*Développez des sites web avec Jave EE*

**Chapitre 1 : Premiers pas avec Java**

1. **Qu’est-ce que Java EE ?**

Bien qu’on dise « J-2-E » à l’oral, l’écriture correcte reste « Java EE ».

JEE est en fait un ensemble de bibliothèques que l’on va rajouter à Java pour le développement de notre projet. Très utilisé en entreprise, notamment par les banques, voir même pour les sites gouvernementaux, c’est un outil de confiance qui a acquis une grande maturité professionnelle.

JEE est aujourd’hui la plateforme de référence pour développer des sites web bien structurés, solides et robustes. Son fonctionnement est basé sur celui des applications web avec notions de clients / serveur :

1. L’utilisateur envoie une requête http au serveur d’application
2. Le serveur d’application la lit et l’envoie au conteneur
3. Le conteneur se charge d’exécuter le code JEE
4. Une réponse http est renvoyée au client par le serveur d’application

On peut finalement comparer le conteneur JEE au moteur PHP. Nous utiliserons dans ce cours « Apache Tomcat » comme serveur d’application, outil gratuit et open source.

1. **Le modèle MVC :**

Comme la plupart des langages informatiques, JEE n’impose aucune structure particulière à notre code. Cependant une bonne pratique de développement ressort dans tous les langages (ou design pattern) et consiste à organiser son code selon le modèle MVC (pour Model Vue Controller) :

1. La requête http de l’utilisateur envoyée au controller
2. Si besoin est, des appels aux modèles (qui contiennent les informations) sont effectués
3. La vue est générée et renvoyée au visiteur

En JEE chacun de ces composantes ont en fait un nom :

* Les Servlets correspondent aux controllers
* Les objets Java (Java Beans) correspondent aux modèles, accompagnés par les bases de données
* Les pages JSP correspondent aux vues (code HTML + code spécifique Java)

**/!\ Attention :** JEE n’est PAS un Framework Java, mais une « spécialisation de [ce] langage » tout comme PHP spécialise le langage HTML. De nombreux frameworks existent comme Spring, JSP, ou encore Struts.

Pour crée un nouveau projet avec Eclipse il faut choisir l’option « New Dynamic Web Project ». L’arborescence d’un projet se compose comme suit :

* Le dossier **« src »** contiendra le code metier, c’est-à-dire les classes Java
* Le dossier **« webcontent »** contient tous les fichiers CSS, HTML, JSP qui formeront le front de notre application
* Le dossier **« WEB-INF »** contient des fichiers de paramètres
* Le dossier **« lib »** contient les librairies externes importées au projet

**Chapitre 2 : Comprendre les Servlets et les JSPs**

1. **Créer un Servlet :**

Le Servlet correspond au contrôleur dans le modèle MVC. Ce sont de simples classes Java qui, par le biais de méthodes (« doGet », « doPost », etc…), reçoivent la requête du visiteur, la traitent en générant une page web et renvoient le tout dans une réponse HTTP. Une servlet n’est finalement rien de plus qu’une classe Java qui étend, donc hérite, de la classe Java « HttpServlet ».

Le fichier web.xml est incontournable en développement JEE. Il est à ranger dans le dossier « WEB-INF ».

1. **Associer une vue à une Servlet :**

Pour associer une vue à une Servlet il faut s’intéresser d’un peu plus près à la méthode « doGet » de la- dite Servlet. Celle-ci est formée par deux paramètres, « request » et « response », le premier objet contient la requête quand la réponse (HTML, JSP, img, etc…) est envoyée via l’objet « response ».

C’est là que la technologie JSP (pour Java Server Pages) entre en jeu ! L’intérêt consiste à mélanger du code Java et du code HTML dans certains fichiers spécifiques. Ils sont eux aussi rangés dans le dossier « WEB-INF », seul présent à la racine du serveur à son lancement.

1. **Communiquer des données avec JSP :**

Lors du développement d’un site web certaines parties restent toujours les mêmes (en-tête, menu, etc…). Avec JEE nous allons centraliser ce code dans des pages JSP spécifiques toutes enregistrées à un seul endroit (un peu comme les inclusions en PHP).

En JEE on peut récupérer des informations de manières dynamiques comme un paramètre d’URL par exemple, avec les méthodes « getAttribute() » et « setAttribute() » depuis la Servlet.

1. **Utiliser une expression langage dans les JSP :**

Il est en réalité une très mauvaise pratique de mélanger du code HTML et du code Java au sein d’un même fichier. Cela peut vite devenir brouillon voire carrément trop compliqué à maintenir. Comme TWIG pour PHP, il est possible d’utiliser en JEE ce qu’on appelle l’Expression Langage (ou EL) qui permet d’insérer des variables ou des conditions de manière plus lisible, directement à l’intérieur des pages JSP : **« ${expression} ».**

1. **Manipuler des Java Beans dans les JSPs :**

Java est un langage orienté objet dont la partie modèle est assurée par de simples classes publiques Java, dont les attributs sont eux privés avec des méthodes publiques pour y accéder. Ce sont les « Java Beans ».

Il est possible avec Eclipse de générer automatiquement les getters et setters depuis une liste d’attributs : **« Clique droit 🡪 Source 🡪 Generate getters and setters ».Chapitre 3 : Comprendre les Servlets et les JSPs**

1. **Qu’est-ce que la JSTL ?**

La JSTL (pour JavaServer Standard Tag Library) est une bibliothèque à rajouter à un projet pour faire du Java dans les pages JSP, mais sous forme xml. Elle se compose en réalité de 5 sous bibliothèques dont la « Core » qui gère les variables, conditions, boucle, etc…

1. **Mettre en place la JSTL :**

Pour utiliser la JSTL il faut déjà télécharger la bibliothèque, la copier dans le dossier « lib » et inclure en début de fichier : **« <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>** » afin de pointer vers cette dernière.

Pour ce genre de directives dont on risque d’avoir besoin dans tous les fichiers, le plus simple est de l’écrire dans un fichier à part et de l’importer automatiquement grâce au fichier de config’ « web.xml ».

1. **JSTL et variable :**

Utiliser JSTL permet de s’affranchir des failles XSS car elle échappe directement les caractères spéciaux XML/HTML et les affiche directement à l’écran sans les traiter. L’expression langage est compatible avec la JSTL.

Avec la JSTL il est possible de :

* Afficher des variables ou une valeur par defaut : **« <c:out value=’${var}’ default=’val’ /> »**
* Créer ou modifier des variables : **« <c:set var=’varName’ value=’varValue’ scope=’varScope’ /> »**
* Modifier des Java Beans : **« <c:set target=’beanSend’ property=’targetProperty’ value=’newValue’ /> »**
* Supprimer des variables : **« <c:remove var=’varName’ scope=’varScope’ /> »**

La propriété « scope » peut prendre 4 attributs :

* Page (accessible uniquement depuis la page)
* Request (accessible depuis la page et celles qui l’appellent ou qu’elle appelle)
* Session (accessible tout au long de la session de l’utilisateur)
* Application (accessible de partout et par tous les utilisateurs)

**/!\ Attention :** Il est pour l’instant obligatoire de fermer avec « / » les balises orphelines en JSTL.

1. **JSTL et conditions :**

Il est possible d’utiliser les conditions avec la JSTL : **« <c:if test=’’${condition}’’ var=’’varName’’> code… </c> »** L’attribut « var », contrairement à celui « test », est optionnel. Il permet d’enregistrer le résultat du test dans une variable accessible dans la suite du code. Par défaut le scope de cette variable est celui de la page mais on peut le modifier avec l’attribut « scope » de la même manière que pour les balises vues précédemment.

Le « else if » n’existe pas en JSTL, pour ça il faut utiliser la balise « <c:choose> » de la manière suivante :

<c:choose>

<c:when test=’’${condition 1}’’> code… </c:when>

<c:when test=’’${condition 2}’’> code… </c:when>

<c:when test=’’${condition 3}’’> code… </c:when>

<c:otherwise> code… </c:otherwise >

</c:choose>

1. **JSTL et boucles :**

En plus des conditions, la JSTL prend bien évidement en compte plusieurs sortes de boucles.

Les boucles « for », ainsi que les tableaux et listes, sont gérés avec « <c:forEach> » :

* <c:forEach i=’’0’’ begin=’’0’’ end=’’10’’ step=’’1’’> (l’attribut « i » est optionnel)
* <c:forEach items=’’varName’’ var=”xName”> (« begin » et « end » sont aussi disponibles)
* Il est possible d’utiliser aussi l’attribut “varStatus” qui va créer une variable contenant plusieurs informations sur la boucle (tour de boucle, index, element courant, longueur, etc…)

Il existe une autre boucle, la boucle « forTokens » qui est uniquement dédiée aux chaines de caractères :

* <c:forToken items=’’chaineADecoupe’’ morceau=’’moreceauAct’’ delims=’’delimiteur’’> code… </c:forToken>

**Chapitre 4 : Développer une application web**

1. **Envoyer des fichiers :**

On ne récupère pas les paramètres d’un formulaire de la même manière que pour récupérer un fichier, qui lui nécessite de prendre en compte la norme http.

En JEE les formulaires qui contiennent un champs d’upload de fichier doivent obligatoirement avoir l’attribut suivant dans leur balise <form> : « enctype=’’multipart/form-data’’ ». Il faut ensuite configurer ledit attribut directement dans le fichier « web.xml ».

Pour récupérer un fichier il faut utiliser la fonction « request.getPart(‘’fileFieldName’’) ».

1. **Gérer les sessions :**

Les sessions servent à suivre un utilisateur jusqu’à la fin de sa visite de notre site web. Ses informations seront stockées dans des variables de sessions qui restent valables d’une page à l’autre du serveur (au contraire des autres variables).

* Pour ce faire il faut d’abord créer un objet de la classe « HttpSession » dans la servlet
* Pour accéder aux variable de session depuis une jsp on utilise « sessionScope.varName »
* Pour terminer une session on utilise « session.invalidate() »

1. **Gérer les cookies :**

Le cookie est une petite information que nous stockons sur l’ordinateur de notre visiteur pour, par exemple, lui éviter d’avoir à retaper son pseudo à chaque fois qu’il veut se connecter à notre site.

* Pour créer un cookie depuis la servlet il faut s’appuyer sur l’objet response :

« response.addCookie(‘’cookieName’’, value) ; »

* Des parametres peuvent être indiqués à la création du cookie pour, par exemple, déterminer sa durée de vie :

« cookie.setMaxAge(60) » *Le cookie expirera dans 60 secondes*

* Pour récuperer tous les cookies liés à notre site et présent sur le PC de notre visiteur il faut, depuis la servlet, utilisé la méthode : « request.getCookies’) ; »

**Chapitre 5 : Enregistrer dans une base de données**

1. **Travailler avec JDBC et une base de données :**

Un site web sans base de données n’est pas très utile. Une bibliothèque JEE existe, nommée « JBDC », et permet de se connecter à n’importe quel type de base de données (MySQL, Oracle, etc…).

Pour que cela fonctionne il faut bien évidemment installer MySQL mais aussi le driver JDBC (procédure dans le cours).

1. **Lire et enregistrer des données en SQL :**

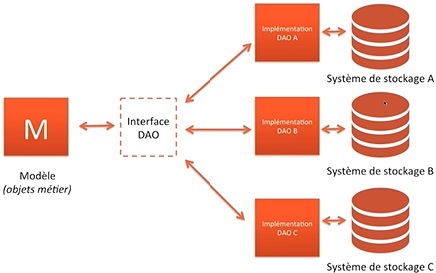
On peut lire et écrire des informations dans la base de données en utilisant directement des requêtes SQL à l’intérieur de notre code Java. En effet, c’est JDBC qui se chargera de faire le lien entre notre application et la base de données.

Pour rappel, les objets Java prennent le rôle des modèles (partie métier) et la servlet prend celui du contrôleur (partie aiguillage) pour enfin envoyer toutes les données à la JSP, qui elle représente donc la vue (partie utilisateur).

1. **Utiliser le modèle DAO :**

En général, il n’est pas bienvenu de retrouver du SQL directement dans les vues ou les contrôleurs. Il est préférable de passer par une interface DAO (pour Data Access Object).

Dans une logique MVC les données de la BDD sont gérées par les modèles. En réalité il ne devrait même pas y avoir de SQL dans les objets Java (qui prennent le rôle de models). L’idée est de séparer au maximum le code SQL qui se retrouvera isolé dans des objets à part, dans lesquels les autres fichiers viendront piocher par le biais de fonctions. Il peut y avoir plusieurs DAO sur un seul projet et dans ce cas-là il faut passer par une interface DAO qui elle fera le lien avec les différents objets d’accès aux données, ou systèmes de stockages :



Une interface en Java est une classe qui définit des noms de méthodes sans les implémenter, car elles le seront par ses classes filles qui viendront surcharger lesdites méthodes dans leur propre classe. Cela permet par exemple de facilement changer de système de stockage sans retoucher à tout le code.