**Compte rendu**

**Matiere :Atelier\_JEE**

**Elaboré par :**

**Feki mohamed feres**

**Ayadi omar**

**Ayadi doha**

**Ben amor emna**

Table de matiers

[Table de matiers 2](#_Toc193056118)

[Atelier 01 7](#_Toc193056119)

[I. Réalisation 7](#_Toc193056120)

[1. Préparer l’environnement de développement 7](#_Toc193056121)

[a. Créer un dossier de travail : 7](#_Toc193056122)

[b. Installer JDK 8 : 7](#_Toc193056123)

[c. Installer Eclipse : 7](#_Toc193056124)

[d. Configurer Tomcat 9 : 7](#_Toc193056125)

[2. Exécuter une application web 7](#_Toc193056126)

[II. Les activités 9](#_Toc193056127)

[1. Activite 1.1 : 9](#_Toc193056128)

[2. Activite 1.2: 10](#_Toc193056130)

[3. Activite 1.3: 11](#_Toc193056131)

[11](#_Toc193056132)

[4. Activite 1.4: 14](#_Toc193056133)

[14](#_Toc193056134)

[Atelier 02 15](#_Toc193056135)

[I. Réalisation 15](#_Toc193056136)

[1. Mise en Place des JSP 15](#_Toc193056137)

[2. Gestion des Formulaires et Communication avec les Servlets 16](#_Toc193056138)

[II. Les activités 17](#_Toc193056139)

[1. Activite 2.1: 17](#_Toc193056140)

[2. Activite 2.2: 18](#_Toc193056141)

[19](#_Toc193056142)

[3. Activite 2.3: 19](#_Toc193056143)

[Atelier 03 20](#_Toc193056144)

[I. Réalisation 20](#_Toc193056145)

[II. Les activités 23](#_Toc193056146)

[1. Activite 3.1: 23](#_Toc193056147)

|  |  |
| --- | --- |
| Tp 4 | Omar ayadi  doha ayadi  Ben amor emna  Feki mohamed feres |
| Activite 4.1 | Ayadi doha |
| Activite 4.2 | Ben amor emna |
| Activite 4.3 | Feki mohamed feres |

Liste figures

[Figure 1:1er affichage 5](#_Toc193056650)

[Figure 2:Code du methode doGet 6](#_Toc193056651)

[Figure 3 Affichage du methode doGet 6](#_Toc193056652)

[Figure 4:Affichage grâce à l'objet PrintWriter 6](#_Toc193056653)

[Figure 5:Affichage au travers une requete 6](#_Toc193056654)

[Figure 6:Affichage des resultas passe en parametre dans la requette 7](#_Toc193056655)

[Figure 7:Resultat du servlete CalculDeMonImc 7](#_Toc193056656)

[Figure 8:Affichage du servlet Renseignement 8](#_Toc193056657)

[Figure 9:Code du classe IMC 8](#_Toc193056658)

[Figure 10:Code du methodes doGet pour s'appuie sur un object de la classe IMC 9](#_Toc193056659)

[Figure 11:Code de connexion.html 9](#_Toc193056660)

[Figure 12:Class User 10](#_Toc193056661)

[Figure 13:Class UserList 10](#_Toc193056662)

[Figure 14:Servlet Connexion.java 11](#_Toc193056663)

[Figure 15:Methodes init()et service(HttpServletRequest request, HttpServletResponse response) 12](#_Toc193056664)

[Figure 16:code page Jsp 13](#_Toc193056665)

[Figure 17:code du methode POST du servlet ConnexionAction 14](#_Toc193056666)

[Figure 18:code CSS 14](#_Toc193056667)

[Figure 19:code du fichier Premier.JSP 15](#_Toc193056668)

[Figure 20:code du connexion.jsp 16](#_Toc193056669)

[Figure 21:visibilte du requette http 17](#_Toc193056670)

[Figure 22:visibilte du session 17](#_Toc193056671)

[Figure 23:visibilite du application 17](#_Toc193056672)

[Figure 24:code du classe Etudiant 18](#_Toc193056673)

[Figure 25:code du methodes doGet du servlet EtudiantManager 18](#_Toc193056674)

[Figure 26:code du fichier affichageEtudiant.jsp 19](#_Toc193056675)

[Figure 27:code du fichier affichageEtudiant2.jsp 19](#_Toc193056676)

[Figure 28:code du fichier ListeProduit.jsp 20](#_Toc193056677)

[Figure 29:l'execution de ListeProduit.jsp 20](#_Toc193056678)

[Figure 30:code du servlet Panier.java 21](#_Toc193056679)

[Figure 31:l'execution du servlet Panier.java 21](#_Toc193056680)

[Figure 32:code du fichier Afficher.jsp 22](#_Toc193056681)

[Figure 33:l'execution du fichierAfficher.jsp 22](#_Toc193056682)

[Figure 34code du fichier Deconnexion.jsp 23](#_Toc193056683)

[Figure 35:l'execution du fichier Deconnexion.jsp 23](#_Toc193056684)

Atelier 01

1. Réalisation
2. Préparer l’environnement de développement
3. Créer un dossier de travail :

D:\Atelier\_JEE.

1. Installer JDK 8 :

Télécharger depuis [Oracle](https://www.oracle.com/java/technologies/downloads/#java8-windows) et configurer la variable JAVA\_HOME dans le PATH.

1. Installer Eclipse :

Télécharger Eclipse 2019-12, choisir la version "Web & Entreprise" et installer sous

D:\Atelier\_JEE\EDI.

1. Configurer Tomcat 9 :

Télécharger depuis [Apache Tomcat](https://tomcat.apache.org/download-90.cgi), extraire dans D:\Atelier\_JEE\Servers et l'ajouter à Eclipse

via Preferences > Server > Runtime Environments

1. Exécuter une application web

Suite au déploiement et au lancement de la commande "Run As \ Run On Server", une

page web statique s'affiche dans le navigateur, affichant le texte : "Première page web

statique"

Figure :1er affichage

La servlet traite avec succès les requêtes correspondant aux URL patterns /HelloWorld et \*.do.

Lors de l’exécution de la méthode doGet() est correctement, ce qui entraîne

l'affichage du message "bonjour" dans le navigateur, validant ainsi le bon

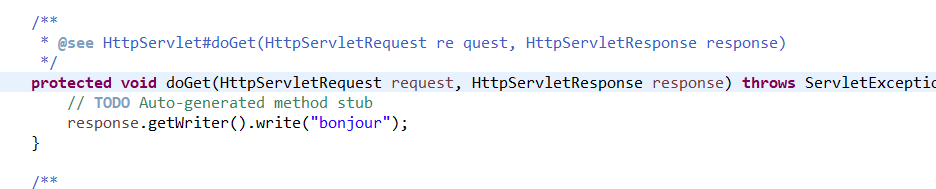
fonctionnement de la servlet. 

Figure :Code du methode doGet

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

Figure Affichage du methode doGet

Une image contenant texte, capture d’écran, Police, logiciel

Le contenu généré par l’IA peut être incorrect.La servlet produit du contenu HTML directement dans la réponse grâce à l'objet PrintWriter.

Figure :Affichage grâce à l'objet PrintWriter

L’appel à **request.getRequestDispatcher().forward()** permet à la servlet de

transférer la requête vers un fichier statique pour afficher son contenu

Figure :Affichage au travers une requete

Une image contenant texte, capture d’écran, logiciel, Police

Le contenu généré par l’IA peut être incorrect.

On peut envoyer des informations à une servlet via la méthode GET. Cela permet d’afficher dynamiquement des données extraites de l’URL et facilite l'interaction entre les pages HTML et les servlets.

Une image contenant texte, capture d’écran, Police, ligne

Le contenu généré par l’IA peut être incorrect.Une image contenant texte, capture d’écran, Police, logiciel

Le contenu généré par l’IA peut être incorrect.

Figure :Affichage des resultas passe en parametre dans la requette

1. Les activités
2. Activite 1.1 :

Resultat du développement de la servlet CalculDeMonImc pour calculer l'IMC à partir du

poids et de la taille fournis via une requête HTTP

Une image contenant texte, capture d’écran, logiciel, Page web

Le contenu généré par l’IA peut être incorrect.

Figure :Resultat du servlete CalculDeMonImc

Lors des tests, la requête GET envoie correctement les paramètres poids et taille, et la servlet affiche l'IMC calculé.

Une image contenant texte, capture d’écran, Police, logiciel

Le contenu généré par l’IA peut être incorrect.Une image contenant texte, capture d’écran, Police, ligne

Le contenu généré par l’IA peut être incorrect.

Figure :Affichage du servlet Renseignement

La différence principale entre **HTTP GET** et **HTTP POST** est :

* **GET** : Les données sont envoyées dans l'URL (visible dans la barre d'adresse du navigateur). Utilisé pour récupérer des informations sans modifier quoi que ce soit.
* **POST** : Les données sont envoyées dans le corps de la requête (non visible). Utilisé pour envoyer des informations .

1. Activite 1.2:

Une image contenant texte, capture d’écran, Police, nombre

Le contenu généré par l’IA peut être incorrect.

Figure :Code du classe IMC

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

Figure :Code du methodes doGet pour s'appuie sur un object de la classe IMC

Q3 \a:

L’objet de la classe **Imc** est déclaré en tant que variable locale dans la méthode **doGet** de

la servlet.

Q3 \b:

L’objet **Imc** est construit dans la méthode **doGet** après la récupération et la

conversion des paramètres poids et taille depuis la requête.

Q3 \c:

La portée de cet objet est **locale à la méthode doGet**. Il n’est accessible qu’à l’intérieur

de cette méthode, ce qui signifie qu’il existe uniquement pendant le traitement de la

requête en cours.

1. Activite 1.3:

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect. une interface  "Connexion.html" qui permet à un utilisateur de saisir ses paramètres deconnexion

Figure :Code de connexion.html

une classe "**User**" caractérisé par un login et un password  avec  un constructeur sans

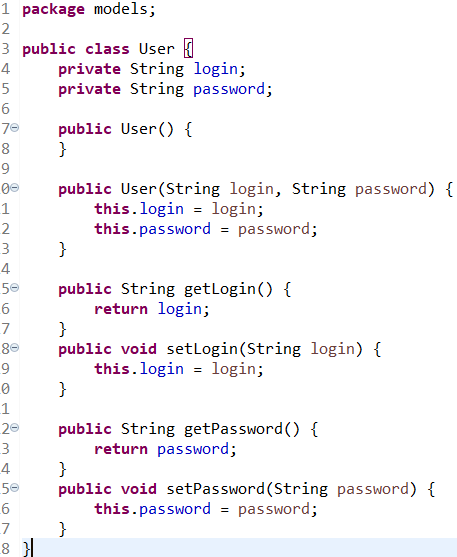
arguments et des getters et des setters.

Figure :Class User

une classe "**UsersList**" et qui permet de gérer une collection de User et initilaiser cette

collection par un ensemble d'utilisateurs de test.  La classe "**UsersList**" présente une méthode

pour vérifier l'existence d'un user donné dans la collection.

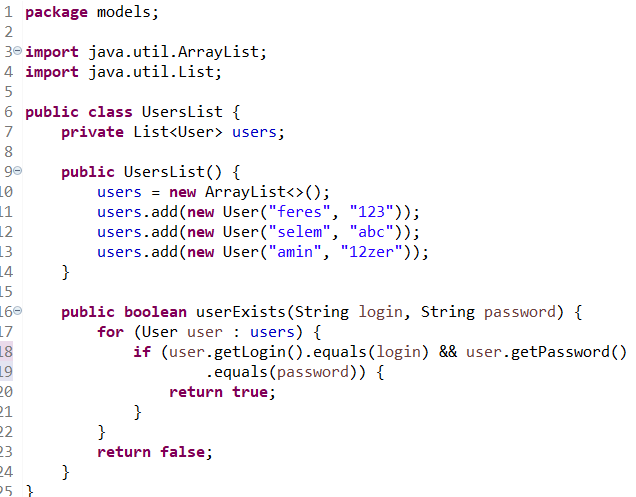


Figure :Class UserList

une servlet "**Connexion.java**" qui permet de récupérer les données du formulaire (en mode

POST) et vérifier si l'utilisateur existe. Si l'utilisateur existe, il sera redirigé vers l'interface

"**Acceuil.html"** sinon un message d'erreur sera affiché

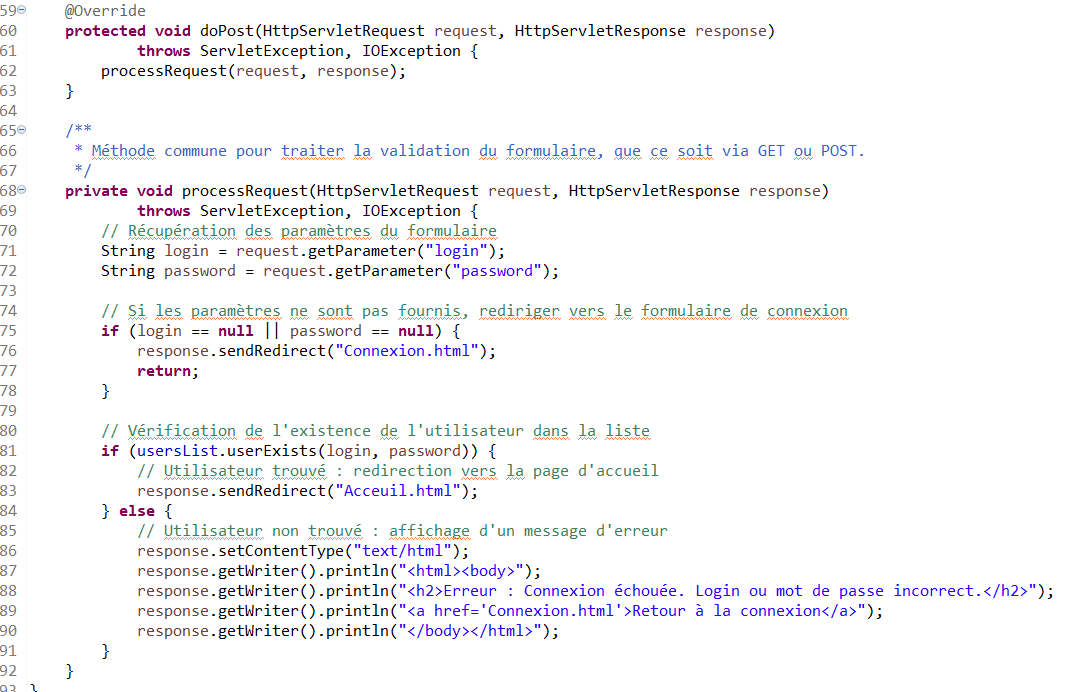


Figure :Servlet Connexion.java

1. Activite 1.4:

init : méthode invoquée lors du premier appel (utile pour les traitements qui s’exécutent une

seule fois).

service: méthode invoquée à chaque appel à la servlet et qui oriente le processus d’exécution

vers la méthode adéquate « doGet » ou « doPost »

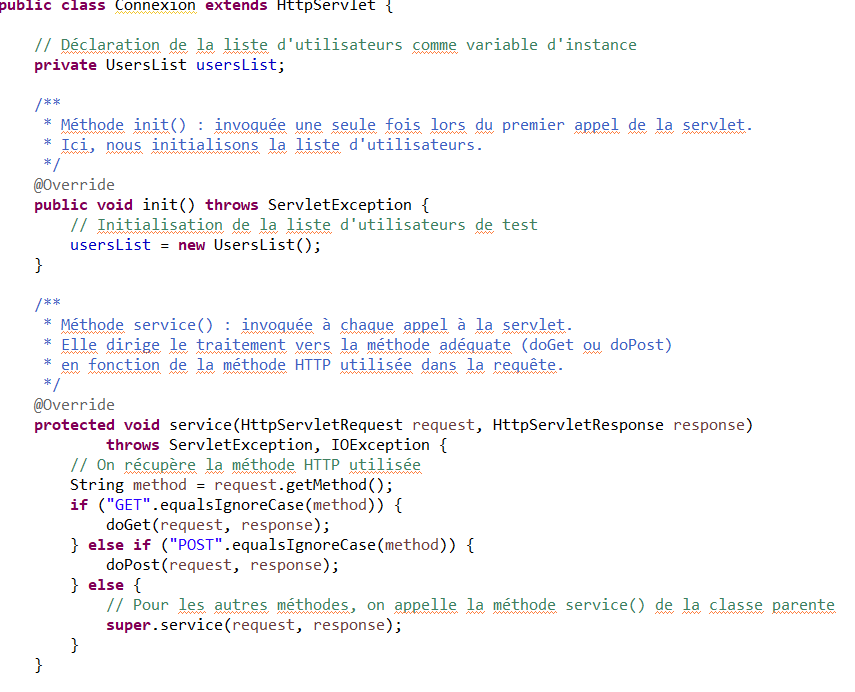


Figure :Methodes init()et service(HttpServletRequest request, HttpServletResponse response)

Atelier 02

1. Réalisation
2. Mise en Place des JSP

Créez une page JSP qui intègre du code HTML et du code Java par l’utilisation des blocs

de scriptlets

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

Figure :code page Jsp

1. Gestion des Formulaires et Communication avec les Servlets

Créez une page de connexion contenant un formulaire qui demande le login et le mot de

passe. Le formulaire est configuré pour envoyer les données à une servlet via la méthode

POST comme la servlet ConnexionAction

Une image contenant texte, capture d’écran, Police, nombre

Le contenu généré par l’IA peut être incorrect.

Figure :code du methode POST du servlet ConnexionAction

Une image contenant texte, capture d’écran, Police, écriture manuscrite

Le contenu généré par l’IA peut être incorrect.Utilisation du fichier css pour afficher les erreurs en rouge

Figure :code CSS

1. Les activités
2. Activite 2.1:

Affiche en **bleu** le texte d’une cellule ayant une valeur paire

-Et Affiche en **rouge** le texte d’une cellule ayant une valeur impaire

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

Figure :code du fichier Premier.JSP

1. Activite 2.2:

Créer une page JSP nommée connexion.jsp contenant un formulaire avec deux champs (login

et password) et un bouton de soumission et afficheage les erreurs et pré-remplir les champs

en cas de validation incorrecte et utilisation de ArrayList<String> pour stocker plusieurs erreurs à la fois.

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

Figure :code du connexion.jsp

1. Activite 2.3:

<http://localhost:8080/web_app_tp02/HelloWorldScope?prenom=Ali>

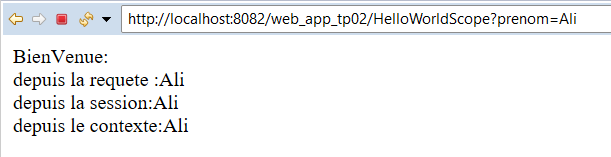


Figure :visibilte du requette http

<http://localhost:8080/web_app_tp02/HelloWorldScope>

Une image contenant texte, capture d’écran, Police, ligne

Le contenu généré par l’IA peut être incorrect.

Figure :visibilte du session

<http://localhost:8080/web_app_tp02/HelloWorldScope>

dans autre navigateur :

Une image contenant texte, capture d’écran, Police, ligne

Le contenu généré par l’IA peut être incorrect.

Figure :visibilite du application

**Request :** La valeur est disponible uniquement dans la requête en cours. Dès que la requête est terminée, elle n'est plus accessible.

**Session :** La valeur est conservée pour toute la durée de la session utilisateur. Elle persiste entre plusieurs requêtes du même navigateur.

**Application :** La valeur est globale à l'ensemble de l'application et accessible par tous les utilisateurs, quels que soient leur navigateur ou leur session.

Atelier 03

1. Réalisation

Une classe Java **Etudiant** a été définie dans le package beans, respectant les conventions d’un Java Bean (constructeur sans paramètres, getters et setters)

Une image contenant texte, capture d’écran, affichage, logiciel

Le contenu généré par l’IA peut être incorrect.

Figure :code du classe Etudiant

la servlet EtudiantManager a été développée pour instancier des objets Etudiant et les

stocker dans différentes portées (request, session, application).

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

Figure :code du methodes doGet du servlet EtudiantManager

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.La page affichageEtudiant.jsp récupère et affiche les données des objets Etudiant à l’aide des méthodes getAttribute().

Figure :code du fichier affichageEtudiant.jsp

Une autre page affichageEtudiant2.jsp utilise la balise <jsp:useBean> pour récupérer automatiquement les objets dans les différentes portées.



Figure :code du fichier affichageEtudiant2.jsp

1. Les activités
2. Activite 3.1:

On se propose de gérer une boutique web de vente de produits informatiques.

Chaque client de la boutique se connecte et ouvre une session pour pouvoir ajouter des

produits à son panier avant de régler le montant de ses achats.

 l’interface ***ListeProduits.jsp***  permet de lister les produits de la boutique:

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

Figure :code du fichier ListeProduit.jsp



Figure :l'execution de ListeProduit.jsp

Une fois l’utilisateur sélectionne un produit et clique sur le bouton d’envoi, les paramètres de la requête seront envoyés à la servlet ***Panier.java*** qui permet d’ajouter le produit sélectionné au panier (Utilisation des variables de session)

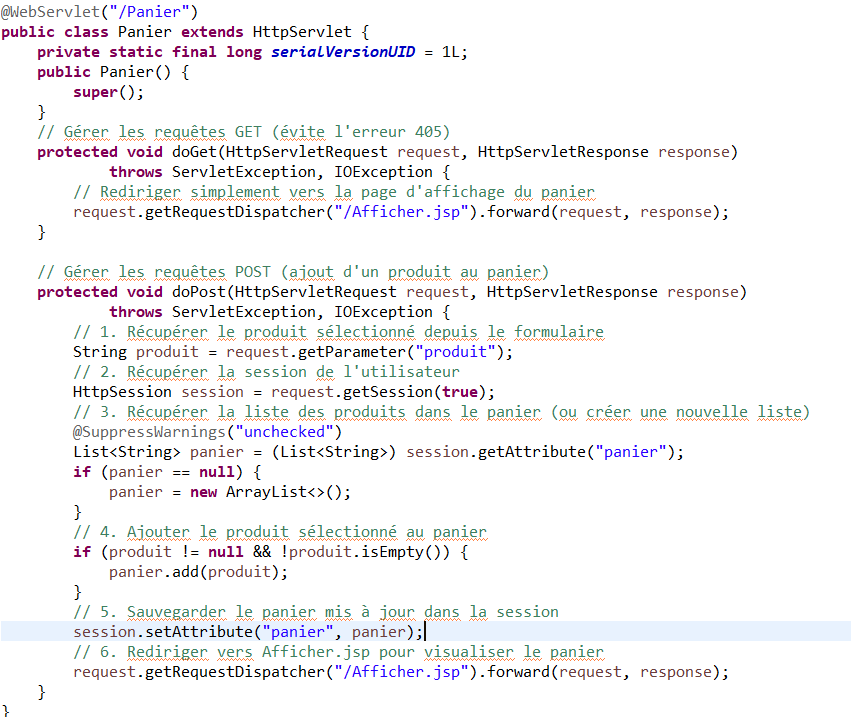


Figure :code du servlet Panier.java



Figure :l'execution du servlet Panier.java

la servlet ***Panier.java***  transmet le contrôle à la JSP ***Afficher.jsp*** qui permet d’afficher le contenu du panier

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

Figure :code du fichier Afficher.jsp

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

Figure :l'execution du fichierAfficher.jsp

L’utilisateur a le choix de retourner à la page ***ListeProduits.jsp***ou bien se déconnecter et accéder à la page ***Deconnexion.jsp*** qui permet de détruire la session

Une image contenant texte, capture d’écran, Police, logiciel

Le contenu généré par l’IA peut être incorrect.

Figure code du fichier Deconnexion.jsp

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

Figure :l'execution du fichier Deconnexion.jsp

Atelier 04

1. Réalisation
2. Préparer l’environnement de développement

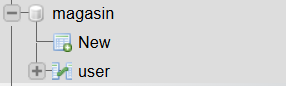


Figure : Création de la base de données « MAGASIN » et de la table « User »

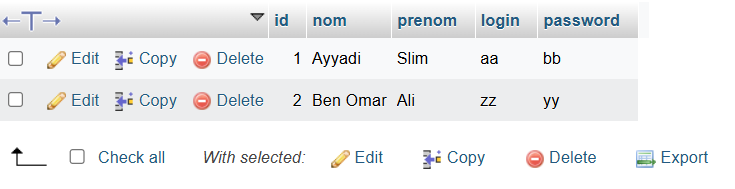
1. nsertion des enregistrements

Figure :Insertion des enregistrements

1. Gérer la partie « Metier »

La couche Métier encapsule la logique de traitement des utilisateurs et interagit avec la base de données.

Les tests prouvent que les services métier sont fiables et fonctionnent.

1. Gérer la partie « web »

Suite à l’execution de la vue « UserConnexion.jsp » :

Figure :resultat execution vue UserConnexion

\*Si les identifiants sont valides, l’utilisateur est redirigé vers la page d’accueil.

\*Si les identifiants sont incorrects, l’utilisateur reste sur la page de connexion avec un message d’erreur.

Suite à l’execution de la vue « accueil.jsp » :

****

Figure :esultat execution accueil.jsp

Affichage des liens "Ajouter un utilisateur" et "Liste des utilisateurs".

L’inclusion de la page entete.jsp est fonctionnelle : la session est bien reconnue et la barre de navigation s'affiche correctement.

Resultat lien :Ajouter un utilisateur :

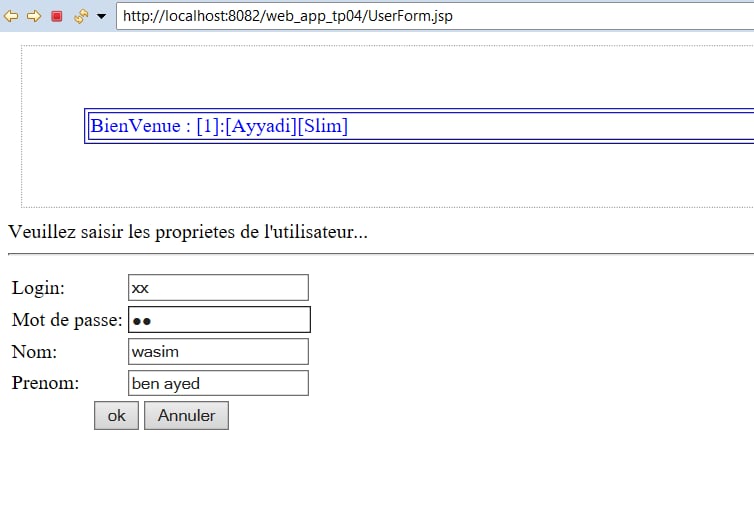


Figure :Resultat lien Ajouter un utilisateur

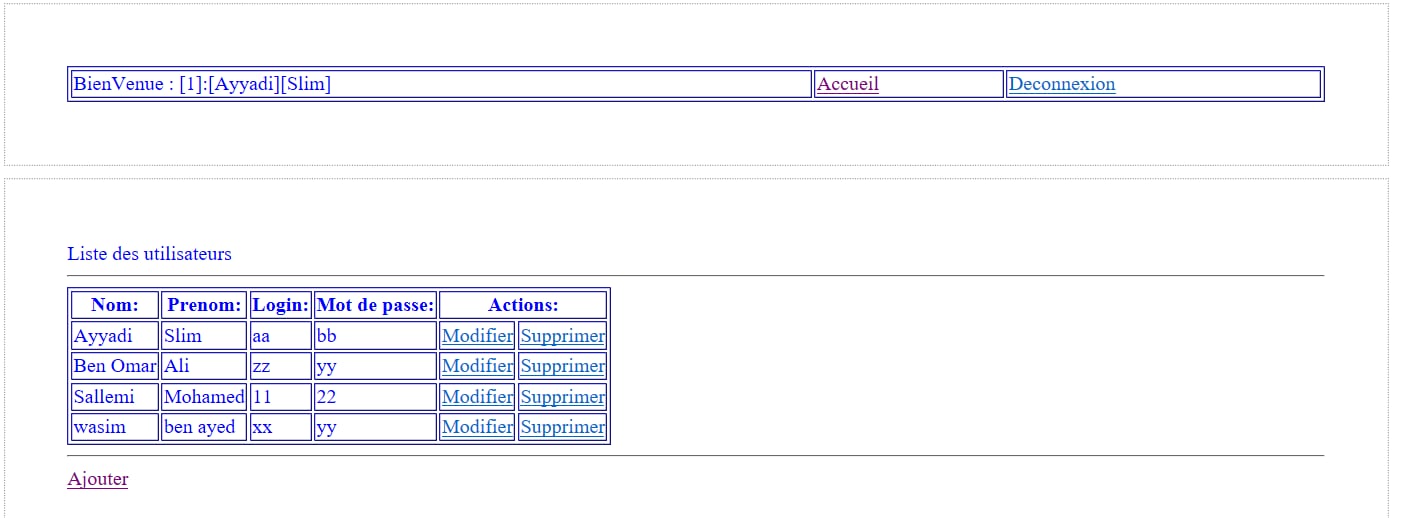


Figure :Liste des Utilisateurs

1. Exercice
2. UserList.jsp :

<%@ page language="java" import="java.util.ArrayList, metier.User"%>

<%@ include file=*"entete.jsp"*%>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">

<html>

<head>

<meta http-equiv=*"Content-Type"*

content="text/html; charset=windows-1256">

<title>Consultation</title>

</head>

<body>

<h1>Liste des utilisateurs</h1>

<hr>

<!-- Formulaire de recherche -->

<form action="UserListController" method="GET">

<label for=*"searchName"*>Nom:</label> <input type=*"text"*

name=*"searchName"* id=*"searchName"* /> <label for=*"searchPrenom"*>Prénom:</label>

<input type=*"text"* name=*"searchPrenom"* id=*"searchPrenom"* /> <input

type="submit" value="Rechercher" />

</form>

<hr>

<!-- Affichage des utilisateurs -->

<table border=*"1"*>

<tr>

<th>Nom:</th>

<th>Prénom:</th>

<th>Login:</th>

<th>Mot de passe:</th>

<th colspan=*"2"*>Actions:</th>

</tr>

<%

ArrayList<User> users = (ArrayList<User>) session.getAttribute("listOfUsers");

**if** (users != **null**) {

**for** (**int** i = 0; i < users.size(); i++) {

User user = users.get(i);

out.println("<tr>");

out.println("<td>" + user.getNom() + "</td>");

out.println("<td>" + user.getPrenom() + "</td>");

out.println("<td>" + user.getLogin() + "</td>");

out.println("<td>" + user.getPassword() + "</td>");

out.print("<td><a href='UserEditionController?id=" + user.getId()

+ "&mode=Edition'>Modifier</a></td>");

out.println("<td><a href='UserEditionController?id=" + user.getId()

+ "&mode=Suppression' onclick='return confirm(\"Voulez-vous vraiment supprimer cet utilisateur ?\")'>Supprimer</a></td>");

out.println("</tr>");

}

}

%>

</table>

<hr>

<a href=*"UserForm.jsp"*>Ajouter un utilisateur</a>

</body>

</html>

1. Modification UserListController.java :

Le code devient comme cela :

package web;

import java.io.IOException;

import java.util.List;

import javax.servlet.ServletConfig;

import javax.servlet.ServletException;

import javax.servlet.annotation.WebServlet;

import javax.servlet.http.HttpServlet;

import javax.servlet.http.HttpServletRequest;

import javax.servlet.http.HttpServletResponse;

import javax.servlet.http.HttpSession;

import metier.User;

import metier.UserMetierImpl;

import metier.UserMetierInterface;

/\*\*

\* Servlet implementation class UserListController

\*/

@WebServlet("/UserListController")

public class UserListController extends HttpServlet {

private static final long serialVersionUID = 1L;

// Déclarer un objet "métier"

UserMetierInterface metier = null;

/\*\*

\* @see HttpServlet#HttpServlet()

\*/

public UserListController() {

super();

}

/\*\*

\* @see Servlet#init(ServletConfig)

\*/

public void init(ServletConfig config) throws ServletException {

// instancier le métier

metier = new UserMetierImpl();

}

/\*\*

\* @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)

\*/

protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

// Récupérer les paramètres de recherche

String searchName = request.getParameter("searchName");

String searchPrenom = request.getParameter("searchPrenom");

// Si les paramètres sont null, les initialiser à des chaînes vides

if (searchName == null) searchName = "";

if (searchPrenom == null) searchPrenom = "";

// Appeler la couche métier pour récupérer les utilisateurs filtrés

List<User> users = metier.searchUsers(searchName, searchPrenom);

// Récupérer une session

HttpSession session = request.getSession(true);

session.setAttribute("listOfUsers", users);

// Rediriger vers la page UserList.jsp

request.getRequestDispatcher("UserList.jsp").forward(request, response);

}

/\*\*

\* @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)

\*/

protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {

doGet(request, response);

}

@Override

protected void service(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

HttpSession session = request.getSession();

User u = (User) session.getAttribute("user");

if (u == null) {

request.getRequestDispatcher("UserConnexion.jsp").forward(request, response);

} else {

super.service(request, response);

}

}

}

1. Modification de la couche métier (UserMetierImpl.java)

Dans la classe UserMetierImpl, nous allons ajouter une méthode searchUsers qui effectuera la recherche des utilisateurs en fonction des critères fournis.

Code modifié de UserMetierImpl.java :

import java.util.ArrayList;

import java.util.List;

public class UserMetierImpl implements UserMetierInterface

// Cette méthode va retourner les utilisateurs filtrés en fonction du nom et du prénom

@Override

public List<User> searchUsers(String name, String prenom) {

List<User> filteredUsers = new ArrayList<>();

// Par exemple, supposons que tu as une base de données ou une liste d'utilisateurs

// Dans cet exemple, je vais utiliser une liste statique de données

// Liste simulée d'utilisateurs

List<User> allUsers = listUsers(); // Tu peux remplacer cette méthode par ta propre logique pour récupérer les utilisateurs depuis ta BD

for (User user : allUsers) {

// Filtrer les utilisateurs en fonction du nom et du prénom

if (user.getNom().contains(name) && user.getPrenom().contains(prenom)) {

filteredUsers.add(user);

}

}

return filteredUsers;

}

@Override

public List<User> listUsers() {

// Logique pour récupérer tous les utilisateurs de la base de données

List<User> users = new ArrayList<>();

// Ajouter des utilisateurs fictifs (remplace ceci par ta logique réelle)

users.add(new User(1, "Mohamed", "Ben Saleh", "11", "22"));

users.add(new User(2, "Sami", "Sallemi", "12", "23"));

// ...

return users;

}

}

Ajouter dans UserMetierInterface:

public User getUserById (int id);

List<User> searchUsers(String name, String prenom);

}

****

Figure :interface de recherche

****

Figure :Resultat de Recherche

1. Les activités
2. Activite 4.1:

L'objectif de cet exercice est d'améliorer l'organisation et la structure du projet "web\_app\_tp04" en respectant les bonnes pratiques de conception logicielle, notamment le principe de séparation des responsabilités

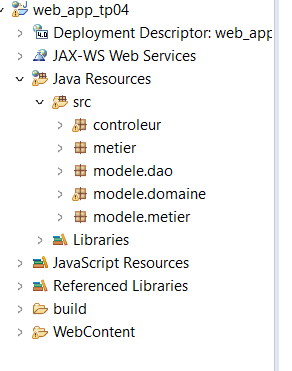


Figure :nouvelle organisation du project

1. Activite 4.2:

L'objectif de cet exercice est d'étendre le projet "**web\_app\_tp04\_bis**" en ajoutant la gestion des **produits**, en suivant la même architecture et les bonnes pratiques mises en place pour la gestion des utilisateurs.  
Concrètement, il s'agit de :

**Créer une nouvelle entité** Produit dans le package modele.domaine, représentant les produits du magasin avec des attributs (id, code, designation, prix).

**Créer une table "produit"** dans la base de données **MAGASIN**, correspondant à cette nouvelle classe.

**Ajouter deux liens** sur la page d'accueil :

Un lien pour **ajouter** un nouveau produit.

Un lien pour **afficher** la **liste des produits** existants.

**Créer un formulaire** formProduit.jsp pour la saisie des nouveaux produits.

**Développer une servlet** ProduitEditionController.java qui récupère les données du formulaire et utilise une classe ProduitMetierImpl pour traiter l'ajout d'un produit en base de données.

**Suivre la même logique** et la même organisation que pour la gestion des utilisateurs afin d'assurer **cohérence**, **réutilisabilité** et **maintenabilité** du projet.

Cet exercice permet ainsi de **consolider la maîtrise de l'architecture MVC** (Modèle-Vue-Contrôleur) et de l'**interaction entre Java EE, JSP et base de données** dans une application web.

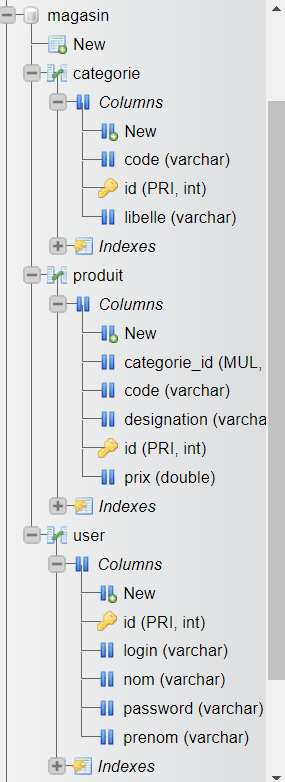


Figure :la nouvelle structure du base de donne

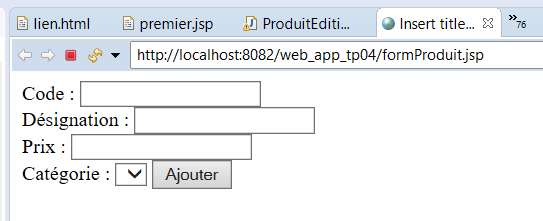


Figure :formulaire dajouter produit

1. Activite 4.3:

Dans le cadre de l'amélioration du projet "web\_app\_tp04\_bis", l'objectif de cet exercice est de mettre en place la gestion des catégories de produits en plus des produits eux-mêmes.  
Pour cela, plusieurs étapes sont demandées :

Achever l'exercice précédent pour assurer une base fonctionnelle.

Créer une nouvelle table Categorie en base de données pour organiser les produits par familles, avec des colonnes adaptées (id, code, libelle).

Définir un Java Bean Categorie dans le package modele.domaine pour représenter cette nouvelle table.

Développer les fonctionnalités métiers dans le package modele.metier afin de pouvoir ajouter, éditer, supprimer et afficher des catégories.

Créer les vues (JSP) permettant de gérer l'interface utilisateur pour les catégories.

Modifier la table Produit pour ajouter une colonne categorie\_id (clé étrangère vers Categorie) et adapter le code Java en conséquence pour intégrer cette nouvelle relation.

Ce travail a pour but de structurer le projet selon une architecture plus réaliste et évolutive, en respectant le modèle de base de données relationnelle et en appliquant une approche propre à la programmation orientée objet et aux applications web dynamiques

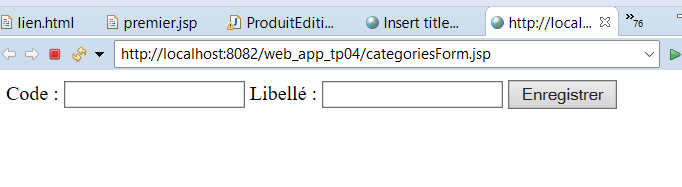


Figure :interface dajouter categorie



Figure :interface de modifier categorie par id

Atelier 7 :

Enregistrer un pilote (driver-jdbc) nommé « com.mysql.driver » associé au module « com.mysql »

Une image contenant texte, capture d’écran, logiciel, Système d’exploitation

Le contenu généré par l’IA peut être incorrect.

Creation un compte administrateur pour accéder à la console d’administration de WildFly

de type « Management User » nom d’utilisateur :jbossadmin

Une image contenant texte, capture d’écran, Police

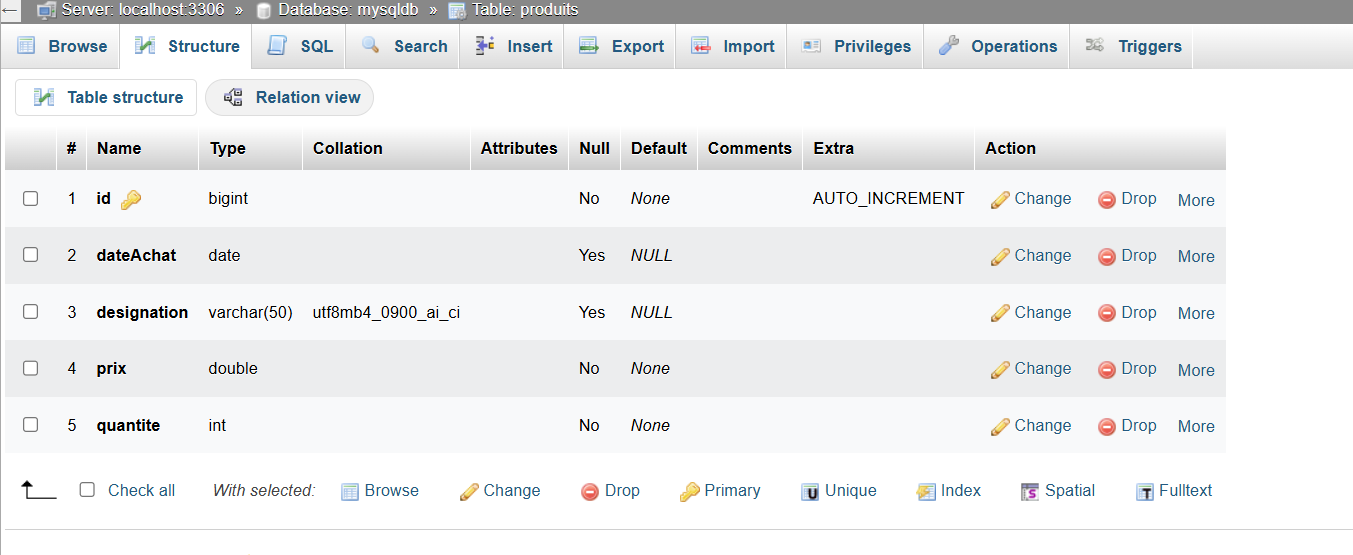
Le contenu généré par l’IA peut être incorrect.

Creation du fichier persistance.xml

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

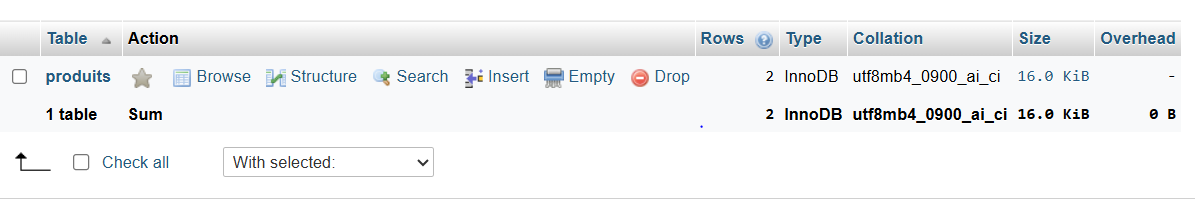
Resultat de la personnalisation de la structure de la table « produit » apartir des annotations de l’entite JPA :



Atelier 8 :

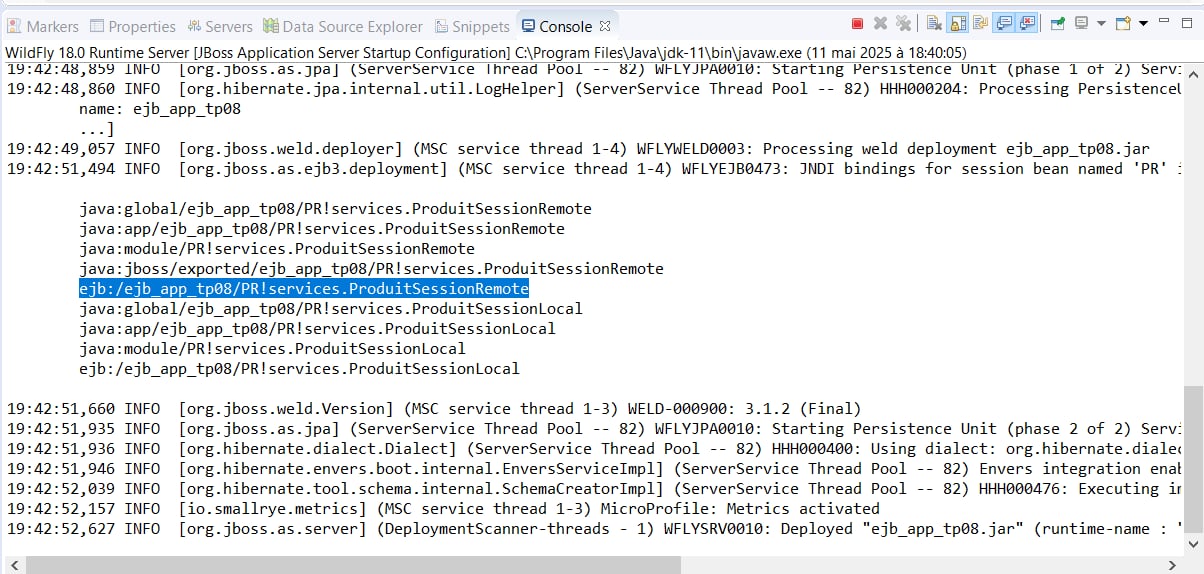
1. Création d’un EJB Entity couche « domaine »

Creation de la table produits



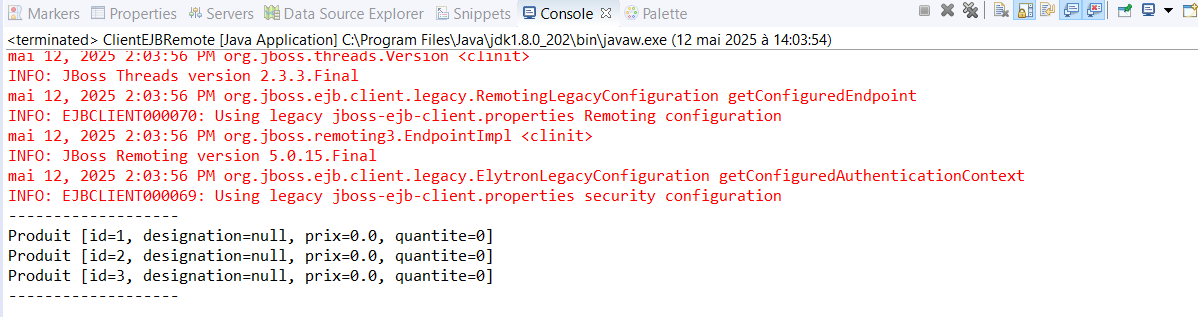
1. Création d’un EJB Session

Resultat de la publication du bean Session



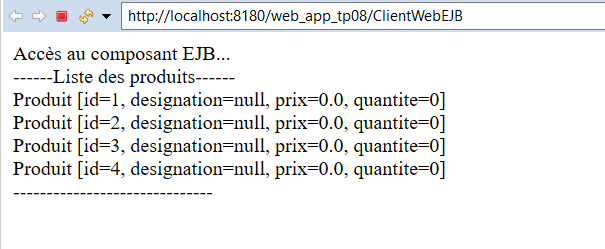
1. Création d’un Client Java

Affichage du produit ajouté apres lancer l’execution du client java :



1. Création un Client WEB (Avec WildFly) :

tester l’invocation à distance au bean Session :



1. Création un Client WEB (Avec Tomcat) :

tester l’invocation à distance au bean Session :

