**JAVA/JEE**

**Les piliers d’une application**

**Présentation**

* Prise en charge des jeux d'affichage (affichage HTML en cas du web).

**Métier**

* Appeler aussi couche applicative, étape où l'exécution des applications se déroule.
* Gérer les requêtes SQL et les résultats.

**Persistance**

* Propre au serveur de BD : Gestion de la correspondance entre le serveur métier avec celui de la base de données.

**Niveaux des piliers d’une application**

1. **Modèle centralisé** : Tout est sur la même machine
2. **Deux niveaux (2-tiers) :**

Client : Présentation / Interface utilisateur.

Serveur : Partie persistance / Gestion physique des données.

On a 3 possibilités pour cette architecture :

* Soit entièrement côté client, intégrés avec la présentation (la partie serveur ne gère que les données).
* Soit entièrement côté serveur.
* Soit découpés entre la partie serveur et la partie client

1. **Trois niveaux (3-tiers)**

Les trois tiers s’exécutent chacun sur une machine différente :

* Présentation (Machine client).
* Applicatif/Métier (Serveur d’application).
* Persistance (Serveur de base de données).

1. **N niveaux (N-tiers)**

Par rapport à 3-tiers : ajoute des couches supplémentaires ;

Précisément la couche métier n'est pas atomique (monolithique), elle est composée d'un ensemble de services ;

**Interaction horizontale :**

Les services (application) métiers peut interagir avec d'autres services métiers (composition de services) ;

**Interaction verticale ;**

Répertoire (LDAP) ;

Messagerie ;

Sécurité

Alors chaque service correspond à 1 service == N niveaux

**JEE et Le N-Tiers**

**Présentation (Client-Side) :**

* Avec client léger(Html) ou client lourd(API).

**Interaction avec la partie applicative sur le serveur (Server-Side) :**

* Via JSP / Servlet pour un client léger :
* S'occupe de la logique de présentation.
* Direct si client lourd (via un middleware type RMI).

**Logique applicative (Business Logic) :**

* Réalisée par composants EJB ;
* Communication via Hibernate ou JDBC.

**JEE : SERVLET & JSP**

**OBJECTIF : PAGES HTML DYNAMIQUES**

**Génération du code HTML à partir du serveur**

* Parfois accompagné du style de la page ;
* Visant un client léger : Navigateur Web ;
* Envoi et récupération des données à partir du navigateur ;

**Procédures :**

* Exécuter un code différent sur une plateforme afin de produire du code HTML ;
* L'affichage obéira à plusieurs critères :
* Selon le type de données ;
* Selon le rôle de l'utilisateur ;

**LES SERVLETS**

Servlet est un programme Java avec des caractéristiques particulières :

* Accessible par une URL donnée ;
* Détecter et répondre à des requêtes de l'utilisateur ;
* Chercher et invoquer le traitement correspondant ;
* Générer une page Html et y mettre la réponse ;

**Applet** : programme Java s'exécutant au niveau client ;

**Servlet** : Programme Java s'exécutant au niveau serveur ;

* C’est une classe Java avec ces méthodes et attributs ;

**HTTPSERVLETREQUEST REQUEST**

Initialisé par l'appel du client (le navigateur)

Données / informations sur la requête envoyée par le client ;

On peut y récupérer notamment :

* Valeurs entrées pour un formulaire ;
* Une session pour gérer un état dédié à chaque client ;
* Cookies du client envoyés avec la requête ;
* Informations sur l'URL utilisée pour l'appel de la Servlet ;
* Login de l'utilisateur s'il s'est identifié.

**HTTPSERVLETRESPONSE RESPONSE**

A initialiser et utiliser par la Servlet pour générer le résultat à envoyer au navigateur client ;

Définir le type MIME des données envoyées (Généralement du code HTML) ;

Récupérer le flux de sortie pour envoyer les données :

* Objet de type PrintWriter avec la méthode getWriter() Flux texte, typiquement pour du HTML
* Objet de type ServletOutputStream avec la méthode getOutputStream () : Flux binaire pour des images, vidéos ...

**SERVLET : GESTION DES SESSIONS**

* Champ caché : Les serveur peut inclure des informations cachées dans un formulaire afin d'identifier la navigation du client :

<input type="hidden" name="sessionid" value="2013">

* Le nom et la valeur de l'entrée sont automatiquement inclus dans la méthode post ou get ;
* Inconvénient : Le cas du href le serveur ne pourra pas poursuivre la navigation.
* **HttpSession** : L'API Servlet fournit une interface HttpSession qui permet de garder trace de l'utilisateur à travers plusieurs pages dans le même site ;
* Le conteneur de Servlet utilise cette interface pour créer une session entre un client et serveur http.
* La session persiste durant une période dans le temps (à définir par le développeur), à travers plusieurs pages et durant plusieurs connexions.

**SERVLETS : GESTION DES COOKIES**

Un **cookie** : fichier texte stocké côté client enregistrant toutes les interactions avec le serveur ;

* Les informations d'un cookie aident le serveur à s'adapter à un client particulier ;
* C'est le serveur qui crée le cookie dans le client. Lors d'une connexion le client envoie le cookie au serveur.

Plusieurs caractéristiques :

* Les informations de navigation ;
* Les données saisies par le client dans un formulaires ;
* Durée de vie à définir par le serveur…

Utiliser les fonctions de l ’API des servlets…

* Créer un cookie : classe Cookie ;
* Ecrire/lire un cookie : addCookie(mycookie), getCookies() ;
* Positionner des attributs d’un cookie : mycookie.setXxx(…);

**WEB.XML ET LES SERVLETS**

Une Servlet est toujours déclarée entre <servlet> et </servlet>.

On doit affecter un nom à chaque Servlet déclarée, dans une balise <servlet-name>.

Ce nom n'est pas nécessairement le nom de la classe de la servlet.

La balise <description> permet de fournir une information sur la Servlet ;

La balise <servlet-class> permet de définir la classe de la Servlet. La classe doit se trouver dans le répertoire WEB-INF/classes de l'application, en respectant les packages.

La balises <init-param> permet de spécifier des paramètres d'initialisation pour une servlet.

Des paramètres chargés en même temps que la servlet, et qu'elle peut récupérer.

La balise <param-name> permet de définir le nom du paramètre, et <param-value> pour spécifier la valeur.

L'application peut récupérer un tel paramètre grâce à la méthode getServletConfig().getInitParameter("random").

La balise <load-on-startup> demande que la servlet soit chargée dès le démarrage du serveur (et non lors de sa première sollicitation). Le nombre entier situé à l'intérieur de ces balises représente l'ordre de chargement.

Le **mapping d'une Servlet** sert à indiquer au serveur quelle Servlet charger pour tel requête du client (telle URL demandée). Rappelons que les URL des servlets sont relatives à l'URL du context (la web app) auquel elles appartiennent.

**JSP :**

**Comment ça marche ?**

•Le développeur a la possibilité d'insérer du code java dans le fichier Html à l'aide des balises prédéfinies ;

•Le moteur des JSP (Serveur d'app) reconnait ces balises, puis procède à leur compilation et leur exécution ;

•Le serveur traite les balises html et java différemment afin de créer une Servlet équivalente à la JSP, l'exécution de cette dernière génère le code Html équivalent au résultat.

**EJB et Java Bean**

Un Bean, c'est un objet qui ne fait généralement pas grand-chose , il est avant tout là pour stocker des données, donc on aura des attributs privés, un constructeur sans argument et/ou un constructeur avec tous les arguments, des get/set/is public pour accéder aux attributs, sans oublier hash Code/equals/toString et c'est tout.

Les EJB ce sont eux qui font le travail, ce sont de grosses classes qui vont vraiment faire la partie opérationnelle de l'application. Il y a plusieurs types d'EJB selon s'ils sont font la relation entre l'application et les données (EJB Entité), les clients (EJB Session), ou d'autres applications (EJB message).

**DAO :**

En gros divise tout en bloc, qui communiquent entre eux uniquement ce qu'ils veulent. La couche d'accès aux données DAO sait comment est fait ta base de données, mais la seule chose qu'elle fait c'est prendre des Beans en paramètres, faire une requête, et donner des Beans en sortie.   
La couche Business elle va faire le lien entre plusieurs DAO, et manipuler les Bean pour donner un tout cohérent.  La couche présentation, elle ne fait que présenter les données, elle ne les calcule pas.

Si dans ta page web tu veux savoir quel temps il fait, la couche présentation va demander au métier de lui renvoyer quel temps il fait. Pour cela la couche métier récupérer l'information, soit dans son cache s’il a déjà l'information, soit auprès d'une DAO qui pourrait être une base de données, ou un autre site web, etc.

**Struts :**

Apache Struts est un [framework](https://fr.wikipedia.org/wiki/Framework) [libre](https://fr.wikipedia.org/wiki/Logiciel_libre) servant au développement d'[applications web](https://fr.wikipedia.org/wiki/Application_web) [Java EE](https://fr.wikipedia.org/wiki/Java_EE). Il utilise et étend l'[API](https://fr.wikipedia.org/wiki/Interface_de_programmation) [Servlet Java](https://fr.wikipedia.org/wiki/Servlet) afin d'encourager les développeurs à adopter l'architecture [Modèle-Vue-Contrôleur](https://fr.wikipedia.org/wiki/Mod%C3%A8le-Vue-Contr%C3%B4leur)(MVC).

Struts 2 regroupe les avantages de deux précédents outils, WebWork et Struts 1, mais c'est une refonte complète.; Cette seconde génération de [framework](https://fr.wikipedia.org/wiki/Framework) MVC ([Modèle-vue-contrôleur](https://fr.wikipedia.org/wiki/Mod%C3%A8le-vue-contr%C3%B4leur)) utilise les notions suivantes : intercepteurs, annotations, langage d'expression [OGNL](https://fr.wikipedia.org/w/index.php?title=OGNL&action=edit&redlink=1) , l'intégration d'outils comme [JSTL](https://fr.wikipedia.org/wiki/JSTL) ([Java Server Pages Standard Tag Library](https://fr.wikipedia.org/wiki/JavaServer_Pages_Standard_Tag_Library)) ou [Spring framework](https://fr.wikipedia.org/wiki/Spring_framework).

Cette infrastructure permet la conception et l'implémentation d'applications Web de taille importante par différents groupes de personnes. En d'autres termes, les designers, développeurs de composants logiciels peuvent gérer leur propre part du projet de manière découplée.

Struts permet la structuration d'une application Java sous forme d'un ensemble d'actions représentant des événements déclenchés par les utilisateurs de l'application. Ces actions sont décrites dans un fichier de configuration de type [XML](https://fr.wikipedia.org/wiki/Extensible_Markup_Language) décrivant les cheminements possibles entre les différentes actions. En plus de cela, Struts permet d'automatiser la gestion de certains aspects comme la validation des données entrées par les utilisateurs via l'interface de l'application. Plus besoin de venir coder le contrôle de chaque donnée fournie par un utilisateur, il suffit de décrire les vérifications à effectuer dans un fichier XML dédié à cette tâche.

En utilisant Struts, le développeur simplifie son travail au niveau des vues et des contrôleurs du modèle MVC. Mais il serait inadapté d'utiliser ce framework dans des projets de petite taille car il introduit une certaine complexité. Struts montre toute sa puissance dans des applications d'une certaine envergure.