|  |  |  |
| --- | --- | --- |
| logo1 | Ministère de l’Enseignement Supérieur et de la recherche scientifique  Direction Générale des Etudes Technologiques  Institut Supérieur des Etudes Technologiques de Sfax  Direction des Etudes et des Stages  Département Technologies de l’Informatique |  |

**Compte rendu JEE**

|  |  |
| --- | --- |
| **Elaboré par  :** | **Charfi Nour**  **Ben Ayedi Malek** |
| **Groupe :** | **DSI 22** |

**Encadré par :** **Mr Nissen Massmoudi**

Année Universitaire : 2022/2023

**Sommaire**

[Chapitre 1 : Préparer l’infrastructure et les outils de développement 6](#_Toc133534805)

[I. L’environnement de développement : 7](#_Toc133534806)

[A. Téléchargement des outils : 7](#_Toc133534807)

[B. Installation des outils 7](#_Toc133534808)

[II. Préparer le logiciel du travail : 8](#_Toc133534809)

[III. Intégration de Tomcat dans Eclipse 8](#_Toc133534810)

[Chapitre2 : Première application web 12](#_Toc133534811)

[I. JEE (Java Entreprise Edition) : 12](#_Toc133534812)

[A. Applications Web : 12](#_Toc133534813)

[B. L’arborescence : 12](#_Toc133534814)

[II. Création du projet web dynamique : 13](#_Toc133534815)

[III. Réalisation du projet : 15](#_Toc133534816)

[A. Créer une page web statique : 15](#_Toc133534817)

[B. Création d’une Servlet et comment réaliser le mapping URL / Servlet : 15](#_Toc133534818)

[C. Appeler un fichier html à partir d’une Servlet : 17](#_Toc133534819)

[D. Passage de paramètres en mode « GET » : 17](#_Toc133534820)

[E. Passage des paramètres d’un formulaire : 18](#_Toc133534821)

[IV. Conclusion : 19](#_Toc133534822)

[Chapitre 3 : LES JAVA SERVER PAGES (JSP) 20](#_Toc133534823)

[I. JSP : Java Server Pages 20](#_Toc133534824)

[II. Création de la première page JSP 20](#_Toc133534825)

[III. Le passage des attributs 21](#_Toc133534826)

[IV. Portée des variables (scope) 23](#_Toc133534827)

[V. Mode connecté 24](#_Toc133534828)

[VI. Conclusion : 25](#_Toc133534829)

[Chapitre 4 : GESTION DE JAVA BEAN 26](#_Toc133534830)

[I. Programmation JEE/JSP 26](#_Toc133534831)

[II. Creation du premiere bean : 26](#_Toc133534832)

[III. La portée des objets 27](#_Toc133534833)

[IV. Conclusion : 31](#_Toc133534834)

[Chapitre 5 : Application Web JEE avec modèle MVC et base de données 32](#_Toc133534835)

[I. Gérer le support physique de stockage des données 32](#_Toc133534836)

[II. Gérer la partie « Metier »: 33](#_Toc133534837)

[III. Gérer la partie « web » 33](#_Toc133534838)

[VI. Conclusion : 38](#_Toc133534839)

[Annexe 39](#_Toc133534840)

**Liste des figures**

[Figure 1 ajout chemin du jdk8 8](file:///C:\Users\Dell\Downloads\CompteRenduJee%20(1).docx#_Toc133534841)

[Figure 2 variable d'environnement 8](file:///C:\Users\Dell\Downloads\CompteRenduJee%20(1).docx#_Toc133534842)

[Figure 3:Nouveau serveur 9](#_Toc133534843)

[Figure 4:Démarrage du serveur 10](#_Toc133534844)

[Figure 5 Erreur rencontré lors de l'exécution 11](file:///C:\Users\Dell\Downloads\CompteRenduJee%20(1).docx#_Toc133534845)

[Figure 6 changement numéro de port 11](file:///C:\Users\Dell\Downloads\CompteRenduJee%20(1).docx#_Toc133534846)

[Figure 7:Création projet web dynamique 13](#_Toc133534847)

[Figure 8:Nommer le projet 14](#_Toc133534848)

[Figure 9:L’arborescence du projet 15](#_Toc133534849)

[Figure 10:Page web statique 15](file:///C:\Users\Dell\Downloads\CompteRenduJee%20(1).docx#_Toc133534850)

[Figure 11:Page web dynamique 15](file:///C:\Users\Dell\Downloads\CompteRenduJee%20(1).docx#_Toc133534851)

[Figure 12:Test1 url-pattern 16](#_Toc133534852)

[Figure 13:Test2 url-pattern 16](#_Toc133534853)

[Figure 14::Exécution du servlet 16](file:///C:\Users\Dell\Downloads\CompteRenduJee%20(1).docx#_Toc133534854)

[Figure 15:Exécution de la servlet HelloWorld 17](#_Toc133534855)

[Figure 16:Appeler un fichier html à partir d’une Servlet 17](file:///C:\Users\Dell\Downloads\CompteRenduJee%20(1).docx#_Toc133534856)

[Figure 17:Exécution servlet qui appelle un fichier html 17](file:///C:\Users\Dell\Downloads\CompteRenduJee%20(1).docx#_Toc133534857)

[Figure 18:Exécution servlet HelloAction 18](file:///C:\Users\Dell\Downloads\CompteRenduJee%20(1).docx#_Toc133534858)

[Figure 19:Exécution lien.html 18](file:///C:\Users\Dell\Downloads\CompteRenduJee%20(1).docx#_Toc133534859)

[Figure 20:Passage des paramètres d'un formulaire avec affichage dans la barre d’adresse 18](#_Toc133534860)

[Figure 21:Passage des paramètres d'un formulaire sécurisées 18](file:///C:\Users\Dell\Downloads\CompteRenduJee%20(1).docx#_Toc133534861)

[Figure 22:Passage des paramètres d'un formulaire sécurisées 18](file:///C:\Users\Dell\Downloads\CompteRenduJee%20(1).docx#_Toc133534862)

[Figure 23:1 ére page jsp 20](#_Toc133534863)

[Figure 24:aspect dynamique"Date" 20](#_Toc133534864)

[Figure 25:Page connexion.jsp 21](#_Toc133534865)

[Figure 26:Un seul message d’erreur “le dernier message écrase les autre” 21](#_Toc133534866)

[Figure 27:Affichage erreur par erreur 22](#_Toc133534867)

[Figure 28:Erreurs specifique des attributs 22](#_Toc133534868)

[Figure 29:Page affichageEtudiant2.jsp 23](#_Toc133534869)

[Figure 30:Execution1 HelloWorldScope 23](#_Toc133534870)

[Figure 31:Execution2 HelloWorldScope dans le même navigateur 23](#_Toc133534871)

[Figure 32:Execution3 HelloWorldScope un autre navigateur 24](#_Toc133534872)

[Figure 33:Lien de deconnexion 25](#_Toc133534873)

[Figure 34:Code bean Etudiant 26](#_Toc133534874)

[Figure 35:Premier modification de la méthode « doGet » de la servlet « EtudiantManager » 27](#_Toc133534875)

[Figure 36::code page « affichageEtudiant.jsp» 27](#_Toc133534876)

[Figure 37:Exécution 1 de la servlet « EtudiantManager » 27](#_Toc133534877)

[Figure 38:Exécution 2 de la servlet « EtudiantManager » 28](#_Toc133534878)

[Figure 39:Exécution de la page « affichageEtudiant.jsp» dans un autre onglet du navigateur web 28](#_Toc133534879)

[Figure 40:Exécution de la page « affichageEtudiant.jsp» dans un autre navigateur web 29](#_Toc133534880)

[Figure 41:Code de la page affichageEtudiant2.jsp 29](#_Toc133534881)

[Figure 42:Exécution de la page affichageEtudiant2.jsp 30](#_Toc133534882)

[Figure 43:Ajout du code dans la page affichageEtudiant2.jsp 30](#_Toc133534883)

[Figure 44:Exécution 3 de de la servlet « EtudiantManager » dansun autre onglet du navigateur web 30](#_Toc133534884)

[Figure 45:Exécution de la page « affichageEtudiant2.jsp» dans un autre onglet du navigateur web 31](#_Toc133534885)

[Figure 46:Exécution de la page « affichageEtudiant2.jsp» aprés l'ajout du code à la fin du corps 31](#_Toc133534886)

[Figure 47:la base de données 32](#_Toc133534887)

[Figure 48:le résultat dans la console et au niveau de la base de données. 33](#_Toc133534888)

[Figure 49: la formulaire de connexion de la page "UserConnexion.jsp" 33](#_Toc133534889)

[Figure 50:Exécution de la page "accueil.jsp" 34](#_Toc133534890)

[Figure 51:Exécution de la page "UserForm.jsp" 35](#_Toc133534891)

[Figure 52:le remplissage de le formulaire avec les informations demandées 35](#_Toc133534892)

[Figure 53:tableau où un lien pour éditer chaque utilisateur est disponible 36](#_Toc133534893)

[Figure 54: tableau où un lien pour éditer chaque utilisateur est disponible 37](#_Toc133534894)

[Figure 55: La confirmation de la déconnexion 37](#_Toc133534895)

[Figure 56: la redirect vers la page "accueil.jsp" la deconnexion 38](#_Toc133534896)

[Figure 57:L'architecture MVC avec servlets, JSP et MySQL 38](#_Toc133534897)

[Figure 58:relation entre JSP et Servlet 43](#_Toc133534898)

**Liste des Tableaux**

[Tableau 1: Table de comparaison entre Servlet et JSP 35](#_Toc133179309)

[Tableau 2:Les fonctionnalités des collections 37](file:///C:\Users\Dell\Downloads\CompteRenduJeeTp1%20(1).docx#_Toc133179310)

[Tableau 3: Collection :un comportement et des fonctionnalités particulières 38](file:///C:\Users\Dell\Downloads\CompteRenduJeeTp1%20(1).docx#_Toc133179311)

# Chapitre 1 : Préparer l’infrastructure et les outils de développement

# L’environnement de développement :

L’objectif de ce chapitre est la mise en place de l’environnement de développement pour réaliser les développements d’applications Java EE pour le Web.

Pour cela, vous aurez au moins besoin :

du kit Java de développement

d’un environnement de développement : par exemple Eclipse

d’un conteneur de Servlet Java EE

## Téléchargement des outils :

La liste des outils requis est :

* **Java 11 JDK**

Le kit de développement Java 11

<https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads.html>**u JDK ?**

Il existe deux distributions de Java par Oracle : JRE (Java Runtime Edition) et le JDK (Java development kit). Le JRE permet uniquement l’exécution des programmes Java. Le JDK fournit en plus un compilateur et des outils utiles aux développeurs. Il est donc nécessaire d’installer le JDK pour développer des applications en Java mais également pour exécuter le serveur TomEE.

* **Eclipse IDE for Enterprise Java Developers**

L’environnement de développement

**https://www.eclipse.org/downloads/download.php?file=/oomph/epp/2019-12/R/eclipse-inst-win64.exe**

Il existe plusieurs distributions d’Eclipse. Il est impératif d’installer **la version pour le développement Java EE**. Si vous avez déjà installé Eclipse, vérifiez dans le menu « Help > About Eclipse » que votre version est bien « Eclipse Java EE IDE for Web Developers » ou « Eclipse IDE for Enterprise Java Developers ».

* **Tomcat**

Le conteneur de servlet Tomcat

[**https://tomcat.apache.org/download-90.cgi**](https://tomcat.apache.org/download-90.cgi)

Pour Windows, téléchargez le fichier **zip** dans la section **Core**. Pour MacOS ou Linux, téléchargez le fichier **tar.gz** dans la section **Core**.

Nous verrons bientôt que les applications Java pour le Web ne sont pas des programmes indépendants (standalone). Les applications Java EE sont déployées et exécutées dans ce que l’on appelle un serveur d’application qui fournit l’environnement d’exécution nécessaire. Tomcat n’est pas à proprement parlé un serveur d’application car il n’offre qu’un nombre de services limités par rapport à un serveur d’application normal. Néanmoins, il est très souvent utilisé en production par les entreprise pour réaliser des application Web en Java.

## Installation des outils

L’installation du JDK dépend de votre plate-forme : il est distribué sous la forme d’un installeur pour Windows et MacOS, et sous la forme d’un package ou d’une archive sous Linux.

Eclipse et Tomcat sont distribués sous la forme d’une archive (tar.gz pour Linux et MacOS et zip pour Windows) que vous pouvez décompresser où vous le souhaitez.

# Préparer le logiciel du travail :

Tout d’abord on a installé la version EDI « eclipse » (eclipse 64 Version 2019-12 R).

Notre version d’éclipse nécessite l’installation du jdk8 qui est en même temps entraine l’ajout le chemin d'accès aux outils de développement Java, tels que le compilateur Java « javac » et l'interpréteur de bytecode Java « java » sur notre système d'exploitation.

Par la suite, on doit ajouter le chemin du jdk8 au début de la valeur de la variable d’environnement « PATH »

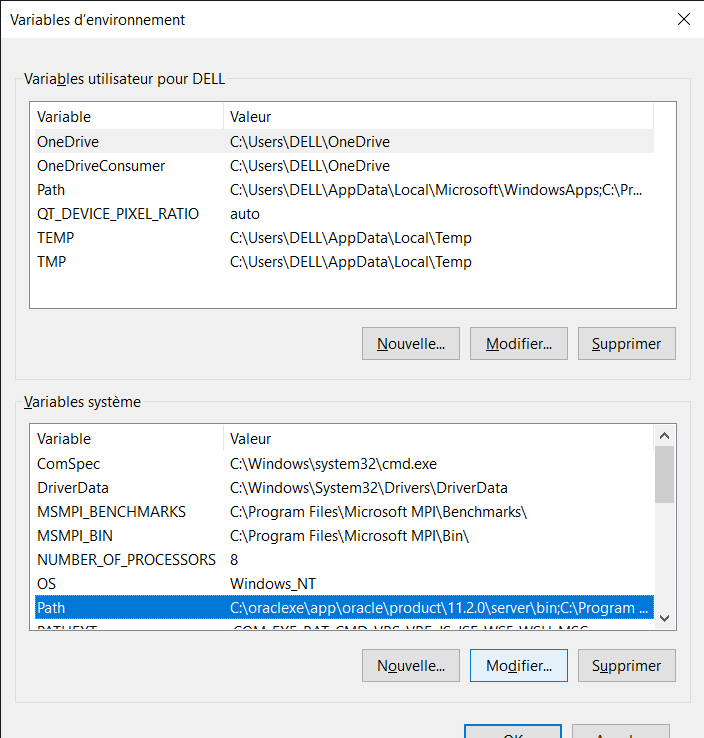
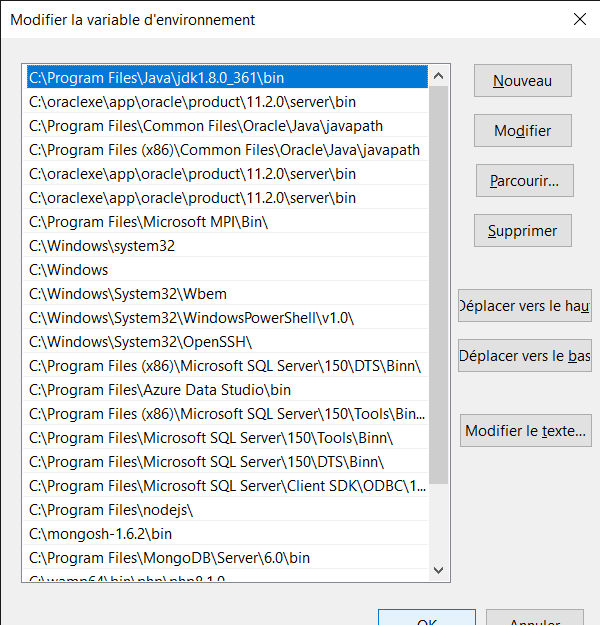


Figure ajout chemin du jdk8

Figure variable d'environnement

# Intégration de Tomcat dans Eclipse

Tomcat est un serveur Web qui peut être exécuté de manière autonome (comme un serveur de production par exemple). Cependant, pour simplifier les tâches de développement, nous allons l’intégrer dans Eclipse afin de pouvoir le démarrer et l’arrêter directement depuis l’IDE.

Dans Eclipse Java EE, il existe une vue pour créer et gérer les serveurs.

Pour ajouter cette vue, il faut passer par le menu **Window > Show View > Servers**.

Nous avons maintenant la vue « Servers » ajoutée à notre perspective de développement. Faites un click droit dans cette vue pour sélectionner **New > Server**.

Nous pouvons maintenant ajouter un serveur Tomcat 9 en sélectionnant **Apache > Tomcat v9.0 Server**.

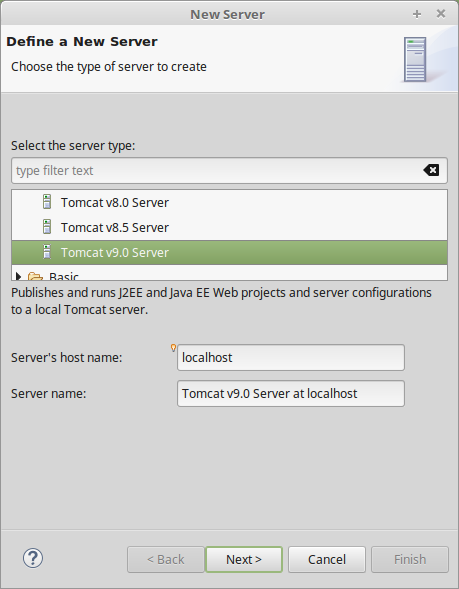


Figure :Nouveau serveur

Cliquer ensuite sur **Next**. Dans la boîte de dialogue **Tomcat Server**, il faut indiquer l’emplacement du serveur sur notre disque. Pour cela, cliquer sur **Browse…** pour sélectionner le répertoire d’installation de Tomcat sur votre disque.

Cliquer ensuite sur **Finish**.

Nous disposons maintenant d’un serveur dans notre vue Servers. Il ne reste plus qu’à le démarrer en faisant un click droit sur son nom et en sélectionnant **Start**.

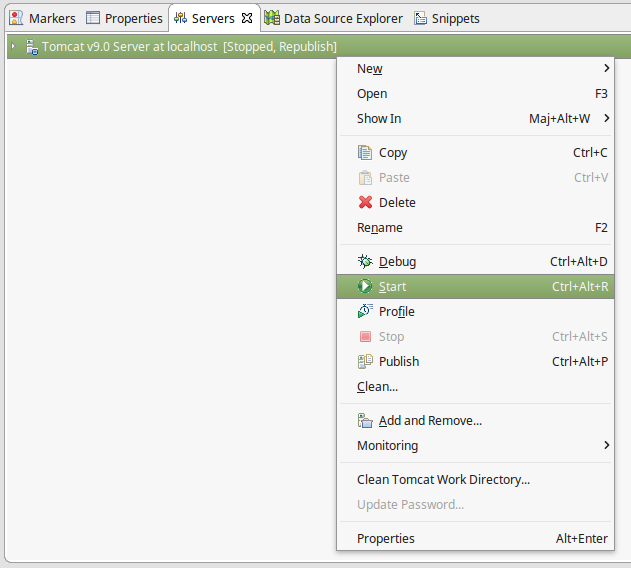


Figure :Démarrage du serveur

Si tout se passe bien, à la fin du lancement, nous pourrons accéder à notre serveur à l’adresse [http://localhost:8080](http://localhost:8080/). Nous devons voir s’afficher une page d’erreur avec le libellé :

État HTTP 404 – Not Found

En effet, le serveur ne contient encore aucune application Java Web à exécuter. Mais le serveur est néanmoins démarré.

Mais malheureusement lors de l’exécution de notre projet un message d’erreur sera affiché

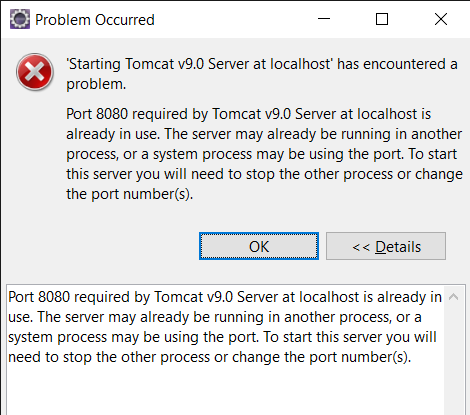


Figure Erreur rencontré lors de l'exécution

Ceci montre que le port 8080 est déjà réserver par un autre programme installé dans ma machine donc j’étais obligé de changer le numéro de port pour le tomcat qui devient 8081.



Figure changement numéro de port

# Chapitre2 : Première application web

# JEE (Java Entreprise Edition) :

JEE (*Java Entreprise Edition*) est la version entreprise de la plate-forme "Java" qui se compose de l'environnement "JSE" ainsi que de nombreuses API et composants destinés à une utilisation "côté serveur" au sein du système d'information de l'entreprise. Il s'agit donc d'une évolution du Java.

Cette section présente l'architecture Java 2 Plateform Enterprise Edition (J2EE) et explique comment vous pouvez l'utiliser pour déployer des applications Web. Si vous connaissez déjà cette architecture, vous pouvez ignorer cette section.

## Applications Web :

Une application Web J2EE est conçue pour être conforme à une spécification J2EE. Pour ajouter des composants Web à un conteneur de servlet J2EE, vous utilisez un package appelé fichier WAR (Web Application Archive). Un fichier WAR est un fichier compressé JAR (archive Java).

Outre les composants Web, il contient en général les ressources suivantes :

* Classes utilitaires côté serveur
* Ressources Web statiques (fichiers de configuration, pages HTML, fichiers image et son, etc.)
* Classes côté client (applets et classes utilitaires)

La structure de répertoires et de fichiers d'une application Web déployée en tant que fichier WAR doit répondre à des règles précises. Un fichier WAR a une structure de répertoire hiérarchique spécifique. Le répertoire supérieur d'un fichier WAR correspond au *répertoire document racine* de l'application.

## L’arborescence :

L’arborescence standardisée d’un fichier WAR :

* META-INF

Ce répertoire contient les meta-informations décrivant le livrable. Il contient en général au moins le fichier MANIFEST.MF qui donne des informations générales : nom du package, auteur, date de création… Ce répertoire est optionnel.

* WEB-INF

Ce répertoire correspond à la section privée de l’application. Il contient le code de l’application ainsi que les fichiers de configuration. Le contenu de ce répertoire n’est pas accessible depuis l’extérieur. Faites le test en tentant de visualiser le fichier web.xml à l’adresse <http://localhost:8080/monappli/WEB-INF/web.xml>. Vous obtenez un code erreur 404 (NOT FOUND). Le fichier est bien présent mais le serveur refuse de répondre à cette adresse !

\*WEB-INF/web.xml

LE fichier de déploiement de l’application Web ! Il fournit des informations au serveur d’application au moment du démarrage de l’application. L’application que nous avons développé était tellement simple que nous n’avons pas eu à nous en préoccuper pour l’instant.

\*WEB-INF/classes

Ce répertoire accueille les fichiers Java compilés (les fichiers .class). Comme nous n’avons pas écrit de code pour notre application, ce répertoire est vide.

\*WEB-INF/lib

Ce répertoire accueille les bibliothèques Java (les fichiers .jar) externes qui sont nécessaires à l’application. Nous verrons que le serveur d’application fournit déjà de nombreuses bibliothèques qu’il ne faut SURTOUT pas incorporer au livrable. Pour toutes les autres, c’est dans ce répertoire qu’elles doivent être placées.

# Création du projet web dynamique :

 A partir d’Eclipse, faire menu File → New → Dynamic Web Project

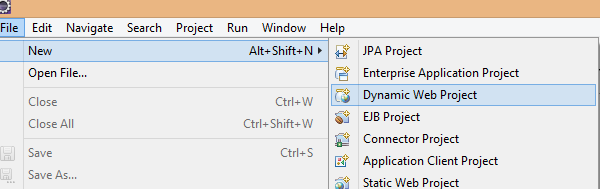


Figure :Création projet web dynamique

 L’écran suivant apparaît :



Figure :Nommer le projet

REMARQUE

Notez que le projet est associé à un serveur d’application, ici Tomcat 9. Cela est nécessaire. C’est une bonne pratique d’intégrer un serveur d’application Java dans Eclipse avant de créer un projet web dynamique.

 Donnez un nom au projet, ici ‘web\_app\_tp01

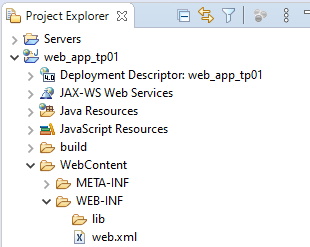
Cliquez sur le bouton Finish et le projet suivant apparaît :  


Figure :L’arborescence du projet

# Réalisation du projet :

## Créer une page web statique :

En premier lieu, on a créer une page web statique «Test.html » dans la répertoire **WebContent .**

Pour lancer l’exécution, cliquer par le bouton droit sur le fichier « **Test.html** » et lancer la commande « **Run As\Run On Server** ».

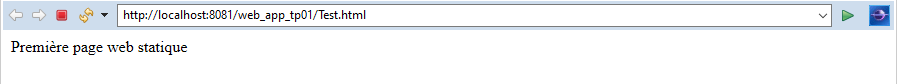
On aboutit cette résultat :

Figure :Page web statique

## Création d’une Servlet et comment réaliser le mapping URL / Servlet :

### Création d’une Servlet :

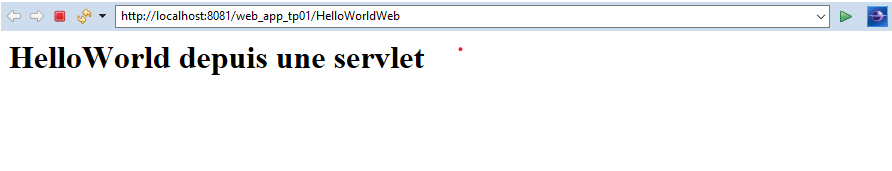
En second lieu, choisir la commande «**New\Package**» pour créer un package nommé «**test**» dans la repertoire « **src** » de notre projet.

Puis sélectionner le package «test» puis choisir la commande «New\Servlet » pour créer notre servlet .

Pour ajouter des fonctionnalités spécifiques à notre servlet, on a redéfinisser les méthodes doGet() et doPost().

Pour exécuter la servlet, cliquer par le bouton droit sur la servlet puis lancer la commande « **Run As\Run On Server** »

Figure :Page web dynamique



Il est possible d’invoquer le même résultat en choisissant un url-pattern « **\*.do** ». On a tester les urls suivantes :

**http://localhost:8081/web\_app\_tp01/test.do**

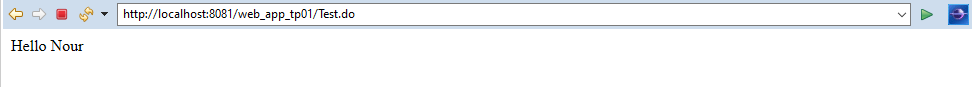


Figure :Test1 url-pattern

**http://localhost:8081/web\_app\_tp01/.do**

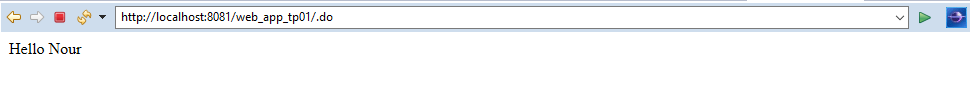


Figure :Test2 url-pattern

### Réaliser le mapping url/Servlet via le fichier web.xml :

On mettre l’instruction « **@WebServlet** » en commentaire dans le code de la servlet

et relancer l’exécution de la servlet.

On remarque que le serveur ne trouve la ressource et n’arrive pas à attribuer un nom logique à la servlet ,une erreur 404 a été déclenchée et affiche la réponse suivante :

Figure ::Exécution du servlet

Une autre manière pour réaliser le mapping url/servlet est d’utiliser le fichier « **web.xml** » (fichier descripteur de déploiement).

On exécute la servlet « HelloWorld » et on a remarquer qu’elle est référencée par l’url-pattern « HelloWeb » par l’adresse suivante: **http://localhost:8081/web\_app\_tp01/HelloWeb**

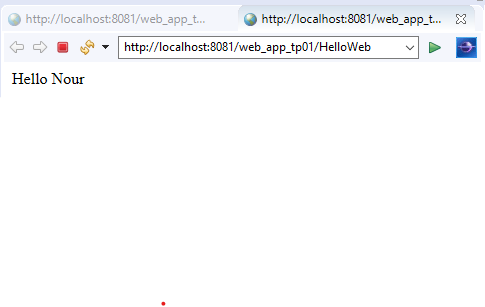


Figure :Exécution de la servlet HelloWorld

Donc le mapping URL/servlet permet de déterminer quelle servlet doit traiter une demande spécifique. L'annotation @WebServlet est une des façons de définir le mapping URL/servlet dans le code source de la servlet. Les « url-patterns » associées à la servlet courante indiquent les URL qui doivent être traitées par cette servlet.

Le fichier de configuration de l'application Web (web.xml) est une autre façon de définir le mapping URL/servlet. Dans ce fichier, vous pouvez définir les URL correspondant à chaque servlet en utilisant la balise <servlet> pour définir la servlet et la balise <servlet-mapping> pour définir le mapping URL/servlet.

## Appeler un fichier html à partir d’une Servlet :

Après avoir saisir le code HTML, on a essayé d’appeler ce code dans une servlet pour l’exécuter c’est pour quoi on a ajouté ce code indiqué dans cette figure dans la méthode « doGet » de notre servlet

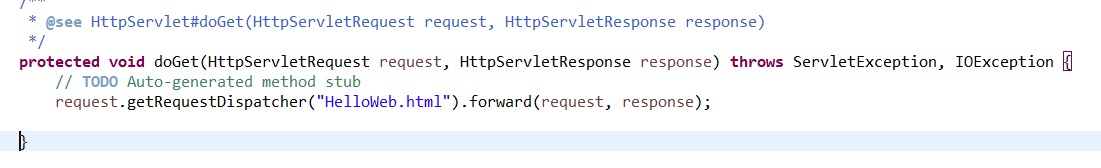


Figure :Appeler un fichier html à partir d’une Servlet

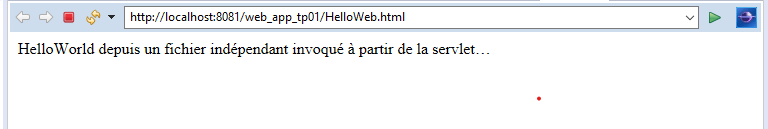
On obtient le résultat suivant :

Figure :Exécution servlet qui appelle un fichier html

La méthode getRequestDispatcher() retourne un objet de type RequestDispatcher qui peut transférer la requête en cours vers une autre ressource en utilisant la méthode forward()

L'appel de la méthode forward(request, response) transfère donc la requête et la réponse vers la ressource cible, en l'occurrence le fichier "HelloWeb.html". Ce fichier va ensuite traiter la requête et générer une réponse, qui sera envoyée au client.

## Passage de paramètres en mode « GET » :

On a créer une nouvelle servlet « **HelloAction** » pour récupérer et afficher la valeur d’un paramètre nommé « **prenom** » passé dans la requête HTTP .

Dans la barre d’adresse du volet d’exécution, saisir l’adresse suivante (passage d’un paramètre nommé «**prenom**» ayant la valeur «**Nour**») :

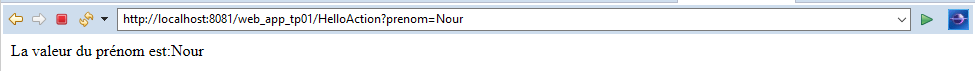
[**http://localhost:8081/web\_app\_tp01/HelloAction?prenom=Nour**](http://localhost:8081/web_app_tp01/HelloAction?prenom=Nour)

Figure :Exécution servlet HelloAction

Puis on a créer un nouveau fichier « **lien.html** » un lien hypertexte qui appelle la servlet «**HelloAction**» en passant le paramètre « **prenom** » avec la valeur « **Nour** ».

Il suffit d’ajouter la ligne suivante :

<a href =*"HelloAction?prenom=Nour"*> Afficher le prénom</a>

On lance l’exécution du fichier « **lien.html** » ,on obtient cette résultat :

Figure :Exécution lien.html

## Passage des paramètres d’un formulaire :

En dernier lieu, on a saisir le code HTML de notre formulaire qui contient deux zones de texte pour saisir un nom et un mot de passe aussi contient deux boutons « ok » et « annuler » La validation du formulaire appelle la servlet ayant comme url-pattern « ConnexionAction » en mode GET.

On ajouter du code dans la servlet « ConnexionAction », dans la méthode « doGet »qui permet de récupérer les paramètres envoyés à partir du formulaire et d’afficher leurs valeurs.

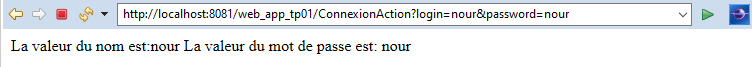


Figure :Passage des paramètres d'un formulaire avec affichage dans la barre d’adresse

Mais, on a remarqué que notre donnée s’affiche dans la barre d’adresse Ce qui rend les données non sécurisées.

Cherchant la sécurité de notre client, on a affecté la valeur « POST » à l’attribut « method » de la balise <form > dans le fichier HTML.

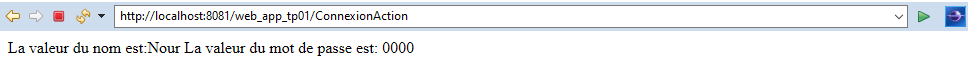
Et pour la servlet, on a changé le code de la méthode « doGet » à la méthode « doPost »

Figure :Passage des paramètres d'un formulaire sécurisées

Figure :Passage des paramètres d'un formulaire sécurisées

# Conclusion :

En résumé, la structure d'un projet Java Web dynamique est organisée en différentes parties qui contiennent des fichiers et des dossiers spécifiques à l'application. Cette structure standard facilite la gestion et le déploiement de l'application.

# **Chapitre 3 : LES JAVA SERVER PAGES (JSP)**

# JSP : Java Server Pages

Nous avons vu que les servlets permettent facilement d’exécuter du code Java pour traiter une requête HTTP. Cependant, l’API servlet n’est pas très pratique pour générer une réponse orientée texte (telle qu’une page HTML). Les Java Server Pages (JSP) ont été la première solution introduite pour offrir une alternative plus simple pour l’écriture de patron (template) de réponse.

# Création de la première page JSP

Tout d'abord on créer un nouveau projet web dynamique nommé

« web\_app\_tp02 » dans notre Workspace .

Puis on a sélectionner le dossier « WebContent » et choisir la commande « New/JSP File ».

en premier lieu , on ajouter un simple message d’aspect statique comme suit la figure

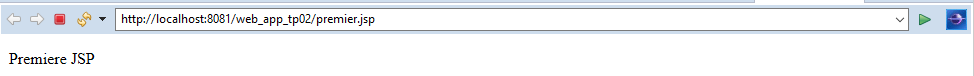


Figure :1 ére page jsp

en second lieu ,on a ajouter du code JAVA (aspect dynamique). Par exemple, on a afficher la date système du serveur.

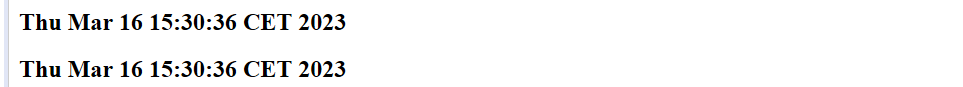


Figure :aspect dynamique"Date"

Les formulaires JSP et Servlet

On réer le formulaire suivant (nommé connexion.jsp )

On créer une feuille de style CSS nommée « style.css » située dans le sous-dossier « css » du « WebContent ».

Créer une servlet nommée « ConnexionAction.java » , dans un package « controller »

La page « connexion.jsp » est associé à style.css



Figure :Page connexion.jsp

# Le passage des attributs

Dans cette partie , il est possible de passer des données dans l'objet "HttpServletRequest" sous forme d'attributs définis par des couples ("key", "value"). La méthode "setAttribute(String key, Object object)" permet d'écrire un attribut avec un nom donné et une valeur souhaitée, tandis que la méthode "Object getAttribute(String key)" permet de lire la valeur d'un attribut à partir de son nom.

Donc ,on a modifier le code de la méthode « doPost() » dans notre servlet “ConnexionAction.java” et tout le code de la page « connexion.jsp ».

Aussi on a ajouter une nouvelle classe « erreur » dans le fichier «style.css».

On aboutit le résultats suivant :



Figure :Un seul message d’erreur “le dernier message écrase les autre”

On a remarquer qu’un seul message d’erreur est affiché (en effet, le dernier message écrase les autres).

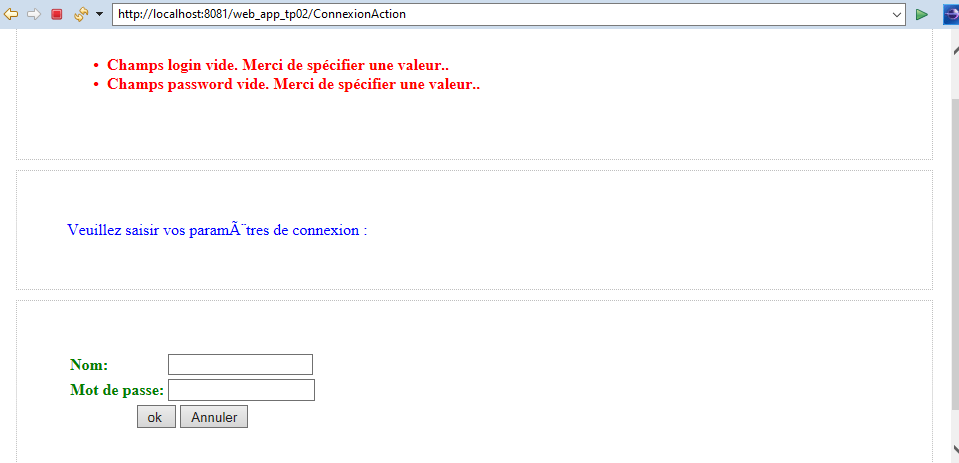


Figure :Affichage erreur par erreur

Pour cela on a créer une nouvelle page « accueil.jsp » et modifier une autre fois la méthode « doPost() » dans la servlet “ConnexionAction.java” et le code de la page « connexion.jsp ».

Tout dabord on a modifier le mode d’envoi des erreurs pour passer tous les messages d’erreur dans un objet « ArrayList ».

En cas d’existence d’erreurs, la servlet redirige appelle la vue « connexion.jsp » en envoyant les attributs nécessaires et on obtient cette résultats :

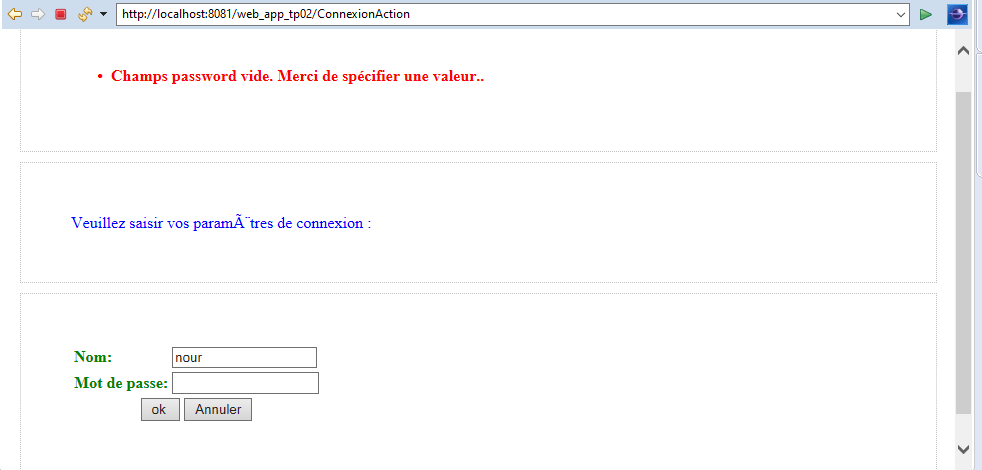


Figure :Erreurs specifique des attributs

Sinon, la servlet appelle la page d’accueil « accueil.jsp » pour afficher un message de BienVenue.

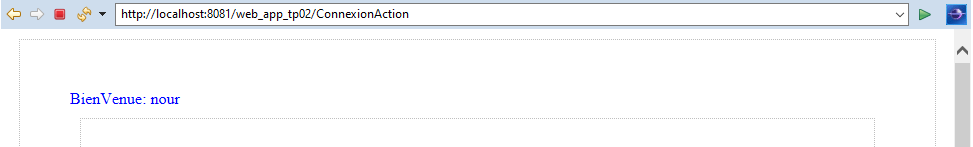


Figure :Page affichageEtudiant2.jsp

# Portée des variables (scope)

En premier lieu ,on a créer une nouvelle servlet « HelloWorldScope » et une nouvelle page « scope.jsp ».

En deuxieme lieu , on a Lancer:

* en premier lieu l’url : <http://localhost:8080/web_app_tp02/HelloWorldScope?prenom=Nour>

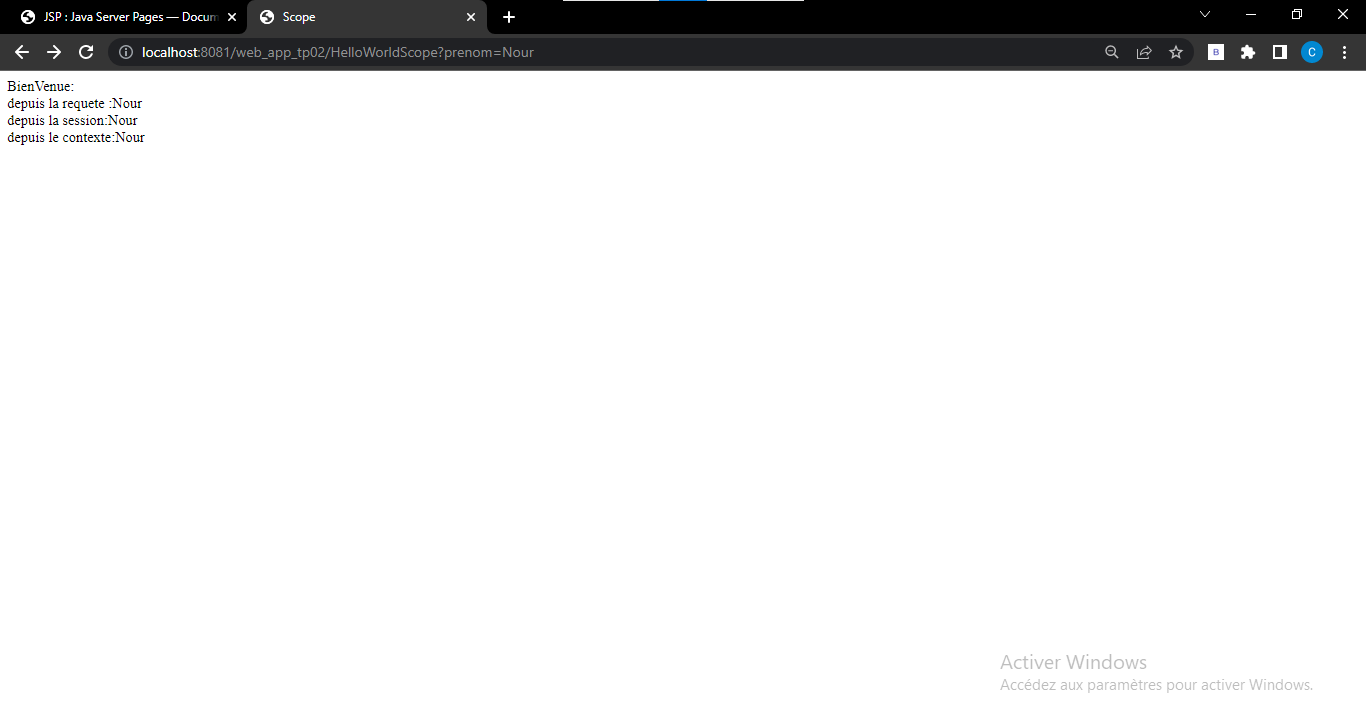


Figure :Execution1 HelloWorldScope

En lançant l'URL <http://localhost:8081/web_app_tp02/HelloWorldScope?prenom=Nour> , la servlet HelloWorldScope sera appelée avec le paramètre "prenom" ayant pour valeur "Nour". La servlet stockera cette valeur dans la session et le contexte de l'application, puis redirigera la requête vers la page JSP "scope.jsp".

* Puis Lancer sur le même navigateur l’url : <http://localhost:8081/web_app_tp02/HelloWorldScope>

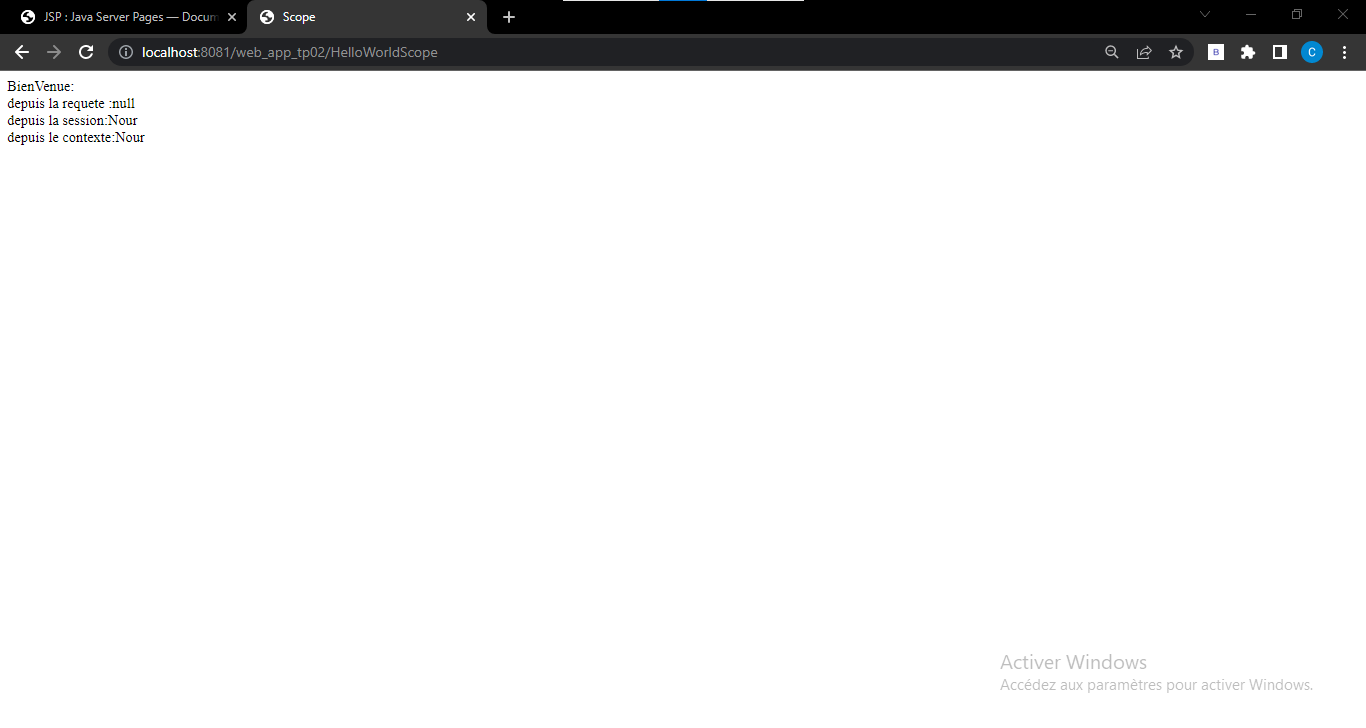


Figure :Execution2 HelloWorldScope dans le même navigateur

En lançant l'URL [http://localhost:8081/web\_app\_tp02/HelloWorldScope](http://localhost:8080/web_app_tp02/HelloWorldScope) sans paramètre, la servlet HelloWorldScope sera appelée sans la valeur du paramètre "prenom". La servlet ne stockera pas de nouvelle valeur dans la session et le contexte de l'application, mais la page JSP affichera la valeur précédemment stockée dans la session et le contexte, s'ils existent.

* Puis Lancer sur un autre navigateur l’url : [http://localhost:8081/web\_app\_tp02/HelloWorldScope](http://localhost:8080/web_app_tp02/HelloWorldScope)

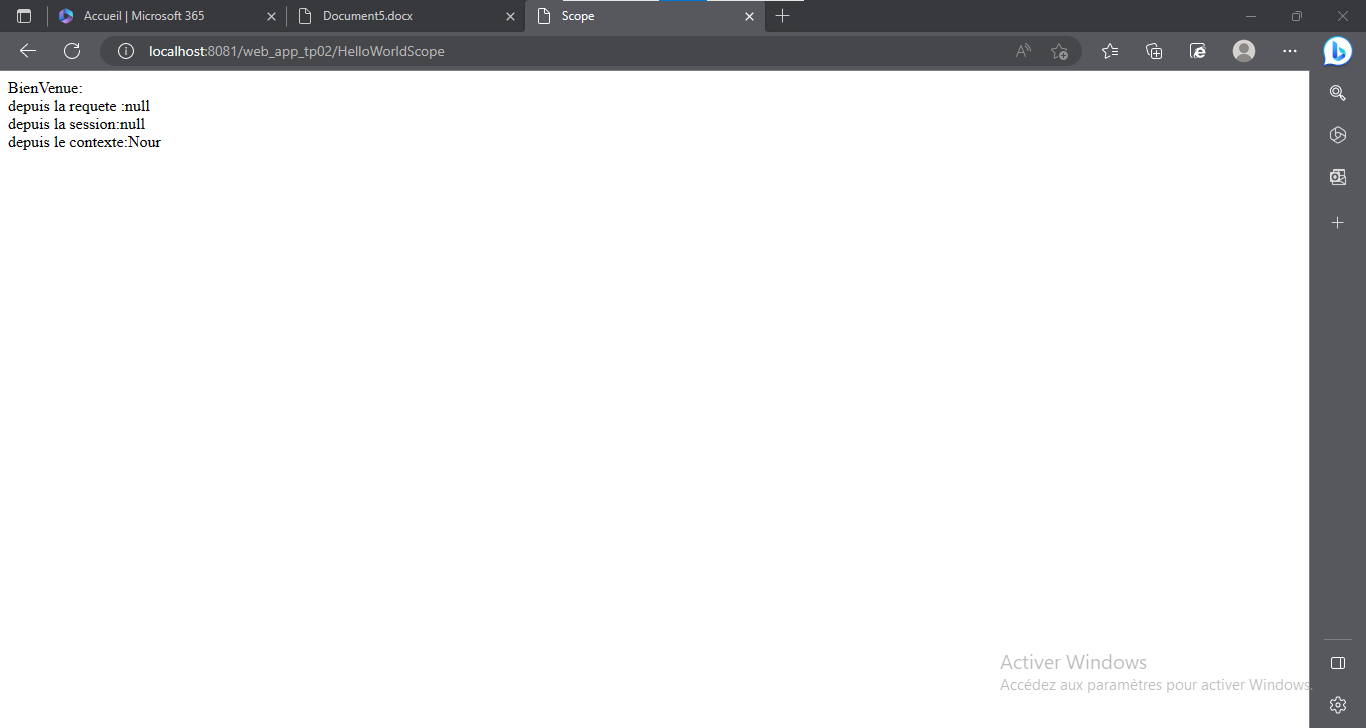


Figure :Execution3 HelloWorldScope un autre navigateur

En lançant l'URL [http://localhost:8081/web\_app\_tp02/HelloWorldScope](http://localhost:8080/web_app_tp02/HelloWorldScope) sur un autre navigateur, la servlet HelloWorldScope sera appelée sans la valeur du paramètre "prenom". La servlet ne stockera pas de nouvelle valeur dans la session et le contexte de l'application, mais la page JSP affichera la valeur précédemment stockée dans la session et le contexte, s'ils existent. Les valeurs attribuées dans la session et le contexte seront partagées entre les différentes sessions et navigateurs.

# Mode connecté

Donc ,on a revenir à la servlet « ConnexionAction » pour mettre les deux paramètres « login » et « password » comme attributs dans la session avant de rediriger la requête vers la page d’accueil et modifier la page « accueil.jsp » avec une nouvelle version.

Et on dernier lieu , on a ajouter un lien « Deconnexion » qui appelle une Servlet « DeconnexionAction » qui mets la valeur « null » à chacun des attributs « login » et « password » de la session et puis redirige l’affichage vers la vue « connexion.jsp »

On aboutit cette resultats :

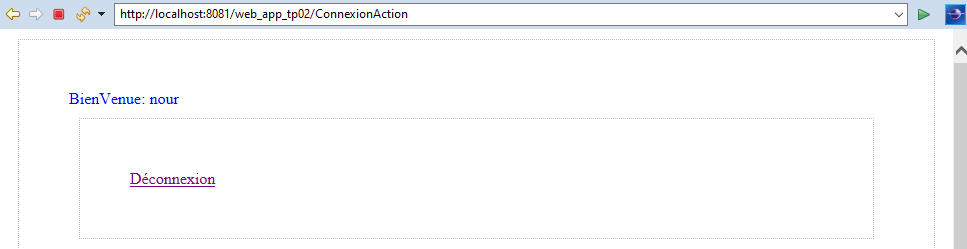


Figure :Lien de deconnexion

# Conclusion :

Les pages JSP s'exécutent, en fait, sous la forme de servlets. Elles disposent du même cycle de vie. Elle sont donc compilées comme les servlets et sont donc plus rapides dans leur traitement. Elles disposent du même support pour la gestion des sessions. Elles peuvent également charger des JavaBeans et appeler leurs méthodes, accéder à des sources de données se trouvant sur des serveurs distants, ou effectuer des calculs complexes. Tout ce que peut faire une servlet une page JSP est capable de le réaliser.

# Chapitre 4 : GESTION DE JAVA BEAN

# Programmation JEE/JSP

JSP (JavaServer Pages) est une technologie utilisée pour la création de pages web dynamiques en Java. Les JSP permettent d'insérer du code Java dans une page HTML, ce qui permet de générer du contenu dynamique à partir de données récupérées depuis une base de données ou d'autres sources de données. Les JSP sont souvent utilisées en combinaison avec des servlets, qui sont des programmes Java qui s'exécutent sur le serveur web et qui gèrent les requêtes des clients.

# Creation du premiere bean :

Tout d'abord on a créer un nouveau projet web dynamique nommé « web\_app\_tp03 » dans notre Workspace .

Puis, on a créer un package nommé «beans » choisir le commande

« New/Class » pour construire un bean nommé « Etudiant »

et une servlet « EtudiantManager » dans un nouveau package nommé « servlets» .

On a compléter le code de ce bean comme suit :

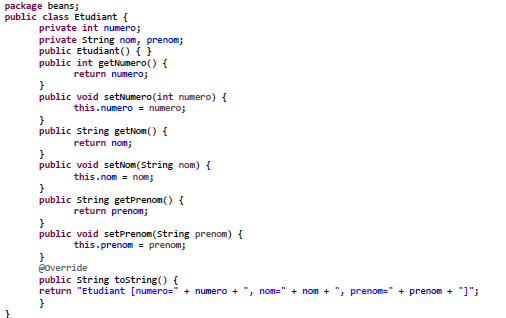


Figure :Code bean Etudiant

On modifier la méthode « doGet » de la servlet « EtudiantManager » par le code suivant :

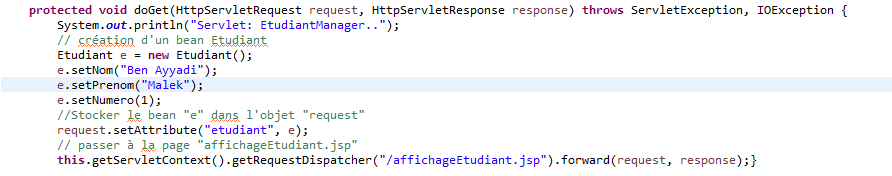


Figure :Premier modification de la méthode « doGet » de la servlet « EtudiantManager »

Puis, on a définir, dans le dossier « WebContent », une page JSP nommée « affichageEtudiant.jsp» ayant le code suivant:

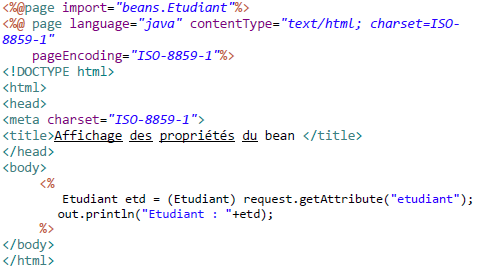


Figure ::code page « affichageEtudiant.jsp»

On obtient le resultat suivant :

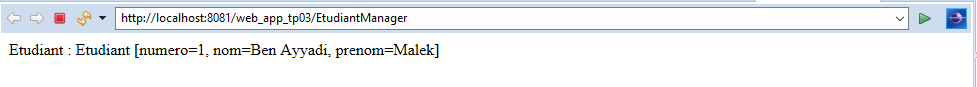


Figure :Exécution 1 de la servlet « EtudiantManager »

# La portée des objets

La portée des objets est un concept important en programmation, notamment en JSP (JavaServer Pages), qui permet de définir la visibilité et la disponibilité des objets dans une application. Elle définit également la durée de vie de ces objets.

Pour cela , on a modifier la méthode « doGet » de notre servlet et notre page JSP pour mieux comprendre la portée des objets et tester leur fonctionnement .

En premier lieu ,on a passer à l’exécution dans un navigateur web en spécifiant l’adresse suivante :

<http://localhost:8081/web_app_tp03/EtudiantManager>

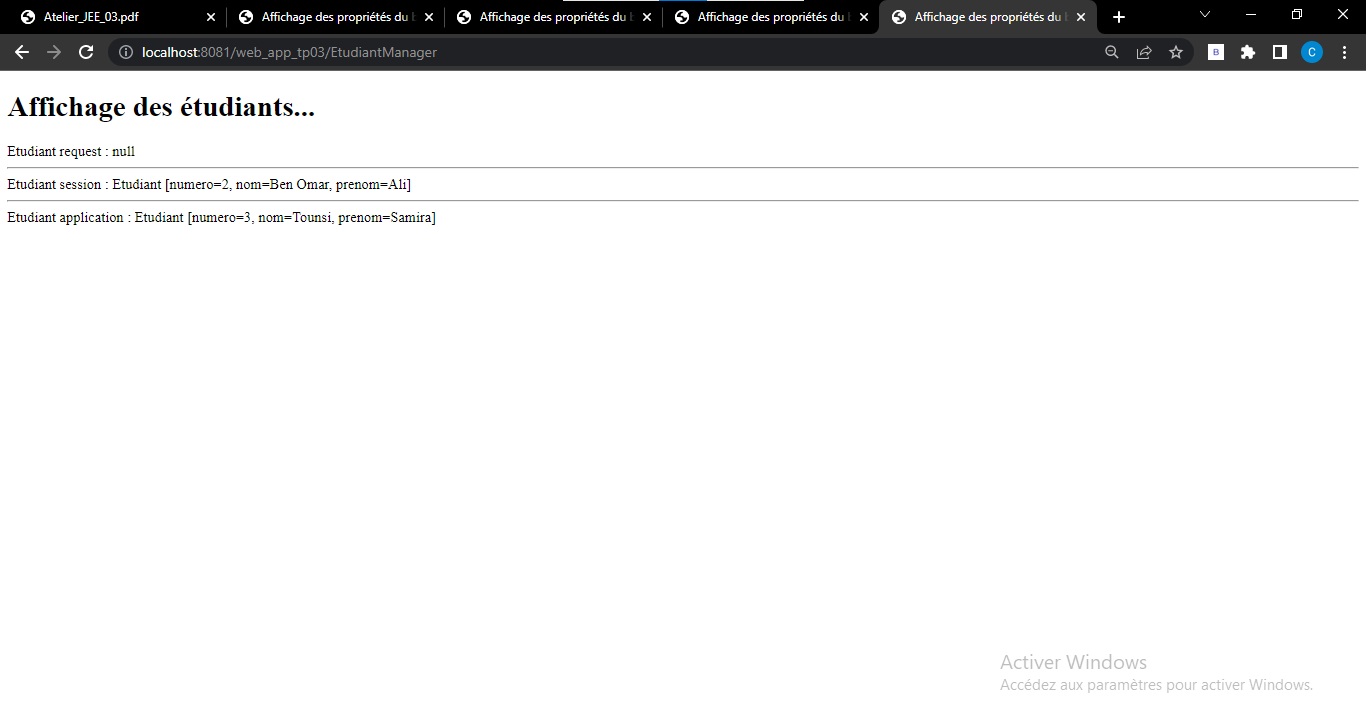


Figure :Exécution 2 de la servlet « EtudiantManager »

L'adresse [http://localhost:8081/web\_app\_tp03/EtudiantManager](http://localhost:8080/web_app_tp03/EtudiantManager) correspond à la page principale de l'application, qui est un formulaire permettant de saisir les informations d'un étudiant et de les enregistrer dans la base de données. En accédant à cette adresse, le formulaire s'affiche dans le navigateur web.

En deuxieme lieu ,on a ouvrir un autre onglet du navigateur web et taper l’adresse suivante :<http://localhost:8081/web_app_tp03/affichageEtudiant.jsp>

L'adresse [http://localhost:8081/web\_app\_tp03/affichageEtudiant.jsp](http://localhost:8080/web_app_tp03/affichageEtudiant.jsp) correspond à une autre page de l'application, qui permet d'afficher la liste des étudiants enregistrés dans la base de données. En accédant à cette adresse à partir du premier onglet, après avoir enregistré quelques étudiants à partir du formulaire, la liste des étudiants s'affiche dans le navigateur web.

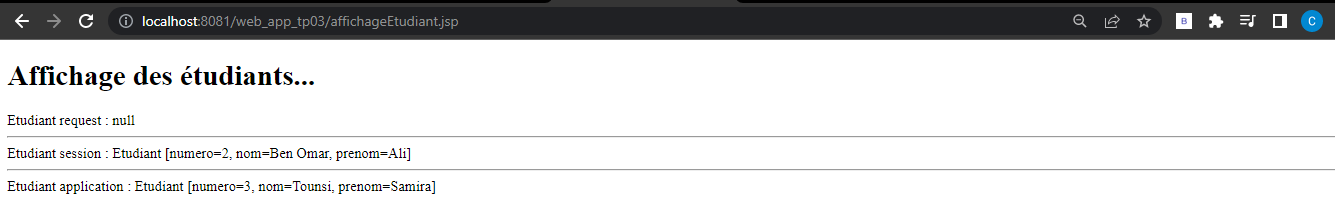


Figure :Exécution de la page « affichageEtudiant.jsp» dans un autre onglet du navigateur web

Et dernier lieu , on a ouvrir un autre navigateur web et taper l’adresse suivante :

<http://localhost:8081/web_app_tp03/affichageEtudiant.jsp>

En accédant à la même adresse precedente à partir du deuxième navigateur web, qui n'a pas été enregistré d'étudiant dans la base de données, la liste est vide.

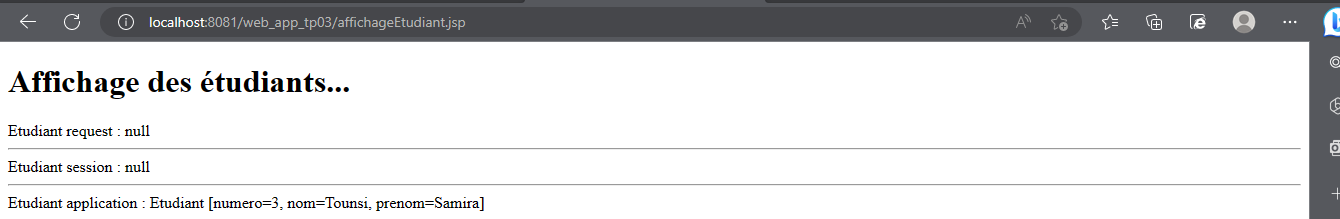


Figure :Exécution de la page « affichageEtudiant.jsp» dans un autre navigateur web

En interprétant le résultat accessible, nous pouvons constater que notre application web fonctionne correctement et que les différentes pages peuvent être à partir de différentes adresses dans le navigateur web. Nous pouvons également constater que les données enregistrées à partir du formulaire sont bien persistées dans la base de données et peuvent être récupérées et affichées à partir de la page d'affichage.

**L’action « useBean »**

**Dans cette partie , on a tester une autre façon de récupérer un bean est donnée par la version « affichageEtudiant2.jsp » qui contienne le code suivant :**



Figure :Code de la page affichageEtudiant2.jsp

On relance l’exécution de « affichageEtudiant2.jsp » et aboutit le résultat suivant :

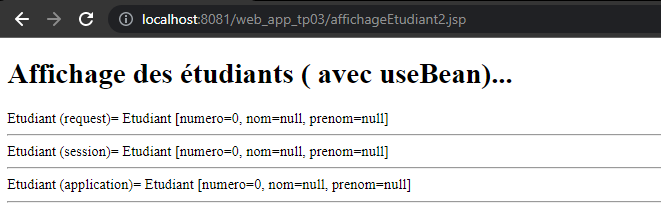


Figure :Exécution de la page affichageEtudiant2.jsp

Il est possible de donner un corps à cette balise, qui ne sera exécuté que si le bean est créé.

Pour se faire, on a ajouter le code suivant à la fin du corps de la page « affichageEtudiant2.jsp »

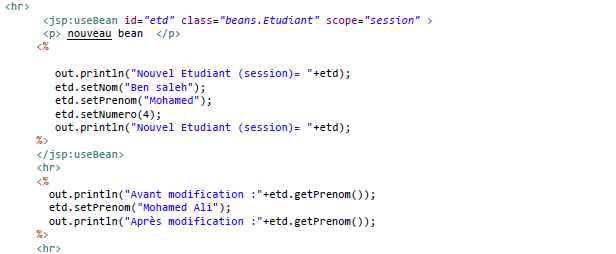


Figure :Ajout du code dans la page affichageEtudiant2.jsp

On passe à l’exécution dans un navigateur web en spécifiant l’adresse suivante :

<http://localhost:8081/web_app_tp03/EtudiantManager>

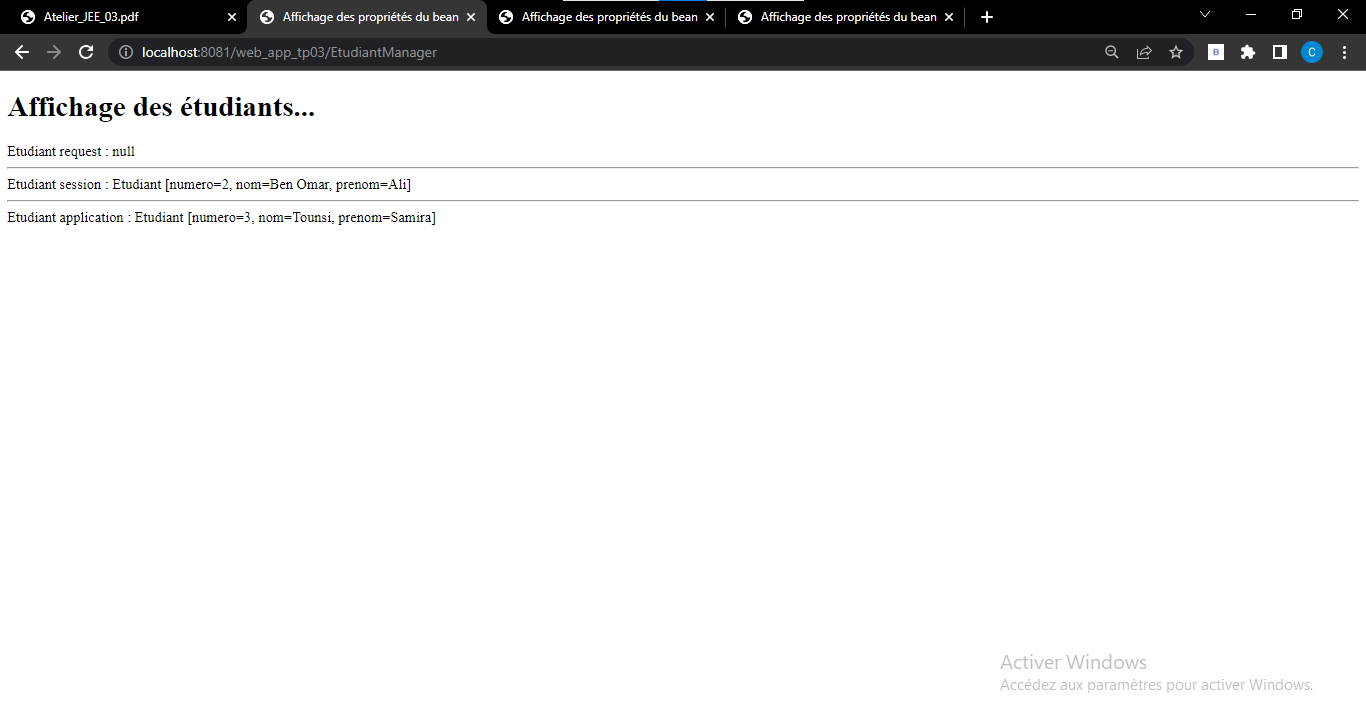


Figure :Exécution 3 de de la servlet « EtudiantManager » dansun autre onglet du navigateur web

Puis on a ouvrir un autre onglet du navigateur web et taper l’adresse suivante :

<http://localhost:8081/web_app_tp03/affichageEtudiant2.jsp>

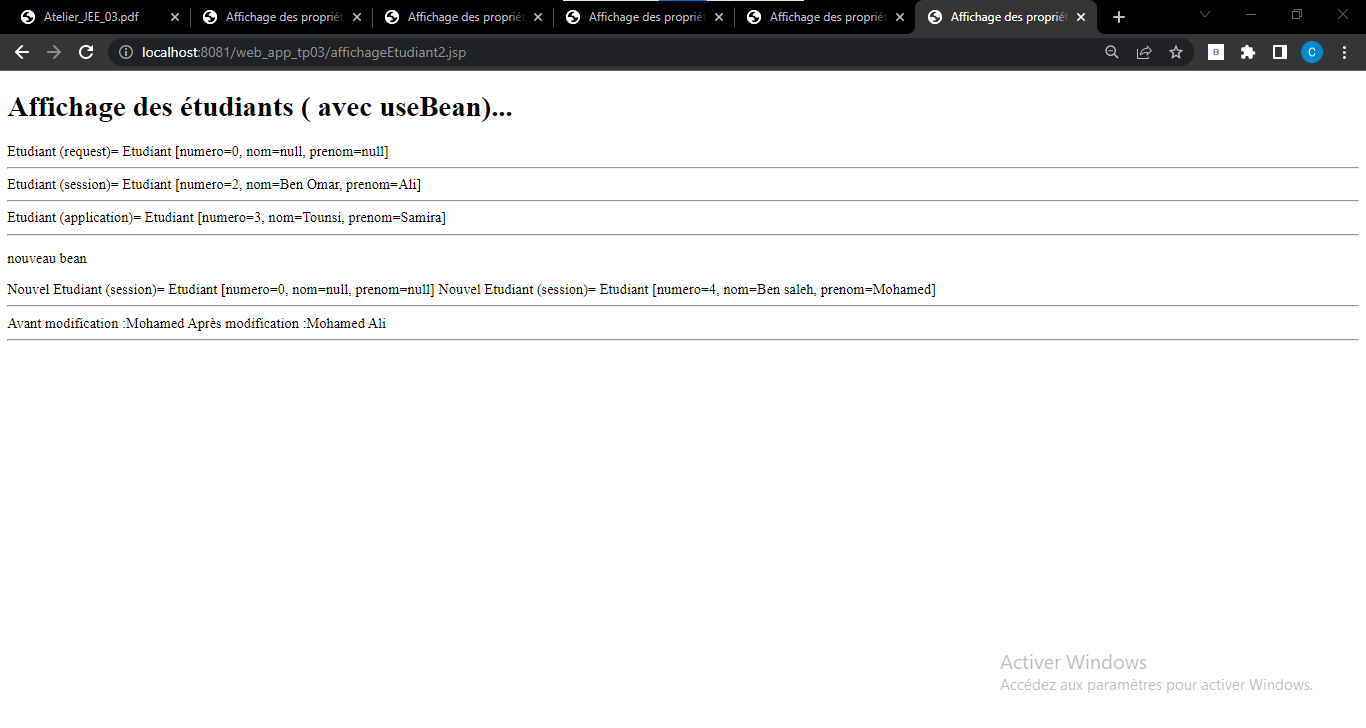


Figure :Exécution de la page « affichageEtudiant2.jsp» dans un autre onglet du navigateur web

Les actions « getProperty» et « setProperty »

Une autre façon pour accèder et modifier les attributs d’un bean est d’utiliser les actions « getProperty » et « setProperty ».

Pour se faire, on a ajouter le code suivant à la fin du corps de la page JSP :

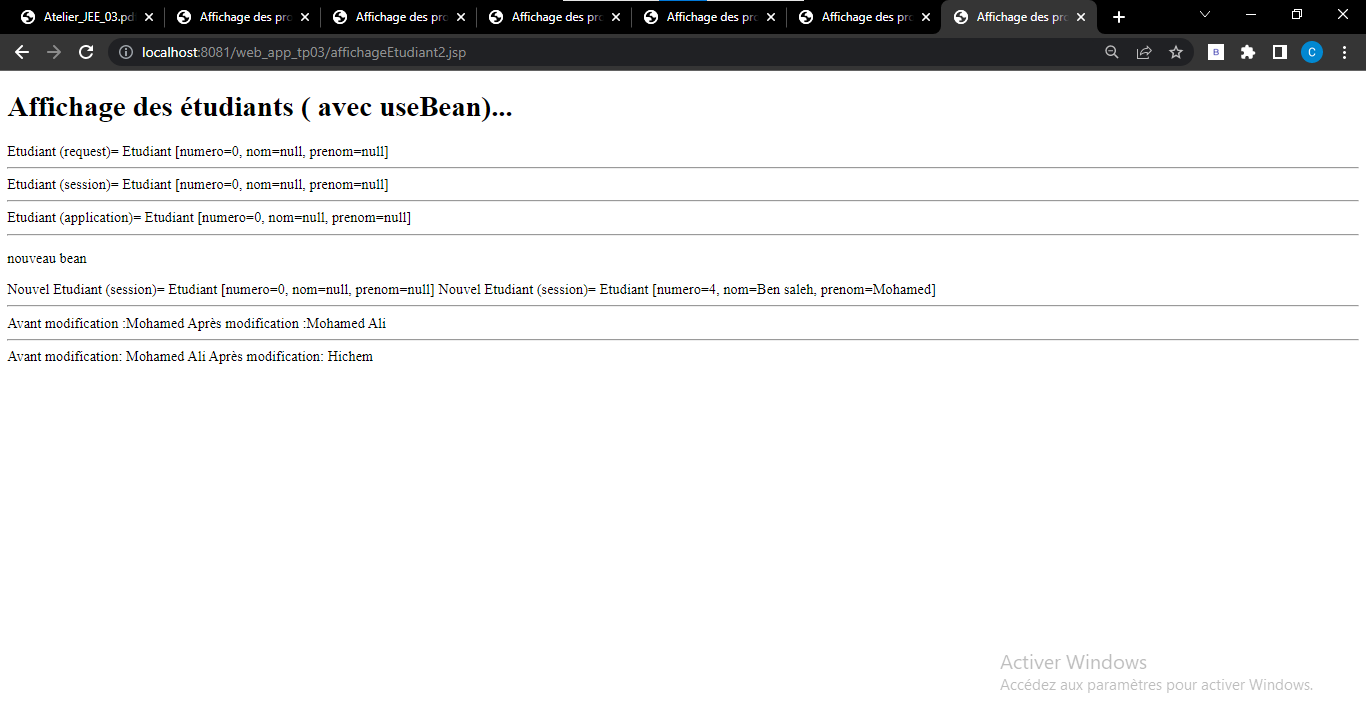


Figure :Exécution de la page « affichageEtudiant2.jsp» aprés l'ajout du code à la fin du corps

# Conclusion :

En résumé, nous avons vu que les servlets permettent facilement d’exécuter du code Java pour traiter une requête HTTP. Cependant, l’API servlet n’est pas très pratique pour générer une réponse orientée texte (telle qu’une page HTML).

Les Java Server Pages (JSP) ont été la première solution introduite pour offrir une alternative plus simple pour l’écriture de patron (template) de réponse.

# Chapitre 5 : Application Web JEE avec modèle MVC et base de données

# Gérer le support physique de stockage des données

En ce qui concerne la gestion du support physique de stockage des données dans une application JEE, il est généralement recommandé d'utiliser un système de gestion de base de données (SGBD) qui gère la persistance des données de manière fiable et sécurisée. Les SGBD les plus couramment utilisés dans l'écosystème JEE sont MySQL, PostgreSQL, Oracle et Microsoft SQL Server.

La gestion du stockage physique des données est généralement effectuée par le SGBD lui-même, qui prend en charge la création et la gestion des fichiers de données sur le disque dur. Cependant, il est important de configurer correctement le SGBD pour garantir des performances optimales et une sécurité adéquate.

En général, il est recommandé de stocker les fichiers de données sur un disque dur dédié ou un ensemble de disques durs configurés en RAID (Redundant Array of Independent Disks) pour garantir une redondance des données et une tolérance aux pannes. Le stockage des données peut également être configuré pour utiliser des disques à état solide (SSD) pour des performances de stockage plus rapides.

En outre, les SGBD modernes disposent de fonctionnalités de sauvegarde et de récupération pour protéger les données contre la perte en cas de panne matérielle ou de catastrophe naturelle. Il est recommandé de configurer régulièrement des sauvegardes pour s'assurer que les données sont protégées contre la perte de données. Les stratégies de sauvegarde peuvent varier en fonction des besoins de l'application et des contraintes de temps de récupération.

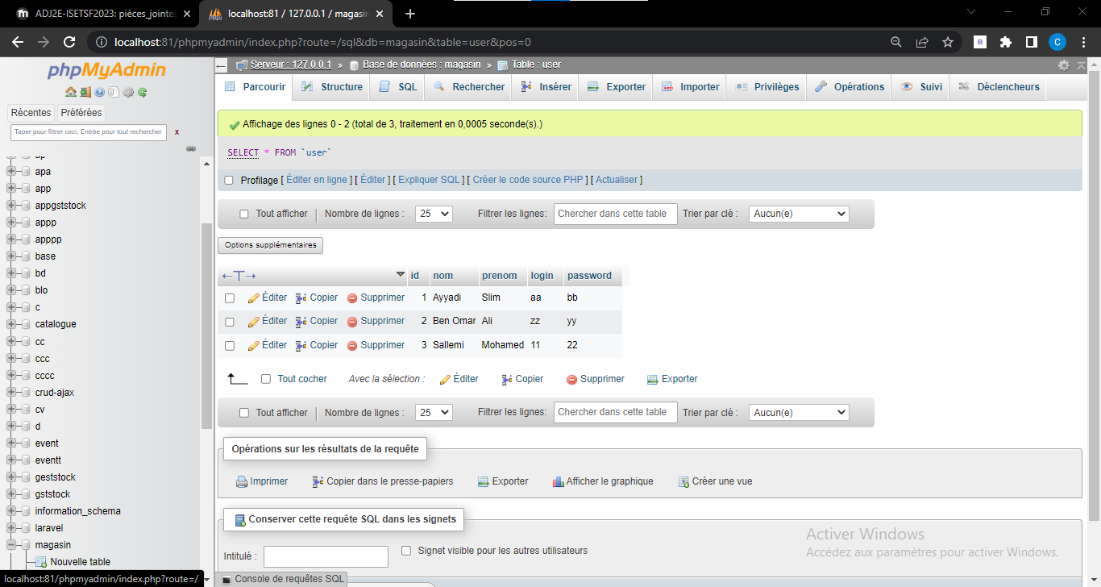


Figure 47:la base de données

# Gérer la partie « Metier »:

Il s'agit d'un ensemble d'instructions pour créer une application web Java avec une base de données. Les instructions consistent à créer un projet web dynamique dans Eclipse, à définir une classe "Utilisateur" qui possède des attributs tels que l'id, le nom et le mot de passe, à créer un package "metier" qui contient une interface pour gérer les objets Utilisateur, à créer une classe DBConnexion pour gérer les connexions à la base de données, et à implémenter UserMetierInterface avec une classe UserMetierImpl pour effectuer des opérations CRUD sur les objets Utilisateur dans la base de données.

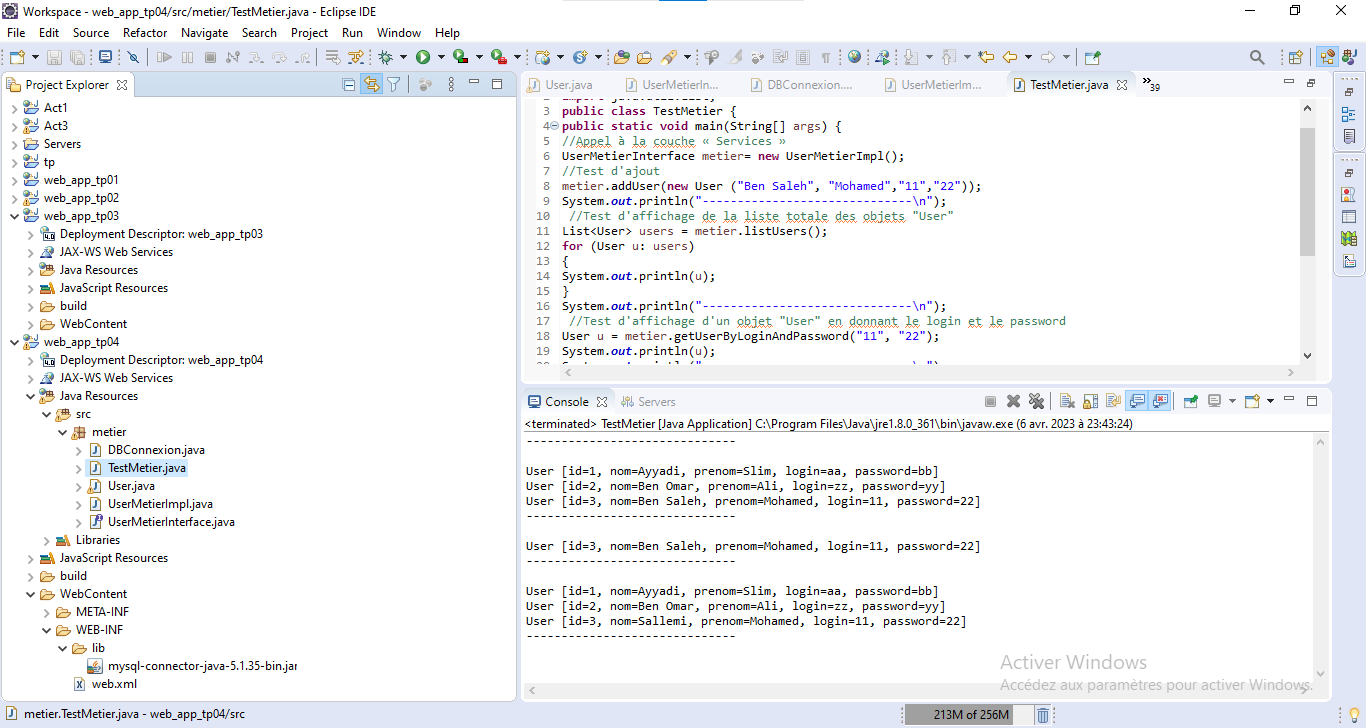


Figure 48:le résultat dans la console et au niveau de la base de données.

# Gérer la partie « web »

Lors de l'exécution de la vue "UserConnexion.jsp", l'utilisateur est présenté avec un formulaire de connexion qui lui permet de saisir son nom d'utilisateur et son mot de passe. Si la saisie est incorrecte, l'affichage est redirigé vers la même vue. Si la saisie est correcte, la servlet "UserController" est appelée en mode "POST". Cette servlet appelle la méthode "getUserByLoginAndPassword(String l, String p)" d'un objet "metier" et vérifie l'existence de l'utilisateur. Si l'utilisateur n'est pas reconnu, l'affichage est redirigé vers la vue "UserConnexion.jsp". Sinon, l'utilisateur est enregistré dans la session et l'affichage est redirigé vers la vue "accueil.jsp" qui inclut le fichier "entete.jsp".

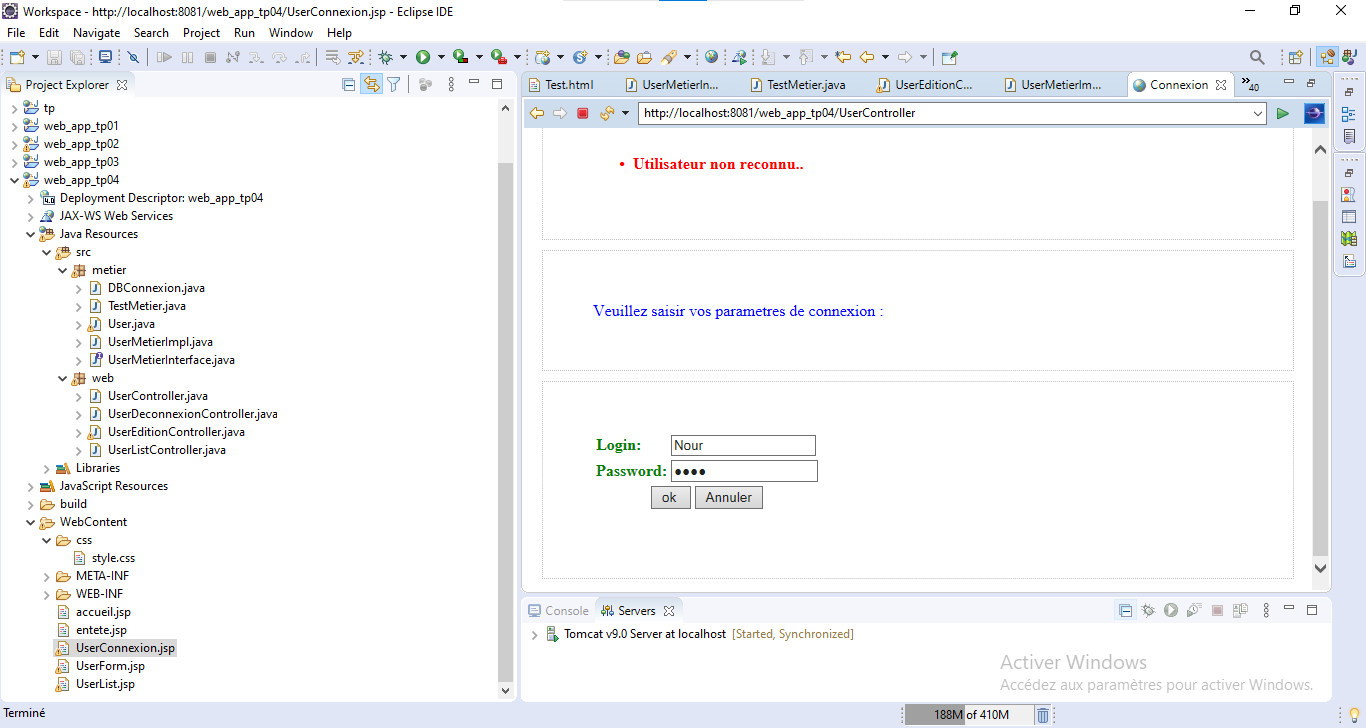


Figure 49: la formulaire de connexion de la page "UserConnexion.jsp"

L'exécution de la servlet Usercontroller.jsp intervient lorsque l'utilisateur soumet un formulaire de connexion à travers la vue UserConnexion.jsp. La servlet traite alors les données reçues en mode POST et vérifie la validité des informations de connexion en appelant la méthode "getUserByLoginAndPassword(String l , String p)" de l'objet métier correspondant. Si les informations sont invalides, l'utilisateur est redirigé vers la vue UserConnexion.jsp pour saisir de nouveau ses identifiants. Sinon, l'utilisateur est enregistré dans la session et redirigé vers la vue accueil.jsp. Cette servlet est donc responsable de l'authentification de l'utilisateur et de la gestion session .

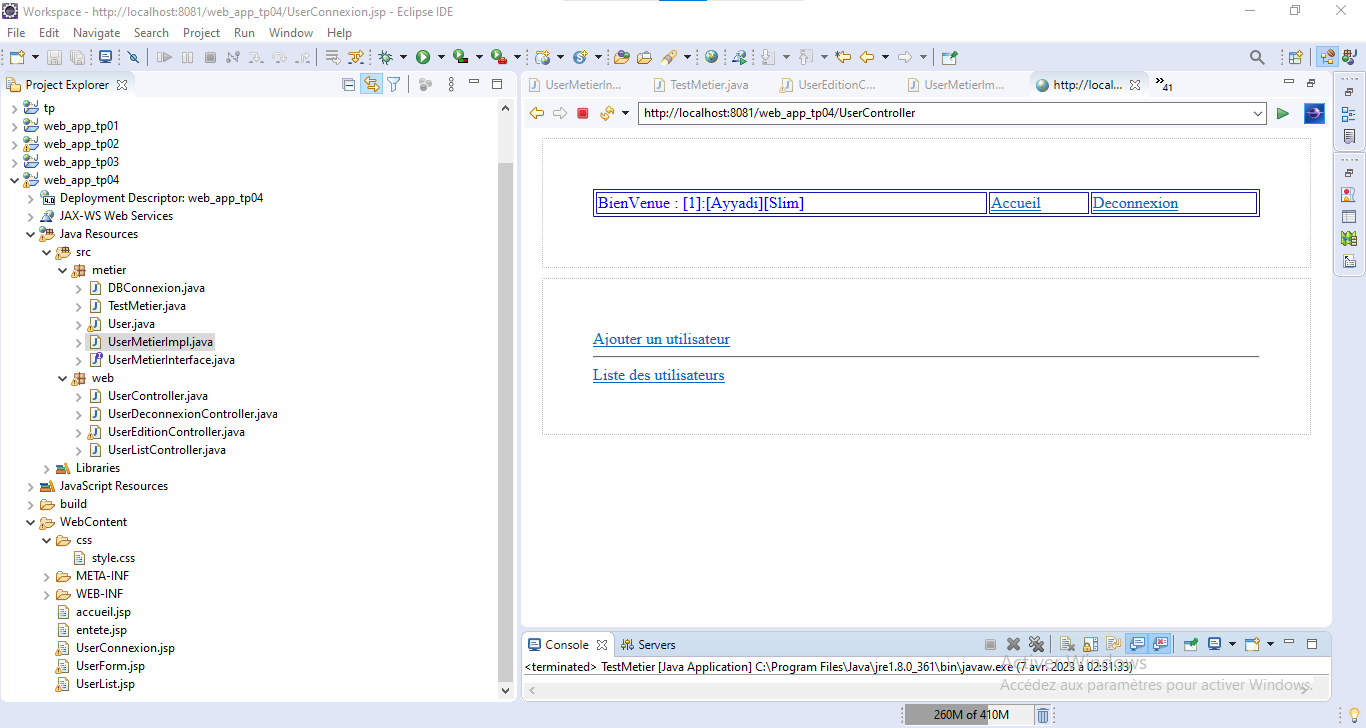


Figure 50:Exécution de la page "accueil.jsp"

Lorsque l'utilisateur clique sur le lien "Ajouter un utilisateur" depuis la page d'accueil, il est redirigé vers la vue "UserForm.jsp". Cette vue contient un formulaire qui permet à l'utilisateur de saisir les informations d'un nouveau utilisateur (nom, prénom, login et mot de passe). Lorsque l'utilisateur soumet le formulaire en mode "POST", les données sont envoyées à la servlet "UserEditionController" qui les récupère et appelle la méthode "saveOrUpdateUser(User u)" d'un objet "metier" pour ajouter le nouvel utilisateur à la base de données. Si la méthode renvoie "false", ce qui indique que l'ajout a échoué, l'utilisateur est redirigé vers la vue "UserForm.jsp" avec un message d'erreur. Sinon, l'utilisateur est redirigé vers la page d'accueil qui affiche un message de confirmation et la liste mise à jour des utilisateurs.

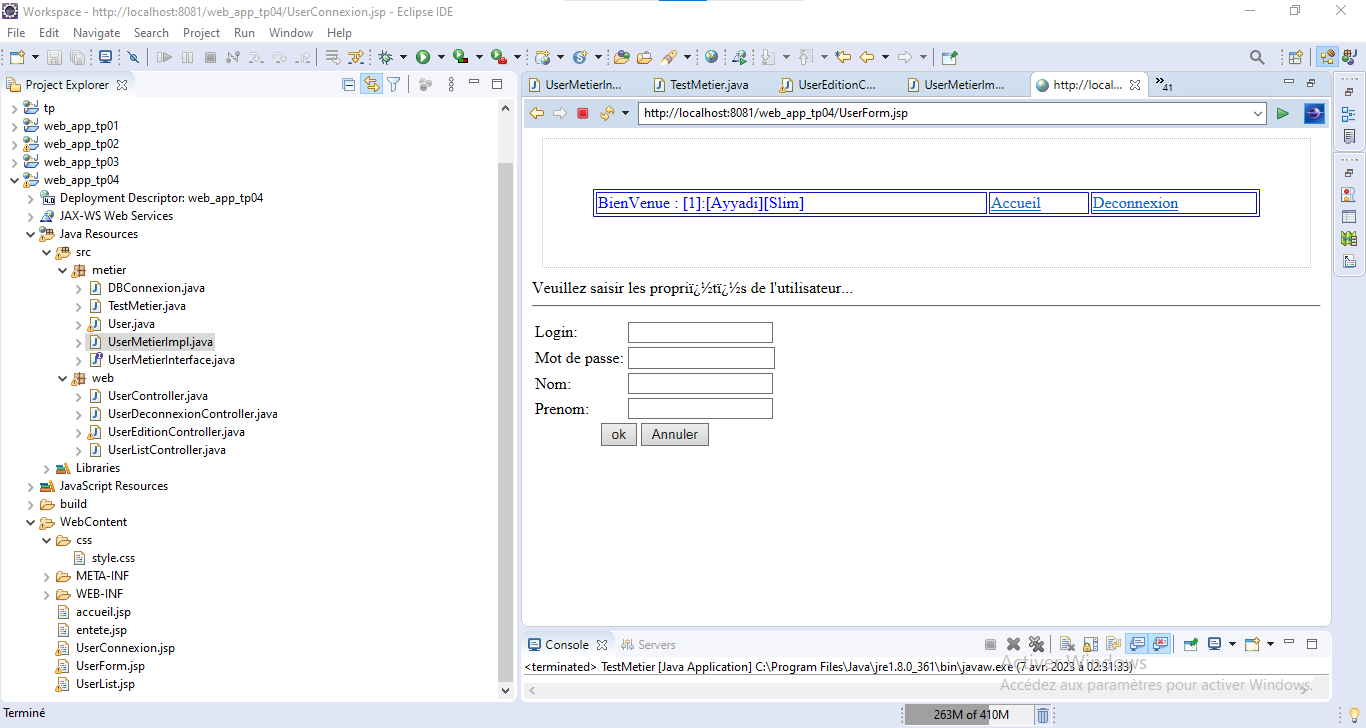


Figure 51:Exécution de la page "UserForm.jsp"

Pour ajouter un nouvel utilisateur à partir de la page "UserForm.jsp", l'utilisateur doit remplir le formulaire avec les informations demandées, notamment le nom, le prénom, le login et le mot de passe. Une fois les informations saisies, l'utilisateur doit cliquer sur le bouton "Enregistrer" qui déclenche la soumission du formulaire à la servlet "UserEditionController". Cette servlet vérifie si les données saisies sont valides et si c'est le cas, elle appelle la méthode "saveUser(User user)" d'un objet "metier" pour enregistrer le nouvel utilisateur dans la base de données. Ensuite, l'affichage est redirigé vers la page d'accueil "accueil.jsp" qui inclut le fichier "entete.jsp". Si les données saisies sont invalides, l'utilisateur est redirigé vers la page "UserForm.jsp" avec un message d'erreur indiquant les champs invalides.

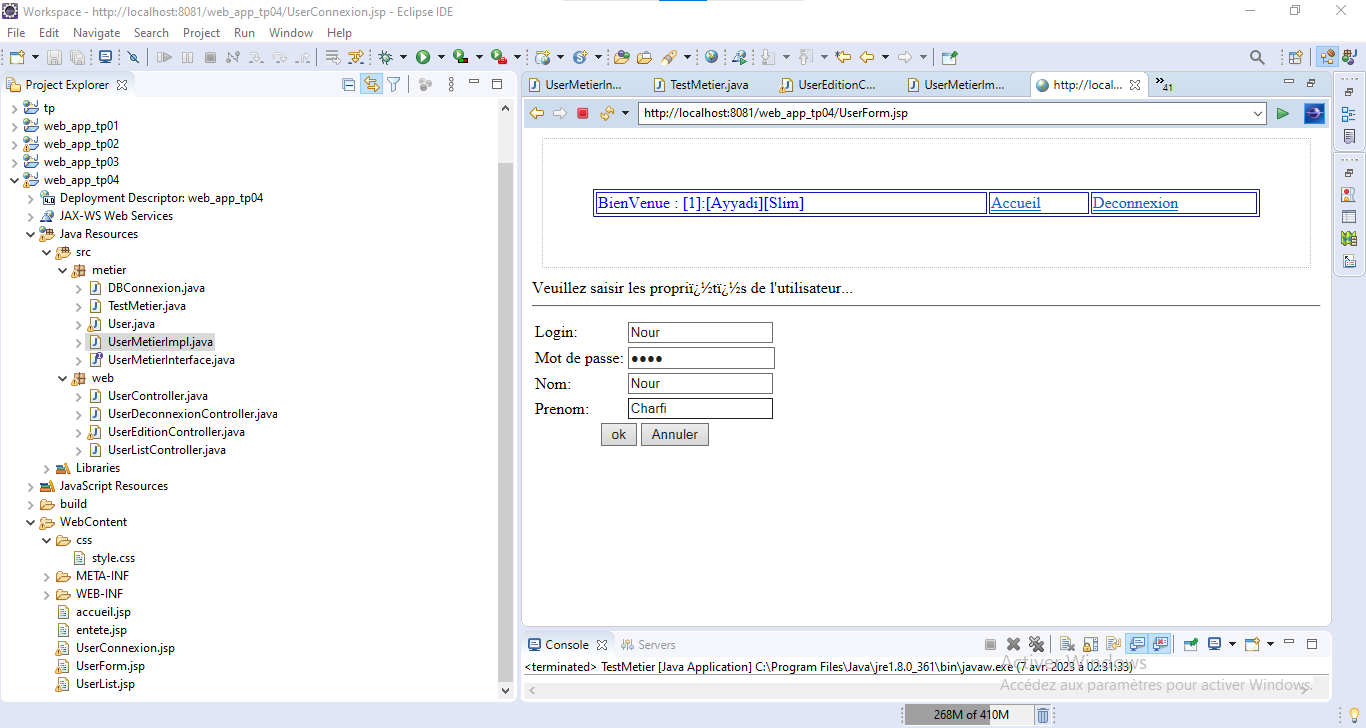


Figure 52:le remplissage de le formulaire avec les informations demandées

Le **UserEditionController** est un contrôleur qui gère l'édition d'un utilisateur existant. Il est accessible à partir de la page **UserList.jsp** où chaque utilisateur est présenté dans un tableau et où un lien pour éditer chaque utilisateur est disponible. Lorsque l'utilisateur clique sur le lien pour éditer un utilisateur, le contrôleur est appelé en mode GET et récupère l'identifiant de l'utilisateur à partir de la requête. Le contrôleur appelle ensuite la méthode **getUserById** de l'objet **metier** pour récupérer les informations de l'utilisateur et les présenter dans le formulaire de modification **UserForm.jsp**. L'utilisateur peut alors modifier les informations et soumettre le formulaire en mode POST.

Une fois que le formulaire est soumis, le contrôleur **UserEditionController** est appelé en mode POST. Le contrôleur récupère les informations du formulaire et appelle la méthode **updateUser** de l'objet **metier** pour mettre à jour les informations de l'utilisateur dans la base de données. Si la mise à jour est réussie, l'utilisateur est redirigé vers la page **UserList.jsp** pour afficher à nouveau le tableau d'utilisateurs avec les mises à jour effectuées. Si la mise à jour échoue, l'utilisateur est renvoyé à la page **UserForm.jsp** avec un message d'erreur.

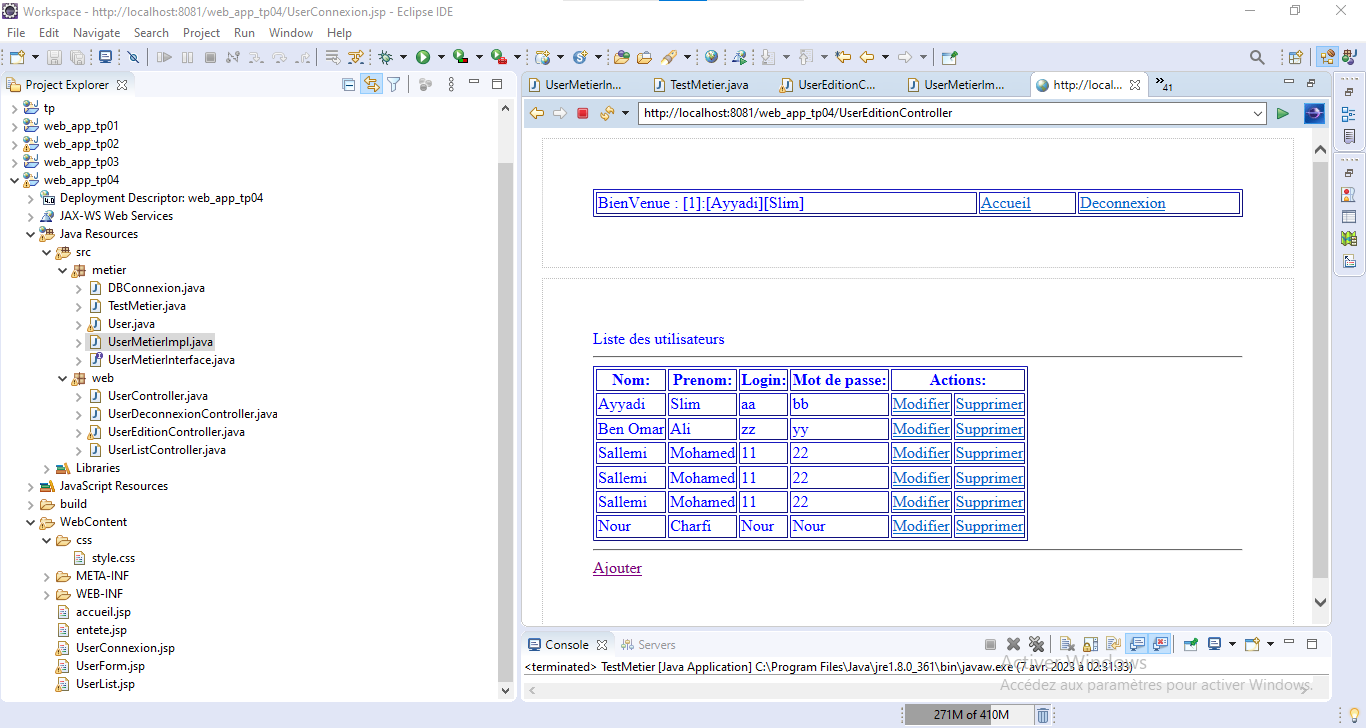


Figure 53:tableau où un lien pour éditer chaque utilisateur est disponible

Le contrôleur "UserEditionController" est également responsable de la suppression d'un utilisateur existant. Lorsque l'utilisateur clique sur le bouton de suppression sur la page "UserEdition.jsp", une requête POST est envoyée à la servlet "UserEditionController" qui récupère l'ID de l'utilisateur à supprimer à partir des paramètres de la requête. Ensuite, la méthode "deleteUserById(int id)" est appelée sur l'objet métier correspondant pour supprimer l'utilisateur de la base de données. Si la suppression est réussie, l'utilisateur est redirigé vers la page de liste d'utilisateurs via la servlet "UserListController". La liste mise à jour des utilisateurs est affichée avec le message de confirmation de suppression. Si la suppression échoue, un message d'erreur est affiché sur la page "UserEdition.jsp".

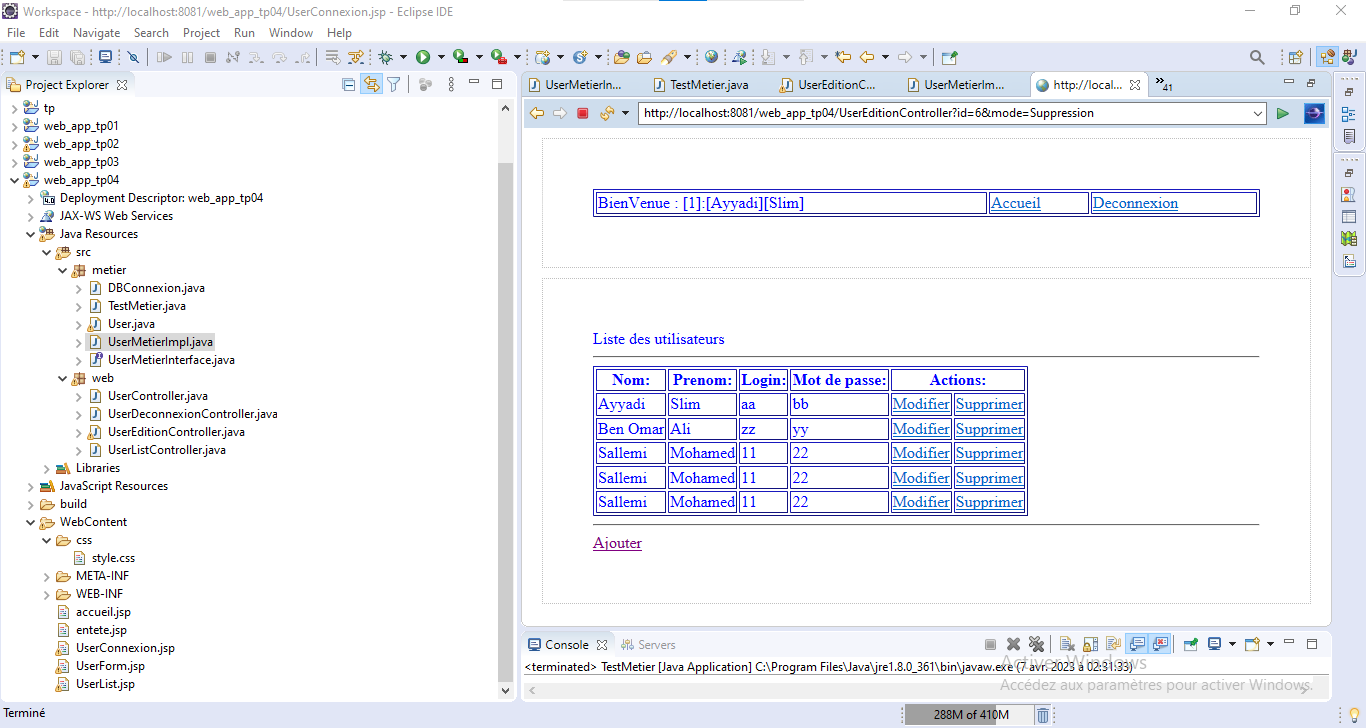


Figure : tableau où un lien pour éditer chaque utilisateur est disponible

La déconnexion de l'utilisateur est gérée par la servlet UserDeconnexionController. Lorsque l'utilisateur clique sur le bouton de déconnexion dans la barre de navigation, cette servlet est appelée en mode GET. Elle supprime la session actuelle en appelant la méthode invalidate() de l'objet HttpSession, puis elle redirige l'utilisateur vers la page de connexion en utilisant la méthode sendRedirect() de l'objet HttpServletResponse. Ainsi, la session de l'utilisateur est effacée et il est déconnecté de l'application.

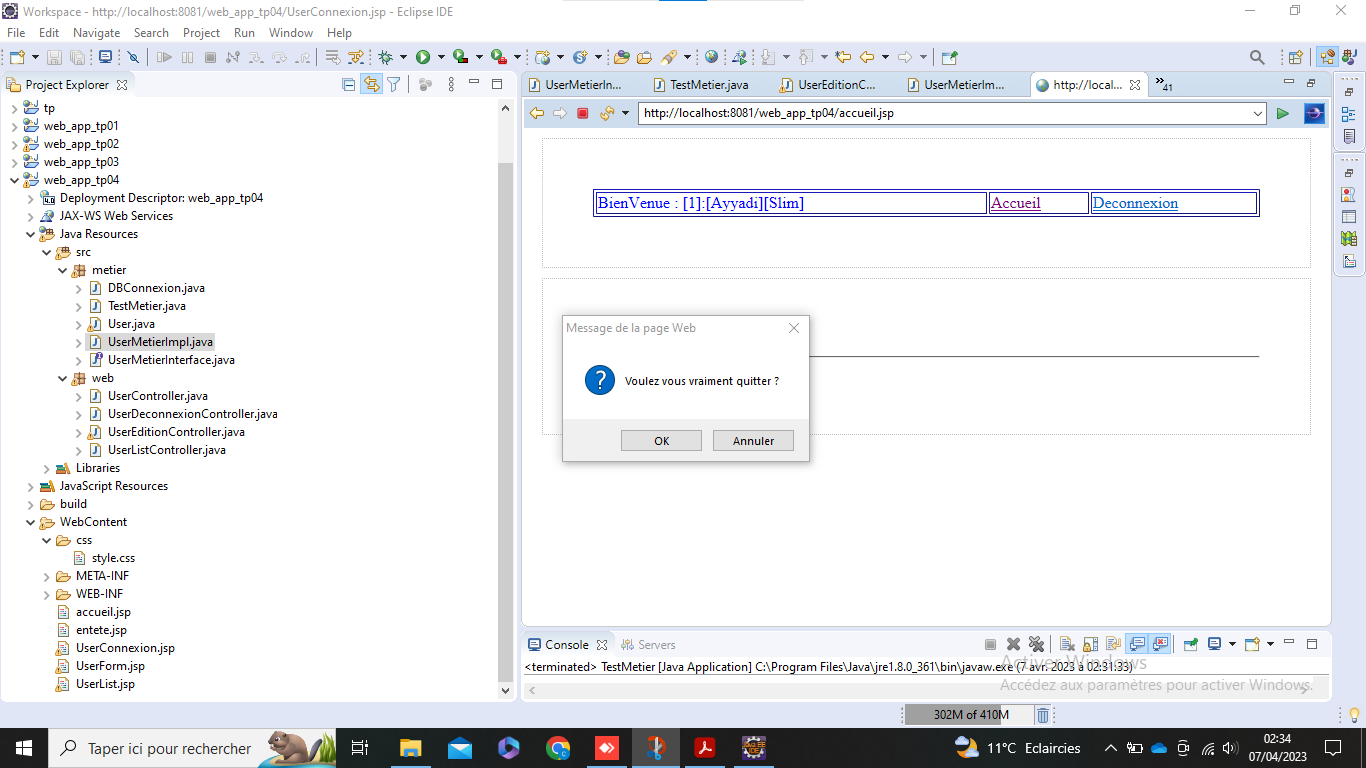


Figure 55: La confirmation de la déconnexion

Après qu'un utilisateur se soit déconnecté, il peut être redirigé vers un contrôleur appelé "UserDeconnexionController". Ce contrôleur sera chargé de terminer la session utilisateur en cours et de rediriger l'utilisateur vers une page de confirmation de déconnexion. Le contrôleur peut également effectuer d'autres tâches de nettoyage, telles que la suppression des cookies de session ou la mise à jour de la base de données pour enregistrer la date et l'heure de la déconnexion de l'utilisateur. La redirection vers ce contrôleur peut être effectuée via une instruction de redirection dans le code de l'application ou via une configuration dans le fichier de configuration de l'application. Cela garantit que l'utilisateur est correctement déconnecté de l'application et que toutes les données de session sont nettoyées avant de rediriger l'utilisateur vers la page de confirmation de déconnexion.

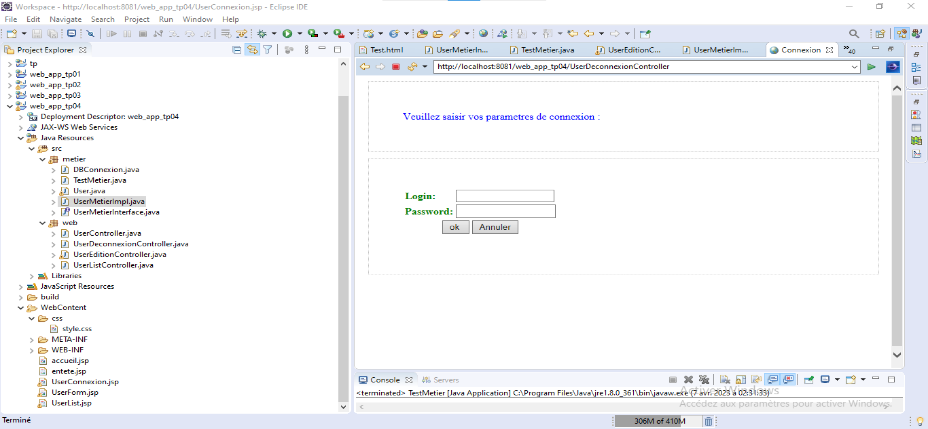


Figure 56: la redirect vers la page "accueil.jsp" la deconnexion

# Conclusion :

En conclusion, ce TP a permis de créer une base de données nommée "MAGASIN" et une table "User" avec les attributs id, nom, prenom, login et password. Ensuite, une classe Java a été créée pour gérer le métier avec la définition d'un bean "User" qui inclut les méthodes setters et getters, les constructeurs et une méthode toString(). Une interface "UserMetierInterface" a été définie pour déclarer les traitements métier sur un objet "User". Une classe "DBConnexion" a été créée pour permettre la connexion à la base de données "MAGASIN". Enfin, une classe "UserMetierImpl" a été mise en place pour donner une implémentation des traitements métier déclarés dans l'interface "UserMetierInterface". Le TP a permis de mettre en pratique les concepts de la programmation orientée objet et de la manipulation de bases de données relationnelles dans un environnement Java.

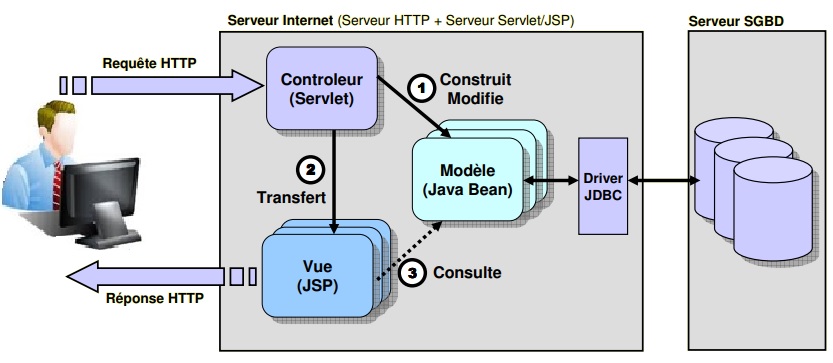


Figure 57:L'architecture MVC avec servlets, JSP et MySQL

# Annexe

* [**Le JRE**](https://www.infoworld.com/article/3304858/what-is-the-jre-introduction-to-the-java-runtime-environment.html) est la partie sur disque de Java qui crée la JVM et y charge les programmes.
* **Le JDK** fournit les outils nécessaires pour écrire des programmes Java qui peuvent être exécutés et exécutés par la JVM et le JRE.
* **Eclipse**  est un projet, décliné et organisé en un ensemble de sous-projets de développements logiciels, de la fondation Eclipse visant à développer un environnement de production de logiciels libre qui soit extensible, universel et polyvalent, en s'appuyant principalement sur Java.
* **Servlet**  est un programme qui s'exécute côté serveur en tant qu'extension du serveur. Elle reçoit une requête du client, elle effectue des traitements et renvoie le résultat. La liaison entre la servlet et le client peut être directe ou passer par un intermédiaire comme par exemple un serveur http.
* **Tomcat** est le logiciel de serveur web préféré des développeurs pour les implémentations Java.
* **Les types des erreurs**

Quand un message d’erreur s’affiche sur un site web que nous visitons, notre réaction habituelle est de fermer tout de suite la fenêtre du navigateur sans même lire le texte descriptif. Lorsqu’il s’agit de votre propre site par contre, ces numéros peuvent vous transmetre une information importante que vous ne devez pas négliger. En les identifiant correctement et en les corrigeant, ces erreurs vous aideront à améliorer votre référencement naturel et de diminuer votre taux de rebond. Les codes d’état HTTP sont utilisés par le serveur web afin d’informer le client du traitement réussi ou échoué de sa demande. Il existe plusieurs types de codes d’état et tous n’affichent pas un message d’erreur. Les codes d’état sont au format de 3 chiffres. Le premier chiffre marque la classe du code d’état:

**1xx : code d’information**

**2xx : code de succès**

**3xx : code de redirection**

**4xx : code d’erreur côté client**

**5xx : code d’erreur côté serveur**

Vous pouvez consulter une liste complète avec les codes d’état [en cliquant ici](https://www.iana.org/assignments/http-status-codes/http-status-codes.xhtml). Voyons ensemble les différent erreurs pour les pages HTML.

* **400 : Bad Request**

Cette page s’affiche quand le client envoie une demande que le serveur n’est pas capable de comprendre. Cela veut dire que le problème vient strictement du côté client : souvent il s’agit d’un système d’exploitation non sécurisé, d’une connexion Internet instable ou d’un problème de navigateur. Pour résoudre ce problème, essayez d’ouvrir la même page dans un autre navigateur, videz votre cache et vérifiez vos mises à jour de sécurité.

* **403 : Forbidden**

L’affichage de ce message indique que l’accès à la ressource demandée est limité et le serveur web refuse de vous transmettre le contenu souhaité. Il apparaît lorsque la navigation sur un fichier ou un répertoire est interdite. Afin de résoudre ce problème, vérifiez d’abord que vous avez spécifié un nom et une extension de fichier de page Web réels et non pas seulement un répertoire. Si l’erreur continue à s’afficher videz votre cache et réessayez.

* **404 : Not Found**

C’est sans doute parmi les messages d’erreur le plus souvent affichés. Il apparaît quand le serveur web n’arrive pas à trouver la ressource demandée par le client : par exemple si un fichier, un répertoire ou une image ont été supprimé. D’autres raisons fréquentes pour l’affichage de ce message sont :

Un URL mal saisi

Le suivi d’un lien qui n’est plus actif

L’utilisation d’un vieux bookmark qui n’est plus d’actualité

* **500 : Internal Server Error**

Ce message apparaît quand le serveur web rencontre une condition inattendue qui l’empêche de répondre à la demande du client. Les raisons les plus fréquentes pour que l’erreur 500 survienne sont :

Un problème lors de l’installation de votre logiciel ou lors de sa mise à jour.

Un nouveau plugin qui vient d’être activé est entré en conflit avec d’autres plugins lors de son installation

Si vous recevez ce message la première chose que vous devez faire est de vérifier vos logs : très souvent vous y trouverez la raison pour laquelle l’erreur s’est affichée.

* **503 : Service Unavailable**

Cette erreur s’affiche lorsque le serveur web ne peut pas lire les requêtes HTTP. Cela peut être dû à une surcharge du serveur ou à une maintenance temporaire.

* **301 : Moved Permanently et 302 : Found**

Le code 301 indique que la ressource que vous recherchez a définitivement changé son adresse. Il montre au client qu’il a été redirigé et qu’il doit utiliser la nouvelle adresse pour ses prochaines requêtes. Le 302 par contre indique au client que la nouvelle adresse est uniquement temporaire.

* **Les API JEE**

Les API sont des Outil logiciel, constitué d'un ensemble de modules dotés de fonctions communes, et qui permet de produire automatiquement des applications Java ou des applets personnalisés.

Parmi ses composants les plus courants, on trouve :

* **EJB** (Enterprise JavaBeans Technology)

L'EJB, environnement d'exécution fournissant des services (sécurité, communications, cycle de vie...), définit comment les composants serveurs sont écrits et fournit un lien standard entre le composant et l'application qui l'utilise. Un client n'accède donc jamais directement à un composant et doit pour cela passer par une interface locale et une interface distance. L'interface locale décrit le cycle d'existence du composant en définissant des méthodes permettant de le trouver, de le créer, de le détruire. Et L'interface distante spécifie les méthodes que ce composant présente au monde extérieur.

* **JDBC (Java Database Connectivity)**

Une API qui permet aux programmes Java d'interagir avec les bases de données relationnelles (type SQL).

* **Servlet et JSP (JavaServer Pages)**

Composants réseau qui sont destinés à une exécution orientée question / réponse. La technologie JSP est une extension de la notion de Servlet permettant de simplifier la génération de pages web dynamiques. JSP est un concurrent direct de l'ASP et du PHP. Un servlet est un composant côté serveur, écrit en Java, dont le rôle est de fournir une trame générale pour l'implémentation de paradigmes "requête / réponse". Ils remplacent les scripts CGI tout en apportant des performances bien supérieures.

* **JMS (Java Message Service)**

Permet à 2 composants d'applications JEE de créer, envoyer et recevoir des messages.

* **JTA (Java Transaction API) et JTS (Java Transaction Service)**

Les spécification JTA et JTS permettent aux composants d'être renforcés par un support des transactions.

* **JavaMailTM Technology**

Permet l'envoi d'e-mails d'une manière indépendante de la plate-forme et du protocole.

* **JAF (JavaBeans Activation Framework)**

Utilisé par JavaMail, JAF permet de déterminer le type d'un flux de données, son accès, et les opérations possibles sur lui.

* **JAXP (Java API for XML)**

Cette API permet de manipuler, créer et modifier des documents XML. JAXP supporte les parseur DOM (Document Object Model) et SAX (Simple API for XML).

* **JEE Connector**

Facilite les connexions aux plates-formes autres que JEE.

* **JAAS (Java Authentication and Authorization Service)**

Permet aux applications JEE d'authentifier et d'autoriser l'exécution par les utilisateurs.

* **JNDI**

Extension Java standard qui fournit une API permettant de localiser et d'utiliser des ressources : il peut y avoir un appel à des services CORBA, DNS, NIS, LDAP...

* **Environnement d'exécution de JEE**

Un des avantages majeurs de JEE est de faire abstraction de l'infrastructure d'exécution. En effet, JEE spécifie les rôles et les interfaces pour les applications, ainsi que l'environnement d'exécution dans lequel les applications sont déployées.

Ceci permet aux développeurs d'application de ne pas avoir à reprogrammer les services d'infrastructure.

* **Attributs Servlets/Pages JSP**

Les attributs fournissent des informations sur les servlets et les pages JSP.

Les attributs **Servlets/Pages JSP** fournissent des informations de performances pour les servlets et les pages JSP (Java™ Server Pages).

Les attributs Servlets JSP incluent par exemple le nombre moyen de demandes simultanées pour un servlet et le temps nécessaire à un servlet pour effectuer une demande. Utilisez les attributs Servlets/Pages JSP pour surveiller les performances et le niveau d'utilisation des servlets et des pages JSP.

* [**Servlet/JSP**](http://www.mtitek.com/tutorials/servlets_jsp/) **| Scopes**

1. **Le scope décrit la portée et la durée de vie des objets créés par une application WEB.**  
   Il y a globalement trois principaux scopes :
   * Il y a le scope REQUEST qui est limité à la durée vie de la requête http, et il privé à cette requête http.  
     Ce qui veut dire que les autres requêtes http ne peuvent pas accéder à ce scope ni à ses objets.
   * Il y a le scope SESSION qui est limité à la durée vie de la session http, et il privé à cette session http.  
     Ce qui veut dire que seules les requêtes http de la même session http peuvent accéder à ce scope et à ses objets.  
     Ce qui veut dire aussi que les objets attachés au scope SESSION persiste entre plusieurs requêtes http.  
     Si des requêtes http sont exécutées au même temps, il est possible d'avoir des accès concurrents sur les objets du scope SESSION.
   * Il y a le scope APPLICATION qui est limité à la durée vie de l'application web, et il privé à cette application web.  
     Ce qui veut dire que seules les requêtes http de l'application web peuvent accéder à ce scope et à ses objets.  
     Ce qui veut dire aussi que les objets attachés au scope SESSION persiste entre plusieurs requêtes et sessions http.  
     Il est possible d'avoir des accès concurrents sur les objets du scope APPLICATION.
2. **Scope REQUEST**

Le scope REQUEST est créé pour chaque requête http reçu par le conteneur de servlets et reste disponible jusqu'à la génération de la réponse http.  
Le scope REQUEST est disponible du moment où la Servlet cible de la requête http est exécutée (appel de la méthode service).  
Si des filtres doivent être exécutés en premier, alors le scope REQUEST est disponible du moment où le premier filtre est exécutée (appel de la méthode doFilter).  
Le scope REQUEST est représenté par une instance de la classe HttpServletRequest.

1. **Scope SESSION**

Le scope SESSION est créé par le conteneur de servlets lorsque l'application décide d'établir une session avec l'utilisateur.  
La session permet d'assurer un suivi des requêtes http de l'utilisateur.  
Le scope SESSION est représenté par une instance de la classe HttpSession.  
Cette instance est disponible en invoquant la méthode getSession du scope REQUEST.

1. **Scope APPLICATION**

Le conteneur de servlets crée un scope APPLICATION pour chaque application web.  
Le scope APPLICATION est unique pour une JVM.  
Le scope APPLICATION est représenté par une instance de la classe ServletContext.

Différence entre JSP et Servlet :

**L**a différence clé entre JSP et Servlet est la suivante: JSP est un langage de script pouvant générer du contenu Web dynamique, tandis que servlet est un programme Java déjà compilé et utilisé pour créer du contenu Web dynamique.

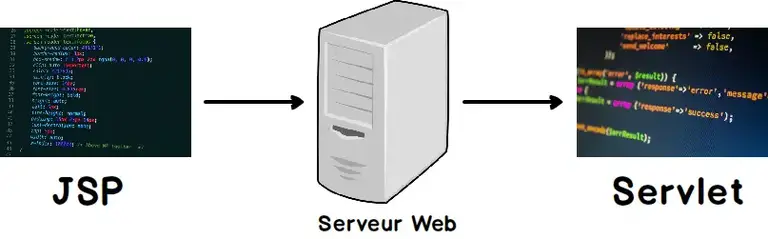


Figure :relation entre JSP et Servlet

|  |  |
| --- | --- |
| **Servlet** | **JSP** |
| C’est un code qui utilise l’approche de Java pour fonctionner. | Il s’agit d’un code basé sur HTML et comprend une approche basée sur les balises. JSP est une interface reposant sur les servlets. Les JSP sont des extensions de servlets permettant de minimiser les efforts des développeurs en matière d’écriture des interfaces utilisateur à l’aide de Java. |
| La programmation d’une servlet est plus difficile que programmer JSP car elle inclut du code HTML dans java. | JSP est comparativement moins difficile à coder car il inclut java dans html. |
| Les servlets jouent le rôle du contrôleur dans l’approche MVC. | JSP jouent le rôle d’une vue dans l’approche MVC dans le but d’afficher la sortie. |
| Les servlets sont plus rapides que JSP. | JSP prend plus de temps que les servlets car la première étape de son cycle de vie réalise la traduction du code JSP en code java, puis la compilation. |
| Les servlets acceptent tous les types de requêtes. | JSP accepte uniquement les requêtes http. |
| Dans les servlets, il est possible de remplacer la méthode service(). | Dans JSP, il n’est pas possible de remplacer la méthode service(). |
| Dans les servlets, la gestion de session n’est pas activée par défaut. Les utilisateurs de JS sont responsables de l’activer explicitement. | La gestion de session est activée automatiquement en cas de JSP. |
| Dans les servlets, tout, y compris la logique métier et la logique de présentation qui doivent être implémenté par les développeurs dans un seul fichier de servlet. | Dans JSP, la logique métier peut être séparée de la logique de présentation avec l’utilisation de javaBeans. |
| La modification dans le cas de servlets prend du temps car elle intègre le rechargement, la recompilation, le redémarrage, etc. du serveur. | La modification de JSP est rapide et facile à mettre en œuvre. Juste on clique sur le bouton d’actualisation. |
| Dans les servlets, il n’existe aucune façon pour exécuter un code JavaScript côté client. | La validation côté client peut être utilisée dans JSP. |
| Dans les servlets, les packages doivent être importés en haut. | Dans JSP, le package peut être importé n’importe où, en haut, au milieu ou en bas. |
| S’exécute sur un serveur Web de type Tomcat. | Les programmes JSP sont compilés dans des servlets Java avant leur exécution. Une fois compilé dans des servlets, le cycle de vie de JSP devient identique à celui d’un servlet. En revanche, JSP contient sa propre API pour le cycle de vie. |
| Reçoit les requêtes HTTP et renvoie des réponses HTTP. | Plus facile à développer (par rapport aux servlets) car il ressemble à HTML. |
| Les balises personnalisées ne peuvent pas être intégrées aux servlets. | Un des principaux avantages de JSP est qu’il peut être utilisé pour créer des balises personnalisées à l’aide de l’API JSP. Il existe un package séparé pour l’écriture de balises personnalisées pouvant être disponible sous la forme de composants réutilisables offrant une grande flexibilité. |
| Servlet a des méthodes de cycle de vie comme init(), service() et destroy(). | JSP présente des méthodes de cycle de vie telles que jspInit(), \_jspService() et jspDestroy(). |
| Servlet est écrit en Java et possède des API supplémentaires spécifiques à son type de traitement. Les servlets suivent toutes les techniques de la programmation orientée objet. | Les JSP utilisent les Javabeans dans des pages Web. |

Tableau : Table de comparaison entre Servlet et JSP

* **JSP**

JSP (JavaServer Pages) est une technologie de programmation web qui permet de créer des pages web dynamiques en utilisant des balises et du code Java. Les fichiers JSP sont des fichiers texte avec l'extension .jsp qui contient du HTML et des balises JSP. Les balises JSP sont utilisées pour insérer du code Java dans une page HTML.

Lorsqu'un navigateur demande une page JSP, le serveur web interprète la page, exécute le code Java et génère une page HTML dynamique qui est renvoyée au navigateur. Cela permet de créer des pages web dynamiques qui peuvent afficher des informations provenant de bases de données, de formulaires, etc.

Les JSP sont souvent utilisés avec les servlets Java pour créer des applications web dynamiques en utilisant le modèle MVC (Modèle-Vue-Contrôleur). Les JSP sont utilisés pour la partie Vue, qui affiche l'interface utilisateur, tandis que les servlets sont utilisés pour la partie Contrôleur, qui gère les actions de l'utilisateur et récupère les données nécessaires à partir du modèle (généralement une base de données ).

* **Les Java Beans**

Les Java Beans sont des composants logiciels réutilisables développés en Java, destinés à être intégrés dans des applications Java ou J2EE. Ils sont conçus pour être facilement manipulés et réutilisés par des outils de développement et des conteneurs d'applications.

Les Java Beans sont généralement des classes Java qui respectent certaines conventions, comme l'utilisation de constructeurs sans argument, l'implémentation de l'interface Serializable pour pouvoir être sauvegardées ou exceptionnelles entre différents serveurs, ou encore la définition de propriétés accessibles par des méthodes getter et poseur.

Les Java Beans peuvent être utilisés dans des applications de différents types, notamment des applications web, des applications de bureau ou des applications mobiles. Ils sont très couramment utilisés dans les environnements de développement Java EE pour faciliter la création de composants réutilisables et modulaires.

Les avantages des Java Beans sont nombreux, notamment leur capacité à réduire la complexité du développement d'applications, leur facilité de réutilisation et d'interopérabilité, ainsi que leur capacité à faciliter la conception de systèmes modulaires et évolutifs.

* **Les collections :**

Les collections sont des objets qui permettent de gérer des ensembles d'objets. Ces ensembles de données peuvent être définis avec plusieurs caractéristiques : la possibilité de gérer des doublons, de gérer un ordre de tri, etc. ...

Une collection est un regroupement d'objets qui sont désignés sous le nom d'éléments.

L'API Collections propose un ensemble d'interfaces et de classes dont le but est de stocker de multiples objets. Elle propose quatre grandes familles de collections, chacune définie par une interface de base :

List : collection d'éléments ordonnés qui accepte les doublons

Set : collection d'éléments non ordonnés par défaut qui n'accepte pas les doublons

Map : collection sous la forme d'une association de paires clé/valeur

Queue et Deque : collections qui stockent des éléments dans un certain ordre avant qu'ils ne soient extraits pour traitement .

* **Présentation du framework collection**

Les tableaux ne peuvent pas répondre à tous les besoins de stockage d'un ensemble d'objets et surtout ils manquent de fonctionnalités. La large diversité d'implémentations proposées par l'API Collections de Java permet de répondre à la plupart des besoins.

Avant Java 1.2 qui a introduit l'API Collections, seules quelques classes du package java.util permettaient de stocker et de gérer des éléments : Array, Vector, Stack, Hashtable, Properties et BitSet. L'interface Enumeration permet de parcourir le contenu de ces objets.

L'API Collections propose de structurer et de définir un ensemble d'interfaces et de classes de type collection. Les collections sont des conteneurs qui permettent de regrouper des objets en une seule entité.

Java propose l'API Collections qui offre un socle riche et des implémentations d'objets de type collection enrichies au fur et à mesure des versions de Java.

L'API Collections possède deux grandes familles chacune définies par une interface :

* java.util.Collection : pour gérer un groupe d'objets
* java.util.Map : pour gérer des éléments de type paires de clé/valeur

Une collection permet de stocker un groupe d'éléments en respectant certaines fonctionnalités selon l'implémentation : de base, elle permet d'ajouter, de supprimer, d'obtenir et de parcours ses éléments.

Les interfaces et les classes de l'API Collections qui ne proposent pas de gestion des accès concurrents sont dans le package java.util. Java 5 propose plusieurs collections dans le package java.util.concurrent telles que CopyOnWriteArrayList, ConcurrentHashMap ou CopyOnWriteArraySet qui permettent des modifications lors de leur parcours.

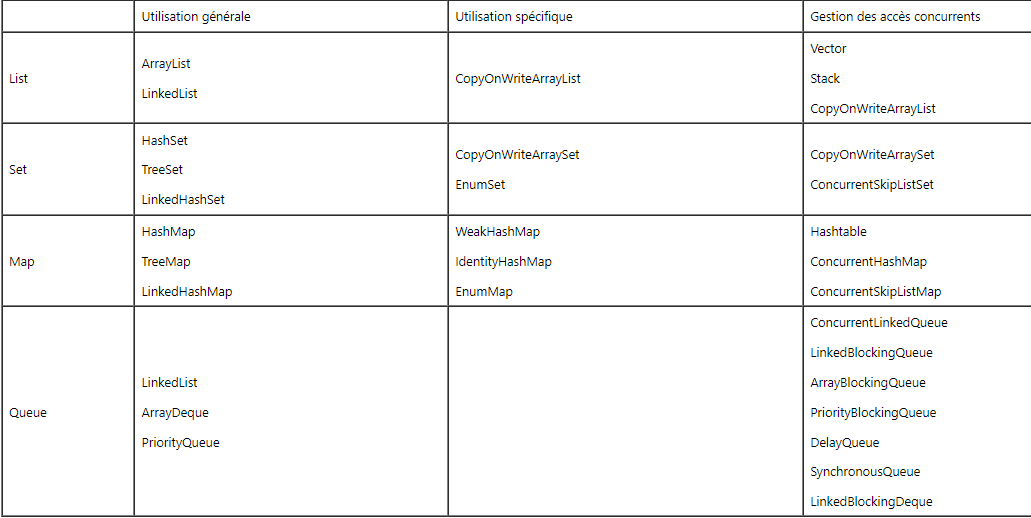
****Les fonctionnalités des collections sont définies dans cinq interfaces de base : Collection, List, Set, Map, Queue.

Tableau :Les fonctionnalités des collections

Elle définit enfin :

* deux interfaces pour le parcours de certaines collections : Iterator et ListIterator.
* une interface et une classe pour permettre le tri de certaines collections : Comparable et Comparator
* des classes utilitaires : Arrays, Collections

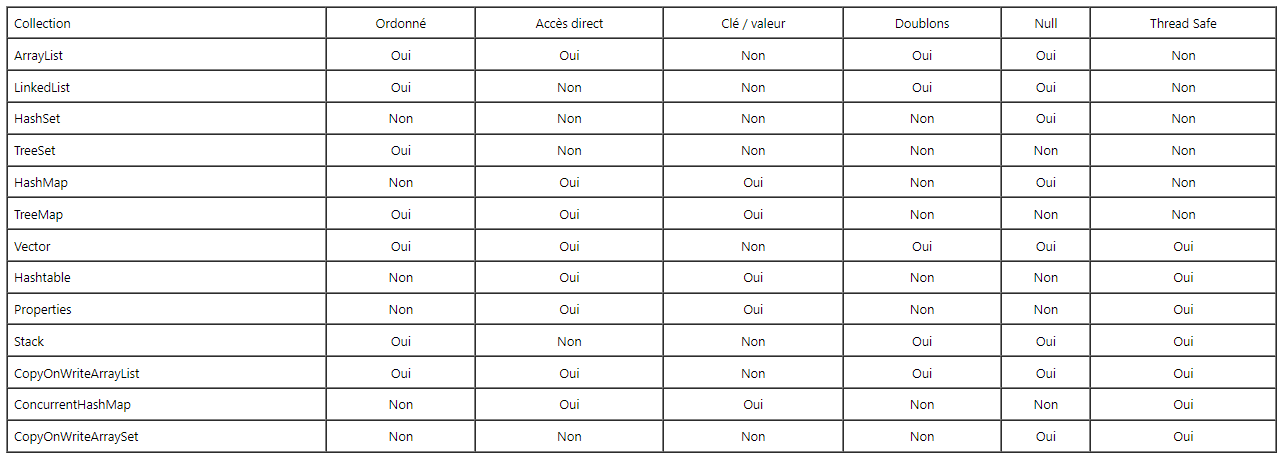
Le framework Collections propose plusieurs implémentations possédant chacune un comportement et des fonctionnalités particulières.

Tableau : Collection :un comportement et des fonctionnalités particulières