Documentation

1) Le Pool de connexion

Le pool de connexion permet une connexion à la base de données plus propre. Celui-ci viens allouer un certain nombre de connexions à l'application qu'il créer dès le lancement du serveur. Ainsi, à chaque requête sur la base, l'application ne créer pas de connexion mais viens se saisir d'une connexion « libre ».

Il doit être configuré de cette manière :
(dans le fichier /Nom_du_Context/META-INF/context.xml) :

```
<Resource name="da2i"
auth="Container"
type="javax.sql.DataSource"
maxTotal="8"
maxIdle="4"
maxWaitMillis="3000"
username="toto"
password="moi"
driverClassName="org.postgresql.Driver"
url="jdbc:postgresql://psqlserv/da2i" />
```

Avec cela on peut alors configurer le temps avant la libération d'une connexions, le nombre maximal de connexion, la source des données ou encore le compte avec lequel la connexion est effectuée.

Et son accès se fait ainsi :

```
Context initCtx = new InitialContext();
Context envCtx = (Context) initCtx.lookup("java:comp/env");
Datasource ds = (DataSource) envCtx.lookup("da2i");
Connection con = ds.getConnection();
```

** Tous les paragraphes de codes fournis sont, bien sûr, à configurer selon l'utilisateur.*

2) Le Realm

Le Realm est un système d'authentification par rôle qui permet une restriction d'accès à des groupes de pages pour les utilisateurs ne s'étant pas connecté ou n'ayant pas les droits.

Il doit être configuré de cette manière :
(dans le fichier /Nom_du_Context/META-INF/context.xml) :

Si le pool de connexion a était implémenter :

```
<Realm className="org.apache.catalina.realm.DataSourceRealm"
debug="99"
dataSourceName="da2i"
localDataSource="true"
userTable="users"
userNameCol="user_name"
userCredCol="user_pass"
userRoleTable="comptes"
roleNameCol="role_name" />
```

Si le pool de connexion n'a pas était implémenter :

```
<Realm className="org.apache.catalina.realm.JDBCRealm"
connectionName="mathieu"
connectionPassword="philippe"
connectionURL="jdbc:sqlite:/tmp/mabase.db"
driverName="org.sqlite.JDBC"
roleNameCol="role_name"
userCredCol="user_pass"
userNameCol="user_name"
userRoleTable="user_roles"
userTable="users"/>
```

(dans le fichier /Nom_du_Context/WEB-INF/web.xml) :

```
<security-constraint>
        <web-resource-collection>
                <web-resource-name>SecuredBookSite</web-resource-name>
                <url-pattern>/classic/*</url-pattern>
        </web-resource-collection>
        <auth-constraint>
                <role-name>admin</role-name>
        </auth-constraint>
</security-constraint>
```

** Tous les paragraphes de codes fournis sont, bien sûr, à configurer selon l'utilisateur.*

```
<security-role>
	<role-name>client</role-name>
</security-role>
<security-role>
	<role-name>admin</role-name>
</security-role>
```

Si une page de connexion n'a pas était créée :

```
<login-config>
	<auth-method>BASIC</auth-method>
	<realm-name>Nom affiché sur le formulaire</realm-name>
</login-config>
```

Si une page de connexion a était créée :

```
<login-config>
	<auth-method>FORM</auth-method>
	<realm-name>Some product</realm-name>
	<form-login-config>
		<form-login-page>connexion.html</form-login-page>
		<form-error-page>connexion_fail.html</form-error-page>
	</form-login-config>
</login-config>
```

Attention dans la page d web de connexion :

```
<form action="j_security_check">
	<input type="text" name="j_username"/>
	<input type="password" name="j_password"/>
	<input type="submit" value="LOGIN" />
</form>
```

** Tous les paragraphes de codes fournis sont, bien sûr, à configurer selon l'utilisateur.*

3) Les requêtes AJAX (JQUERY)

Les requêtes Ajax permette d'interroger une ressource de votre context de façon asynchrone (sans rafraîchir votre page). Pour effectuer cela, il est utile d'avoir de vague notion de Javascript.

Il doit être configuré de cette manière :

dans votre page web :

On importe la libraire Ajax et notre script Javascript :

```
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
<script src="main.js"></script>
```

dans votre fichier main.js :

```
$.ajax({
	type: 'POST/GET/PUT/DELETE',
	url: 'url_ressource',
	data: 'nom_param1=' + param1 + '&nom_param2=' + param2,
	error: function(response) {},
	success: function(response) {}
});
```

** Tous les paragraphes de codes fournis sont, bien sûr, à configurer selon l'utilisateur.*

4) JSTL

Les JSTL expressions sont des balises qui permettent les tags de structures (itération et conditionnement), Internalisation, exécution de requêtes SQL ou encore l'utilisation de documents XML.

Il doit être configuré de cette manière :

dans votre répertoire de librairie insérer jstl-1.2.jar à télécharger sur :

http://www.java2s.com/Code/Jar/j/Downloadjstl12jar.htm

dans votre page web :

On importe la librairie et on lui donne un préfixe.

```
<%@ taglib uri = "http://java.sun.com/jsp/jstl/core" prefix = "c" %>
```

On peux ensuite écrire :

```
<c:choose>
	<c:when test=" boolean_random == true ">
		code html
	</c:when>
	<c:otherwise>
		code html
	</c:otherwise>
</c:choose>
```

ou encore :

```
<c:forEach var="entry" items="${requestScope['myErrorList']}">
	<p>Coucou</p><br/>
</c:forEach>
```

** Tous les paragraphes de codes fournis sont, bien sûr, à configurer selon l'utilisateur.*