TP nº 9

Consignes les exercices ou questions marqués d'un ⋆ devront être d'abord rédigés sur papier (afin de se préparer aux épreuves écrites du partiel et de l'examen). En particulier, il est recommandé d'être dans les même conditions qu'en examen : pas de document autres que le cours ni de calculatrice. Tous les TPs se font sous Linux.

1 Préambule

Le but TP est le suivant :
- dernière chance de configurer Eclipse pour un intégrer un serveur Tomcat
- écrire une application Web plus complexe selon le modèle MVC : utilisation de JSP+JSTL pour la vue, JDBC pour le modèle et HttpServlet pour le contrôleur, utilisation des filtres pour l'authentification et sortie JSON.
- **il est interdit de renommer les classes et packages Java**
- **il ne faut pas importer le projet Eclipse du TP avant d'avoir correctement configuré Eclipse !**

2 Configuration d'Eclipse

Si vous n'avez pas correctement configuré Eclipse, suivez les instructions du TP 7.

3 Authentification reloaded

Ce TP fait suite au TP8 et contient donc son corrigé :
- Classe utilitaire `Pair` publique avec des *getters* pour accéder aux attributs.
- Classe `MovieDB` publique qui contient le modèle et possède une méthode `getMovieList(int from, in to)` renvoyant la liste des films ordonnés, entre les positions `from` et `to` sous la forme d'un `Vector<Pair<Integer, String>>`.
- Class `MovieListServlet` qui contient le code du contrôleur, récupérant les paramètres de requête, appelant le modèle et stockant les résultats dans l'objet `request` pour que la vue les affiche
- Fichier `movie_list.jsp` contenant la vue. Par rapport au TP8, la vue est déplacée dans `WEB-INF/jsp/`. Le fichier n'est donc plus accessible directement depuis une requête HTTP, il doit être la cible d'une redirection interne faite dans le contrôleur.

Le but du TP est d'ajouter à ce code un mécanisme d'authentification modulaire : on ne doit pas modifier les fichiers existants pour assurer l'authentification. On va pour cela utiliser des filtres. Les fichiers ajoutés sont :
- `login.jsp` : un simple fichier contenant un formulaire de saisie de login et mot de passe dont la cible est `CheckServlet`.
- `CheckServlet` : un contrôleur vérifiant le login et le mot de passe.
- `LoginFilter` : un filtre appelé pour chaque requête HTTP. Si la cible de la requête HTTP est un fichier censé être protégé, le filtre vérifie dans la session que l'utilisateur est authentifié, sinon il redirige vers `login.jsp`
- `LogoutServlet` : un contrôleur cible du bouton `déconnexion` se trouvant dans `movie_list.jsp`. Il détruit la session et redirige vers la page de login.
- `MovieWSServlet` : un service Web pour l'exercice 4.

Questions

1. Ajouter un formulaire au fichier `login.jsp` permettant de saisir des paramètres `login` et `pass` et accédant à la page `CheckServlet` via la méthode `post`.
2. Ajouter dans `CheckServlet` du code qui vérifie que le `login` et le `pass` sont corrects (en les comparant à des constantes). S'ils sont corrects, `CheckServlet` effectue une redirection HTTP vers `MovieListServlet` sinon on effectue une redirection vers `login.jsp`. De plus en cas d'authentification réussie, une variable de session `"status"` valant `"connected"` est stockée dans la session.
3. ⋆ On souhaite maintenant que chaque page teste automatiquement que `status` vaut `"connected"`. Implémenter cela au moyen d'un `Filter` (classe `LoginFilter`). Si un accès est invalide, alors on doit être redirigé vers la page `login.jsp`. On prendra un soins particulier à la manière de rediriger la requête selon les cas.

> **Réponse:** On utilise la garde `urlPatterns` de `LoginFilter` pour forcer *toutes* les requêtes à passer par le filtre. On écrit ensuite du code dans le filtre pour ignorer certaines URLs pour lesquelles l'authentification n'est pas nécessaire (exemple `login.jsp`, `CheckServlet`, ...). En cas d'accès authorisé, on utilise l'objet `chainFilter` pour passer la main à l'élément suivant dans la chaîne de filtre. En cas d'accès non authorisé, on utilise une redirection HTTP pour forcer le client à rechercher la page de login. **Attention**, si le filtre n'est pas implémenté correctement, même la page `login.jsp` sera bloquée (i.e. une page vide s'affiche car rien n'est renvoyé au client dans le corps de la réponse).

4. ⋆ On suppose qu'un utilisateur navigue sur `login.jsp` sans aucun *cookie* (première visite) et se connecte avec succès. Donner exactement la liste des actions internes au serveur Web et externes pour arriver sur la liste des films (requêtes HTTP effectuées, méthodes exécutées, pages chargées etc...).

> **Réponse:**
>
> (a) Accès interne à `LoginFilter`, comme la cible est `login.jsp` on laisse passer la requête.
>
> (b) Accès externe à `login.jsp` . La cible du formulaire est `CheckServlet`.
>
> (c) Accès interne à `LoginFilter`, comme la cible est `CheckServlet` on laisse passer la requête.
>
> (d) Accès externe à `CheckServlet` (méthode `doPost`). Validation du login/mot de passe et mise à jour des variables de sessions. Redirection HTTP (externe) vers `MovieListServlet`
>
> (e) Accès interne à `LoginFilter`, comme la cible est `MovieListServlet` on vérifie la variable de session `status` (qui est correctement placée par `CheckServlet` à l'étape précédente). On charge l'élément suivant dans la chaîne de filtre.
>
> (f) Accès externe à `MovieListServlet` (méthode `doGet`).

5. ⋆ Que peut on dire sur l'exécution de `LoginFilter` ? Modifier le code pour retirer les appels superflus.
6. Ajouter à la page `movie_list.jsp` un bouton de déconnection. Ce dernier accédera à `LogoutServlet` qui redirigera vers `login.jsp` après avoir détruit la session.

4 Web Service

On souhaite implémenter un Web Service permettant de lister les ids de films, puis pour un film donné, son titre. Attention, on fera en sorte que les Servlets de cet exercices ne requière pas d'être authentifié (soit en désactivant le filtre de l'exercice précédant, soit en lui faisant ignorer les servlets créés ici).

— Créer un servlet `MovieWSServlet` similaire qui prend deux paramètres `from=`i et `to=`j en GET et qui renvoie un fichier JSON de la forme :

```
{  "status" : "success",
   "size" : n,
   "ids" : [ "1234", ...,  "4849" ]
}
```

où `n` est la taille du tableau `ids`, ce dernier contenant les mid des films triés par titre entre les positions i et j.

— Étendre le servlet pour qu'il soit associé aux URLs de la forme : `MovieWSServlet/mid1234` et qui affiche en réponse :

```
{  "status" : "success",
   "mid" : "1234",
   "title" : "Le titre du film"
 }
```

— En cas d'erreur (film inexistant, exception) le JSON suivant est renvoyé :

```
{  "status" : "error",
   "message" : "message d'erreur"
 }
```