

Programmation d'applications d'entreprise

Chapitre 4 – Java Server Faces: JSF
UFR Sciences et Techniques - Antenne de Blois
Sameh.kchaou@univ-tours.fr
Supports de cours adaptés de Anthony Bocci

JSF

JSF = **J**ava **S**erver **F**aces

Framework MVC de présentation des applications Web en Java :

- Librairie de composants graphiques
- Fonctionnalités se basant sur ces composants

Implémentations

- Oracle – Mojarra (<https://javaserverfaces.java.net>)
- Apache – MyFaces (<https://myfaces.apache.org>)
- Oracle – ADF Faces
(<https://www.oracle.com/technology/products/adf/adffaces/index.html>)
- JBoss RichFaces (<https://labs.jboss.com/portal/jbossrichfaces>)

Framework

L'objectif principal d'un framework MVC est de simplifier le flux de contrôle de l'application.

- Clarté et meilleure organisation (séparation présentation - traitement - données)

Développement rapide (se concentrer sur le côté métier).

Framework orienté Action Controller

- Se base sur la notion d'actions
- Raisonnement sur le couple (requête, réponse)
- Le développeur écrit des classes qui représentent des actions effectuées par l'utilisateur (ex : détails commande)
- Écrire plus de code
- Un meilleur contrôle sur le processus
- Exemple de framework : Struts

Framework orienté View Controller

- Concept novateur : suit un modèle orienté composants
- Abstraction des concepts de traitement de l'application sous forme de collection de composants.
- Pas besoin d'écrire autant de code
- Moins de possibilités de contrôle sur le processus
- Exemple de framework : JSF

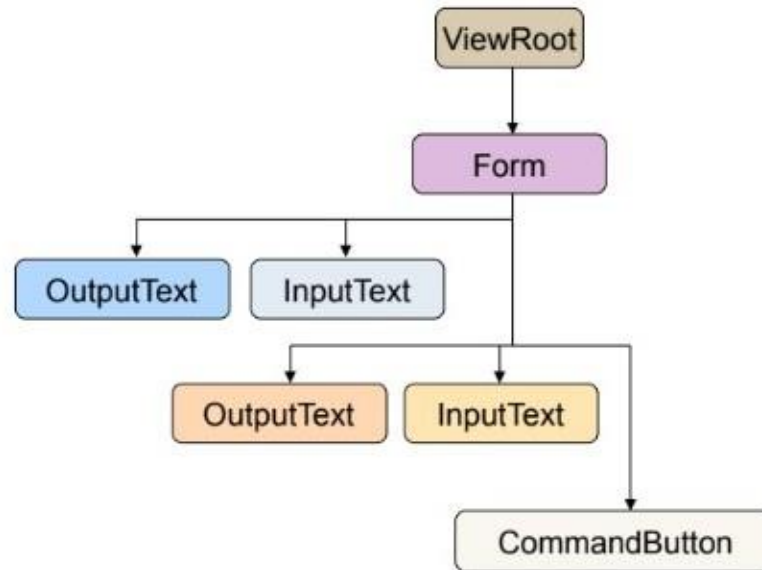
Framework

Faut-il toujours utiliser un framework ?

- Pour des applications simples ou des prototypes, l'utilisation d'un framework peut alourdir le développement.
- Dans des projets complexes avec plusieurs fonctionnalités et des besoins en évolutivité, un framework est souvent utile pour assurer la clarté, la maintenabilité et la cohérence du code.

Nouvelle vision

Représentation de la page JSF sous forme d'arbre de composants, et accessible via le contexte de l'application



JSF

- Un ensemble riche de classes pour spécifier l'état et le comportement des composants de l'interface utilisateur.
- Un **modèle de rendu** qui définit comment rendre les composants de différentes manières.
- Un **modèle de conversion** qui définit comment enregistrer des convertisseurs de données sur un composant.
- Un **modèle d'événement et d'écouteur** qui définit comment gérer les événements des composants.
- Un **modèle de validation** qui définit comment enregistrer les validateurs sur un composant .

Composants de l'architecture JSF

Contrôleur (Faces Servlet) :

- Servlet principale « **front-controller** » de l'application qui sert de contrôleur.
 - Toutes les requêtes de l'utilisateur passent systématiquement par elle, qui les examine et appelle les différentes actions correspondantes.
- Java beans particuliers (managed beans)

Vue pages web (JSP, HTML) :

- JSF peut générer plusieurs formats de réponse
- JSF utilise les **Facelets**
- Facelet est formée d'une arborescence de composants UI

Modèle JavaBean :

- Classes Java spécialisées qui synchronisent les valeurs avec les composants UI
- Accèdent à la **logique métier** et gèrent la **navigation** entre les pages.

Faces-config.xml :

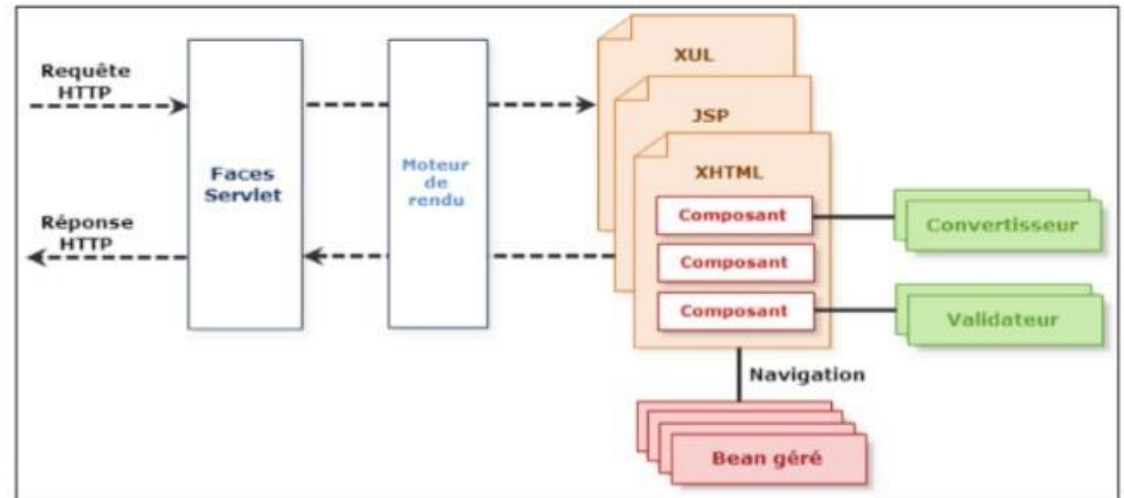
- Fichier de configuration de l'application définissant les **règles de navigation** et les différents managed beans utilisés

Composants de l'architecture JSF

Pas de servlet spécifique à chaque page

Moteur de rendu (Renderer) : décode la requête de l'utilisateur pour initialiser les valeurs du composant et encode la réponse pour créer une représentation du composant pour le client.

Convertisseurs et validateurs : permettent de valider les champs de saisie textuelle et de les convertir vers d'autres types d'objets



Les vues JSF

Simple pages XHTML nommées **Facelets**.

Facelet est un fichier XML

Trois extensions : .jsf, .xhtml ou .faces.

Ensemble de balises JSF constituant la page :

- Composants graphiques : les boutons, les champs de texte, les cases à cocher, les menus déroulants, etc.
- Composants de conversion
- Composants de validation

Expressions EL

Une Facelet contient des expressions EL

`{personneBean.login}`

Une page JSP contient des expressions EL

`{personneBean.login}`

EL permet de relier les composants de la vue aux beans.

- des champs HTML d'interaction utilisateur (saisie de données, listes, etc.) à des propriétés des beans (`<h:inputText value="{personneBean.login}" />`)
- des boutons à leurs méthodes d'action (`<h:commandButton value="Se connecter" action="{personneBean.connecter}" />`)

Expression EL

➤ *Fonctionnement de l'EL dans JSF (Résumé) :*

- Une EL permet d'accéder simplement aux Beans des différents scopes de l'application (page, request, session et application)
- Forme d'une EL dans JSF **`#{expression}`**
- `#{MyBean.attribut}` indique à JSF de chercher un objet qui porte le nom de **MyBean** dans son contexte puis invoque
 - **`getAttribut()`** pour récupérer la valeur de l'attribut
 - **`setAttribut()`** pour mettre à jour la valeur de l'attribut

Les vues JSF

Le *framework* JSF intègre nativement trois bibliothèques standard : **Core**, **HTML** et **UI**.

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

h : désigne la bibliothèque HTML (<h:inputText> , <h:commandButton>)

f : désigne la bibliothèque Core (<f:facet>)

ui : désigne la bibliothèque de templating et de composition

Composants graphiques

```
<h:inputText id="username" value="#{user.name}" />

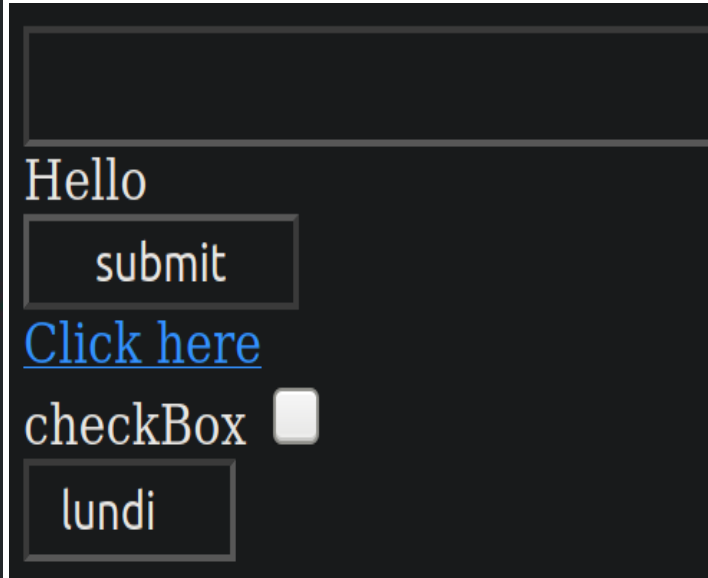
<h:outputText value="hello"></h:outputText>

<h:commandButton id="submitbutton" value="submit"
  action="#{personMB.login()}" />

<h:outputLink value="https://x.org">Click here</h:outputLink>

<h:selectBooleanCheckbox value="#{bean.value}"> checkBox
</h:selectBooleanCheckbox>

<h:selectOneMenu value="#{bean.day}">
  <f:selectItems value="#{bean.choixPossibles}" />
</h:selectOneMenu>
```



A visual rendering of the JSF code shown on the left. It features a dark-themed user interface. At the top is a wide, empty text input field. Below it, the word "Hello" is displayed. Underneath "Hello" is a rectangular button with the text "submit". Below the button is a blue, underlined link that says "Click here". Below the link is the text "checkBox" followed by an unchecked checkbox. At the bottom is another rectangular button with the text "lundi".

Exemple de page JSF

```
><html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://xmlns.jcp.org/jsf/core">
<head>
  <title>Exemple de Page JSF</title>
</head>
<body>

  <h:form>
    <!-- Champ de texte pour l'utilisateur -->
    <h:inputText id="username" value="#{user.name}" />

    <!-- Affichage de texte statique -->
    <h:outputText value="hello" />

    <!-- Bouton de soumission -->
    <h:commandButton id="submitbutton" value="submit" action="#{personMB.login()}" />

    <!-- Lien cliquable -->
    <h:outputLink value="https://x.org">Click here</h:outputLink>

    <!-- Case à cocher -->
    <h:selectBooleanCheckbox value="#{bean.value}"> checkBox</h:selectBooleanCheckbox>

    <!-- Menu déroulant avec des options dynamiques -->
    <h:selectOneMenu value="#{bean.day}">
      <f:selectItems value="#{bean.choixPossibles}" />
    </h:selectOneMenu>
  </h:form>

</body>
</html>
```


Managed bean

C'est un Java bean géré par JSF

Permet de faire le lien entre l'IHM et le code métier de l'application.

Doit contenir des accesseurs (getters) et des mutateurs (setters) pour les champs de l'IHM.

Définition dans la classe Java par les **annotations**.

- `@ManagedBean`
- `@SessionScoped`

Managed bean

❖ *Différents types de portée pour les Managed Beans :*

- **Request Scope (@RequestScoped)** : persiste durant une seule requête HTTP dans une application web.
- **Session Scope (@SessionScoped)** : persiste pour plusieurs requêtes HTTP d'une même session dans une application web.
- **Application Scope (@ApplicationScoped)** : persiste via toutes les interactions utilisateurs avec l'application web.

Managed bean : exemple

Par défaut, utilisable sous le nom « loginBean » dans une vue.
Nom de la classe avec la première lettre en minuscule.

```
1 import javax.inject.Named;
2 import javax.enterprise.context.SessionScoped
3
4 @Named
5 @SessionScoped
6 public class LoginBean
7 {
8     private String identifiant;
9
10    public String getIdentifiant()
11    {
12        return identifiant;
13    }
14
15    public void setIdentifiant(String identifiant)
16    {
17        this.identifiant = identifiant;
18    }
19 }
```

Managed bean : exemple

Exemple d'utilisation dans une page JSF :

```
<h:form>  
<h:inputText value="#{loginBean.identifiant}"  
  h:commandButton value="Se connecter"  
  action="#{loginBean.login}" />  
</h:form>
```

appeler une méthode login dans le bean LoginBean
lorsque l'utilisateur soumet le formulaire

```
3  
4 @Named  
5 @SessionScoped  
6 public class LoginBean  
7 {
```

Managed bean : CDI(Contexts and Dependency Injection)

À partir de JSF 2.3, il est recommandé d'utiliser un injecteur de dépendances externe au framework. Exemple : [Weld 3.1.x](#).

Ajouter **weld-jsf.jar** et **weld-servlet-shaded.jar** dans **WEB-INF/lib/**.

Pour activer l'injection de dépendances, créer un fichier **WEB-INF/beans.xml**.

Activation de CDI : beans.xml

WEB-INF/beans.xml

```
<beans xmlns="http://xmlns.jcp.org/xml/ns/javaee"  
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee  
    http://xmlns.jcp.org/xml/ns/javaee/beans\_2\_0.xsd"  
  bean-discovery-mode="all">  
</beans>
```

Exemple :

- ❖ **L'objectif:** Créer une page JSF qui contient un formulaire permettant à l'utilisateur de saisir deux nombres (numberX et numberY). Lorsqu'il clique sur le bouton "Add", une méthode nommée add() du bean numberBean sera appelée pour traiter les données (par exemple, pour additionner les deux nombres et afficher le résultat).
- 1. Créez un bean « numberBean » avec 3 propriétés numberX, numberY et result de type int.
- 2. Ajoutez une méthode add() dans numberBean qui effectue l'addition de numberX et numberY, et qui stocke le résultat dans la propriété result.
- 3. Créez un formulaire JSF (<h:form>) contenant deux champs de saisie pour les valeurs numberX et numberY : Utilisez les expressions EL pour lier chaque champ de saisie aux propriétés correspondantes du bean.
- 4. Ajoutez un bouton pour déclencher la méthode add() lorsque l'utilisateur clique sur "Add".
- 5. Ajoutez un composant <h:outputText> pour afficher le résultat de l'addition (la valeur de result dans le bean) une fois le bouton cliqué.

Exemple :

Web.xml

Fichier de déploiement de l'application

Déclare la servlet principale : javax.faces.webapp.FacesServlet

- point d'entrée d'une application JSF
 - Préfixe /faces/ (<http://localhost/myAppli/faces/index.jsp>)
 - Suffixes *.jsf ou *.faces (<http://localhost/myAppli/index.jsf>)

Spécifier le nom et le chemin du fichier de configuration

- Nom du paramètre : javax.faces.application.CONFIG_FILES
- Exemple : /WEB-INF/faces-config.xml

```
<context-param>  
    <param-name>javax.faces.application.CONFIG_FILES</param-name>  
    <param-value>/WEB-INF/faces-config.xml</param-value>  
</context-param>
```

Web.xml

Spécifie où l'état de l'application doit être sauvegardé

- Nom du paramètre : javax.faces.STATE_SAVING_METHOD
- Valeurs possibles : client ou server

```
<context-param>  
    <param-name>javax.faces.STATE_SAVING_METHOD</param-name>  
    <param-value>client</param-value> <!-- ou "server" -->  
</context-param>
```

Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
  http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd"
  id="WebApp_ID" version="2.5">
```

```
<!-- Nom de l'application -->
```

```
<display-name>HelloNormandyJug</display-name>
```

Nom de l'application

```
<!-- Utiliser les fichiers *.xhtml -->
```

```
<context-param>
```

```
<param-name>javax.faces.DEFAULT_SUFFIX</param-name>
```

```
<param-value>.xhtml</param-value>
```

```
</context-param>
```

Extension des pages

```
<!-- Servlet JSF -->
```

```
<servlet>
```

```
<servlet-name>Faces Servlet</servlet-name>
```

```
<servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
```

```
<load-on-startup>1</load-on-startup>
```

```
</servlet>
```

Servlet utilisée

```
<!-- Mapping de Servlet JSF -->
```

```
<servlet-mapping>
```

```
<servlet-name>Faces Servlet</servlet-name>
```

```
<!-- URL prises en compte -->
```

```
<url-pattern>*.jsf</url-pattern>
```

```
</servlet-mapping>
```

Mapping de la servlet

```
<!-- Point d'entrée -->
```

```
<welcome-file-list>
```

```
<welcome-file>login.jsf</welcome-file>
```

```
</welcome-file-list>
```

Page d'accueil

```
</web-app>
```

Faces-config.xml

Décrit essentiellement 5 éléments principaux

- 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 validateurs et des convertisseurs **<validator>** et **<converter>**
- Autres éléments liés à des nouveaux composants JSF **<render-kit>**

Navigation

Faces-config.xml indique au contrôleur le schéma de navigation.

<navigation-rule> pour paramétrer les règles de navigation

<from-view-id> indique la vue source où est effectuée la demande de redirection.

<navigation-case> précise une page vue destination pour chaque valeur de clé

<from-outcome> : la valeur de la navigation

<to-view-id> : la vue demandée

Navigation

Deux types de navigation

- Navigation statique: La valeur de la clé outcome est connue au moment de l'écriture de la vue
 - Pas besoin du fichier de configuration « faces-config » pour déclarer la navigation !
- Navigation dynamique: La valeur de la clé outcome est inconnue au moment de l'écriture des vues.
 - Utiliser « faces-config » pour la navigation
 - Les valeurs outcomes peuvent être calculées par des ManagedBeans

Navigation statique

Clé outcome définie dans la page JSF :

En cliquant sur le bouton, on serait redirigé vers la page avec la clé outcome (valeur de action du composant h:commandButton)

```
<h:form>  
  <h:commandButton action="page2" value="Move to page2.xhtml" />  
</h:form>
```

chercher s'il y a une vue avec le nom « page2 »
sans avoir recours au fichier de navigation

Navigation dynamique

Clé outcome est défini dans le managedBean

```
<h:form>
a <h:inputText label="a" id="a" value="#{addControl.first}"/>
b <h:inputText label="b" id="b" value="#{addControl.second}"/>
<h:commandButton value="add" action="#{addControl.add2()}/>
</h:form>
```

dans add2.xhtml

```
<h:body>
Addition result :
<h:outputText value="#{addControl.result}"/>
<h:link value="back to index" outcome="index"/>
</h:body>
```

dans addResult.xhtml

managed bean:

```
@RequestScoped
public class AddControl {
    private int first, second, result;
    ....
    public String add2() {
        result= first+ second;
        return "success";
    }
}
```


Navigation dynamique

faces-config.xml

```
<navigation-rule>  
  <from-view-id>/add2.xhtml</from-view-id>  
  <navigation-case>  
    <from-outcome>success</from-outcome>  
    <to-view-id>/addResult.xhtml</to-view-id>  
  </navigation-case>  
</navigation-rule>
```

Faces-config.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<faces-config xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-facesconfig_2_0.xsd"
  version="2.0">

  <application>
    <message-bundle>presentation/messages</message-bundle>
    <locale-config>
      <supported-locale>en</supported-locale>
      <supported-locale>fr</supported-locale>
    </locale-config>
  </application>

  <navigation-rule>
    <from-view-id>/login.xhtml</from-view-id>
    <navigation-case>
      <from-outcome>succes</from-outcome>
      <to-view-id>/WEB-INF/pages/accueil/accueil.xhtml</to-view-id>
    </navigation-case>
    <navigation-case>
      <from-outcome>erreur</from-outcome>
      <to-view-id>/login.xhtml</to-view-id>
    </navigation-case>
  </navigation-rule>

</faces-config>
```

Fichier de ressources
par défaut

I18N

Navigation

Les validateurs

Vérifier la validité des données

Applicable sur l'ensemble des composants de saisies

Exemples de validateurs :

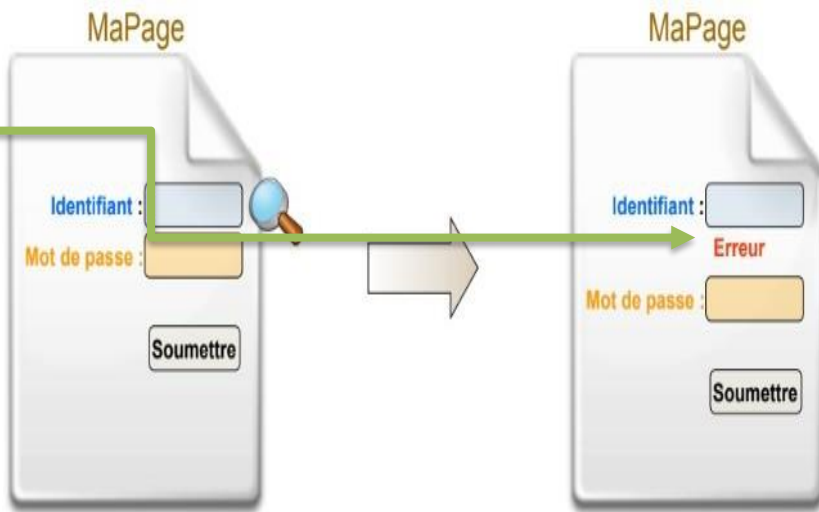
- Valider la présence de saisie
- Valider que la saisie est conforme à une plage de valeurs
- Valider le format de saisie (expression régulière)
- Valider la longueur de la saisie

Validateurs

Tag	Function
f:validateDoubleRange	Attributs minimum et maximum
f:validateLongRange	Attributs minimum et maximum
f:validateLength	Attributs minimum et maximum
f:validateRegEx	Une expression régulière
f:validator	Un validateur personnalisé l'attribut binding pointe vers une classe qui implémente Validator
f:validateBean	Un validateur personnalisé l'attribut binding pointe vers une instance de BeanValidator
f:validateRequired	Ne doit pas être nul (même effet que <i>required</i>)

Exemple

```
<h:form id="loginForm">
  <h3>Connexion</h3>
  <!-- Champ pour le nom d'utilisateur -->
  <h:outputLabel for="username" value="Nom d'utilisateur : " />
  <h:inputText id="username" value="#{authBean.username}" required="true"
    requiredMessage="Le nom d'utilisateur est obligatoire." />
  <h:message for="username" style="color: red;" />
  <br/><br/>
  <!-- Champ pour le mot de passe -->
  <h:outputLabel for="password" value="Mot de passe : " />
  <h:inputSecret id="password" value="#{authBean.password}" required="true"
    requiredMessage="Le mot de passe est obligatoire."
    <f:validateLength minimum="8" maximum="20" />
  </h:inputSecret>
  <h:message for="password" style="color: red;" />
  <br/><br/>
  <!-- Bouton de soumission -->
  <h:commandButton value="Se connecter" action="#{authBean.authenticate}" />
</h:form>
```



Validateur personnalisé

Le validateur doit implémenter l'interface
javax.faces.validator.Validator

Le validateur doit être déclaré soit dans faces-config.xml, soit en l'annotant avec **@FacesValidator("ID")**

```
@FacesValidator("emailValidator")
public class EmailValidator implements Validator<Object> {

    @Override
    public void validate(FacesContext context, UIComponent component, Object value)
        throws ValidatorException {
        String email = value.toString();
        // Validation simple pour vérifier si l'entrée ressemble à une adresse email
        if (!email.matches("^[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\\.[A-Za-z]{2,}$")) {
            throw new ValidatorException(new FacesMessage(FacesMessage.SEVERITY_ERROR,
                "Email invalide", "Veuillez saisir une adresse email valide."));
        }
    }
}
```

Utilisation dans JSF

```
<h:form>
```

```
  <h3>Inscription</h3>
```

```
  <!-- Champ pour l'adresse email -->
```

```
  <h:outputLabel for="email" value="Adresse Email :" />
```

```
  <h:inputText id="email" value="#{userBean.email}">
```

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

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

```
  <h:message for="email" style="color: red;" />
```

```
  <br/><br/>
```

```
  <!-- Bouton de soumission -->
```

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

```
</h:form>
```

```
@FacesValidator("emailValidator")
```

id du valideur

Convertisseurs

Permet la conversion des données :

- IHM vers ManagedBean
- ManagedBean vers IHM

Exemples :

- Conversion de date
- Conversion de nombre

Il est facile de créer son propre convertisseur.

Exemple

MaPage



Nom :

Prénom :

Date de naissance :

PersonneBean

```
/**
 * Nom de l'utilisateur
 */
private String nom;

/**
 * Prénom de l'utilisateur
 */
private String prenom;

/**
 * Date de naissance de l'utilisateur
 */
private Date dateNaissance;
```

```
<h:inputText id="dateNaissance" value="#{personneBean.dateNaissance}">
  <f:convertDateTime for="dateNaissance" type="date" pattern="dd/MM/yyyy"/>
</h:inputText>
```

Convertisseurs

Conversion automatique

- associer le composant à une propriété d'un managed bean avec un type standard
 - BigDecimalConverter, BooleanConverter, DateTimeConverter, DoubleConverter, EnumConverter, ...

Conversion manuelle

- `<h:inputText converter="javax.faces.convert.IntegerConverter"/>`

le champ de saisie est explicitement lié à un convertisseur qui garantit que la valeur entrée sera convertie en un entier

Conversion personnalisée

- Classe Java qui implémente l'interface `javax.faces.convert.Converter`.
- L'annotation `@FacesConverter`.
- Deux méthodes principales :
 - `getAsObject()`
 - `getAsString()`
- Associer le convertisseur au composant JSF.

```
@FacesConverter("customPersonConverter")
public class PersonConverter implements Converter {
    @Override
    public Object getAsObject(FacesContext context, UIComponent component, String value) {
        // Conversion de l'IHM vers un objet (ex. : ID → Objet)
        if (value == null || value.isEmpty()) {
            return null;
        }
        return new Person(Integer.parseInt(value), "NomExemple");
    }
    @Override
    public String getAsString(FacesContext context, UIComponent component, Object value) {
        // Conversion de l'objet vers l'IHM (ex. : Objet → ID)
        if (value == null) {
            return "";
        }
        return String.valueOf(((Person) value).getId());
    }
}
```

Conversion personnalisée

```
@FacesConverter("customPersonConverter")  
public class PersonConverter implements Converter {  
    @Override
```

Id du convertisseur

```
<h:selectOneMenu value="#{bean.selectedPerson}" converter="customPersonConverter">  
    <f:selectItems value="#{bean.personList}" var="person"  
        itemValue="#{person}" itemLabel="#{person.name}" />  
</h:selectOneMenu>
```

Events & Listeners

- JSF utilise le modèle évènement/écouteur (event/listner) du JavaBeans (utilisé by Swing)
- Les composants UI génèrent des évènements et les écouteurs sont enregistrés pour ces évènements
- Dans une application JSF, l'intégration de la logique applicative consiste à assigner l'écouteur approprié au composant qui génère des évènements
- Il y a 3 évènements standards
 - Value-change events
 - Action events
 - Data model events

Value-Change Events

Les évènements value-change sont générés lorsque l'utilisateur saisie une nouvelle donnée dans un composant `inputText`

```
<h:inputText valueChangeListener="#{myForm.processValueChanged}" />
```

Lorsque l'utilisateur modifie le contenu du champs de saisie et soumet le formulaire, JSF génère un évènement value-change

```
public void processValueChanged(ValueChangeEvent event){  
    HtmlInputText sender = (HtmlInputText)event.getComponent();  
    sender.setReadOnly(true);  
}
```

Action events

Générés lorsque l'utilisateur active des composants boutons ou lien, appelés aussi sources d'actions, intégrant des contrôles de boutons ou hyperlinks

Action events sont traités par les action listeners

Action Listeners

- affectent la navigation
 - Typiquement, ils réalisent des traitements et retournent une clé outcome, utilisée par le système de navigation de JSF pour sélectionner la page suivante
- N'affecte pas la navigation
 - Manipulent des composants dans la même vue, sans modifier la vue courante.

Action methods

```
<h:commandButton type="submit" value="Login" action="#{loginForm.login}"/>
```

Lorsque on clique sur le bouton, l'événement action est déclenché et la méthode login du ManagedBean loginForm est exécutée.

```
public class LoginForm {  
    // ...  
    public String login(){  
        // Si le login est valide  
        if (/*...*/) {  
            return "success";  
        } else{  
            return "failure";  
        }  
    }  
    // ...  
}
```


Action Listener Methods

Pour exécuter un code métier qui n'est pas associé à la navigation.

On associe une méthode ActionListener avec le composant

```
<h:commandButton id="redisplayCommand" type="submit"  
    value="Redisplay" ActionListener="#{myForm.doIt}"/>
```

Contrairement aux méthodes d'action, les actions listeners ont accès au composant qui a déclenché l'évènement.

```
public void doIt(ActionEvent event){  
    HtmlCommandButton button = (HtmlCommandButton)event.getComponent();  
    button.setValue("It's done!");  
}
```

Exercice

- ❖ Supposez que l'on dispose d'un Bean avec une méthode login et des propriétés username, password et birthDate. Créez un formulaire dans une page JSF avec les champs suivants :
 - **Login** : chaîne de caractères obligatoire avec une longueur comprise entre 5 et 30 caractères.
 - **Mot de passe** : champ masqué avec une longueur minimale de 8 caractères.
 - **Date de naissance** : champ texte pour la saisie d'une date au format jj/MM/aaaa.
- Si les données sont invalides, affichez un message d'erreur spécifique sous chaque champ.

Exercise

JSF et AJAX

AJAX: **A**synchronous **J**avascript and **X**ML

Une requête Ajax envoyée par le client est lancée « en arrière-plan » ; l'utilisateur peut continuer à travailler avec la page en cours

La réponse du serveur à la requête ne modifie qu'une partie de la page en cours

JSF et AJAX

En JSF 2, l'utilisation d'ajax est simple <f:ajax>

<f:ajax> Ajoute des fonctionnalités Ajax au composant JSF dans lequel il est inclus

Un événement déclencheur sur ce composant provoque l'envoi d'une requête Ajax au serveur sans attendre la soumission du formulaire

Par exemple,

une liste déroulante peut envoyer une requête Ajax quand l'utilisateur fait un nouveau choix ; la réponse du serveur ne mettra à jour qu'une partie de la page, par exemple une 2 ème liste déroulante dont les valeurs dépendent du choix fait par l'utilisateur dans la 1ère liste

JSF et AJAX

les attributs <f:ajax>

- event : événement déclencheur de la requête
 - Dépend du composant
 - Par exemple, valueChange pour les <h:inputText> ou click pour les <h:commandButton>
- execute : composants pris en compte pour l'envoi de la requête (espace séparateur) ; valeur par défaut @this, le composant qui contient <f:ajax>
- render : composants mis à jour au moment de la réponse du serveur (espace séparateur) ; valeur par défaut @none

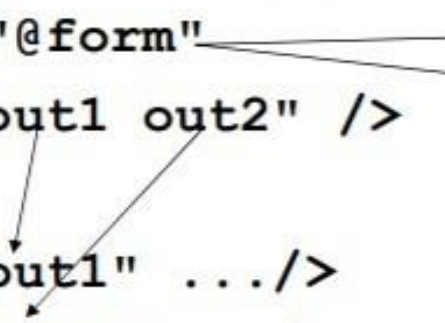
Render et execute

une liste de valeurs séparées par des espaces :

- id d'un élément dans le même formulaire
- id d'un élément dans un autre formulaire, avec notation « :idparent:idElement »
- @this désigne l'élément sur lequel porte la balise ajax
- @form désigne le formulaire dans son ensemble
- @all désigne tous les composants
- @none aucun composant

Exemple

```
<h:form>
  <h:inputText value=... />
  <h:commandButton value="Enregistrer" ...>
    <f:ajax execute="@form"
              render="out1 out2" />
  </h:commandButton>
  <h:outputText id="out1" .../>
  <h:outputText id="out2" .../>
</h:form>
```



Formulaire englobant

The diagram consists of a box labeled "Formulaire englobant" with two arrows pointing from it to the `<f:ajax>` tag and the `render="out1 out2"` attribute in the code snippet above.

Exercice

1. Créez un Managed Bean « User » avec :

- Une propriété name pour capturer le nom entré par l'utilisateur.
- Une méthode sayHello() pour générer un message de salutation.

2. Créez un formulaire JSf avec :

- un composant de texte statique affichant le texte "Veuillez entrer votre nom ».
- un champ de texte permettant à l'utilisateur d'entrer son nom.
- Un bouton qui, lorsqu'il est cliqué, appelle la méthode sayHello() et déclenche un événement AJAX pour mettre à jour le composant dynamique.
- un composant de texte dynamique avec le nom saisi.

Exercise

Quiz !

Qu'est-ce que JSF ?

- a) Un framework de gestion des bases de données
- b) Un framework pour développer des interfaces utilisateur web en Java
- c) Un moteur de template pour HTML
- d) Un framework uniquement pour les applications mobiles

Quiz !

Quel fichier est généralement utilisé pour définir les vues JSF ?

- a) web.xml
- b) .jsp
- c) .xhtml
- d) .html

Quiz !

Quel est le rôle de l'annotation @ManagedBean dans JSF ?

- a) Déclarer une variable d'instance
- b) Analyser une expression EL
- c) Déclarer un bean géré par JSF
- d) Créer une vue dynamique

Quiz !

Dans JSF, quel fichier est utilisé pour configurer la navigation entre les pages ?

- a) faces-config.xml
- b) web.xml
- c) application.properties
- d) beans.xml

Quiz !

Quelle balise JSF permet d'envoyer une requête de type POST au serveur ?

- a) `<h:commandLink>`
- b) `<h:outputLink>`
- c) `<h:commandButton>`
- d) `<h:inputText>`

Quiz !

Que permet l'Expression Language (EL) dans JSF ?

- a) Manipuler des objets Java en back-end
- b) Récupérer et manipuler des données dans des pages JSP ou XHTML
- c) Interagir avec la base de données
- d) Valider les formulaires HTML

Quiz !

Comment accéder à la valeur d'une propriété d'un bean géré dans JSF ?

- a) `{bean.property}`
- b) `{bean.property}`
- c) `{bean.getProperty()}`
- d) `{bean.getProperty()}`

Quiz !

Dans JSF, quelle balise est utilisée pour afficher un message d'erreur de validation ?

- a) `<h:messages>`
- b) `<h:inputText>`
- c) `<h:outputText>`
- d) `<h:commandButton>`

Quiz !

Comment valider un formulaire en utilisant JSF et Beans ?

- a) La validation se fait uniquement côté client avec JavaScript
- b) En utilisant des annotations de validation dans le Managed Bean et en affichant les erreurs avec `<h:messages>`
- c) Par des expressions EL dans le fichier .xhtml
- d) En déclarant des règles de validation dans faces-config.xml

Quiz !

Quelle annotation est utilisée pour valider un champ dans un Managed Bean JSF ?

- a) @Size
- b) @Email
- c) @NotNull
- d) @Valid