    <navigation-case>
        <from-outcome>payment</from-outcome>
        <if> #{paymentController.registerCompleted}</if>
        <to-view-id>payment.xhtml</to-view-id>
    </navigation-case>
    <navigation-case>
        <from-outcome>payment</from-outcome>
        <to-view-id>register.xhtml</to-view-id>
    </navigation-case>
</navigation-rule>
```

---

# TP : Lab\_Navigation

# Beans Managés

- Définition et Rôle du Bean managé
- Caractéristiques du bean mangé
- Comprendre les annotations : `@ManagedBean`, `@ManagedProperty`
- Instanciation des Beans Managés, méthodes et property des Beans
- Paramètres par défaut attribut: "name" et "scope »
- Initialisation des propriétés de type "Collection »
- Durée de vie, scope : `@SessionScoped`, `@RequestScope`, `ViewScoped...`

# Beans Managés

## Rôle dans l'architecture JSF

Rappelons qu'un Bean est une classe Java respectant un ensemble de directives:

- Un constructeur public sans argument
- Les propriétés d'un Bean sont accessibles au travers de méthodes
- getXXX et setXXX portant le nom de la propriété

L'utilisation des Beans dans JSF permet:

- l'affichage des données provenant de la couche métier
- le stockage des valeurs d'un formulaire
- la validation des valeurs
- l'émission de messages pour la navigation (reçus par facesconfig.xml)

# Beans Managés

## Caractéristiques du bean mangé

JSF « manage » le bean c'est à dire :

Qu'il l'instancie automatiquement,

Nécessité d'avoir un constructeur par défaut

Qu'il contrôle son cycle de vie

- ✓ Ce cycle dépend du « scope » (request, session, application, etc.)

Appelle les accesseurs avec :

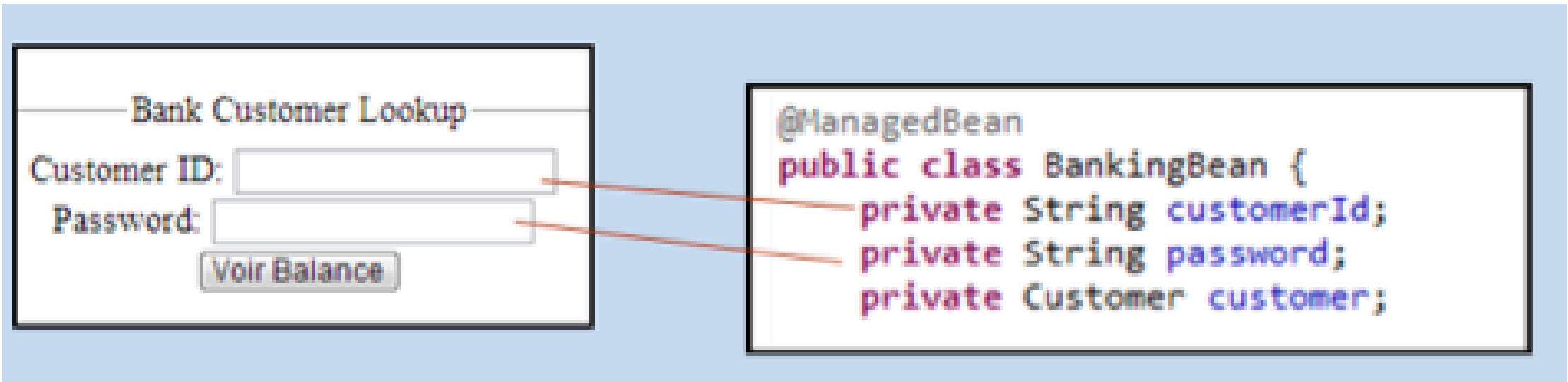
- ✓ les méthodes setter
- ✓ les méthodes getter

# Beans Managés

## Des propriétés

Une paire pour chaque élément input de formulaire,

Les setters sont automatiquement appelés par JSF lorsque le formulaire sera soumis.



# Beans Managés

## Des méthodes « action »

Une par bouton de soumission dans le formulaires (un formulaire peut avoir plusieurs boutons de soumission),

La méthode sera appelée lors du clic sur le bouton par JSF

```
public String showBalance() {
    if (!password.equals("secret")) {
        return ("wrong-password");
    }
    customer = LookupService.findCustomer(customerId);
    if (customer == null) {
        return ("unknown-customer");
    } else if (customer.getBalance() < 0) {
        return ("negative-balance");
    } else if (customer.getBalance() < 10000) {
        return ("normal-balance");
    } else {
        return ("high-balance");
    }
}
```

# Beans Managés

## Des propriétés pour les données résultat

Seront initialisées par les méthodes « action » après un traitement métier,

Il faut au moins une méthode get sur la propriété afin que les données puissent être affichées dans une page de résultat.

```
private Customer customer;

private static CustomerService lookupService = new CustomerServiceImpl();

//...

customer = lookupService.findCustomer(customerId);
```

# Beans Managés

## Injection de dépendance dans JSF

Deux méthodes :

- par xml(faces-config.xml)
- par annotation

### Par xml(faces-config.xml)

JSF réalise l'injection de dépendances via l'utilisation de l'EL dans faces-config.xml.

Exemple :

```
<managed-bean>
    <managed-bean-name>address</managed-bean-name>
    <managed-bean-class>package.Address</managed-bean-class>
</managed-bean>
<managed-bean>
    <managed-bean-name>person</managed-bean-name>
    <managed-bean-class>package.Person</managed-bean-class>
    <managed-property>
        <property-name>address</property-name>
        <property-class>package.Address</property-class>
        <value># {address} </value>
    </managed-property>
</managed-bean>
```

# Beans Managés

## Par annotation

```
@ManagedBean (name="address")  
@SessionScoped  
Public class Address implements Serializable {
```

Puis

```
@ManagedBean  
@SessionScoped  
public class Person implements Serializable {  
  
    @ManagedProperty (value="#{address}")  
    private Address address;  
  
    //must provide the setter method  
    Public void setMessageBean(MessageBean messageBean) {  
        this.messageBean = messageBean;  
    }  
    //...  
}
```

# Beans Managés

## Instanciation des Beans

Deux possibilités :

- Déclarer le bean dans le fichier de config de JSF(faces-config.xml, optionnel depuis la version 2.0)
- Utiliser des annotations (fortement conseillé depuis la version 2.0)

### Déclarer dans faces-config.xml

Trois éléments essentiels sont à préciser

- **<managed-bean-name>** définit un nom qui servira d'étiquette quand le Bean sera exploité dans les pages JSP
- **<managed-bean-class>** déclare le nom de la classe de type package.class
- **<managed-bean-scope>** précise le type de scope utilisé pour le Bean (none, application, session, request)

# Beans Managés

## EXEMPLE DE DECLARATION DANS FACES-CONFIG.XML

```
<managed-bean>
    <description>Client Bean</description>
    <managed-bean-name>myCli</managed-bean-name>
    <managed-bean-class>com.formation.ClientBean</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

## Utiliser des annotations

```
@ManagedBean(name="myCli")
@SessionScoped
public class Client {...}
```

---

**TP : Lab\_ManagedBean**

# Beans Managés

## Injection de dépendance dans JSF

Deux méthodes :

- par xml(faces-config.xml)
- par annotation

### Par xml(faces-config.xml)

JSF réalise l'injection de dépendances via l'utilisation de l'EL dans faces-config.xml.

Exemple

```
<managed-bean>
    <managed-bean-name>address</managed-bean-name>
    <managed-bean-class>package.Address</managed-bean-class>
</managed-bean>
<managed-bean>
    <managed-bean-name>person</managed-bean-name>
    <managed-bean-class>package.Person</managed-bean-class>
    <managed-property>
        <property-name>address</property-name>
        <property-class>package.Address</property-class>
        <value># {address} </value>
    </managed-property>
</managed-bean>
```

# Beans Managés

## Par annotation

```
@ManagedBean (name="address")  
@SessionScoped  
Public class Address implements Serializable {
```

Puis

```
@ManagedBean  
@SessionScoped  
public class Person implements Serializable {  
  
    @ManagedProperty (value="#{address}")  
    private Address address;  
  
    //must provide the setter method  
    Public void setMessageBean(MessageBean messageBean) {  
        this.messageBean = messageBean;  
    }  
    //...  
}
```

# Beans Managés

## Les scopes

Ils indiquent la durée de vie des managed Beans.

Valeurs possibles :

- **request,**
- **session,**
- **application,**
- **view,**
- **conversation,**
- **aucun**
- **ou custom**

**RequestScope** = valeur par défaut.

On les spécifie dans :

- faces-config.xml
- ou sous forme d'annotations de code

# Beans Managés

## Annotations pour les Scopes(1/2)

### `@RequestScoped`

On crée une nouvelle instance du bean pour chaque requête.

### `@SessionScoped`

On crée une instance du bean et elle durera le temps de la session.

Le bean doit être Sérifiable.

### `@ApplicationScoped`

Met le bean dans « l'application », l'instance sera partagée par tous les utilisateurs de toutes les sessions.

# Beans Managés

## Annotations pour les Scopes(2/2)

### @ViewScoped

La même instance est utilisée aussi souvent que le même utilisateur reste sur la même page, même s'il fait un refresh (reload) de la page !

A été conçu spécialement pour les pages JSF faisant des appels Ajax.

### @NoneScoped

Le bean est instancié mais pas placé dans un Scope.

Utile pour des beans qui sont utilisés par d'autres beans qui sont eux dans un autre Scope.

### @CustomScoped(value=« #{{uneMap}} »)

Le bean est placé dans la HashMap et le développeur gère son cycle de vie.

# JSF Expression Language Unified EL

Sert à relier des propriétés ou des méthodes de beans avec des valeurs de pages JSF

Une expression EL est entourée par `#{...}`

## Exemple

Bean	Test Page
<pre>@ManagedBean public class SimpleBean {     private String[] colors =         { "red", "orange", "yellow" };      public String getMessage() {         return("Hello, World");     }      public String[] getColors() {         return(colors);     } }</pre>	<pre>&lt;!DOCTYPE ...&gt; &lt;html xmlns="http://www.w3.org/1999/xhtml"       xmlns:h="http://java.sun.com/jsf/html"&gt; ... &lt;ul&gt;     &lt;li&gt;Message: #{simpleBean.message}&lt;/li&gt;     &lt;li&gt;First color: #{simpleBean.colors[0]}&lt;/li&gt; &lt;/ul&gt; ...</pre>

# JSF Expression Language Unified EL

## Syntaxe du langage

Evaluation d'une propriété de type « **a.b** »

**a** peut être :

- un tableau
- une liste
- une map
- un objet quelconque

Syntaxe spéciale pour les 3 premiers types.

**a.b avec a=objet « ordinaire »**

*Si a est un objet ‘(qui n’est ni un tableau, ni une liste, ni une map)*

*Alors a.b désigne une propriété b de a*

a.b peut désigner le getter a.getB ou le setter a.setB selon les cas

```
<h:inputText value="#{bean.nom}" />
```

## JSF Expression Language Unified EL

la valeur affichée au départ sera celle renvoyée par getNom

la valeur tapée par l'utilisateur sera rangée dans le modèle par la méthode setNom

**a.b avec a=Map**

*Si a est une map*

*Alors a["cle"] ou a.cle désigne la valeur de la map de clé*

*Si a est une liste ou tableau,*

*Alors a[i] ou a.i désigne l'élément numéro i (commence à 0)*

# JSF Expression Language Unified EL

## Opérateurs

- **Arithmétiques** : + - \* / (ou div) % (ou mod)
- **De comparaison** : < <= > >= == != (ou lt le gt ge eq ne)
- **Logiques** : && || ! (ou and or not)
- **empty** ; empty a est vrai si a est null, un tableau ou une chaîne vide, une collection ou une map de longueur 0
- **Ternaire ? :** (comme en Java)

# Beans Managés

---

TP : Lab\_Injection

## Gestion des ressources avec JSF2

- Rôle du répertoire clé : 'resources'
- Gérer des feuilles de style avec : h:outputStylesheet
- Gérer des images : h:graphicImage
- Gérer des scripts javascript : h:outputScript
- Rôle des attributs communs:
  - 'library'
  - 'name'

## Gestion des ressources avec JSF2

➤ La plupart des composants ont besoin de ressources externes pour s'afficher correctement :

- <h:graphicImage> a besoin d'une image,
- <h:commandButton> peut afficher une image pour représenter le bouton,
- <h:outputScript> référence un fichier JavaScript

et surtout les composants peuvent également appliquer des styles CSS.

### Convention : répertoire 'resources'

➤ Avec JSF, une ressource est un élément statique qui peut être transmis aux éléments afin d'être : affiché (images) ou traité (CSS) par le navigateur.

## Gestion des ressources avec JSF2

- JSF 2.0 permet d'assembler directement les ressources dans un fichier jar séparé, avec un numéro de version et une locale, et de les placer à la racine de l'application web, sous le répertoire suivant :

**resources/<identifiant\_ressource>**

ou

**META-INF/resources/<identifiant\_ressource>**

<identifiant\_ressource> est formé de plusieurs sous-répertoire indiqués sous la forme :

**[locale/] [nomBibliothèque/] nomRessource [/versionRessource]**

Tous les éléments entre crochets sont facultatifs.

# Gestion des ressources avec JSF2

Exemple :

**resources/css/principal.css**

**resources/images/logo.png**

**resources/javascript/jsf.js**

Prendre en compte ces différentes ressources

```
<h:outputStylesheet library="css" name="principal.css" />
<h:outputScript library="javascript" name="jsf.js" target="head" />
<h:graphicImage library="images" name="logo.png" />
```

## Gestion des messages utilisateur

### Messages d'erreur ou d'information

- Les conversions et validations peuvent conduire à l'affichage de messages d'erreur
- Les convertisseurs et validateurs standards produisent des messages d'erreur standard
- Il est possible de modifier les messages d'erreur standard
- Lorsqu'une action s'est bien déroulée et que la page courante reste la même, il faut le signaler à l'utilisateur par un message d'information.

# Gestion des messages utilisateur

## Affichage de messages dans l'interface web

### <h:messages>

- Réserve un endroit de la page pour afficher tous les messages pour tous les composants et les messages qui ne sont pas liés à un composant particulier.
- De nombreux attributs permettent de définir le style CSS et la mise en page et d'indiquer si on veut seulement un résumé ou les détails du message

### Exemple :

```
<h:messages errorClass="erreur" />
```

## Gestion des messages utilisateur

### <h:message>

Un attribut obligatoire **for** indique l'identificateur du composant qui est lié à ce message.  
Habituellement placé près du composant dans la page JSF

```
<h:message for="nom" />
```

### Structure des messages JSF : la classe *FacesMessage*

- Un message peut être généré automatiquement en liaison avec une erreur de conversion ou de validation.
- Il peut aussi être généré par du code Java en utilisant la classe *FacesMessage*
- Il suffit de passer une instance de *FacesMessage* à la méthode *addMessage* de *FacesContext*
- Le message sera affiché si la page courante est réaffichée

# Gestion des messages utilisateur

## Ajouter un message

`FacesContext.getCurrentInstance().addMessage` permet d'ajouter un message par programmation  
Les paramètres de la méthode addMessage permettent d'indiquer  
L'*id* du composant avec lequel le message est associé (peut être *null*)

*Le message* (classe FacesMessage)

## Id d'un composant

Exemple de désignation d'un composant :  
si le composant a l'id « *montant* » dans le formulaire d'id « *transfert* »,  
l'id du composant à passer en 1er paramètre de addMessage est « *transfert:montant* »

# Gestion des messages utilisateur

## Classe FacesMessage

Le constructeur peut prendre en paramètre

La « *gravité* » du message par une constante de la classe `FacesMessages.Severity` :

- `SEVERITY_FATAL` (erreur grave),
- `SEVERITY_ERROR`, `SEVERITY_WARN`(avertissement),
- `SEVERITY_INFO` (information ; valeur par défaut)
- Message résumé (summary) affiché par défaut par `<h:message>`
- Message complet (detail) affiché par défaut par `<h:messages>`

## Gestion des messages utilisateur

### Exemple de code

```
FacesContext.getCurrentInstance ().addMessage (   
component.getClientId (FacesContext.getCurrentInstance ()),  
new FacesMessage (FacesMessage.SEVERITY_INFO,"Détails du message", null));
```

# Internationaliser une application JSF 2

## Internationaliser une application JSF 2

- **Le fichier propriété**
- Nous pouvons facilement délocaliser les messages afin de simplifier le code et de faciliter l'internationalisation du logiciel.
- JSF utilise des fichiers de messages en lieu et place de textes en dur dans vos fichiers xhtml.
- Ce fichier de messages est un fichier au format *properties*.

Un fichier properties est un simple fichier texte qui contient une suite de **couples clé/valeur** représenté comme suit:

clé1=valeur1

clé2=valeur2

clé3=valeur3

# Internationaliser une application JSF 2

## Déclaration du fichier de messages (1/2)

Pour notre exemple, nous allons appeler notre fichier :

messages.properties

Et il sera placé dans le répertoire:

com/formation

Pour la déclaration 2 manières de faire :

- Soit dans vos fichier **xhtml**
- Soit dans le déclarer dans le fichier **faces-config.xml**

# Internationaliser une application JSF 2

## Déclaration du fichier de messages (2/2)

La première façon est de déclarer votre fichier dans vos fichiers *xhtml* de la façon suivante:

```
<f:loadBundle basename="com.formation.messages" var="messages" />
```

La deuxième façon est de le déclarer dans le fichier *faces-config.xml*

```
<application>
    <resource-bundle>
        <base-name> com.formation.messages </base-name>
        <var>messages</var>
    </resource-bundle>
</application>
```

# Internationaliser une application JSF 2

## Utilisation dans le code xhtml

### Exemple 1 :

```
# {messages.Name}
```

### Exemple 2 :

```
<h:outputLabel value="# {messages.Name}" />
```

### Exemple 3 :

Il est également possible d'utiliser des messages avec des paramètres.

Dans le fichier properties, le message avec paramètres se déclare comme suit:

```
facebook.hello=Hello {0} {1}
```

# Internationaliser une application JSF 2

Les chiffres entre accolades sont les numéros des paramètres utilisés dans le fichier xhtml:

```
<h:outputFormat value="# {messages ['facebook.hello']} >
  <f:param value="# {personnesListController.facebookProfile.firstName}" ></f:param>
  <f:param value="# {personnesListController.facebookProfile.lastName}" ></f:param>
</h:outputFormat>
```

Le paramètre 0 correspond au ***firstName*** et le paramètre 1 correspond au ***lastName***.

# Internationaliser une application JSF 2

## Utilisation dans le code Java

Pour charger ce fichier, le code est:

```
 ResourceBundle bundle = ResourceBundle.getBundle("com.formation.messages");
```

Puis

```
 String message = bundle.getString (key);
```

# Internationaliser une application JSF 2

## Rappel internationalisation Java

- Le but de l'internationalisation est de permettre à l'utilisateur de changer la langue des textes de votre application facilement.

### ➤ Configurer les langues disponibles

La première chose à faire pour rendre votre application multi langues, c'est de lister les langues disponible.

- Dans votre fichier *faces-config.xml* ajoutez ceci:

```
<application>
    <locale-config>
        <default-locale>fr</default-locale>
        <supported-locale>en</supported-locale>
        <supported-locale>es</supported-locale>
    </locale-config>
    <resource-bundle>
        <base-name>com.formation.messages</base-name>
        <var>messages</var>
    </resource-bundle>
</application>
```

# Internationaliser une application JSF 2

## Créer vos fichiers de langues

Lorsque vous n'utilisez pas l'internationalisation, le fichier de messages se nomme comme suit:

```
messages.properties
```

Format pour l'internationalisation

```
messages_langue.properties
```

**Exemple :**

```
messages_fr.properties  
messages_en.properties  
messages_es.properties
```

# Internationaliser une application JSF 2

## Rendre multi langues les pages JSF

Modifier les pages JSF pour que les langues soient prises en compte.

Pour ce faire, il faut ajouter dans notre page une balise **f:view** comme suit:

```
<f:view locale="fr" />
```

L'attribut locale sert à définir la langue en cours.

Vous pouvez également remplacer le « fr » de l'attribut par une propriété d'un bean:

```
<f:view locale="# {personnesListController.language}" />
```

Dans ce cas, la variable « **language** » doit être du type *java.util.Locale*.

# Internationaliser une application JSF 2

## Changer la locale de la vue en cours

```
FacesContext facesContext = FacesContext.getCurrentInstance ();
facesContext.getViewRoot ().setLocale (locale);
```

## TP : Lab\_Properties

# Les composants de la bibliothèque JSF 2

JSF met à disposition un certain nombre de classes composants couvrant la plupart des besoins classiques. Une page est une arborescence de classes héritant de *javax.faces.component.UIComponent*.  
JSF 2.2 couvre 4 librairies de base :

Uri	Préfixe classique	Description
<a href="http://xmlns.jcp.org/jsf/html">http://xmlns.jcp.org/jsf/html</a>	<b>H</b>	Contient les composants et leurs rendus HTML ( <b>h:commandButton, h:inputText, etc.</b> )
<a href="http://xmlns.jcp.org/jsf/core">http://xmlns.jcp.org/jsf/core</a>	<b>F</b>	Contient les actions personnalisées indépendantes d'un rendu particulier ( <b>f:convertNumber,</b> <b>f:validateDoubleRange, f:param, etc.</b> )
http://xmlns.jcp.org/jsf/facelets	<b>Ui</b>	Marqueurs pour le support des modèles de page.
http://xmlns.jcp.org/jsf/composite	<b>Cc</b>	Sert à déclarer et à définir des nouveaux composants personnalisés

# Les composants de la bibliothèque JSF 2

## Bibliothèque core

La bibliothèque **core** contient des balises qui sont indépendante du rendu HTML.

Elles sont utiles pour proposer des traitements et des réglages spécifiques pour les balises relatives à la vue (bibliothèque HTML).

Balises	Description
<code>&lt;f:attribute&gt;</code>	Propose un attribut particulier (clé/valeur) sur une balise parente.
<code>&lt;f:param&gt;</code>	Spécifie un paramètre particulier associé à la chaîne paramétrée proposée par la balise parente.
<code>&lt;f:facet&gt;</code>	Gestion d'une zone délimitée de la page Web.
<code>&lt;f:actionListener&gt;</code>	Ajoute un événement de type validation sur un composant parent.
<code>&lt;f:setPropertyActionListener&gt;</code>	Propose une propriété sur un événement de type validation associé à un composant parent.

# Les composants de la bibliothèque JSF 2

## Bibliothèque core

<b>&lt;f:valueChangeListener&gt;</b>	Ajoute un événement de type changement de valeur sur un composant parent.
<b>&lt;f:phaseListener&gt;</b>	Ajoute un événement de type changement de phase sur un composant parent.
<b>&lt;f:event&gt;</b>	Ajoute un système d'événement sur un composant parent.
<b>&lt;f:converter&gt;</b>	Ajoute un convertisseur personnalisé sur un composant parent.
<b>&lt;f:convertDateTime&gt;</b>	Ajoute un convertisseur prédéfini de type date ou heure sur un composant parent.
<b>&lt;f:convertNumber&gt;</b>	Ajoute un convertisseur prédéfini numérique sur un composant parent.
<b>&lt;f:validator&gt;</b>	Ajoute un validateur personnalisé à un composant parent.
<b>&lt;f:validateDoubleRange&gt;</b>	Ajoute un valideur prédéfini qui limite les valeurs numériques réelles sur un composant parent.

# Les composants de la bibliothèque JSF 2

## Bibliothèque core

<b>&lt;f:convertNumber&gt;</b>	Ajoute un convertisseur prédefini numérique sur un composant parent.
<b>&lt;f:validator&gt;</b>	Ajoute un validateur personnalisé à un composant parent.
<b>&lt;f:validateDoubleRange&gt;</b>	Ajoute un validateur prédefini qui limite les valeurs numériques réelles sur un composant parent.
<b>&lt;f:validateLength&gt;</b>	Ajoute un validateur prédefini qui limite le nombre de caractères saisie sur un composant parent.
<b>&lt;f:validateLongRange&gt;</b>	Ajoute un validateur prédefini qui limite les valeurs numériques entières sur un composant parent.
<b>&lt;f:validateRequired&gt;</b>	Vérifie qu'une valeur est présente sur un composant parent.
<b>&lt;f:validateRegex&gt;</b>	Vérifie que la valeur saisie respecte une expression régulière sur un composant parent.
<b>&lt;f:validateBean&gt;</b>	Utilise un bean de validation qui impose un certain nombre d'éléments à prendre en compte.
<b>&lt;f:loadBundle&gt;</b>	Prise en compte d'une ressource qui propose l'ensemble des messages paramétrés (gestion de la nationalité, par exemple).

# Les composants de la bibliothèque JSF 2

## Bibliothèque core

<code>&lt;f:selectItems&gt;</code>	Propose l'ensemble d'éléments qui vont remplir les radios boutons, les cases à cocher, la liste déroulante, etc.
<code>&lt;f:selectItem&gt;</code>	Propose un élément spécifique.
<code>&lt;f:verbatim&gt;</code>	Permet de donner le texte qui possède des balises sans interprétation particulière.
<code>&lt;f:viewParam&gt;</code>	Défini un paramètre pour la vue qui sera initialisé à l'aide du paramètre de la requête.
<code>&lt;f:ajax&gt;</code>	Permet d'implémenter des commandes ajax.
<code>&lt;f:view&gt;</code>	Utile si nous désirons prendre en compte la nationalité ou la gestion événementielle de phase



# Les composants de la bibliothèque JSF 2

## Exemple 1:

```
<h:outputFormat value="param0 : {0}, param1 : {1}" >  
    <f:param value="Number 1" />  
    <f:param value="Number 2" />  
</h:outputFormat>
```

## Exemple 2:

```
<h:dataTable value="#{auteurs}" var="élément" headerClass="inverse" columnClasses="ligne">  
    <h:column>  
        <f:facet name="header">Nom </f:facet>  
        # {élément.nom}  
    </h:column>  
    <h:column>  
        <f:facet name="header">Prénom </f:facet>  
        # {élément.prenom}  
    </h:column>  
</dataTable>
```

# Les composants de la bibliothèque JSF 2

## Bibliothèque html

Cette bibliothèque intègre cette fois-ci des composants visuels, qui assurent donc le rendu de la page web.

**Ensemble des attributs qui leurs sont communs.**

# Les composants de la bibliothèque JSF 2

## Bibliothèque core



Ensemble des attributs qui leurs sont communs.

Attributs	Description
<b>id</b>	Permet d'identifier le composant pour interagir avec un autre.
<b>binding</b>	Lie ce marqueur (ce composant dans sa globalité) directement avec le bean géré, et permet donc une interaction en coulisse.
<b>rendered</b>	Attribut très intéressant qui permet d'afficher ou pas la balise (équivalent d'un if). Valeurs possibles true ou false.
<b>value</b>	Attribut certainement le plus utilisé, qui permet d'être en relation directe avec une propriété particulière du bean géré déterminé par une expression EL (# {bean.propriété}).
<b>valueChangeListener</b>	Un écouteur spécifique qui entre en action lors d'un changement de valeur. La méthode désignée dans cet attribut est alors automatiquement sollicitée.
<b>converter</b>	Mise en relation avec un convertisseur personnalisé (classe qui implémente l'interface Converter) et qui permet de passer d'une chaîne de caractères (seule type compatible avec le protocole HTTP) vers un autre type quelconque et vice versa.

# Les composants de la bibliothèque JSF 2

## Bibliothèque core

<b>validator</b>	Mise en relation avec un validateur personnalisé (classe qui implémente l'interface Validator, ou éventuellement une méthode avec une signature spécifique) qui permet d'envoyer la valeur désignée dans l'attribut value au bean géré (qui travaille en coulisse) que si la valeur est correcte.
<b>required</b>	Une valeur est impérativement requise dans ce champ pour passer la phase de validation.
<b>converterMessage,</b> <b>validatorMessage,</b>	Messages personnalisés qui s'affichent automatiquement (en relation avec les balises <h:message> ou <h:messages>) lorsque un
<b>requiredMessage</b>	problème survient lors de la phase de conversion, de validation ou lorsque <u>q'une</u> valeur de saisie est oubliée.

## Les composants de la bibliothèque JSF 2

Nous pouvons les regrouper par catégories :

- Structure de la page HTML (head, body, form, outputStylesheet, outputScript)
- Dispositions (panelGrid, panelGroup)
- Tables (dataTable et column)
- Les entrées : (input...)
- Les sorties (output...)
- Les images (graphicImage)
- Les commandes (commandButton et commandLink)
- Les requêtes GET HTTP (button, link, outputLink)
- Les sélections (checkbox, listbox, menu, radio)
- Messages d'erreur (message, messages)

# Les composants de la bibliothèque JSF 2

## Structure de la page HTML

Balises	Description
<code>&lt;h:body&gt;</code>	Génère un élément <code>&lt;body&gt;</code> HTML.
<code>&lt;h:head&gt;</code>	Génère un élément <code>&lt;head&gt;</code> HTML.
<code>&lt;h:form&gt;</code>	Génère un élément <code>&lt;form&gt;</code> HTML.
<code>&lt;h:outputScript&gt;</code>	Génère un élément <code>&lt;script&gt;</code> HTML.
<code>&lt;h:outputStylesheet&gt;</code>	Génère un élément <code>&lt;link&gt;</code> HTML.

# Les composants de la bibliothèque JSF 2

## Structure de la page HTML

### Exemple 1:

```
<h:outputStylesheet library="css" name="style.css" />
```

Rendus :

```
<link type="text/css" rel="stylesheet"
      href="/JavaServerFaces/faces/javax.faces.resource/style.css?ln=css" />
```

### Exemple 2:

```
<h:outputScript library="js" name="common.js" />
```

Rendus :

```
<script type="text/javascript"
src="/JavaServerFaces/faces/javax.faces.resource/common.js?ln=js"></script>
```

# Les composants de la bibliothèque JSF 2

## Grilles et tableaux

Les données doivent souvent être affichées sous forme de tableau.

Balises	Description
<b>&lt;h:dataTable&gt;</b>	Représente un ensemble de données qui seront affichés dans un élément <table> HTML.
<b>&lt;h:column&gt;</b>	Produit une colonne de données dans un composant <h:dataTable>.
<b>&lt;h:panelGrid&gt;</b>	Produit également un élément <table> HTML.
<b>&lt;h:panelGroup&gt;</b>	Conteneur de composants pouvant s'imbriquer dans un <h:panelGrid>

JSF fournit donc le marqueur **<h:dataTable>** permettant ainsi de parcourir une liste d'éléments (dont la taille est variable) afin de générer automatiquement un tableau.

# Les composants de la bibliothèque JSF 2

## EXEMPLE 1

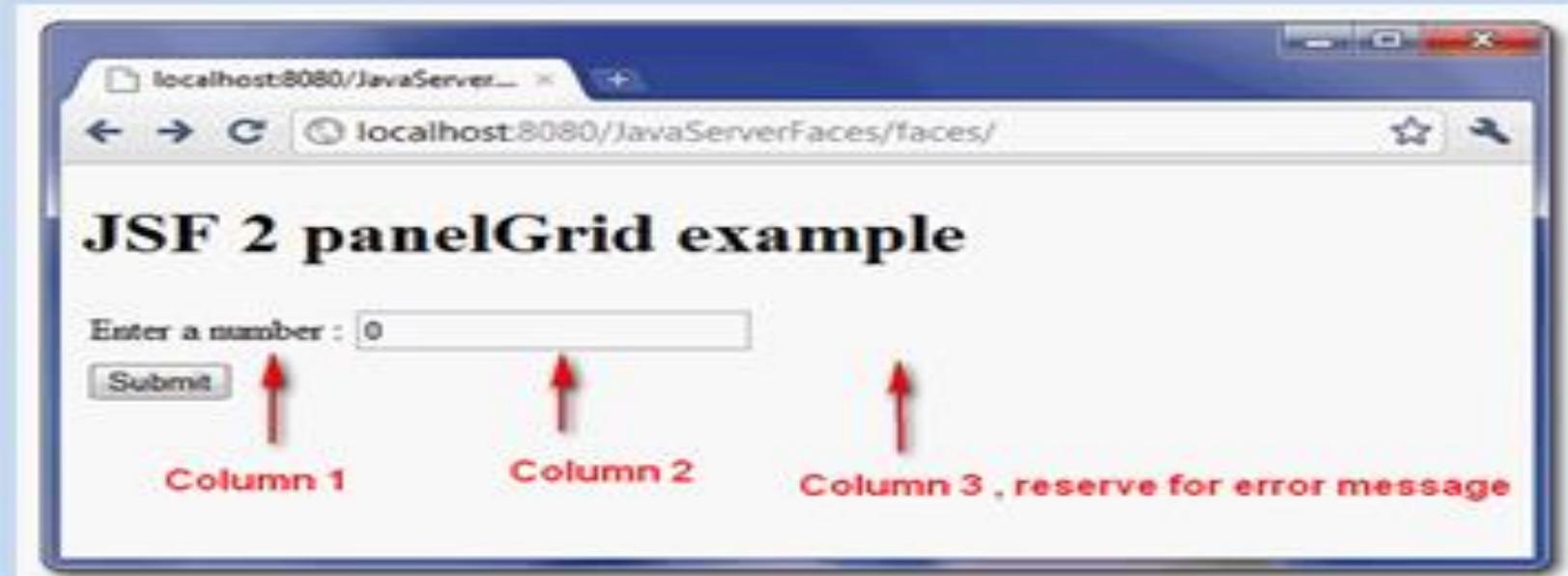
```
<h:dataTable value="#{userData.employees}" var="employee">
    <h:column>
        <f:facet name="header">Name</f:facet>
        #{employee.name}
    </h:column>
    <h:column>
        <f:facet name="header">Department</f:facet>
        #{employee.department}
    </h:column>
    <h:column>
        <f:facet name="header">Age</f:facet>
        #{employee.age}
    </h:column>
    <h:column>
        <f:facet name="header">Salary</f:facet>
        #{employee.salary}
    </h:column>
</h:dataTable>
```

```
@ManagedBean(name = "userData")
@SessionScoped
public class UserData implements Serializable {
    private static final long serialVersionUID = 1L;
    ...
    public ArrayList<Employee> getEmployees() {
        return employees;
    }
}
```

## Les composants de la bibliothèque JSF 2

### EXEMPLE 2

```
<h:form>
    <h:panelGrid columns="3">
        Enter a number :
        <h:inputText id="number" value="#{dummy.number}"
                     size="20" required="true"
                     label="Number">
            <f:convertNumber />
        </h:inputText>
        <h:message for="number" style="color:red" />
    </h:panelGrid>
    <h:commandButton value="Submit" action="result" />
</h:form>
```



# Les composants de la bibliothèque JSF 2

## Les entrées

Les entrées sont des composants qui affichent leur valeur courante (actuelle) et permettent à l'utilisateur de saisir différentes informations textuelles.

Il peut s'agir de champs de saisie, de zone de texte ou de composants pour entrer un mot de passe ou des données cachées.

Balises	Description
<b>&lt;h:inputHidden&gt;</b>	Représente un élément d'entrée HTML de type caché (non affiché).
<b>&lt;h:inputSecret&gt;</b>	Représente un élément d'entrée HTML de type mot de passe.
<b>&lt;h:inputText&gt;</b>	Représente un élément d'entrée HTML de type texte.
<b>&lt;h:inputTextarea&gt;</b>	Représente une zone de texte HTML

## Les composants de la bibliothèque JSF 2

**Exemple :**

```
<h:inputHidden value="#{bean.valeurCachéeRécupérée}" />
<h:inputSecret value="#{bean.motDePasse}" maxlength="8" />
<h:inputText value="#{bean.propriété}" />
<h:inputText value="#{bean.propriété}" size="40" />
<h:inputTextarea value="#{bean.propriété}" rows="5" cols="20" />
```

# Les composants de la bibliothèque JSF 2

## Les sorties

Balises	Description
<b>&lt;h:outputLabel&gt;</b>	Génère une balise <label> HTML.
<b>&lt;h:outputLink&gt;</b>	Génère une balise <a> HTML.
<b>&lt;h:outputText&gt;</b>	Génère tout simplement un texte littéral.
<b>&lt;h:outputFormat&gt;</b>	Génère un texte littéral paramétré.
<b>&lt;h:graphicsImage&gt;</b>	Génère une balise <img> HTML.

# Les composants de la bibliothèque JSF 2

**Exemple :**

## Les graphiques

- Il n'existe qu'un seul composant pour afficher les images `<h:graphicsImage>`.
- Ce marqueur génère un élément `<img>` HTML pour afficher une image que les utilisateurs n'auront pas le droit de manipuler

**Exemple :**

```
<h:graphicsImage library="images" name="logo.gif" />
<h:graphicsImage url="/resources/images/logo.gif" />
<h:graphicsImage value="# {bean.logo}" />
```

# Les composants de la bibliothèque JSF 2

## Les commandes - boutons et hyperliens

Balises	Description
<b>&lt;h:commandButton&gt;</b>	Représente un élément HTML pour un bouton de type submit ou reset. La méthode HTTP de la requête est de type POST.
<b>&lt;h:commandLink&gt;</b>	Représente un élément HTML pour un lien agissant comme un bouton submit. Ce composant doit être placé dans un formulaire. La méthode HTTP de la requête est de type POST.
<b>&lt;h:button&gt;</b>	Ce composant est similaire au marqueur <b>&lt;h:commandButton&gt;</b> mais c'est La méthode GET HTTP qui est proposée pour soumettre la requête.
<b>&lt;h:link&gt;</b>	Ce composant est similaire au marqueur <b>&lt;h:commandLink&gt;</b> mais c'est La méthode GET HTTP qui est proposée pour soumettre la requête.
<b>&lt;h:outputLink&gt;</b>	Génère une balise <b>&lt;a&gt;</b> HTML. A la différence de <b>&lt;h:commandLink&gt;</b> , il n'est pas nécessaire de placer cette balise dans un formulaire, il s'agit d'un simple lien statique sans interprétation particulière pas JSF, ni d'appel de méthode spécifique du bean géré.

# Les composants de la bibliothèque JSF 2

## Exemple :

```
<h:commandButton value="Valider votre commande" action="# {bean.validation}" />
<h:commandButton value="Réinitialiser le formulaire" type="reset" />
<h:commandButton image="Image.png" action="page.xhtml" />
<h:commandLink value="Naviguer vers votre nouvelle page" action="# {bean.méthode}" />
<h:commandLink value="Naviguer vers votre nouvelle page" action="page.xhtml" />
```

## Les balises de sélections

➤ Les composants de sélection permettent de choisir une ou plusieurs valeurs dans une liste.

Graphiquement, ils sont représentés par :

- des cases à cocher,
- des boutons radio,
- des listes
- ou des combos box

# Les composants de la bibliothèque JSF 2

## Les balises de sélections

Balises	Description
<b>&lt;h:selectBooleanCheckbox&gt;</b>	Génère une case à cocher représentant une valeur booléenne unique. Cette case sera initialement cochée ou décochée selon la valeur de sa propriété checked.
<b>&lt;h:selectManyListBox&gt;</b>	Génère un composant à choix multiples, dans lequel nous pouvons choisir une ou plusieurs options.
<b>&lt;h:selectManyCheckbox&gt;</b>	Génère une liste de cases à cocher.
<b>&lt;h:selectManyMenu&gt;</b>	Génère une balise <select> HTML.
<b>&lt;h:selectOneListBox&gt;</b>	Génère un composant à choix unique, dans lequel nous ne pouvons choisir qu'une seule option.
<b>&lt;h:selectOneMenu&gt;</b>	Génère un composant à choix unique, dans lequel nous ne pouvons choisir qu'une seule option. N'affiche qu'une option à la fois.
<b>&lt;h:selectOneRadio&gt;</b>	Génère une liste de boutons radio.

# Beans Managés

---

**TP : Lab\_DataTable**

# VALIDATION DES COMPOSANTS AVEC JSF 2

- Principe de la validation JSF 2
- l'interface "javax.faces.convert.Validator »
- utilisation des validator standard JSF 2: attribut "validator »
- Mise en œuvre d'un validateur personnalisé
- Validation avec implémentation JSR 303 : Hibernate validator

# VALIDATION DES COMPOSANTS AVEC JSF 2

## Principe de la validation JSF

- JSF offre de nombreuses facilités pour valider les données utilisateur.
- JSF propose en standard un mécanisme de validation des données :
  - contrôle de présence,
  - de type de données,
  - de plage de valeurs,
  - de format, ...
- La validation peut se faire de deux façons :
  - au niveau de certains composants
  - ou avec des classes spécialement développées pour des besoins spécifiques.

# VALIDATION DES COMPOSANTS AVEC JSF 2

## Principe de la validation JSF

- Ces classes s'attachent à un composant et sont réutilisables.
- Ces validations sont effectuées côté serveur.
- La validation n'est possible que sur les composants qui implémentent EditableValueHolder, C'est-à-dire :
  - les champs textes,
  - les checkbox,
  - les radio buttons,
  - les listes
  - et les combo box.

# VALIDATION DES COMPOSANTS AVEC JSF 2

## Principe de la validation JSF

Les différents types de validations :

- Méthode de validation dans un backing-bean
- Les **validators** standards de l'API JSF
  - Validation automatique implicite
  - Validation automatique explicite
- Modèle de programmation pour écrire un validator personnalisé
- Validation avec implémentation JSR 303 : Hibernate validator

# VALIDATION DES COMPOSANTS AVEC JSF 2

## Méthode de validation dans un backing-bean

### Etapes :

- Déclarer le propriété avec le type String
- Traiter la validation dans la méthode setXX(..) et/ou dans la méthode action du bean
- Si exception retourner null pour rester sur la même page
- Créer le(s) message(s) d'erreur(s)
- Afficher dans la pages

# VALIDATION DES COMPOSANTS AVEC JSF 2

## Méthode de validation dans un backing-bean

```
@ManagedBean  
public class Client {  
    private String userID = "";  
    // Get + Set  
  
    public String doBid() {  
        FacesContext context = FacesContext.getCurrentInstance();  
        if (getUserID().equals("")) {  
            context.addMessage(null,  
                new FacesMessage("UserID required"));  
        }  
  
        if (context.getMessageList().size() > 0) {  
            return(null);  
        } else {  
            doBusinessLogicForValidData();  
            return("ok");  
        }  
    }  
    private void doBusinessLogicForValidData() {}  
}
```

### EXEMPLE

```
<h:form>  
    <h:messages styleClass="error"/>  
    <table>  
        <tr>  
            <td>User ID:  
            <h:inputText value="#{client.userID}"/></td></tr>
```

# VALIDATION DES COMPOSANTS AVEC JSF 2

## Les validators standards de l'API JSF

### ➤ Validation automatique implicite

#### Etapes :

- Déclarer les propriétés du JavaBean avec les types primitives (int, long, double...)
- Préciser l'attribut « required » sur le composant JSF
- Si error de conversion et/ou absence de valeur ➔ JSF construit automatiquement les messages d'erreurs
- Afficher les messages sur la page

# VALIDATION DES COMPOSANTS AVEC JSF 2

## Les validators standards de l'API JSF

EXEMPLE

```
@ManagedBean  
public class BidBean2 {  
    private Double bidAmount;  
    private Integer bidDuration;
```

```
<tr>  
    <td>Bid Amount:  
        $<h:inputText value="#{bidBean2.bidAmount}"  
                      required="true"  
                      requiredMessage="You must enter an amount"  
                      converterMessage="Amount must be a number"  
                      id="amount"/></td>  
        <td><h:message for="amount" styleClass="error"/></td></tr>  
<tr>
```

# VALIDATION DES COMPOSANTS AVEC JSF 2

## Validation automatique explicite

- Pour effectuer un contrôle plus précis sur les données, il faut utiliser un validateur.
- Ils sont utilisables grâce aux balises :
  - <f:validateDoubleRange>, permet de tester si un nombre à virgule se trouve dans une borne.
  - <f:validateLongRange>, permet de tester si un nombre entier se trouve dans une borne.
  - <f:validateLength>, permet de tester la longueur d'une chaîne.
  - <f:validateRequired>
  - <f:validateRegex>, attribut pattern
  - <f:validateBean>, attribut validationGroups

# VALIDATION DES COMPOSANTS AVEC JSF 2

## Validation automatique explicite

Toutes ces balises possèdent les attributs « minimum » et « maximum » qui définissent les bornes dans lesquelles doit se trouver une valeur pour être valide.

### Exemple :

```
<h:inputText  
    id="carte" value="# {bb.carte}"  
    requiredMessage="Carte obligatoire"  
    validatorMessage="Numéro de carte invalide">  
    <f:validateLength minimum="16"/>  
</h:inputText>
```

# VALIDATION DES COMPOSANTS AVEC JSF 2

## Modèle de programmation pour écrire un validator personnalisé

➤ Il existe 2 manières de la faire :

- soit en utilisant une méthode(déclarée dans le Bean)
- soit en implémentant l'interface « `javax.faces.validator.Validator` ».

# VALIDATION DES COMPOSANTS AVEC JSF 2

## En utilisant une méthode

- La méthode doit être référencé dans le ManagedBean.

Cette méthode qui gère la validation doit avoir le prototype suivant :

```
public void methodName(javax.faces.context.FacesContext context,  
javax.faces.component.UIComponent component, Object value) {  
    //Gestion de l'événement (validation)  
}
```

- Si erreur de validation, la méthode doit lancer « javax.faces.validator.ValidatorException ».
- Le constructeur prend en paramètre un objet de type « **javax.faces.application.FacesMessage** » ( message qui est destiné à l'utilisateur).

La méthode peut ensuite être référencée grâce à l'attribut « **validator** ».

# VALIDATION DES COMPOSANTS AVEC JSF 2

```
<h:inputText validator="#{nomDuBean.methodName}" />
```

## En implémentant l'interface

- Il faut implémenter l'interface « javax.faces.validator.Validator ».

La seule méthode à redéfinir est :

```
public void validate (FacesContext context, UIComponent component, Object value);
```

Pour utiliser ce validateur, il faut le déclarer dans le fichier « faces-config.xml » à l'aide de la balise `<validator>`.

# VALIDATION DES COMPOSANTS AVEC JSF 2

## Exemple :

```
<validator>
    <!-- Identifiant du validateur dans les pages JSF -->
    <validator-id>emailValidator</validator-id>
    <!-- Classe du validateur -->
    <validator-class>com.formation.validation.EmailValidator</validator-class>
</validator>
```

Le validateur peut ensuite s'utiliser grâce à la balise `<f:validator>`.

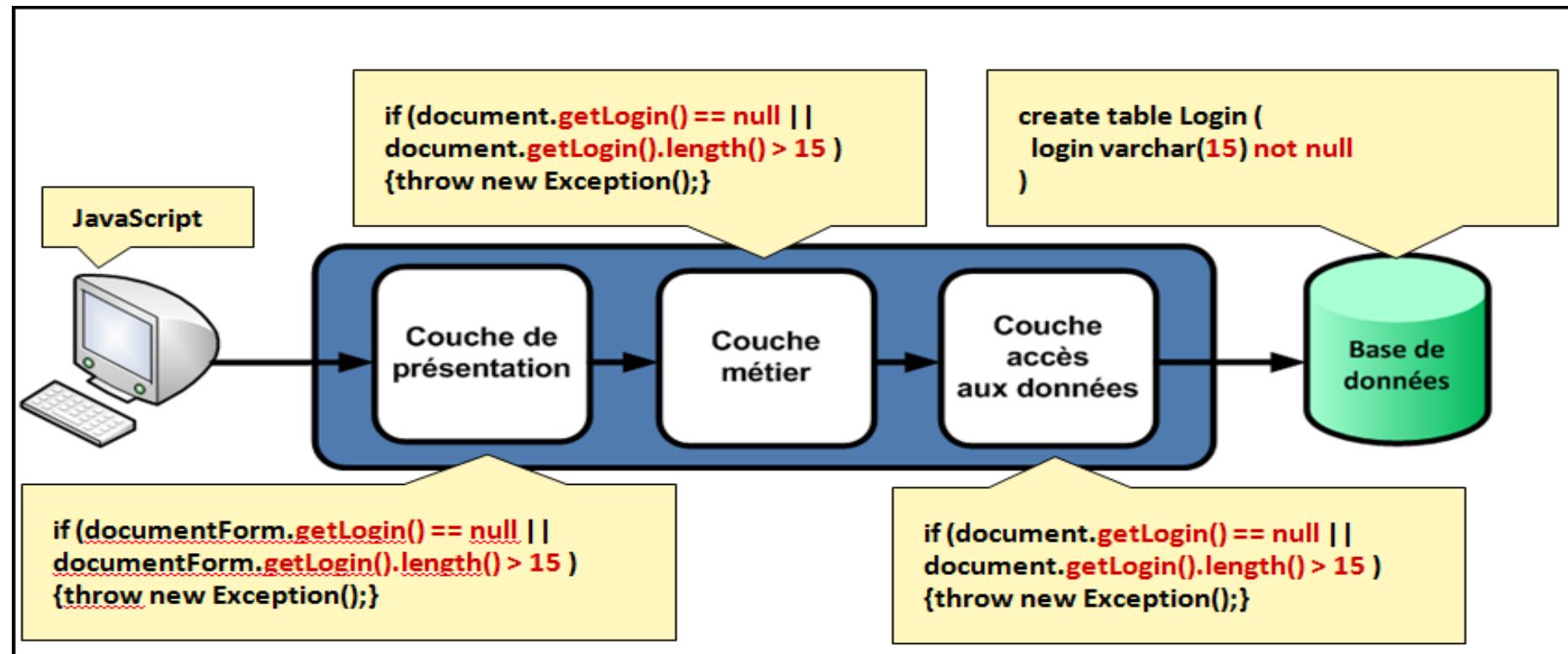
## Exemple :

```
<h:inputText>
    <f:validator validatorId="emailValidator" />
</h:inputText>
```

# VALIDATION DES COMPOSANTS AVEC JSF 2

## Validation avec implémentation JSR 303 : Hibernate validator

### Le constat actuel



### Problèmes et solution

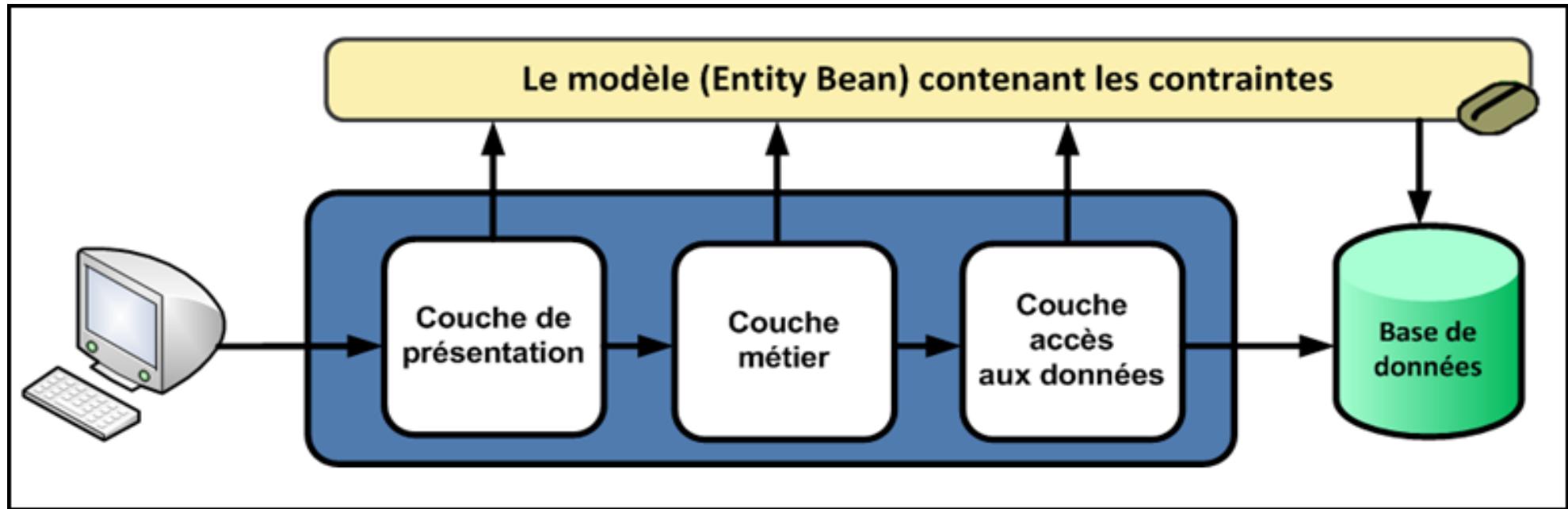
#### Problèmes

- Duplication du même code sur l'ensemble des couches
- Risque de non-consistance, Augmentation des contrôles à l'exécution

# VALIDATION DES COMPOSANTS AVEC JSF 2

## Solution

- Utiliser « Bean Validation » définie par la JSR 303



## Présentation de la JSR

- La JSR 303 définit un modèle de métadonnées et une API pour valider les Beans Java.
- Cette validation s'effectue en utilisant les annotations mais il est possible d'utiliser des fichiers XML.

*Implémentation de référence : Hibernate Validator*

# VALIDATION DES COMPOSANTS AVEC JSF 2

## EXEMPLE

```
public class Utilisateur {  
  
    @NotNull  
    private String login;  
  
    @Size(min = 8, max = 16)  
    private String mdp;  
  
    @Past  
    private Date dateDeNaissance;  
  
    //Constructeurs, getters et setters  
}
```

**TP : Lab\_Conversion**

## TP : Lab\_Validation

# Gestion des évènements

## ➤ Utilité

- Avec les événements, JSF essaie de reproduire la programmation des interfaces utilisateurs non Web (type Swing)
- Des événements sont générés par JSF et des écouteurs peuvent réagir à ces événements.

## ➤ Types d'événements

- Générés par des actions de l'utilisateur (de type « application »)
- Générés à certains moments du cycle de vie (de type « système »)

# Gestion des évènements

## Événements « application »

### ➤ Deux catégories :

- ValueChangeEvent(*ValueChange Listener*) : lorsque la valeur d'un composant a été modifiée par l'utilisateur (champ de saisie de texte, menu, liste déroulante,...)
- ActionEvent(*ActionListener*) : provoqué par un clic sur un bouton ou un lien (ce qui peut provoquer un changement de page)

# Gestion des évènements

## ActionListener

- Tous les composants qui implémentent « **ActionSource** » (**UICommand**) peuvent générer un événement.
- Rappel du cycle de vie d'une requête JSF :



- Les événements de type ActionListener sont traités pendant la phase « Invocation de l'application ».

## Gestion des évènements

- Si le composant qui déclare l'événement a l'attribut « **immediate** » à **true** alors l'événement est traité pendant la phase « Application des paramètres... ».
- Il existe 2 manières de prendre en charge l'événement côté serveur.
  - La première solution est d'associer une méthode d'un JavaBean générique.
  - La deuxième est de créer une implémentation de l'interface

### Méthode générique

Pour créer une méthode qui gère un événement, il faut qu'elle ait le prototype suivant :

```
public void nomMethode(javax.faces.event.ActionEvent e);
```

## Gestion des évènements

Pour utiliser la méthode il suffit de faire référence à elle grâce à l'attribut « `actionListener` ».

```
<h:commandButton value="Send" actionListener="#{nomDuBean.nomMethode}" />
```

# Gestion des évènements

## Interface ActionListener

- Créer une classe qui implémente « javax.faces.event.ActionListener ».

```
package com.formation;
public class GereEvent implements ActionListener {
    public void processAction(ActionEvent actionEvent) {
        //Gestion de l'événement
    }
}
```

- Cet ActionListener peut se référencer de 2 manières différentes. Il est possible d'utiliser l'attribut « actionPerformed » comme vu précédemment.
- Mais on peut aussi utiliser la balise. <f:actionListener>.

```
<h:commandLink value="avec l'interface">
    <f:actionListener type="com.formation.GereEvent" />
</h:commandLink>
```

# Gestion des évènements

## ValueChange Listener

- Les composants qui implémentent « EditableValueHolder », c'est-à-dire toutes les classes filles de UIInput :

- UISelectBoolean,
- UISelectMany,
- UISelectOne

possèdent une valeur éditable par l'utilisateur.

Rappel le cycle de vie d'une requête JSF :



## Gestion des évènements

Les événements de type **ValueChangeListener** sont traités pendant la phase « Validation des entrées utilisateurs ».

- Si le composant qui déclare l'événement a l'attribut « **immediate** » à **true**, alors l'événement est traité pendant la phase « Application des paramètres ... ».

### Méthode générique

Une méthode qui gère l'événement doit avoir le prototype suivant :

```
public void check(javax.faces.event.ValueChangeEvent event);
```

La méthode peut ensuite être référencée grâce à l'attribut « **valueChangeListener** ».

## Gestion des évènements

```
<h:InputText valueChangeListener="#{nomDuBean.nomMethode}" />
```

L'autre solution consiste à implémenter « javax.faces.event.ValueChangeListener » et de redéfinir la méthode « processValueChange (ValueChangeEvent) ».

# Gestion des évènements

**Exemple :**

```
package com.formation;

public class BeanValueChangeListener implements ValueChangeListener {
    public void processValueChange(ValueChangeEvent event) {
        //Gestion de l'événement
    }
}
```

L'attribut « type » de celle-ci permet de préciser la classe de l'écouteur qu'on utilise.

**Exemple :**

```
<h:inputText value="valeur">
    <f:valueChangeListener type="com.formation.BeanValueChangeListener" />
</h:inputText >
```

## Gestion des évènements

### Événements « système »

- JSF 2.0 introduit une nouvelle gestion d'événements qui est capable de prendre en compte les notifications globales d'une application web.
- Nous pouvons ainsi prendre en compte certaines notifications dans des phases bien particulières

# Gestion des évènements

Classe Evenements	Source de l'événement	Description
<b>PostConstructApplicationEvent</b>	<b>Application</b>	Après le démarrage de l'application web.
<b>PreDestroyApplicationEvent</b>		Avant qu'elle se termine définitivement.
<b>PostAddToViewEvent</b>	<b>UIComponent</b>	Après l'ajout du composant dans l'arbre de vue.
<b>PreRemoveFromViewEvent</b>		Avant qu'il soit enlever de l'arbre de vue.
<b>PostRestoreStateEvent</b>	<b>UIComponent</b>	Après la restauration du composant.
<b>PreValidateEvent</b>	<b>UIComponent</b>	Avant et après la validation d'un composant.
<b>PostValidateEvent</b>		
<b>PreRenderViewEvent</b>	<b>UIViewRoot</b>	Avant que le rendu de l'élément racine de la page ne soit effectué.
<b>PreRenderComponentEvent</b>	<b>UIComponent</b>	Avant que le rendu du composant ne soit effectué.
...	...	...

# Gestion des évènements

## Mise en œuvre

- Vous devez donc implémenter une méthode spécifique qui doit alors posséder un attribut de type *ComponentSystemEvent*.

```
public class MyManagedBean {  
    ...  
    public void méthodeEvénement(ComponentSystemEvent evt) {  
        ...  
    }  
}
```

L’attribut « type » de celle-ci permet de préciser la classe de l’écouteur qu’on utilise.

Exemple :

```
<h:inputText value="valeur">  
    <f:valueChangeListener type="com.formation.MyManagedBean" />  
</h:inputText >
```

**TP : Lab\_Evenement**

# Vues JSF 2 avec Facelets

- Facelets: Framework léger utilisant une syntaxe au format XML, et permet une programmation plus aisée des pages JSF et offre des fonctionnalités couplées avec JSF non supportées avec JSP.
- Facelets: est fournit en standard avec JSF 2 pour le développement des vues et permet:
  - L'utilisation de XHTML pour la création de pages Web.
  - Le support compatible des librairies de tags JavaServer Faces et JSTL.
  - Le support du langage [Expression Language](#) (EL).
  - Gestion de la navigation de façon implicite en réduisant le code et en facilitant la mise en place du routage entre les pages.
  - de développer des composants JSF personnels et ce sans avoir à écrire la moindre ligne de code en Java, Ceci se fait en réalisant des compositions de composants existants.

# Vues JSF 2 avec Facelets

## Template : industrialiser la création d'écrans (1/1)

- Les applications doivent utiliser des chartes graphiques pour uniformiser la présentation des pages Web.
- Les pages utilisent alors le template comme structure et y injectent leur contenu spécifique.
- Le but est de factoriser les éléments communs à un ensemble de pages qu'on déclarera plutôt dans une page unique, appelée template.
- Les Templates évitent la répétition de code entre les pages qui suivent un même schéma de présentation.
- Facelets permet de faire cela grâce à son système de template.

# Vues JSF 2 avec Facelets

## Template : industrialiser la création d'écrans (2/2)

Le templating offre plusieurs avantages :

- Uniformiser la structure des pages

Simplifier la mise à jour par :

- une modification dans le template se propage automatiquement dans toutes les pages qui l'utilisent.
- Gain en productivité :
  - ✓ moins de code à écrire
  - ✓ Le qu'une page ne contient que ce qui lui est propre.

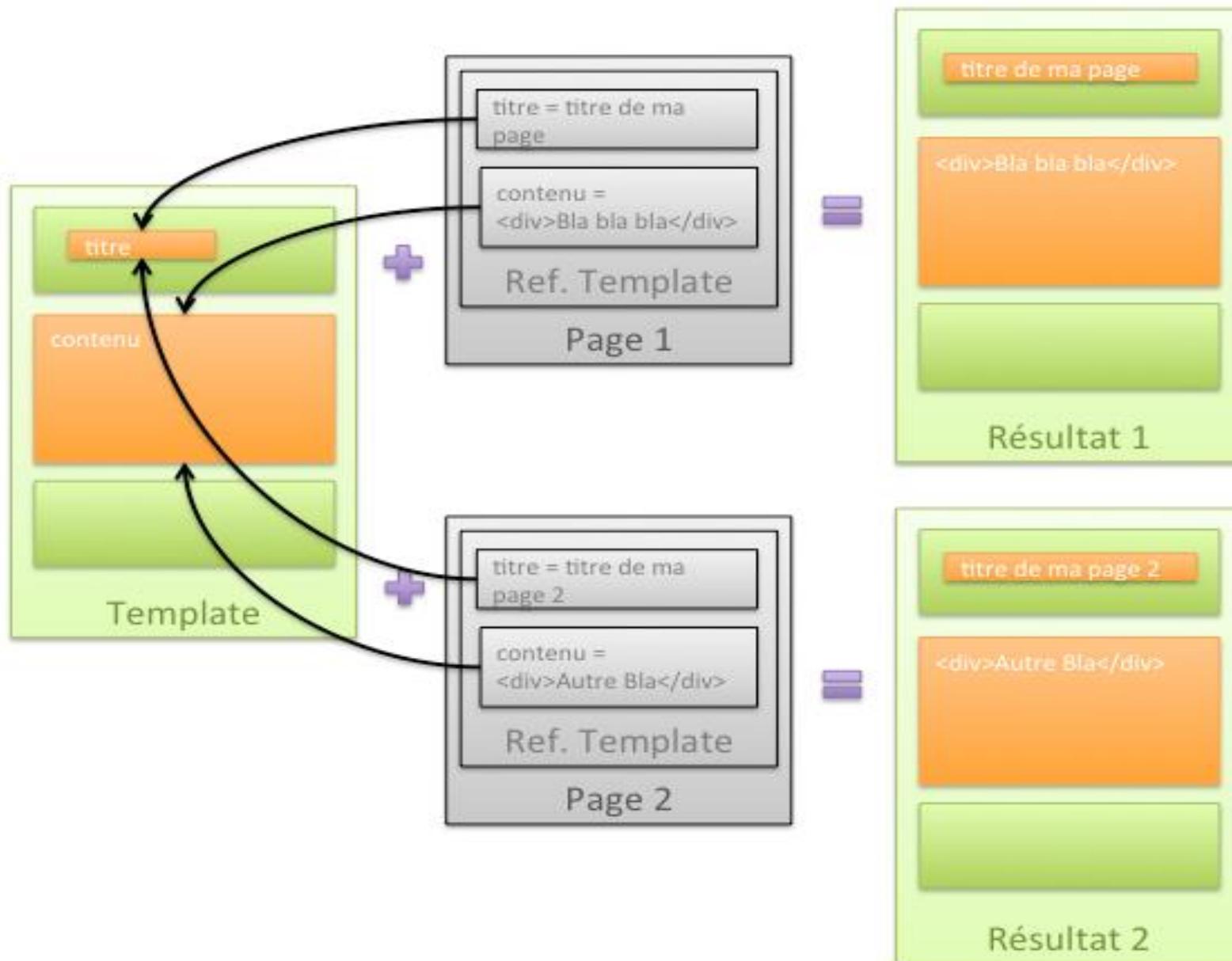
# Vues JSF 2 avec Facelets

## Fonctionnement

### Balises utilisées

- <ui:include>
- <ui:define>
- <ui:insert>
- <ui:composition>
- <ui:param>
- <ui:insert>

Les balises <ui:insert> indiquent les endroits où le code spécifique pourra être injecté.



# Vues JSF 2 avec Facelets

## Exemple : template.xhtml

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:t="http://myfaces.apache.org/tomahawk">
<head>
    <title><ui:insert name="titre" /></title>
</head>
```

```
<body>
    <p>
        <ui:insert name="contenu" />
        <ui:debug />
    </p>
</body>
</html>
```

# Vues JSF 2 avec Facelets

## Exemple : template.xhtml( suite)

Vous devez définir la bibliothèque de marqueurs nécessaires à savoir ici la bibliothèque facelets :

`xmlns:ui="http://java.sun.com/jsf/facelets"`.

La balise `<ui:debug />` intéressante, en phase de développement, lorsque vous souhaitez connaître l'arborescence des composants constituant la vue en cours.

Ce mode débug est activé par l'utilisateur en utilisant un raccourci clavier (par défaut

CTRL+SH

### Debug Output

/alea.xhtml

+ Component Tree

+ Scoped Variables

La balise `<ui:insert>` peut contenir une valeur par défaut :

```
<ui:insert name="body">
    <!-- code par défaut -->
</ui:insert>
```

M.Mbengue

# Vues JSF 2 avec Facelets

## Exemple : ma\_page.xhtml( suite)

<ui:composition> et <ui:define>

La balise <ui:composition> indique le template qui va être utilisé.

Les balises <ui:define> permettent de définir les contenus spécifiques à la page (élément identifié via l'attribut name).

Exemple de page utilisant le template : ma\_page.xhtml

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html">

    <ui:composition template="/template.xhtml">
        <ui:define name="titre">Titre de ma page</ui:define>

        <ui:define name="contenu">
            Texte, formulaires etc.
        </ui:define>
    </ui:composition>
</html>
```

# Vues JSF 2 avec Facelets

## Résultat obtenu :

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:t="http://myfaces.apache.org/tomahawk">
<head>
    <title>Titre de ma page</title>
</head>
<body>
    <p>
        Texte, formulaires etc.
    </p>
</body>
</html>
```

# Beans Managés

---

TP : Lab\_Template

## JSF 2 et AJAX

- ❶ AJAX (Asynchronous JavaScript And XML) est une technique de développement Internet basée sur le langage JavaScript.
- ❶ Il permet d'effectuer des requêtes HTTP sans recharger les pages, appelées de façon asynchrone, afin d'effectuer des rafraîchissements partiels des pages.
- ❶ Ajax repose sur l'utilisation des technologies XHTML et CSS ainsi que sur DOM (Document Object Model) pour la représentation des objets et enfin sur l'objet JavaScript XMLHttpRequest pour la réalisation des requêtes en arrière-plan.
- ❶ Les services sont appelés en arrière plan, que ce soit l'affichage ou les actions (consultation, modification, suppression) et les résultats sont retournés sous la forme d'une variable.
- ❶ Il existe une pléthore de librairies Ajax comme jQuery, Prototype, GWT, ExtJS ou autres, plus ou moins simples à utiliser  
*mais pas adaptées en standard au Framework JSF*

## JSF 2 et AJAX

- Depuis JSF 2, le support Ajax est natif.
- Il n'est plus nécessaire de développer du code JavaScript pour manipuler l'objet XMLHttpRequest, mais d'inclure une librairie spécifique proposée par le Framework
- JSF 2 propose un mécanisme standard pour l'ajout de fonctionnalités Ajax aux applications Web.

## JSF 2 et AJAX

- Avec JSF 2.x, nous devons inclure dans nos pages XHTML l'unique fichier Ajax jsf.js présente dans la librairie javax.faces.

```
<h:outputScript name="ajax.js" library="javax.faces" target="head"/>
```

- Cette librairie permet d'initialiser les dialogues en incluant les fichiers nécessaires au fonctionnement.
- La principale fonction utilisée avec JSF dans nos pages est `jsf.ajax.request(ELEMENT, EVENT)`, responsable de l'envoi des informations au serveur et de l'affichage de la réponse

## JSF 2 et AJAX

➤ La méthode `jsf.ajax.request(ELEMENT, EVENT, OPTIONS)` possède trois paramètres :

- **ELEMENT** : représente un composant JSF XHTML lié à l'événement.

Le mot-clé `this` est souvent utilisé pour faire référence à l'élément courant.

- **EVENT** : représente les événements JavaScript pouvant survenir sur le composant.

Le mot-clé `event` est souvent utilisé pour faire référence aux événements par défaut (événement `onclick`).

- **OPTIONS** : représente une liste optionnelle d'options sous la forme d'un tableau contenant une paire nom / valeur séparés par des espaces.

# JSF 2 et AJAX

```
</h:inputText>
```

```
<h:commandButton value="#{b.valider}" styleClass="boutonvalider"
onclick="jsf.ajax.request(this, event, {execute: 'identifiant motdepasse', render: 'reponse'});
return false;"/>
<h:outputText id="reponse" value="#{authentificationBean.reponseAuthentification}" />
```

```
@ManagedBean
public class AuthentificationBean {
    private String identifiant;
    private String motdepasse;
    // injecter le Web Bean
    @ManagedProperty(value = "#{serviceAuthentification}")
    private ServiceAuthentificationBean serviceAuthentificationBean;

    // verification de l'authentification
    public String getReponseAuthentification() {
        if (identifiant != null && motdepasse != null) {
            // authentification OK
            if (serviceAuthentificationBean.verifierAuthentification(
                identifiant, motdepasse))
                {
                    return "Authentification réalisée avec succès";
                }
        }
        return "Erreur lors de l'authentification";
    }
}
```

## JSF 2 et AJAX

Les attributs **execute** représentent une liste de composants à envoyer au serveur.

les attributs **render** représentent une liste de composants à recevoir du serveur en association avec un composant de la page.

## JSF 2 et AJAX

- Lorsque le client clique sur le bouton, le cycle de vie de la page passe par les six phases d'une requête courante.
- La réponse Ajax est envoyée au client pendant la dernière phase du cycle de vie et retourne un contenu au format XHTML, XML ou JSON.
- Le contenu de la page ne change pas, seul le contenu de la balise <h:outputText/> change par l'intermédiaire du getter authentificationBean.reponseAuthentification().
- Les valeurs des composants sont liées avec la requête et la réponse Ajax par l'intermédiaire des paramètres execute et render.
- La notation des ID des composants doit donc être unique et permet de faire la correspondance avec les composants DOM qui interagissent de façon asynchrone avec le serveur.

## Utilisation simplifiée d'Ajax

- Il existe une seconde méthode standard pour utiliser Ajax avec JSF.
- La balise <f:ajax/> propose un ensemble de services évolués parfaitement adaptés aux traitements asynchrones.
  - La balise <f:ajax/> est associée à un composant d'action, comme un bouton ou un lien, et utilise un ID de composant pour afficher le résultat.

```
<h:commandButton action="">
    <f:ajax render="idSortie"/>
</h:commandButton>
<h:outputText id="idSortie"/>
```

- Lorsque l'utilisateur clique sur le bouton, l'action précisée est déclenchée en tâche de fond.
- Le composant précisé avec l'attribut **render** est alors remplacé dans le DOM par la nouvelle valeur.

## balise <f:ajax>

```
<f:ajax render="id1 id2" execute="id3 id4" event="evenementdeclencheur" onevent="fonctionJavaScript"/>
```

possède les quatre attributs suivants :

**render** : les éléments à afficher ou à actualiser dans la page, précisés par leur ID. - Les ID des composants sont séparés par des espaces.

➤ Il existe quatre valeurs génériques : **@this**, **@form**, **@none** et **@all**.

**@this**: permet d'actualiser le composant courant (encapsulant la balise) et correspond à la valeur par défaut.

**@form**: permet d'envoyer tout le formulaire.

**@none**: précise qu'aucune valeur n'est envoyée mais permet juste un déclenchement et l'actualisation des composants.

## JSF 2 et AJAX

**@all:** permet d'actualiser tous les composants de la page.

**execute** : les éléments à envoyer au serveur par l'intermédiaire de leur setter.

**event** : l'évènement DOM à l'écoute de l'évènement déclencheur (keyup, blur, mousedown...).

**onevent** : une fonction JavaScript à lancer en association avec le traitement Ajax.

- Il n'est pas nécessaire de préciser le préfixe **on** sur chaque évènement : **mouseover, mousedown, keyup, blur** ou autres.
- L'annotation **@form** est très souvent utilisée pour envoyer la totalité des composants du formulaire.

## balise <f:ajax>

- L'attribut **onevent** est également très intéressant
- permet de déclencher une méthode JavaScript en association avec le service Ajax.
  - Le nom de la méthode JavaScript n'a pas d'importance, mais celle-ci reçoit en paramètre un objet data contenant le statut et les propriétés de la requête Ajax.
  - L'attribut **status** contient les valeurs suivantes :
    - **begin** (avant l'exécution),
    - **complete** (pendant l'exécution)
    - **success** (après l'exécution).
  - L'attribut **source** contient les valeurs du DOM utilisé dans la requête Ajax.

# Beans Managés

---

**TP : Lab\_Ajax**

## Composants Additionnels

- Les deux jeux de composants standards de JSF s'avèrent trop limités et insuffisants pour le développement d'applications d'entreprise.
- Il est possible dès lors d'utiliser des jeux de composants additionnels qui offrent de nouveaux composants plus riches.

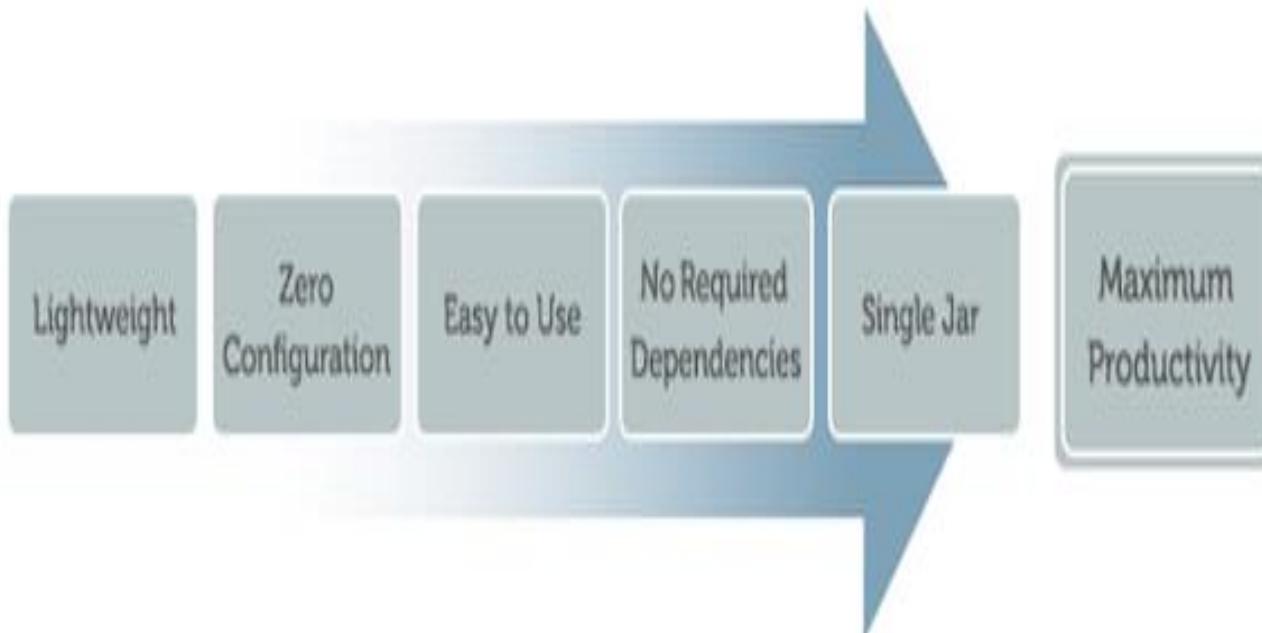
On peut citer par exemple:

- JBoss ***RichFaces*** et ***Ajax4JSF***, composants open-source supportant Ajax
- **Primefaces** , composants open-source supportant Ajax(avec version mobile)
- **ICEfaces**, un jeu de composants open-source supportant Ajax
- Apache ***Tomahawk***, composants(non ajax) très riche et open-source
- **RCFaces**, un jeu de composants très riche AJAX et open-source

# Composants Additionnels

## Configuration composants Primefaces

Flexible et permet un développement rapide des IHMs



## Rapid Development

- ✓ Simple
- ✓ Effective
- ✓ Powerful

# Composants Additionnels

---

## Installation PrimeFaces

Pas de configuration spécifique

Téléchargez le Fichier jar sur : <http://www.primefaces.org/downloads.html>

## Utilisation

Importez l'espace de nom de Primefaces dans votre page vue.xhtml

# Composants Additionnels

## Installation PrimeFaces

Importez l'espace de nom de primefaces.

```
<html ....  
xmlns:p=http://primefaces.org/ui ">
```

### Exemple

```
<h:body>  
<h:form>  
  <h:panelGrid columns="2" styleClass="borderAll">  
    ...  
    <h:outputText value="date" />  
    <p:calendar pattern="dd/MM/yyyy" />  
  </h:panelGrid>  
  <h:commandButton value="identification" styleClass="button" />  
</h:form>  
</h:body>
```

## Composants Additionnels

Vous pouvez personnaliser le thème utilisé dans PrimeFaces en ajoutant le code suivant dans web.xml

```
<context-param>
<param-name>primefaces.THEME</param-name>
<param-value>NOM_THEME</param-value>
</context-param>
```

Avec **NOM\_THEME** nom du thème à utiliser.

La liste les thèmes disponibles se trouve à l'adresse  
<http://www.primefaces.org/themes.html>.

Ci-dessous quelques exemples des thèmes de PrimeFaces:

# Composants Additionnels



# Composants Additionnels

## Configuration composants Richfaces

RichFaces est LA librairie de composants pour qui développe des applications JSF.

**Attention :** Pour bien fonctionner une page (ou template) doit respecter la structure suivant :

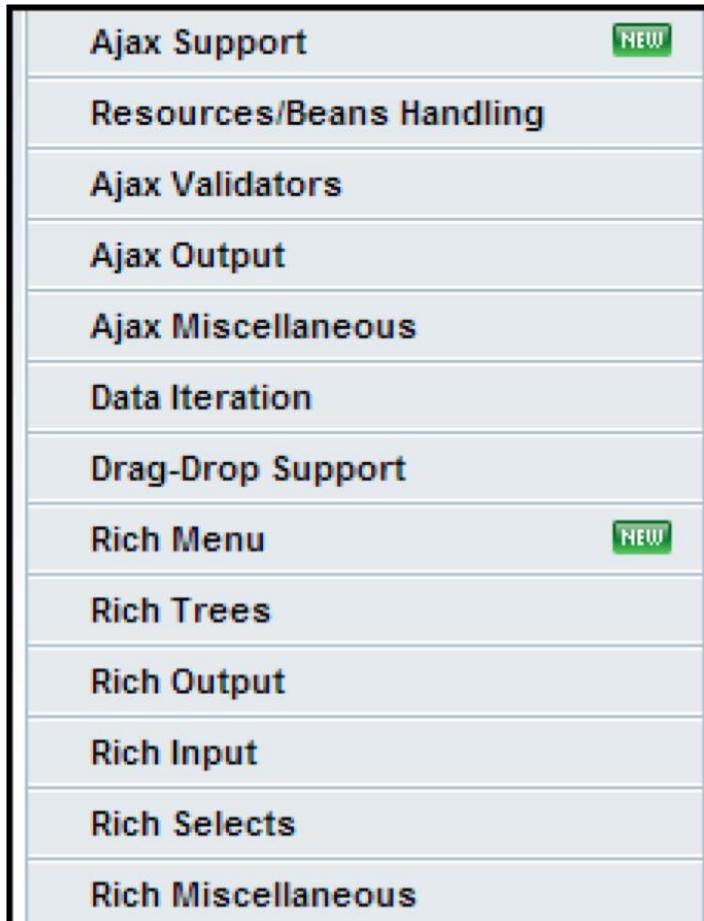
```
<f:view>
    <h:head> ... </h:head>
    <h:body> .... </h:body>
</f:view>
```

# Composants Additionnels

## Configuration composants Richfaces

### Demo

<http://livedemo.exadel.com/richfaces-demo/>



## Composants Additionnels

---

### Installation RichFaces

Pas de configuration spécifique

Fichier jar sur : <http://www.primefaces.org/downloads.html>

### Utilisation

Dans la facelet nous importons simplement l'espace de nom de Primefaces et nous pouvons utiliser ses composants de la même façon que les composants standard

# Composants Additionnels

## Installation RichFaces

```
<html xmlns="...  
    xmlns:a4j="http://richfaces.org/a4j"  
    xmlns:rich="http://richfaces.org/rich">
```

### Exemple

The image shows the RichFaces component palette on the left and a web application screenshot on the right.

**Component Palette (Left):**

- Ajax Output
- Ajax Miscellaneous
- Data Iteration
  - Column
  - Column Group
  - Columns
  - Data Definition List
  - Data Filter Slider
  - Data Grid**
  - Data List
  - Data Ordered List
  - Data Scroller
  - Data Table
  - Extended Data Table
  - Repeat
  - Scrollable Data Table
  - Table Filtering
  - Table Sorting
- Drag-Drop Support
- Rich Menu
- Rich Trees

**Web Application Screenshot (Right):**

The application title is "DataGrid example". The page header says "Car Store". The main content displays a DataGrid with 12 rows of car information. Each row contains a "Chevrolet Corvette" or "Chevrolet Malibu" card with the following details:

Car Model	Price	Mileage	VIN	Stock
Chevrolet Corvette	40813	77799.0	UVOCMHZUQCYREIP	AECUOC
Chevrolet Corvette	30764	72280.0	AJPSAHGXMSOYCC	PKAOVO
Chevrolet Corvette	29816	48361.0	JHEWAIHQHQCWEMG	ZQAMXJ
Chevrolet Corvette	17455	23412.0	RHWCNDSOSSPVJ	DYQJCP
Chevrolet Corvette	81099	52535.0	GSLEFATURUOVOSJ	ALFQDL
Chevrolet Malibu	17186	37605.0	KNFQUSAGSFJORGGO	AUJWISI
Chevrolet Malibu	21294	48199.0	JVZSOLIQZFKXWQ	HDOELSA
Chevrolet Malibu	16637	62078.0	HISVFZPYNA04VBQ	UTTHXKZ
Chevrolet Malibu	33758	62078.0	VWKOYOFQXIZITQ8	LYTROWP
Chevrolet Malibu	21294	48199.0	JVZSOLIQZFKXWQ	HDOELSA

Pagination at the bottom shows pages 1 through 10.

# Beans Managés

---

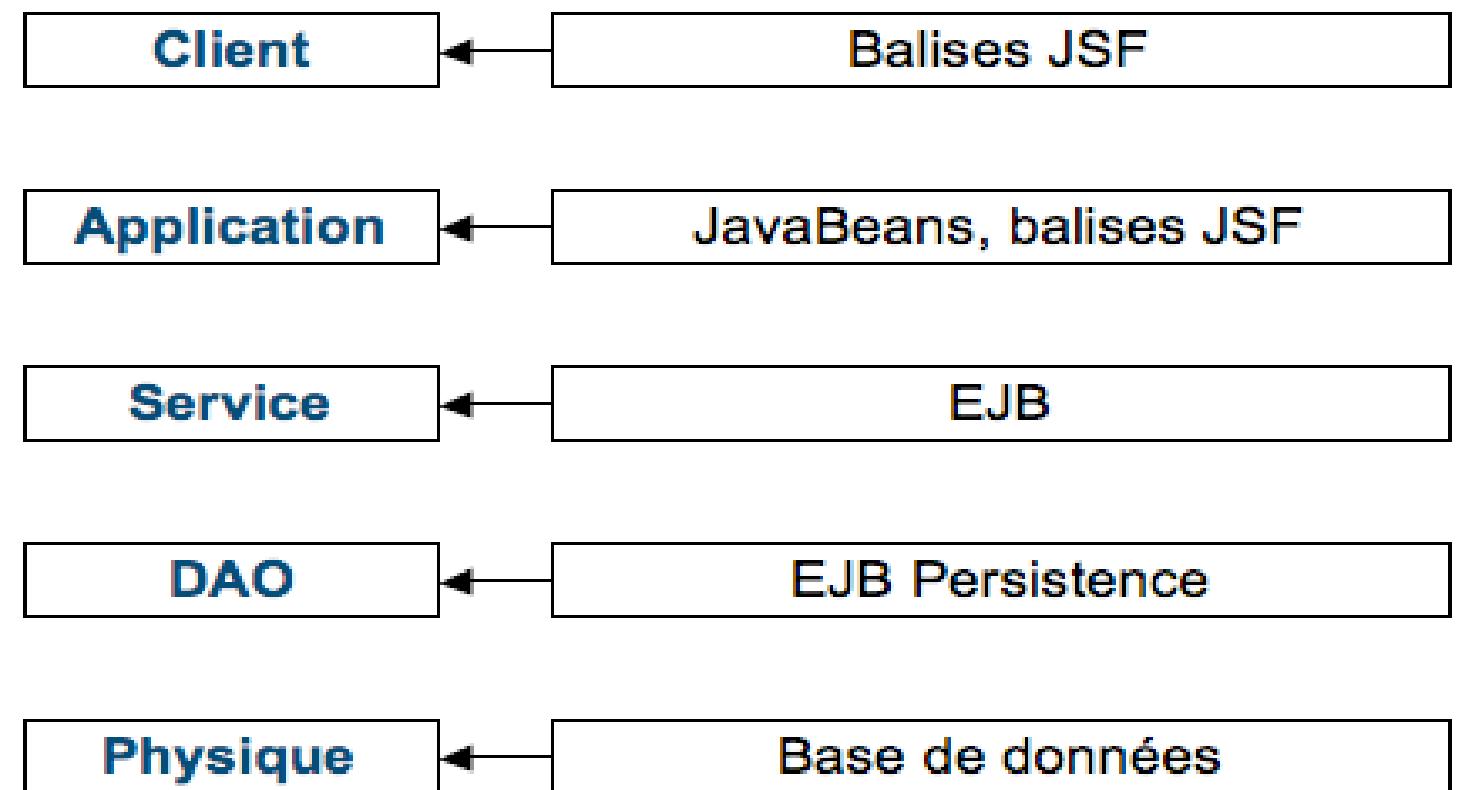
TP : Lab\_PrimeFaces

# Intégrer JSF avec les autres technologies

## Intégration avec EJB3

- Il est tout à fait possible d'interagir avec des EJBs ou tout autre type de couche métier.
- Ce sont les JavaBeans qui vont vous permettre d'accéder à une couche Service.

Voici un exemple d'architecture JSF + EJB :



# Intégrer JSF avec les autres technologies

## Injection d'EJB

- L'injection d'EJB permet de récupérer une instance de l'interface Local ou Remote.
- Elle s'utilise grâce à l'annotation « `@EJB` ».

Exemple :

```
@EJB  
private Store store;
```

- Il est aussi possible d'utiliser l'injection de ressource pour récupérer une instance sur le contexte JNDI courant et ensuite y rechercher l'EJB.

Exemple :

```
@Resource  
private SessionContext ctx;  
  
Store store = (Store) ctx.lookup ("Store");
```

# Intégrer JSF avec les autres technologies

## Injection de WebService

- Il est possible de récupérer un WebService qui hérite de « javax.xml.ws.Service ».
- L'injection s'utilise grâce à l'annotation « @WebServiceRef ».

### Exemple :

```
@WebServiceRef  
public StockQuoteService stockQuoteService;
```

## Injection d'unité de persistance

- Il est possible de récupérer une instance d'un EntityManager.
- Comme en EJB3, l'injection se fait grâce à l'annotation « @PersistenceContext ».
- Le champ « unitName » permet de renseigner le nom de l'unité de persistance à utiliser. Pour être valide, ce nom doit être déclaré dans le fichier « persistence.xml ».

### Exemple :

```
@PersistenceContext (unitName="InventoryManagement")  
EntityManager em;
```

# Intégrer JSF avec les autres technologies

## Intégration avec Spring: Configuration du web.xml

```
<!--  
    <context-param>  
        <param-name>contextConfigLocation</param-name>  
        <param-value>/WEB-INF/xxx/springConfig.xml</param-value>  
    </context-param>  
-->  
  
<!--  
    <listener>  
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>  
    </listener>  
-->
```

- **org.springframework.web.context.ContextLoaderListener** qui démarre le context Spring
  - ➔ charge en mémoire le fichier de configuration spring
- Si **springConfig.xml** directement ou sous répertoire de WEB-INF alors param-value de la forme

# Intégrer JSF avec les autres technologies

## Intégration avec Spring: Configuration du web.xml

```
<param-value>/WEB-INF/nomSousRépertoire /springConfig.xml</param-value>
```

Si **springConfig.xml** est dans le classpath **alors** param-value de la forme

```
<param-value>/WEB-INF/classes /springConfig.xml</param-value>
```

Si **springConfig.xml** une librairie (jar de WEB-INF/lib) **alors** param-value de la forme

```
<param-value>classpath*: /springConfig.xml</param-value>
```

## Configuration de faces-config.xml

- Indique à Jsf que les beans ne sont plus managés par lui mais par Spring.

```
<application>
    <variable-resolver>org.springframework.web.jsf.DelegatingVariableResolver</variable-resolver>
</application>
```

## Injection de bean

```
<managed-bean>
    <managed-bean-name>monBean</managed-bean-name>
    <managed-bean-class>com.formation.MonBean</managed-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
    <managed-property>
        <property-name>beanSpring</property-name>
        <value># {springBean}</value>
    </managed-property>
</managed-bean>
```

TP : JSF\_JDBC

# Beans Managés

---

**TP : JSF\_JPA**

# Beans Managés

---

**TP : JSF\_SPRING**

# Création de composants visuels personnalisés (UI)

La création d'un composant avec facelets se déroule en trois phases comme suit :

- Création de la page xhtml qui définit le composant.
- Création d'un descripteur de bibliothèque de composants (fichier *.taglib.xml*).
- Déclaration du fichier *taglib.xml* dans *web.xml*.

## Créer la page xhtml du composant (*myComponent.xhtml*)

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:h="http://java.sun.com/jsf/html">

    <ui:composition>
        <h:outputText value="# {titre}" styleClass="titre" />
        <h:inputText value="# {valeur}" styleClass="zoneTexte" />
    </ui:composition>
</html>
```

## Création de composants visuels personnalisés (UI)

Les paramètres `# {titre}` et `# {valeur}` seront passés en paramètre lors de l'appel de ce composant.

Les fichiers de définition des composants vont généralement dans un sous dossier de WEB-INF.

**Exemple : *WEB-INF/composants***

➤ **Création du taglib.xml(myTag.taglib.xml)**

- C'est un fichier *xml* qui décrit une bibliothèque de composants.

Il définit :

- son namespace
- la liste des composants qu'il contient

# Création de composants visuels personnalisés (UI)

**Exemple :**

```
<?xml version="1.0"?>
<!DOCTYPE facelet-taglib PUBLIC
"-//Sun Microsystems, Inc.//DTD Facelet Taglib 1.0//EN"
"facelet-taglib_1_0.dtd">
<facelet-taglib>
    <namespace>http://www.formation.com/jsfEtFacelets/monJeu</namespace>
    <tag>
        <tag-name>myComp</tag-name>
        <source>composants/myComponent.xhtml</source>
    </tag>
</facelet-taglib>
```

Le *namespace* est indiqué dans l'élément `<namespace>`, et pour chaque composant dans le jeu de composants, on ajoute un élément `<tag>` avec le nom du composant et le fichier *xhtml* qui contient sa définition.

Le fichier *taglib.xml* doit être dans ou dans un sous dossier de **WEB-INF**.

# Création de composants visuels personnalisés (UI)

## Déclaration dans web.xml

Il s'agit ici d'indiquer à facelets :  
l'emplacement du ou des fichiers *taglib.xml*  
et le chemin vers le *taglib.xml* comme valeur.

```
<context-param>
    <param-name>facelets.LIBRARIES</param-name>
    <param-value>/WEB-INF/myTag.taglib.xml</param-value>
</context-param>
```

## Utilisation du composant

Pour ce faire, on doit importer le jeu de composants en question via son namespace en lui associant un préfixe et utiliser les composants qu'il contient.

# Création de composants visuels personnalisés (UI)

## EXEMPLE

```
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:mj="http://www.formation.com/jsfEtFacelets/monJeu">
<head>
    <title>Ma première page XHTML avec facelets</title>
</head>
<body>
    <f:view>
        <mj:myComp titre="Hello World" valeur="#{unManagedBean.unChamp}" />
    </f:view>
</body>
</html>
```