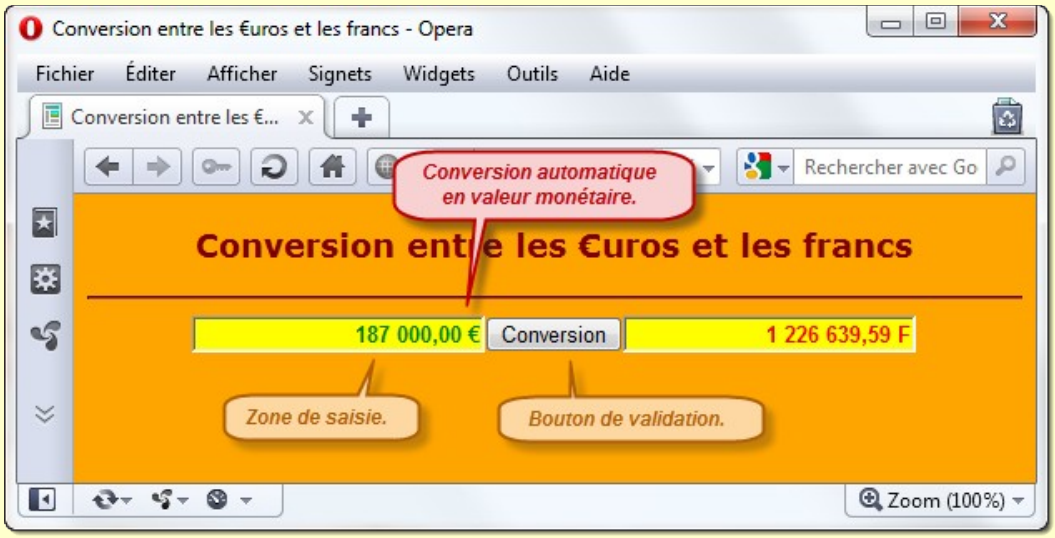
Pour afficher graphiquement les informations provenant du serveur, nous avons besoin d'une interface utilisateur. Les applications utilisant les interfaces pour interagir avec l'utilisateur sont de différents types : applications de bureau, applications web s'exécutant dans un navigateur ou applications mobiles sur un terminal portable. Nous nous intéressons ici sur les **interfaces web**.

- **Les pages web statiques** sont composées de HTML pur contenant éventuellement des graphiques eux aussi statiques (JPG, PNG, par exemple).
- **Les pages web dynamiques** sont en revanche composées en temps réel (**à la volée**) à partir de données calculées à partir d'informations fournies par l'utilisateur.

INTRODUCTION À JSF

Les applications JSF sont des applications web classiques qui interceptent le protocole HTTP via la servlet Faces et produisent des documents HTML à la volée.

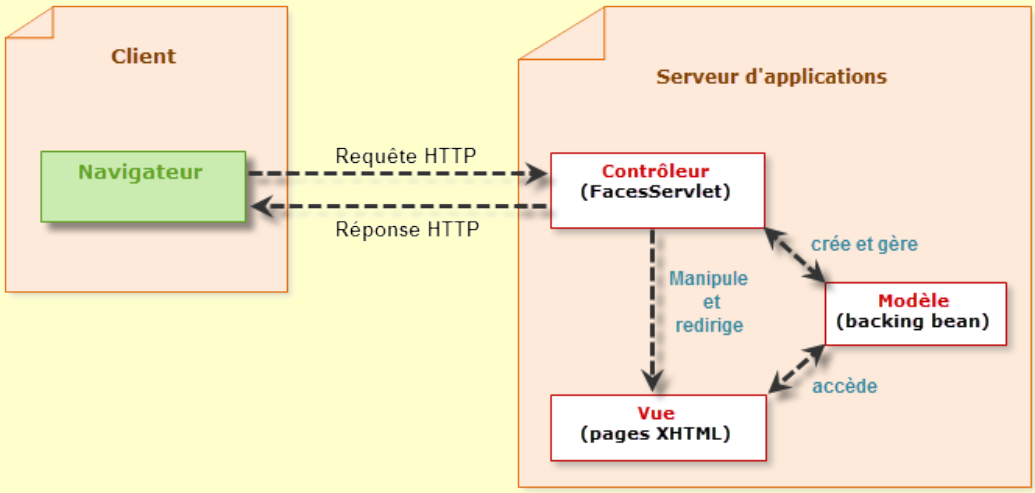
***JSF** fournit un ensemble de widgets standard (**boutons, liens hypertextes, cases à cocher, zone de saisie, tableaux dynamiques, etc.**) et facilite son extension par l'ajout de composants tiers.*



MODÈLE MVC

JSF utilise le patron de conception **MVC** (***Modèle-Vue-Contrôleur***). **MVC** permet de découpler la ***vue*** (*la page*) et le ***modèle*** (*le traitement des données affichées dans la vue*). Le ***contrôleur*** prend en charge les actions de l'utilisateur qui pourraient impliquer des modifications dans le ***modèle*** et dans les ***vues***.

*Comme le montre la figure ci-dessous, la partie **modèle** de **MVC** représente les données (**et leurs différents traitements**) de l'application ; la **vue** correspond à l'interface utilisateur et le **contrôleur** gère la communication entre les deux (**utilisation du protocole HTTP**) :*



- Le ***modèle*** est représenté par le contenu, qui est quelque fois stocké dans une base de données et affiché dans la vue ; il ne se soucie pas de l'aspect que verra l'utilisateur. Avec JSF, il peut être formé d'un simple **bean** (JavaBean), d'appels **EJB**, d'entités **JPA,** etc.

```java
package bean;

import javax.faces.bean.*;

@ManagedBean
@ViewScoped
public class Conversion {
   private final double TAUX = 6.55957;
   private double euro;
   private double franc;

   public double getEuro() { return euro; }
   public void setEuro(double euro) { this.euro = euro; }

   public double getFranc() { return franc; }
   public void setFranc(double franc) { this.franc = euro * TAUX; }

   public void euroFranc() { franc = euro * TAUX; }
}
```

Propriété euro

Propriété franc.

euroFranc() réalise la conversion.

✗ La ***vue*** JSF est la véritable page XHTML (XHTML est réservée aux interfaces web, mais il pourrait s'agir d'autre type de vue, comme WML pour les dispositifs mobiles). Une ***vue*** fournit une représentation graphique d'un ***modèle*** et un ***modèle*** peut avoir plusieurs ***vues*** pour prévoir un affichage sous forme de formulaire ou sous forme de liste, par exemple.

```xml
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/
<html xmlns="http://www.w3.org/1999/xhtml"
    xmlns:h="http://java.sun.com/jsf/html"
    xmlns:f="http://java.sun.com/jsf/core">
  <h:head>
    <title><h:outputText value="Conversion entre les €uros et les francs" /></title>
  </h:head>

  <h:outputStylesheet name="principal.css" />

  <h:body>
    <h3><h:outputText value="Conversion entre les €uros et les francs" /></h3>
    <hr />
    <h:form>
      <h:inputText value="#{conversion.euro}" styleClass="saisie">
        <f:convertNumber type="currency" currencySymbol="€"/>
        <f:validateDoubleRange minimum="0.0" />
      </h:inputText>

      <h:commandButton value="Conversion" action="#{conversion.euroFranc}" />

      <h:inputText value="#{conversion.franc}" readonly="true" styleClass="resultat">
        <f:convertNumber type="currency" currencySymbol="F"/>
      </h:inputText>
    </h:form>
  </h:body>
</html>
```

Propriété euro du bean géré conversion

exécution de la méthode euroFranc() du bean géré conversion.

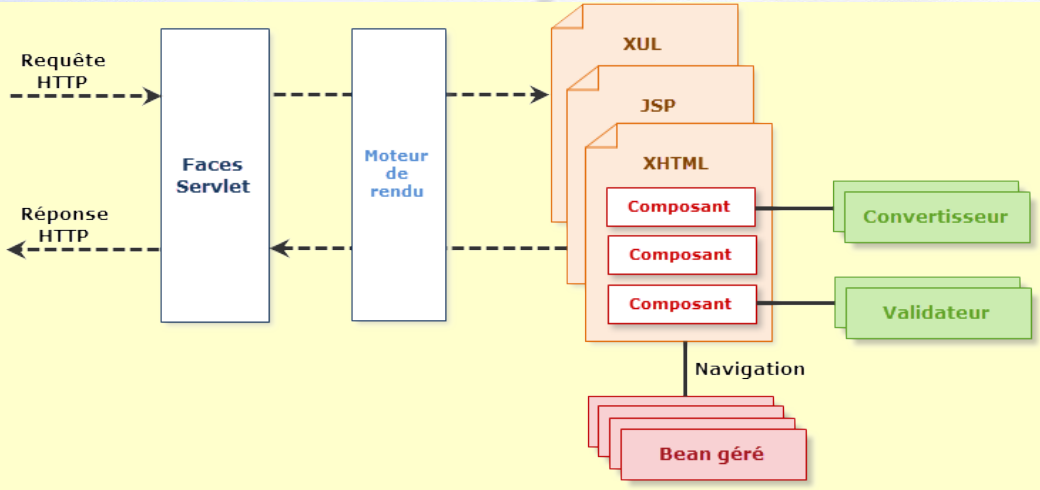
Propriété franc du bean géré conversion

✗ Lorsqu'un utilisateur manipule une ***vue***, celle-ci informe un ***contrôleur*** des modifications souhaitées. Ce ***contrôleur*** se charge alors de rassembler, convertir et valider les données, appelle la logique métier puis produit le contenu en XHTML. Avec JSF, le contrôleur est un objet ***FacesServlet***.

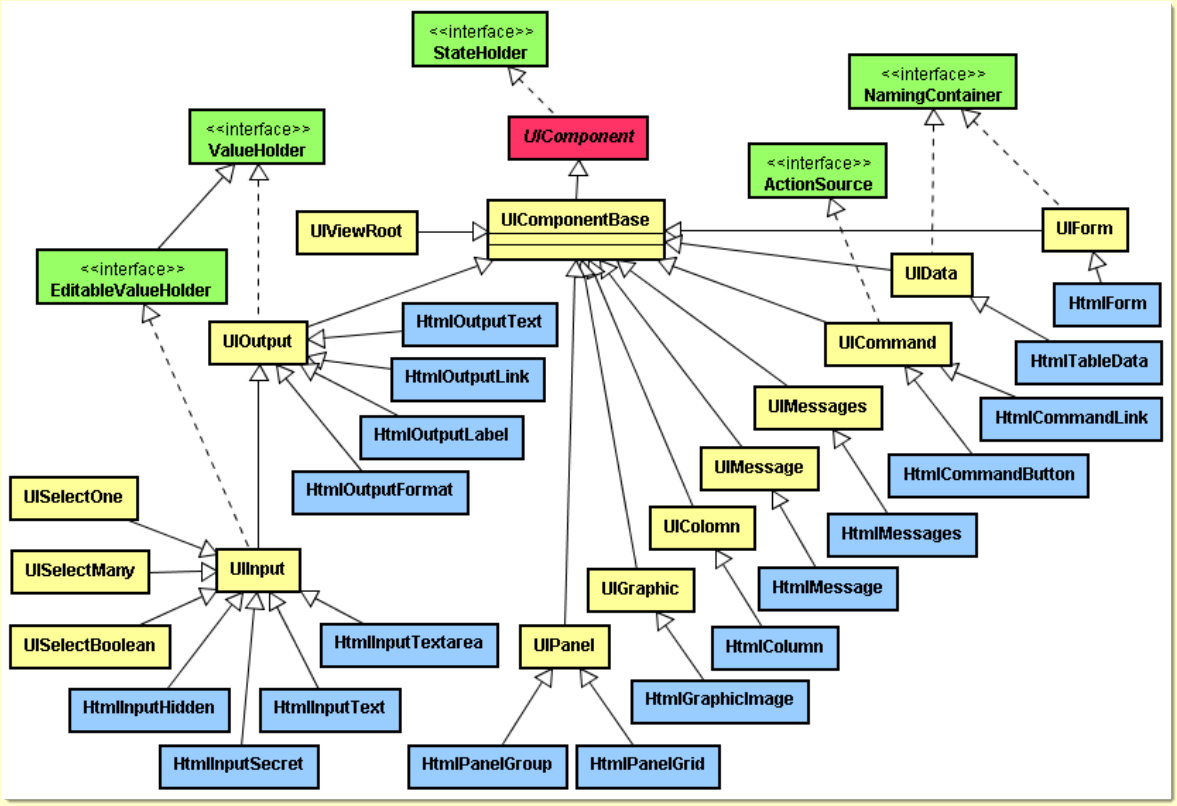
*Cette **servlet** est déjà fabriquée. Nous n'avons aucun code à écrire en son sein. Nous pouvons toutefois influencer son comportement au moyen d'annotations spécifiques, comme par exemple **@ManagedBean**. Dans ce cas là, la **servlet** s'occupera alors de générer un nouvel objet correspondant à la classe du bean géré avec une durée de vie adaptée.*

✗ ARCHITECTURE INTERNE DE JSF

FacesServlet : est la servlet (***composant côté serveur spécialisé pour la prise en compte du protocole HTTP***) principale 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 du ***modèle*** en utilisant les ***beans gérés***. Il est possible de configurer le comportement de cette servlet au travers ***d'annotations spécifiques*** (***sur les beans gérés, sur les convertisseurs, sur les composants internes, sur le moteur de rendu et sur les validateurs***).



✗ **Pages et composants** : Le framework ***JSF*** doit envoyer une page sur le dispositif de sortie du client (un navigateur, pas exemple) et exige donc une technologie d'affichage appelée ***Facelets***. ***Facelets*** est formée d'une arborescence de composants (également appelés widgets ou contrôles) fournissant des fonctionnalités spécifiques pour interagir avec l'utilisateur (***champ de saisie, boutons, liste, etc.***).



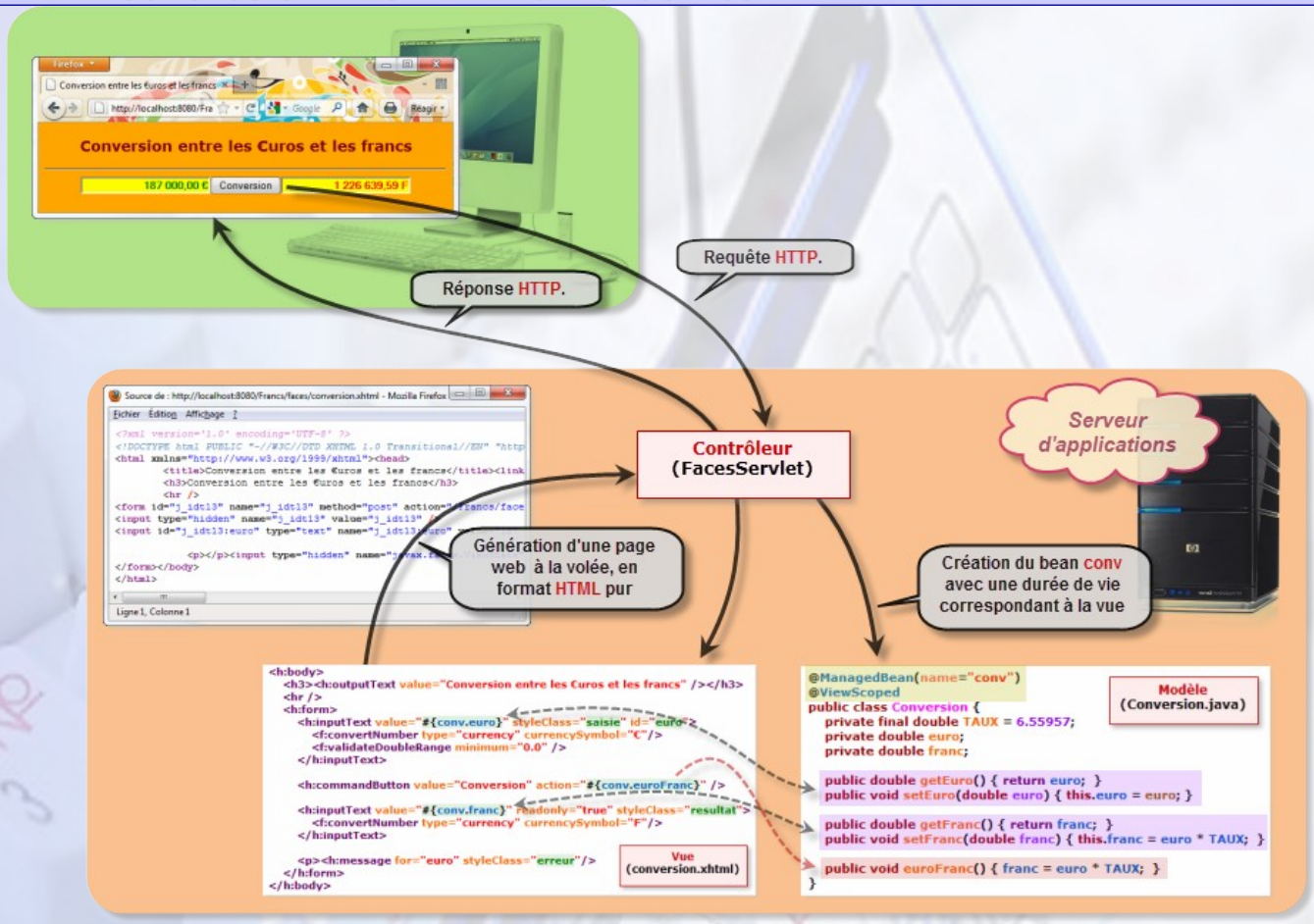
✗ **Un moteur de rendu** s'occupe d'afficher un composant et de traduire la saisie d'un utilisateur en valeur de composants. Nous pouvons donc le considérer comme un traducteur placé entre le client et le serveur : il décode la requête de l'utilisateur pour initialiser les valeurs du composant et encode la réponse pour créer une représentation du composant que le client pourra comprendre et afficher.

✗ **Convertisseurs** : Lorsque la page est affichée, l'utilisateur peut s'en servir pour entrer des données. Comme il n'y a pas de contraintes sur les types, un moteur de rendu ne peut pas prévoir l'affichage de l'objet. Voilà pourquoi les convertisseurs existent : ils traduisent un objet (***Integer, Date, Enum, Boolean, etc.***) en ***chaîne de caractères*** afin qu'il puisse s'afficher (***protocole HTTP est un protocole uniquement textuel***) et, inversement, construisent un objet à partir d'une chaîne qui a été saisie. JSF fournit un ensemble de convertisseurs pour les types classiques, mais vous pouvez en toute liberté développer les vôtres.

✗ **Validateurs** : Parfois, les données doivent également être validées avant d'être traitées par le ***modèle*** en coulisse : c'est le rôle des validateurs ; nous pouvons ainsi associer un ou plusieurs validateurs à un composant unique afin de garantir que les données saisies sont correctes. Nous pouvons ainsi imposer des valeurs limites à ne pas dépasser ou d'autres contraintes encore plus sophistiquées.

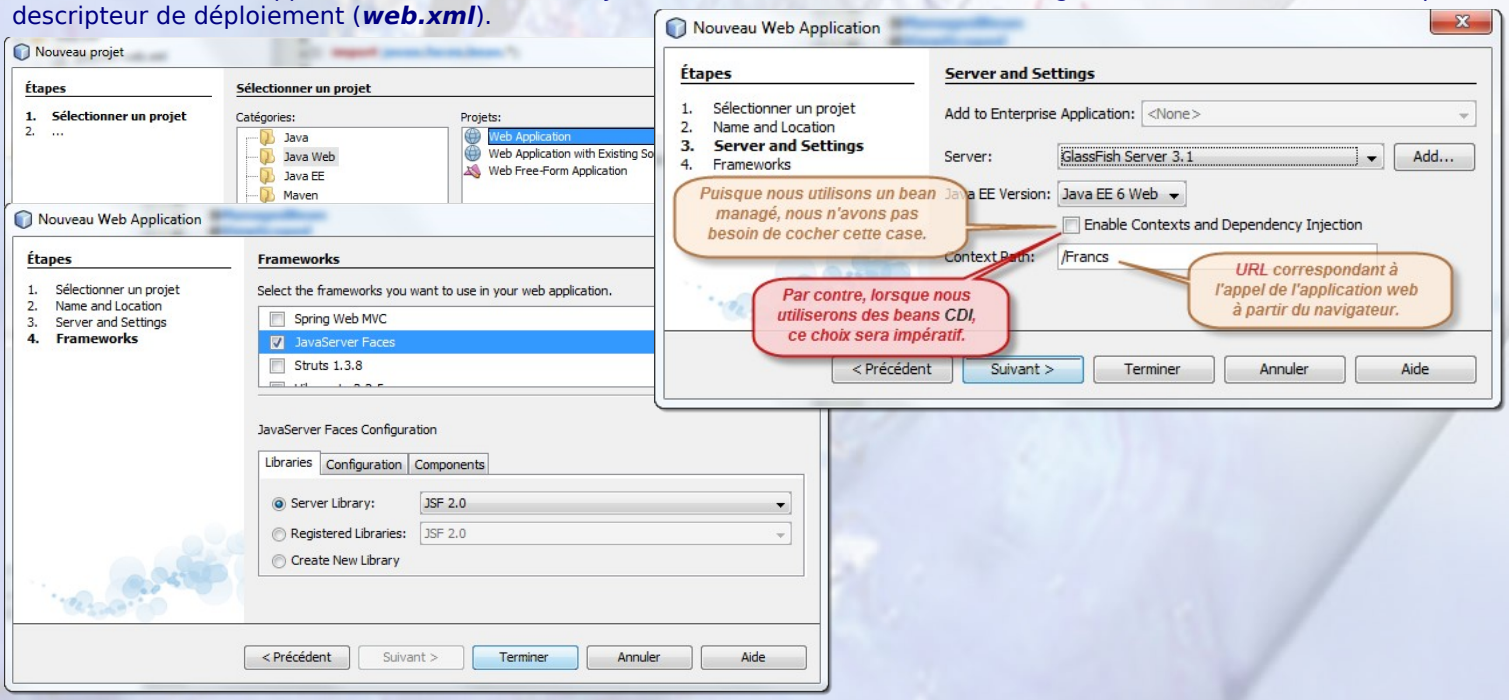
✗ **Beans gérés et navigation** : Tous les concepts que nous venons de présenter - qu'est-ce qu'une page, qu'est-ce qu'un composant, comment sont-il affichés convertis et validés - sont liés à une page unique, mais les applications web sont généralement formées de plusieurs pages et doivent réaliser un traitement métier (en appelant une couche EJB, par exemple). Le passage d'une page à une autre, l'invocation d'EJB et la synchronisation des données avec les composants sont pris en charge par les ***beans gérés***.

PREMIÈRE CONCLUSION DE CETTE TECHNOLOGIE JSF



MISE EN ŒUVRE DU PROJET DE L'APPLICATION WEB – CONVERSION MONÉTAIRE

Nous allons valider tous les différents concepts que nous venons de découvrir durant tout le début de cette étude au travers du projet de conversion sur lequel nous avons également travaillé. Nous en profiterons pour découvrir l'architecture d'une application web traité avec JSF et de voir comment se traite la gestion des ressources ainsi que le descripteur de déploiement (***web.xml***).

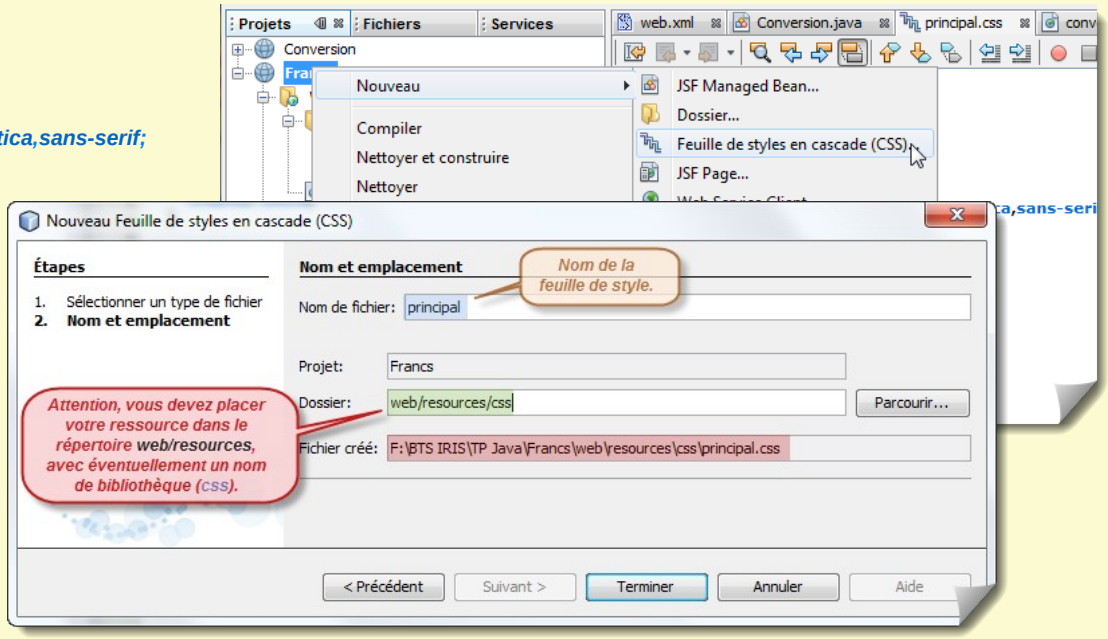


LES RESSOURCES

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. 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.

resources/<identifiant_ressource>

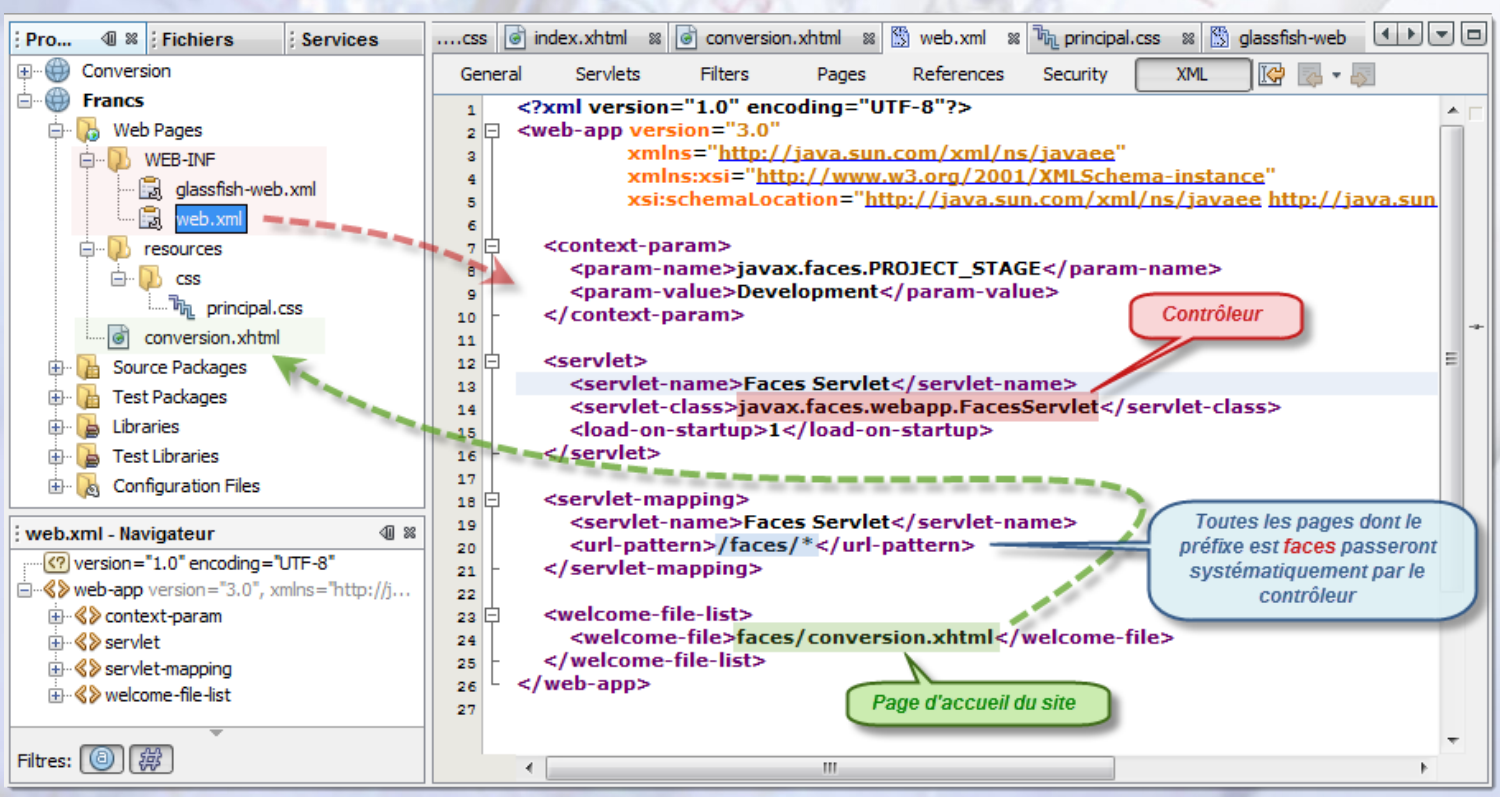
```
body {
    background-color: orange;
    color: maroon;
    font-family: Verdana,Arial,Helvetica,sans-serif;
    text-align: center;
}

.saisie, .resultat {
    background-color: yellow;
    text-align: right;
    font-weight: bold;
    color: green;
}

.resultat, .erreur  {
    color: red;
}
```



Le fait de choisir une feuille de style plutôt que d'intégrer les styles directement dans la vue me paraît être une attitude de bon développeur. Il faut toujours séparer la mise en forme de votre page web par rapport à son ossature, surtout si vous devez en construire plusieurs. Toutes les pages respecterons ainsi la même charte graphique. A tout moment, vous pouvez changer cette charte sans remettre en cause les structures de développement de vos différentes vues. Pour finir, lorsque vous prenez une feuille de style à part entière, vous bénéficiez d'un éditeur CSS intégré, ce qui n'est pas négligeable.

DESCRIPTEUR DE DÉPLOIEMENT

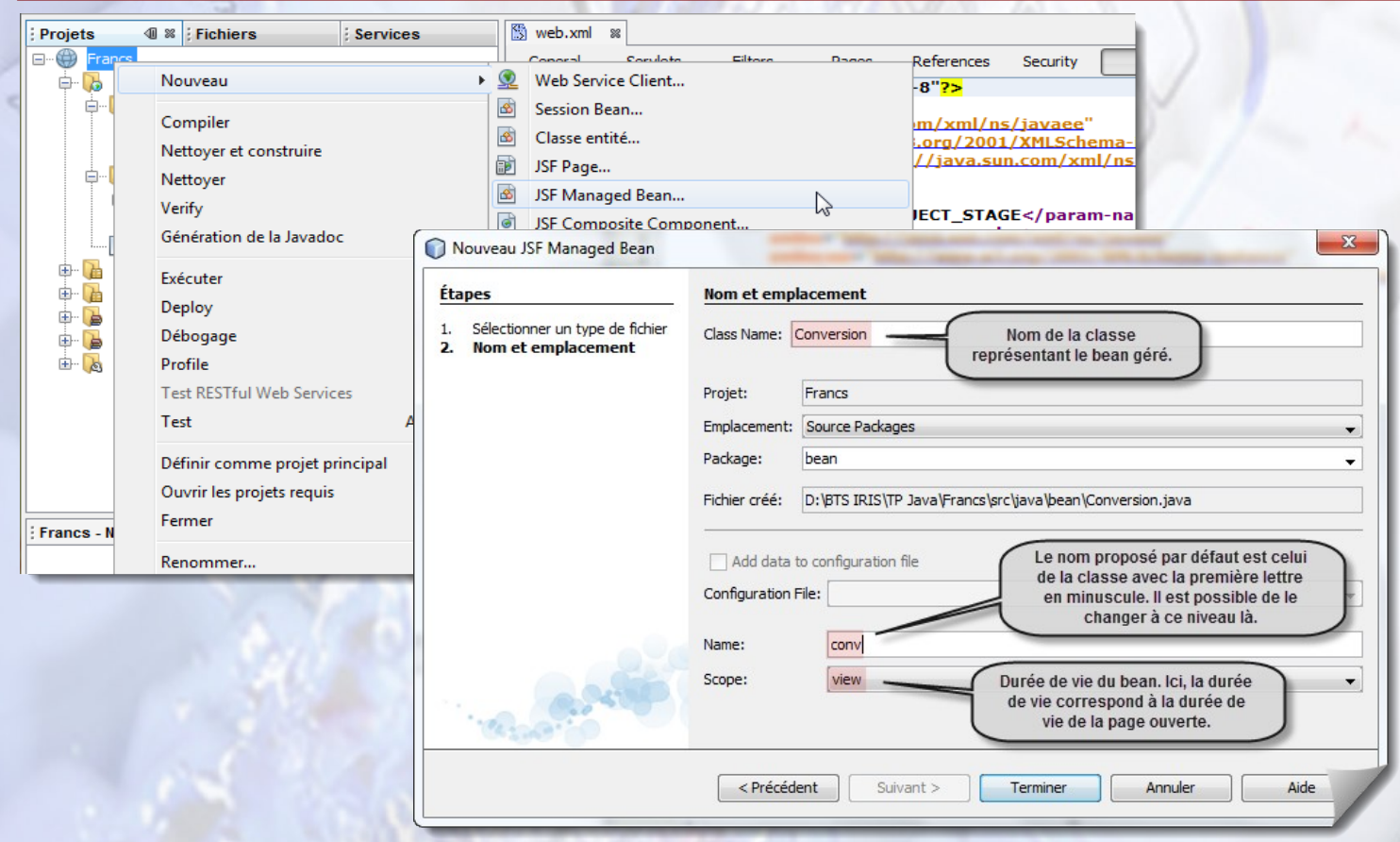
Les requêtes qui doivent être traitées par la **FacesServlet** sont redirigées à l'aide d'une association de servlet dans le descripteur de déploiement (**web.xml**) qui se situe dans le répertoire **WEB-INF**. Les pages web, les beans gérés, les convertisseurs, etc. doivent être assemblés à l'aide de ce fichier.



Le bean géré Conversion

Comme nous l'avons évoqué plus haut dans ce chapitre, le modèle **MVC** encourage la séparation entre le ***modèle***, la ***vue*** et le ***contrôleur***. Avec Java EE, les pages JSF forment la ***vue*** et la **FacesServlet** est le ***contrôleur***. Les **beans gérés**, quant à eux, sont une passerelle vers le ***modèle***.

- **Les beans gérés** sont des ***classes Java annotées***. Ils constituent le cœur des applications web car ils exécutent la logique métier (ou la délèguent aux EJB, par exemple), gèrent la navigation entre les pages et stockent les données. Une application JSF typique contient un ou plusieurs beans gérés qui peuvent être partagés par plusieurs pages.
- Les données sont stockées dans les attributs du **bean géré**, qui, en ce cas, est également appelé **"*backing bean*"**. Un backing bean définit les données auxquelles est lié un composant de l'interface utilisateur (la cible d'un formulaire, par exemple). Pour établir cette liaison, nous utilisons EL, le langage d'expressions.
- Ecrire un **bean géré** est simple puisqu'il s'agit simplement de créer une classe Java annotée par ***@ManagedBean***.
- Les **beans gérés** sont des classes Java prises en charge par la **FacesServlet**. Les composants de l'interface utilisateur sont liés aux propriétés du bean et peuvent invoquer des méthodes d'action.
- La présence de l'annotation ***@ManagedBean*** sur une classe l'enregistre automatiquement comme un **bean géré**. C'est la **FacesServlet** qui gère cette classe. Par défaut, le nom du **bean géré** (objet) est celui de la classe commençant par une minuscule. Il est possible de choisir un autre nom en spécifiant l'attribut **name** de l'annotation ***@ManagedBean***.
- Les composants de l'interface utilisateur étant liés aux propriétés du **bean géré**, changer son nom par défaut a des répercussions sur la façon d'appeler une propriété ou une méthode.
- Les objets créés dans le cadre d'un **bean géré** ont une certaine durée de vie et peuvent ou non être accessibles aux composants de l'interface utilisateur ou aux objets de l'application. Cette durée de vie et cette accessibilité sont regroupées dans la notion de portée. La portée que je choisi pour le **bean géré** est ***@ViewScoped.*** Cet objet est alors disponible dans une vue donnée jusqu'à sa modification. Son état persiste jusqu'à ce que l'utilisateur navigue vers une autre vue, auquel cas il est supprimé.



- Mise à part les annotations, nous retrouvons une classe normale avec des **propriétés** et une **méthode de traitement**.
- Ces **propriétés** sont indispensables pour que la page web puisse y accéder et qu'il y ait une **interaction** possible entre le ***modèle*** et la ***vue***.
- L'annotation ***@ManagedBean*** est également indispensable pour stipuler que cette classe doit être prise en compte par le contrôleur (être gérée). C'est finalement l'équivalent d'un dispositif de configuration puisque nous demandons ainsi que le nom de l'objet correspondant s'appelle conv. Par ailleurs, nous configurons également la durée de vie de cet objet pour qu'il soit en adéquation avec la vue au moyen de l'annotation ***@ViewScoped***.
- Grâce à ces simples annotations, il est très facile de changer de nom d'objet ainsi que sa durée de vie.

bean.Conversion.java

```
package bean;

import javax.faces.bean.*;

@ManagedBean(name="conv")
@ViewScoped
public class Conversion {
	private final double TAUX = 6.55957;
	private double euro;
	private double franc;

	public double getEuro() { return euro;  }
	public void setEuro(double euro) { this.euro = euro; }

	public double getFranc() { return franc;  }
	public void setFranc(double franc) { this.franc = euro * TAUX;  }

	public void euroFranc() { franc = euro * TAUX;  }
}
```

x LA VUE CONVERSION.XHTML

Le modèle, vous venez de le découvrir est très simple. Nous nous intéressons maintenant à notre dernier élément de l'application web, à savoir la ***vue***, plus précisément la page **conversion.xhtml**.

Lorsque vous travaillez avec la vue, vous disposez d'un certain nombre de composants (marqueurs) déjà construits qui sont installés dans des bibliothèques spécifiques qu'il faut prendre en compte lors de l'élaboration de vos pages web si vous désirez les utiliser.

URI	Préfixe classique	Description
http://java.sun.com/jsf/html	h	Contient les composants et leurs rendus HTML (h:commandButton, h:inputText, etc.)
http://java.sun.com/jsf/core	f	Contient les actions personnalisées indépendantes d'un rendu particulier (f:convertNumber, f:validateDoubleRange, f:param, etc.)
http://java.sun.com/jsf/facelets	ui	Marqueurs pour le support les modèles de page.
http://java.sun.com/jsf/composite	composite	Sert à déclarer et à définir des nouveaux composants personnalisés.
http://java.sun.com/jsp/jstl/core	c	Les pages Facelets peuvent éventuellement utiliser certains marqueurs issus de JSP (c:if, c:forEach et c:catch). A éviter si possible.
http://java.sun.com/jsp/jstl/functions	fn	Les pages Facelets peuvent utiliser tous les marqueurs de fonctions issus de la technologie JSP.

conversion.xhtml

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
	xmlns:h="http://java.sun.com/jsf/html"
	xmlns:f="http://java.sun.com/jsf/core">

	<h:head><title><h:outputText value="Conversion entre les €uros et les francs" /></title></h:head>

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

	<h:body>
		<h3><h:outputText value="Conversion entre les €uros et les francs" /></h3>
		<hr />
		<h:form>
			<h:inputText value="#{conv.euro}"
					styleClass="saisie"
					id="euro"
					converterMessage="Il faut une valeur monétaire"
					validatorMessage="Uniquement les nombres positifs"
					required="true"
					requiredMessage="Précisez votre valeur monétaire">

				<f:convertNumber type="currency" currencySymbol="€"/>
				<f:validateDoubleRange minimum="0.0" />
			</h:inputText>
			<h:commandButton value="Conversion" action="#{conv.euroFranc}" />
			<h:inputText value="#{conv.franc}" readonly="true" styleClass="resultat">
				<f:convertNumber type="currency" currencySymbol="F"/>
			</h:inputText>
			<p><h:message for="euro" styleClass="erreur"/></p>
		</h:form>
	</h:body>
</html>
```

- Au début, dans la balise **<html>**, vous spécifiez les bibliothèques qui vous serons utiles pour la constitution de votre page.
- Par exemple, sauf cas très rare, nous avons toujours besoin des marqueurs relatifs au rendu, comme les zones de saisie, les boutons etc. Aussi, nous proposons l'espace de nom **h** (pour HTML) associé à la bibliothèque **xmlns:h=**"http://java.sun.com/jsf/html".
- Pour les marqueurs qui réalise du traitement en coulisse, comme les convertisseurs et les validateurs, nous introduisons la bibliothèque **xmlns:f=**"http://java.sun.com/jsf/core". (Nous aurions pu prendre l'espace de nom c, mais il est généralement déjà utilisé par la JSTL).
- La balise **<h:outputStylesheet library="css" name="principal.css" />** nous permet de prendre en compte la feuille de style **principal.css** que nous avons mise en œuvre au préalable, dans la bibliothèque (répertoire) **css**. Vous pouvez placer votre feuille de style directement dans le répertoire **resources** (donc sans bibliothèque), auquel cas vous n'avez plus à spécifier l'attribut **library**.
- Nous retrouvons pratiquement le même code que nous avons déjà consulté mainte fois. Vous remarquez toutefois la présence d'un nouveau marqueur **<h:message for="euro" styleClass="erreur"/>** qui permet d'avertir l'utilisateur d'une mauvaise utilisation sur la zone de saisie des €uros.
- Cette balise doit toujours être rattachée à une autre au moyen de l'attribut **for**. De la même façon, la balise en question, comme ici **<h:inputText value="#{conv.euro}" id="euro">**, doit posséder l'attribut **id** à l'intérieur duquel vous spécifiez un nom de variable à prendre en compte.
- Toujours dans cette balise, vous avez la possibilité de spécifier les messages qui s'afficheront automatiquement lors d'une erreur de saisie particulière. Ici, nous proposons des messages d'erreur de conversion, de validation et de champ non vide, respectivement avec les atttributs **converterMessage**, **validatorMessage** et **requiredMessage**.