

Leçon 9 – Sécurité

Voilà un vaste sujet qui couvre de nombreux aspects; nous n'en aborderons que quelques uns.

1 SÉCURISER LE CHEMIN DE NAVIGATION

Si l'utilisateur suit le chemin de navigation que vous avez planifié tout va bien. Mais que se passe-t'il si il essaie de s'en détourner ? Examinons deux cas récurrents.

Le problème des bookmarks

Par exemple que se passe-t'il si il revient sur une page bookmarquée lors d'une navigation précédente ?

En fait, ce problème ne se pose pas par défaut avec JSF puisque l'adresse dans la barre du navigateur ne change pas. En effet, la « front servlet » gérée par JSF s'arrange pour rediriger vers une page JSP sans changer l'adresse vue par le navigateur. Dès lors, l'utilisateur qui bookmarque la page reviendra sur la page d'accueil lors d'une prochaine visite.

Toutefois, ce comportement n'est pas toujours celui qui est souhaité. Il est parfois utile, pour le confort de l'utilisateur, de permettre de bookmarquer une page. On l'indique à JSF via la balise « <redirect/> » ajoutée à une règle de navigation. Lorsque JSF applique cette règle, plutôt que d'envoyer directement la page au navigateur, il lui envoie une demande de redirection qui conduit le navigateur à initier une nouvelle demande pour la nouvelle page. L'adresse dans la barre du navigateur est ainsi actualisée.

Dans un pareil cas, il faut bien veiller à ce que toute l'information soit disponible lors d'un accès direct à cette page.

Le problème d'un accès direct à la page

Même si l'adresse d'une page n'apparaît pas dans la barre d'adresse du navigateur, rien n'empêche à l'utilisateur de la taper directement s'il la connaît. Comment l'en empêcher ?

La réponse a déjà été donnée en deuxième année, il suffit de mettre dans page dans le dossier « WEB-INF ». Malheureusement, il y a un petit problème d'incompatibilité avec la technologie JSF : cela n'est pas possible pour une page contenant une balise « form ». Il existe toutefois un truc pour y arriver tout de même. Nous vous renvoyons au très bon tutoriel « <http://www.coreservlets.com/JSF-Tutorial/> » et plus particulièrement à la section 2 (contrôle de la navigation)

2 AUTHENTIFICATION ET RÔLE

Les techniques précédentes supposent un site ouvert à tous, sans restriction ni distinction. Dans la plupart des cas réels, on doit pouvoir identifier la personne qui visite un site mais aussi lui présenter un site différent en fonction de son rôle (client, fournisseur, employé, gestionnaire du site, ...). Voyons comment mettre cela en oeuvre.

Un peu de terminologie

Java EE utilise les termes suivants

- **User principal** : identifiant de la personne (login)
- **Rôle** : identifie la catégorie à laquelle appartient le visiteur. Un « User principal » peut appartenir un rôle et un rôle peut, bien sûr, être partagé par plusieurs « User principal ».
- **realm** : mode d'authentification d'un utilisateur. Cela peut se faire via
 - un fichier de mots de passe sur le serveur d'application
 - un serveur LDAP
 - un serveur d'authentification de Windows
 - ...

Définir les rôles

Avant toute chose, il faut définir qui pourra jouer quel rôle. Cela se fait en deux phases.

1. Nommer les différents rôles dans le fichier « web.xml » du module « Application d'Entreprise » (Netbeans offre une interface facilitant l'introduction de la liste des rôles)

```
<security-role>
  <role-name>Client</role-name>
</security-role>
<security-role>
  <role-name>Employe</role-name>
</security-role>
```

2. Associer des utilisateurs ou des groupes à des rôles dans le fichier « sun-web.xml » (pour le serveur d'application de Sun). La balise « group-name » fait référence à un groupe défini au moment de l'authentification.

```
<security-role-mapping>
  <role-name>Client</role-name>
  <group-name>user</group-name>
</security-role-mapping>
<security-role-mapping>
  <role-name>Admin</role-name>
  <principal-name>bob</group-name>
</security-role-mapping>
```

Restreindre l'accès à des pages

Le point précédent ayant été réalisé, il est possible de restreindre l'accès à une partie du site à tout qui possède le bon rôle (via des balises dans le fichier « web.xml ») (à nouveau, Netbeans offre un interface facilitant cette tâche)

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>
      <url-pattern>/faces/client/*</url-pattern>
    </web-resource-name>
  </web-resource-collection>
  <auth-constraint>
    <role-name>Client</role-name>
    <role-name>Admin</role-name>
  </auth-constraint>
</security-constraint>
```

Dans cet exemple, toutes les pages commençant par « /faces/client/ » ne peuvent être demandées que par une personne ayant le rôle « Client » ou « Admin »

S'authentifier

Que se passe-t'il lorsqu'une personne demande une page à l'accès restreint ? Si il a le bon rôle, la page lui est montrée. Dans le cas contraire (ou si elle ne s'est pas encore authentifiée), une procédure d'authentification lui est proposée à l'issue de laquelle la page sera montrée ou refusée. Tout cela se fait automatiquement et est contrôlé (une fois encore) via le fichier « web.xml ».

Quels choix avons-nous ? Beaucoup (cf. l'interface proposée par Netbeans) ! Retenons deux possibilités : la boîte de dialogue et une page dédiée à l'authentification. Ce deuxième cas demande un peu d'explications. Il faut fournir

- Une page dédiée contenant le code suivant (ou une variante respectant les noms des champs de saisie et de l'action)

```
<form method="POST" action="j_security_check">
  identifiant : <input type="text" name="j_username"/>
  Mot de passe : <input type="password" name="j_password"/>
  <input type="submit" value="Login"/>
</form>
```

- Une page montrée en cas d'erreur

S'authentifier via le serveur d'application

Pour demander au serveur d'application (de Sun) de prendre en charge l'authentification des visiteurs, il faut éditer un fichier sur le serveur.

1. Se connecter à la console d'administration du serveur d'application (via Netbeans par exemple)
2. Suivre « configuration » / « security » / « realm » et enfin « file »
3. Choisir « Manage Users » puis « Add User »

Tests dans les beans

Du code Java dans les beans peut effectuer des tests sur l'utilisateur via des méthodes du contexte

- FacesContext.getCurrentInstance().getExternalContext().getRemoteUser()
- FacesContext.getCurrentInstance().getExternalContext().isCallerInRole(String roleName);

3 SÉCURITÉ AU NIVEAU DU CONTENEUR EJB

Ce dont nous avons parlé jusqu'à présent concernait le conteneur Web. Cela ne joue donc pas pour un client lourd ascendant à notre application. Il est dès lors important de gérer également la sécurité à ce niveau. Heureusement, une grosse partie du travail peut être récupérée, à savoir la définition des rôles et leur lien avec une méthode d'authentification.

A partir de là, des annotations permettent de restreindre des beans (entités ou sessions) à des rôles particuliers. La contrainte peut même être mise sur des méthodes particulières de ces beans.

- **@RolesAllowed** : indique les rôles qui peuvent utiliser ce bean ou cette classe
- **@PermitAll** : tout le monde peut utiliser ce bean ou cette classe

```
@Stateless
@RolesAllowed("superuser")
public class AdminServiceBean implements AdminService {
    public void adminTask() {
        System.out.println("Admin method called");
    }

    @RolesAllowed("user")
    public void sharedTask() {
        System.out.println("Shared admin/user method called");
    }

    @PermitAll
    public void safeTask() {
        System.out.println("PermitAll method called");
    }
}
```

4 EXERCICE

Modifiez votre application de vente pour qu'une personne doivent s'identifier pour pouvoir passer une commande.

Pour vos recherches, nous vous conseillons de vous rendre sur le site suivant qui est clair et complet : <http://www.coreservlets.com/JSF-Tutorial/>