# Sommaire

# TP 4- Validation et messages

**Objectifs :**

Manipuler les tags de validation des champs de saisie.
Mettre en place de fichiers de stockage de libellés internationalisés (fichiers properties).

4.a) Dans la page `login.jsp`, mettre en place des tags de validation pour les champs `login` et `password`.
En cas d'erreur, les messages d'erreur affichés sont les messages par défaut.

Critères de validation :

- Chaque champ est obligatoire.
- La taille de chaque champ est limitée à 8 caractères.

⇨ Positionner l'affichage des éventuels messages d'erreur dans la page JSP à l'aide de tags `<h:message … />`

 Tester.

4.b) Utilisation de fichiers internationalisés pour les messages d'erreur.

⇨ Mise en place

- Dans le répertoire `WEB-INF/src/messages`, importer les fichiers `messages.properties` et `messages_en.properties`.

⇨ Déclarer les fichiers nouvellement créés dans le `faces-config.xml`. La langue par défaut doit être le français.

 Tester que les messages d'erreur sont maintenant personnalisés.

4.c) Utilisation de fichiers internationalisés pour les libellés de champs de formulaire.

Les fichiers properties importés dans la question précédente contiennent les libellés internationalisés nécessaires pour la page `login.jsp`.

⇨ Utiliser ces libellés dans la page `login.jsp` (message de bienvenue, libellés des champs, boutons).

 Tester

4.d) Utilisation d'une feuille de styles `css` pour les messages d'erreur

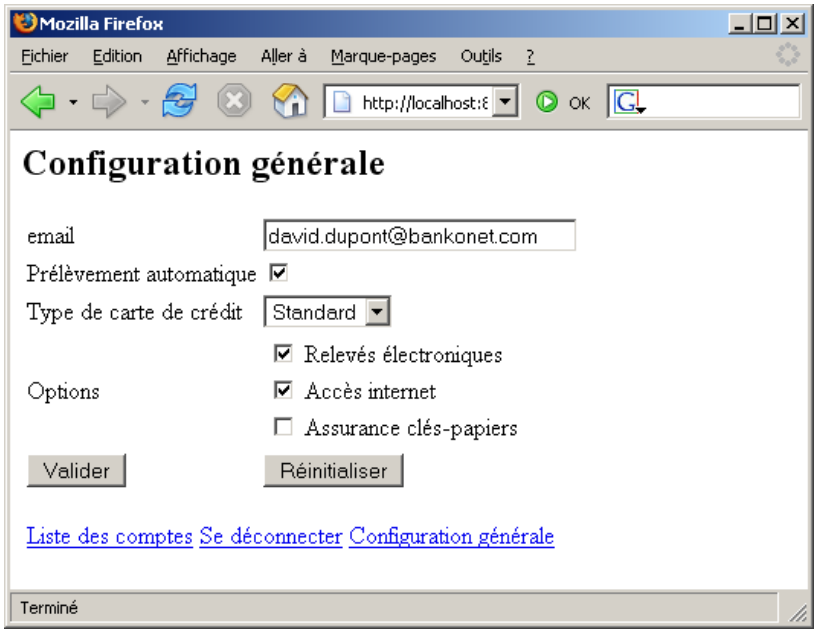
⇨ Récupérer le fichier `bankonetStyles.css` dans votre espace de travail.

⇨ Appliquer cette feuille de styles à vos messages d'erreur.

 Tester

# TP 5- Formulaires avancés

**Objectifs :**
Comprendre les possibilités de JSF pour la génération de cases à cocher, listes de choix etc.




*Pour cet exercice, il est conseillé de ne pas travailler avec des libellés internationalisé. En effet, l'externalisation des libellés compliquerait fortement le code.*

## 5.a) Affichage de la page de configuration générale du client.

⇨ Récupérer la classe `ConfigGeneraleBean` et la placer dans le package `com.bankonet.bean` .

- Cette classe contient l'initialisation de l'attribut `allOptions` et sa méthode d'accès.

⇨ Dans `ConfigGeneraleBean`, créer la méthode `init()`, et l'implémenter comme suit :

- Dans la couche service , effectuer l'appel approprié pour récupérer un objet de type `ConfigGenerale`. Cet objet doit être stocké comme variable d'instance de `ConfigGeneraleBean`.
- Si la récupération s'est bien passée, rediriger vers l'alias `initConfigSuccess`.

⇨ Méthodes getters / setters dans `ConfigGeneraleBean`

*Principe de la délégation : ConfigGeneraleBean doit déclarer la plupart des méthodes contenues dans ConfigGenerale, en changeant éventuellement le comportement de certaines méthodes (rajout des spécificités JSF).*
*Dans ce cas, il n'est pas souhaitable d'utiliser l'héritage. On lui préfère la délégation, mécanisme beaucoup moins contraignant. ConfigGeneraleBean doit stocker une instance de ConfigGenerale, et y faire appel à chaque fois que c'est nécessaire.*

*Il existe un assistant Eclipse pour la délégation : clic droit > Source > Generate DelegateMethods.*

- D'après l'impression d'écran ci-avant, écrire (ou générer) les méthodes getter/setter appropriées dans la classe `ConfigGeneraleBean`.

⇨ Page `configGenerale.jsp`

- D'après l'impression d'écran ci-avant, créer et implémenter la page `configGenerale.jsp`. Les boutons de soumission de formulaire ne sont pas actifs pour le moment.
- Dans `pagePrincipale.jsp`, rajouter un lien « Configuration générale ».

⇨ Impacter le fichier `faces-config.xml`.

 Tester l'affichage de la page de configuration générale.

### 5.b) Soumission du formulaire

⇨ Dans `ConfigGeneraleBean`, créer la méthode `update()`. Cette méthode sera appelée lors de la soumission du formulaire dans `configGenerale.jsp`.

- Dans la couche service , effectuer l'appel approprié pour mettre à jour la configuration générale.

⇨ En cas de succès, on retournera vers `pagePrincipale.jsp`. Impacter le fichier `faces-config.xml`.

 Tester la mise à jour.

### 5.c) Validation d'un email

⇨ créer une classe `EmailValidator` qui implémente l'interface `Validator`.

⇨ Dans le fichier `faces-config.xml`, déclarer un nom logique pour ce validateur.

⇨ En s'inspirant des exemples fournis dans le chapitre « Validation et conversion », utiliser ce validator dans `configGenerale.jsp` (ne pas oublier de déclarer un tag `h:message` pour l'affichage d'un éventuel message d'erreur de validation).

 Tester.

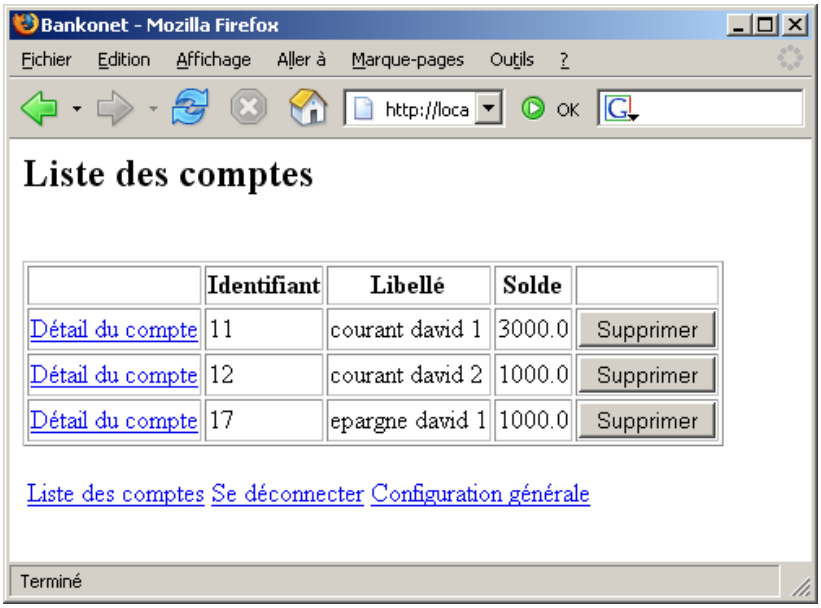
# TP 6- Barre de navigation

**Objectifs :**

Savoir mettre en place une barre de navigation avec JSF. Connaître le tag `f:subview`.

*Exemple d'inclusion de page JSP :*

```
<jsp:include page="/pageInc.jsp" flush="true"/>
```




## 6.a) Page `navigation.jsp`

⇨ Créer la page `navigation.jsp`.

- Cette page sera incluse par les autres pages. Elle ne doit pas contenir de balises englobantes de type <html> … </html>.
- Dans `navigation.jsp`, déclarer un tag englobant `<f:subview>`. Il doit contenir les liens qui étaient dans la page `pagePrincipale.jsp`. Ces liens doivent être disposés horizontalement (cf. ci-dessus).
- Positionner l'inclusion dans la page `compteListe.jsp`.

Tester.