

TP04

Application Web JEE avec modèle MVC et base de données

Objectifs

Implémenter le modèle MVC à travers une application Web JAVA avec accès à une base de données

1. Préparer et configurer le support physique

- Créer une base de données
- Préparer un utilitaire d'accès à la base de données

2. Mettre en place le modèle

- Définir les objets métier
- Exposer les fonctionnalités DAO
- Lier le modèle au contrôleur

Travail demandé

A. Gérer le support physique de stockage des données

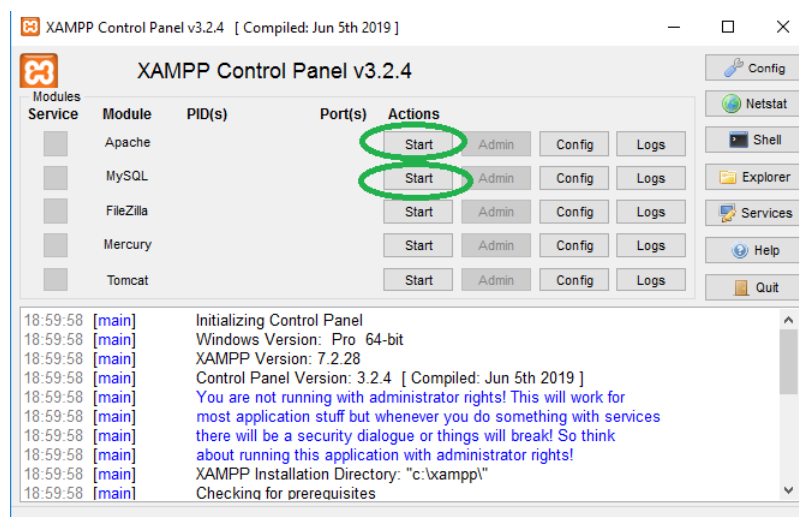
1. Installer le logiciel « **xampp** » à partir de l'adresse suivante :

<https://www.apachefriends.org/xampp-files/7.2.28/xampp-windows-x64-7.2.28-0-VC15-installer.exe>

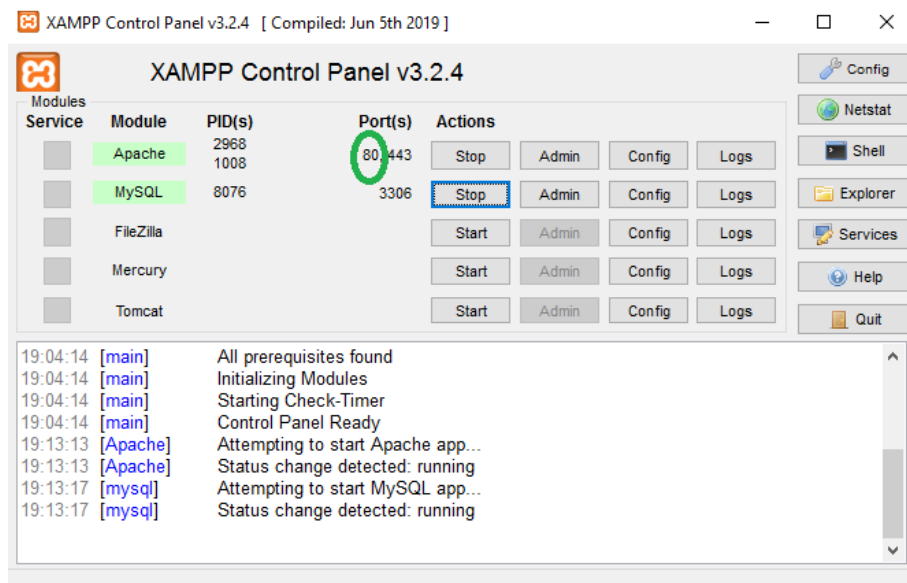
2. Double-cliquer sur le fichier pour lancer l'installation.

3. Un panneau de contrôle de « **Xampp** » s'affiche pour indiquer les états des différents modules.

4. Par défaut, les services des modules sont arrêtés :

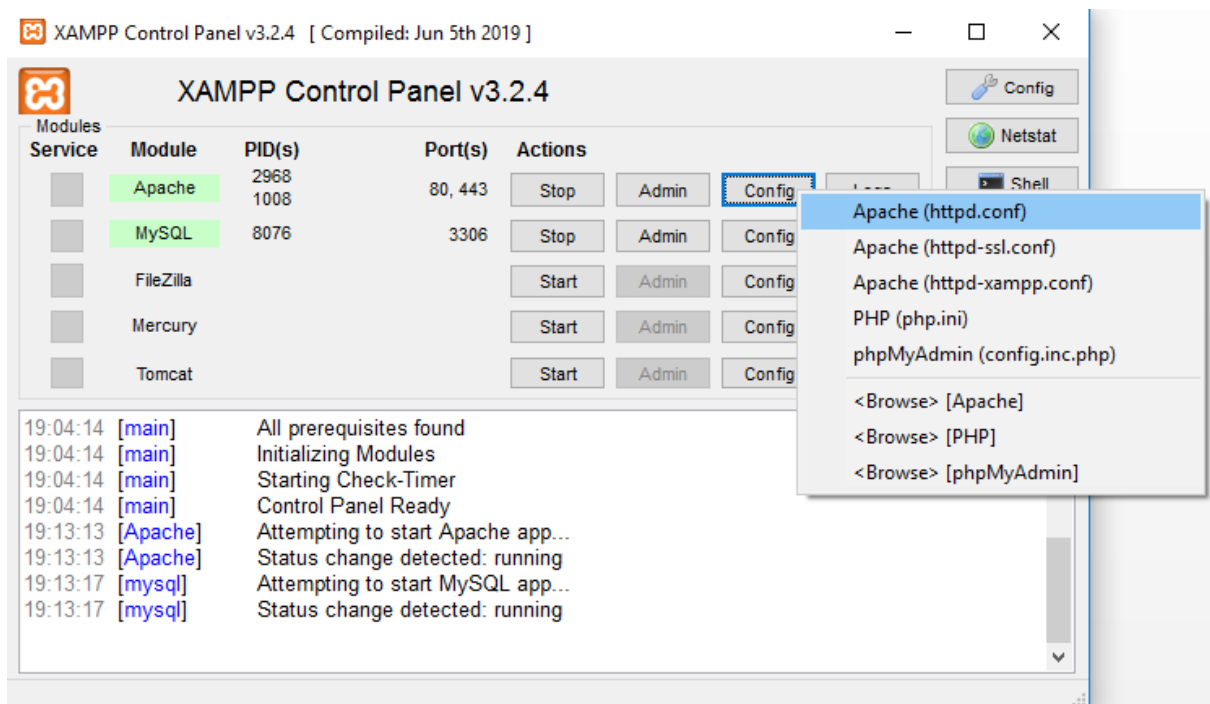


5. Démarrer les deux modules «**Apache** » et « **MySQL** » Remarquer que le module **Apache** est lancé avec le port « **80** ».

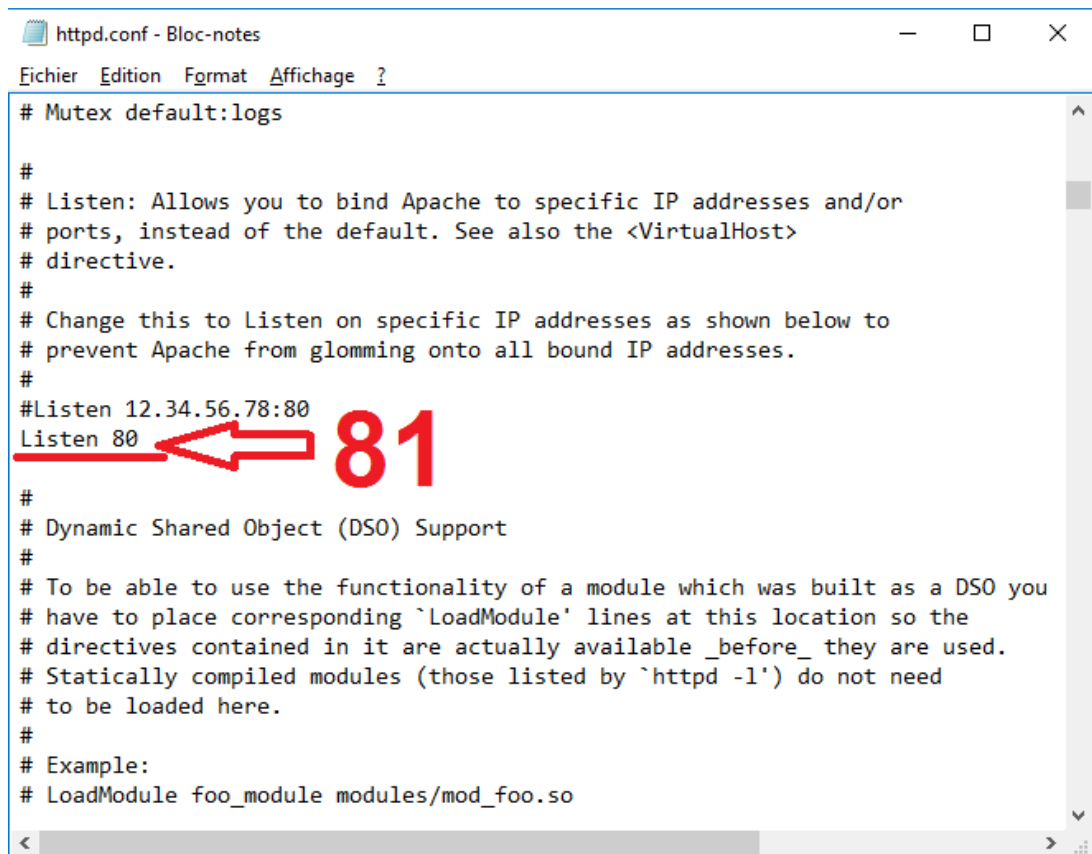


6. Changer le port HTTP à « **81** » (afin de libérer le port « **80** » pour notre serveur web Tomcat. Ainsi, suivre les instructions suivantes :

- ❖ Cliquer sur le bouton «**Config**» qui correspond au module «**Apache**»
- ❖ Choisir l'option «**Apache (httpd.conf)**»



❖ Ainsi, ouvrir le fichier « **httpd.conf** » et chercher l'instruction « **Listen 80** » et changer la valeur du port d'écoute du module « **Apache** » à « **81** » et enregistrer.

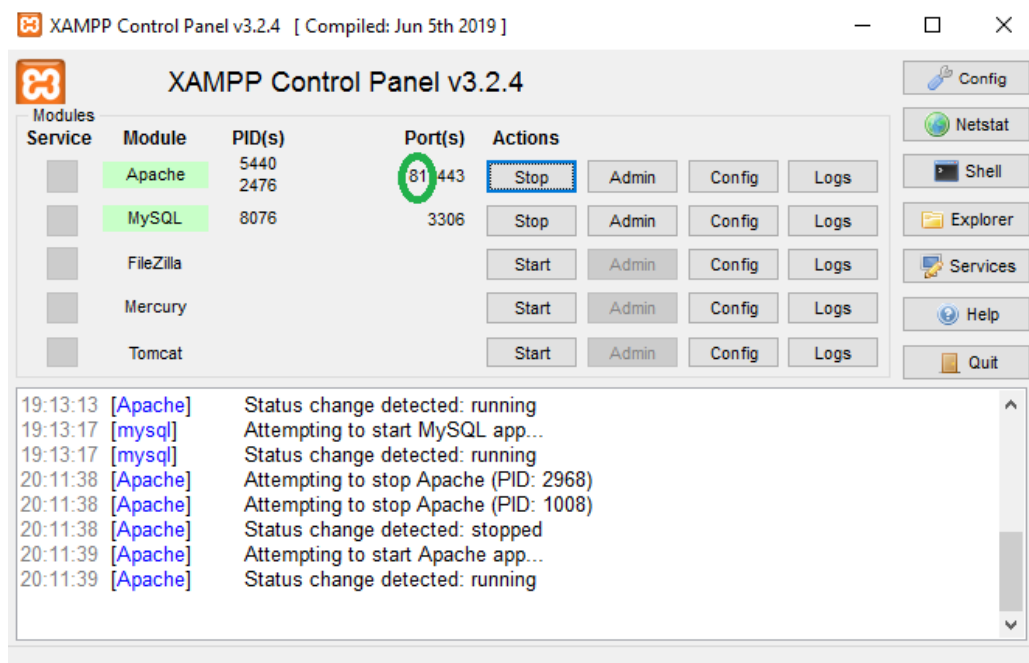


```
# Mutex default:logs

#
# Listen: Allows you to bind Apache to specific IP addresses and/or
# ports, instead of the default. See also the <VirtualHost>
# directive.
#
# Change this to Listen on specific IP addresses as shown below to
# prevent Apache from glomming onto all bound IP addresses.
#
#Listen 12.34.56.78:80
Listen 80

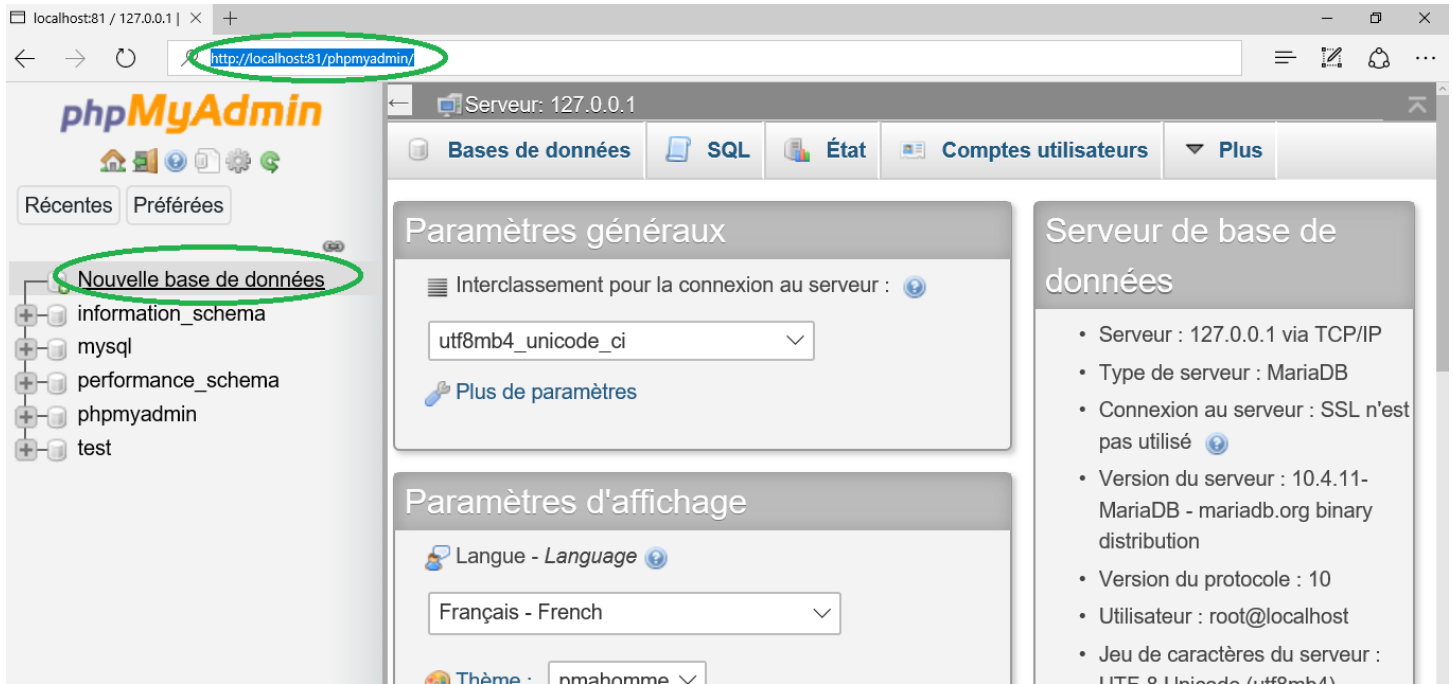
#
# Dynamic Shared Object (DSO) Support
#
# To be able to use the functionality of a module which was built as a DSO you
# have to place corresponding 'LoadModule' lines at this location so the
# directives contained in it are actually available _before_ they are used.
# Statically compiled modules (those listed by 'httpd -l') do not need
# to be loaded here.
#
# Example:
# LoadModule foo_module modules/mod_foo.so
```

❖ Arrêter le module « **Apache** » puis le démarrer de nouveau. Remarquer le lancement de « **Apache** » avec le port **81**



❖ Pour lancer l'application « **phpmyadmin** » ouvrir le navigateur web et spécifier l'url suivante :

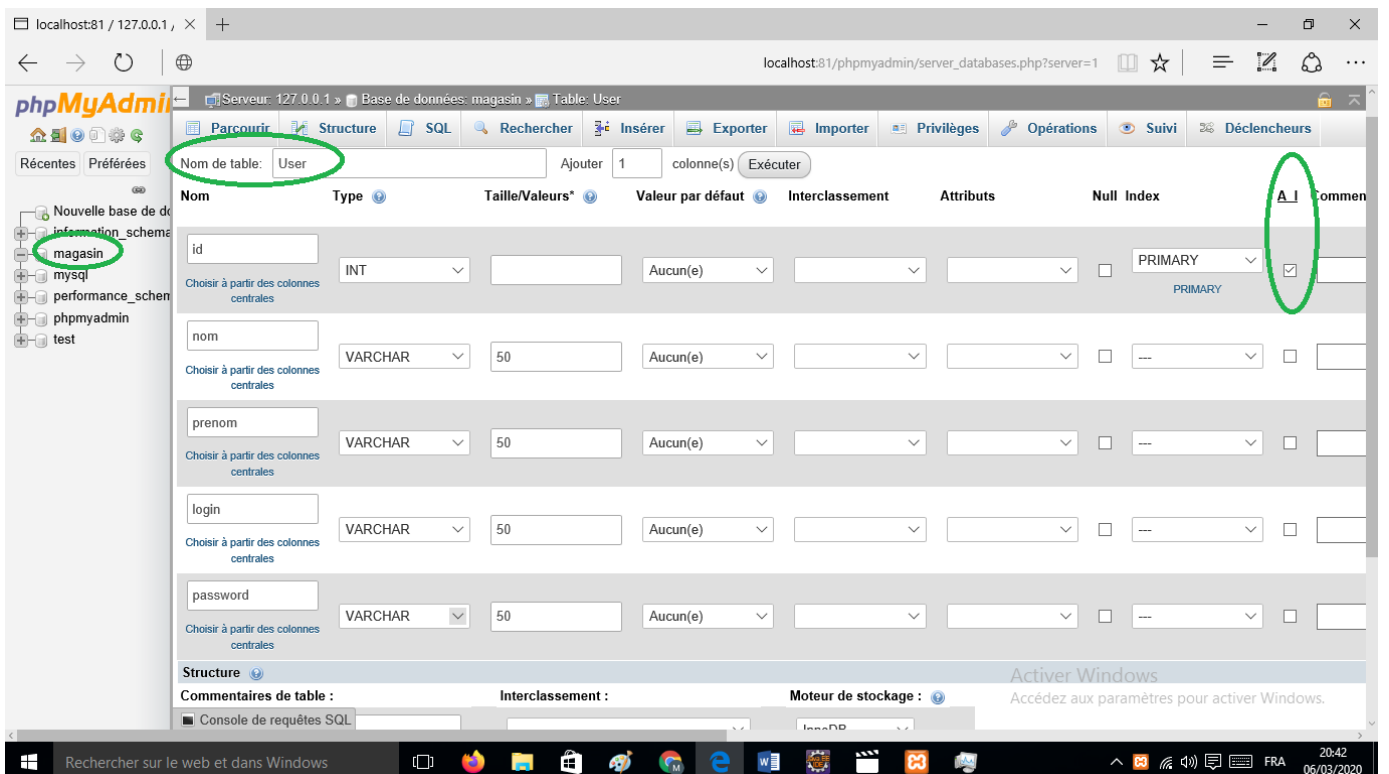
<http://localhost:81/phpmyadmin/>



7. Créer une base de données nommée « **MAGASIN** »

8. Créer une table « **User** » définie par :

- « **id** » : clé primaire (int)
- « **nom** » (varchar)
- « **prenom** » (varchar)
- « **login** » (varchar)
- « **password** » (varchar)



localhost:81 / 127.0.0.1, X +

localhost:81/phpmyadmin/db_structure.php?server=1&db=magasin&table=User

Seigneur: 127.0.0.1 » Base de données: magasin » Table: user

Parcourir Structure SQL Rechercher **Insérer** Exporter Importer Plus

Structure de table Vue relationnelle

#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action
1	id	int(11)			Non	Aucun(e)		AUTO_INCREMENT	Modifier Supprimer
2	nom	varchar(50)	utf8mb4_general_ci		Non	Aucun(e)			Modifier Supprimer
3	prenom	varchar(50)	utf8mb4_general_ci		Non	Aucun(e)			Modifier Supprimer
4	login	varchar(50)	utf8mb4_general_ci		Non	Aucun(e)			Modifier Supprimer
5	password	varchar(50)	utf8mb4_general_ci		Non	Aucun(e)			Modifier Supprimer

Tout cocher Avec la sélection : Parcourir Modifier Supprimer Primaire Unique Index

Texte entier Ajouter à la liste centrale Supprimer de la liste centrale de colonnes

Imprimer Suggérer des optimisations de structure Suivre la table Déplacer des colonnes Normaliser

Ajouter 1 colonne(s) après password Exécuter

9. Choisir l'action « **insérer** » pour ajouter deux enregistrements :

- ❖ (id=1, nom= Ayyadi, prenom=Slim, login=aa, password= bb)
- ❖ (id=2, nom= Ben Omar, prenom=Ali, login=zz, password= yy)

Seigneur: 127.0.0.1 » Base de données: magasin » Table: user

Parcourir Structure SQL Rechercher **Insérer** Exporter Importer

✓ Affichage des lignes 0 - 1 (total de 2, traitement en 0,0014 seconde(s).)

```
SELECT * FROM `user`
```

Profilage [Éditer en ligne] [Éditer] [Expliquer SQL] [Créer le

Tout afficher | Nombre de lignes : 25 | Filtrer les lignes: Chercher dans cette table | Trier par

+ Options

	id	nom	prenom	login	password
<input type="checkbox"/> Éditer Copier Supprimer	1	Ayyadi	Slim	aa	bb
<input type="checkbox"/> Éditer Copier Supprimer	2	Ben Omar	Ali	zz	yy

Tout cocher Avec la sélection : Éditer Copier Supprimer Exporter

B. Gérer la partie « Metier »

10. Créer, dans le workspace « **D:\Atelier_JEE\workspace** », un nouveau projet web dynamique nommé « **web_app_tp04** ».

11. Créer sous « **src** » un package « **metier** »

12. Définir le bean « **User** » qui englobe :

- les attributs « **id** », « **nom** », « **prenom** », « **login** » et « **password** »,

- les méthodes « **setters** » et « **getters** »,
- Les constructeurs,
- Une méthode « **toString()** » qui retourne une représentation textuelle de l'objet « **User** ».

```
package metier;

import java.io.Serializable;

public class User implements Serializable
{
    private int id;
    private String nom;
    private String prenom;
    private String login;
    private String password;
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public User() {
        super();
        // TODO Auto-generated constructor stub
    }
    public User(int id, String nom, String prenom, String login, String
password) {
        super();
        this.id = id;
        this.nom = nom;
        this.prenom = prenom;
        this.login = login;
        this.password = password;
    }
    public User(String nom, String prenom, String login, String password) {
        super();
        this.nom = nom;
        this.prenom = prenom;
        this.login = login;
        this.password = password;
    }
    @Override
    public String toString() {
        return "User [id=" + id + ", nom=" + nom + ", prenom=" + prenom + ",
login=" + login + ", password=" + password + "]";
    }
    public String getNom() {
        return nom;
    }
    public void setNom(String nom) {
```

```

        this.nom = nom;
    }
    public String getPrenom() {
        return prenom;
    }
    public void setPrenom(String prenom) {
        this.prenom = prenom;
    }
    public String getLogin() {
        return login;
    }
    public void setLogin(String login) {
        this.login = login;
    }
    public String getPassword() {
        return password;
    }
    public void setPassword(String password) {
        this.password = password;
    }
}

```

13. Définir dans le package « **metier** » une interface « **UserMetierInterface** » qui déclare les traitements (les services) à réaliser sur l'objet « **User** »

```

package metier;

import java.util.List;

//une interface qui déclare la liste des traitements métier sur un "User"
public interface UserMetierInterface
{
    //Ajouter un objet "User"
    public void addUser(User u);
    //Retourner la liste de tous les objets "User"
    public List<User> listUsers();
    //Retourner l'objet "User" ayant le login et le password passés en paramètres
    public User getUserByLoginAndPassword(String l, String p);
    //Mettre à jour un objet "User" déjà existant
    public void updateUser(User u);
    //Supprimer un objet "User" identifié par son "id"
    public void deleteUser(int id);
}

```

14. Construire une classe « **DBConnexion** » qui permet de retourner une seule instance de connexion à la base de données « **MAGASIN** » :

```

package metier;

import java.sql.Connection;
import java.sql.DriverManager;

public class DBConnexion
{
    private static Connection connection =null;
    static
    {
        try {
            //Charger le pilote d'accès à la BD
            Class.forName("com.mysql.jdbc.Driver");
            //Etablir la connexion à la BD "MAGASIN"

            connection=DriverManager.getConnection("jdbc:mysql://localhost:3306/MAGASIN",
"root","");

        } catch (Exception e) {
            // Message d'erreur en cas de problème de connexion
            System.out.println("Problème de connexion à la BD..");
            e.printStackTrace();
        }
    }
    public static Connection getConnection()
    {
        //return l'objet de connexion
        return connection;
    }
}

```

15. Donner une implémentation des traitements « **métier** » dans une classe
« **UserMetierImpl** »

```

package metier;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class UserMetierImpl implements UserMetierInterface{
    @Override
    public void addUser(User u) {
        //récupérer une connexion à la BD
        Connection conn= DBConnexion.getConnection();
        try {
            // préparer la requête SQL
            PreparedStatement ps = conn.prepareStatement(" insert into user
values (0,?,?,?,?,?)");

```



```

        // passer les paramètres
        ps.setString(1, u.getNom());
        ps.setString(2, u.getPrenom());
        ps.setString(3, u.getLogin());
        ps.setString(4, u.getPassword());
        // exécuter la requête
        ps.executeUpdate();
    } catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

@Override
public List<User> listUsers() {
    //Définir une liste vide pour stocker les objets "User"
    List<User> users= new ArrayList<User>();
    //récupérer une connexion à la BD
    Connection conn= DBConnexion.getConnection();
    try {
        // préparer la requête SQL
        PreparedStatement ps = conn.prepareStatement(" select * from
User");

        // Récupérer le résultat de la requête
        ResultSet rs= ps.executeQuery();
        if (rs!=null)
        {
            //parcourir le résultat
            while(rs.next())
            {
                //Construire un objet "User" puis lui affecter les
attributs

                // et enfin l'ajouter dans la liste
                User u = new User();
                u.setId(rs.getInt("id"));
                u.setNom(rs.getString("nom"));
                u.setPrenom(rs.getString("prenom"));
                u.setLogin(rs.getString("login"));
                u.setPassword(rs.getString("password"));
                // ajouter l'objet "User" dans la liste
                users.add(u);
            }
        }
        ps.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
    //retourner la liste
    return users;
}

@Override
public User getUserByLoginAndPassword(String l, String p) {
    //récupérer une connexion à la BD
    Connection conn= DBConnexion.getConnection();
    User u= null;

```

```

        try {
            // préparer la requête SQL
            PreparedStatement ps = conn.prepareStatement(" select * from
User where login =? and password = ?");
            ps.setString(1, l);
            ps.setString(2, p);
            ResultSet rs= ps.executeQuery();
            if (rs!=null)
            {
                while(rs.next())
                {
                    u = new User();
                    u.setId(rs.getInt("id"));
                    u.setNom(rs.getString("nom"));
                    u.setPrenom(rs.getString("prenom"));
                    u.setLogin(rs.getString("login"));
                    u.setPassword(rs.getString("password"));

                }
            }
            ps.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
        return u;
    }

    @Override
    public void updateUser(User u) {
        //récupérer une connexion à la BD
        Connection conn= DBConnexion.getConnection();
        try {
            // préparer la requête SQL
            PreparedStatement ps = conn.prepareStatement(" update user set
nom= ?, prenom=?, login=?, password =? where id=? ");
            ps.setString(1, u.getNom());
            ps.setString(2, u.getPrenom());
            ps.setString(3, u.getLogin());
            ps.setString(4, u.getPassword());
            ps.setInt(5, u.getId());

            ps.executeUpdate();
        } catch (SQLException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    @Override
    public void deleteUser(int id) {
        //récupérer une connexion à la BD
        Connection conn= DBConnexion.getConnection();
        try {
            // préparer la requête SQL

```

```

        PreparedStatement ps = conn.prepareStatement(" delete from user
where id=? ");
        ps.setInt(1, id);
        ps.executeUpdate();
    } catch (SQLException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}
}

```

16. Créer une classe « **TestMetier** » afin de tester les traitements ainsi définis :

```

package metier;
import java.util.List;

public class TestMetier {
    public static void main(String[] args) {

        //Appel à la couche « Services »
        UserMetierInterface metier= new UserMetierImpl();
        //Test d'ajout
        metier.addUser(new User ("Ben Saleh", "Mohamed","11","22"));
        System.out.println("-----\n");

        //Test d'affichage de la liste totale des objets "User"
        List<User> users = metier.listUsers();
        for (User u: users)
        {
            System.out.println(u);
        }
        System.out.println("-----\n");

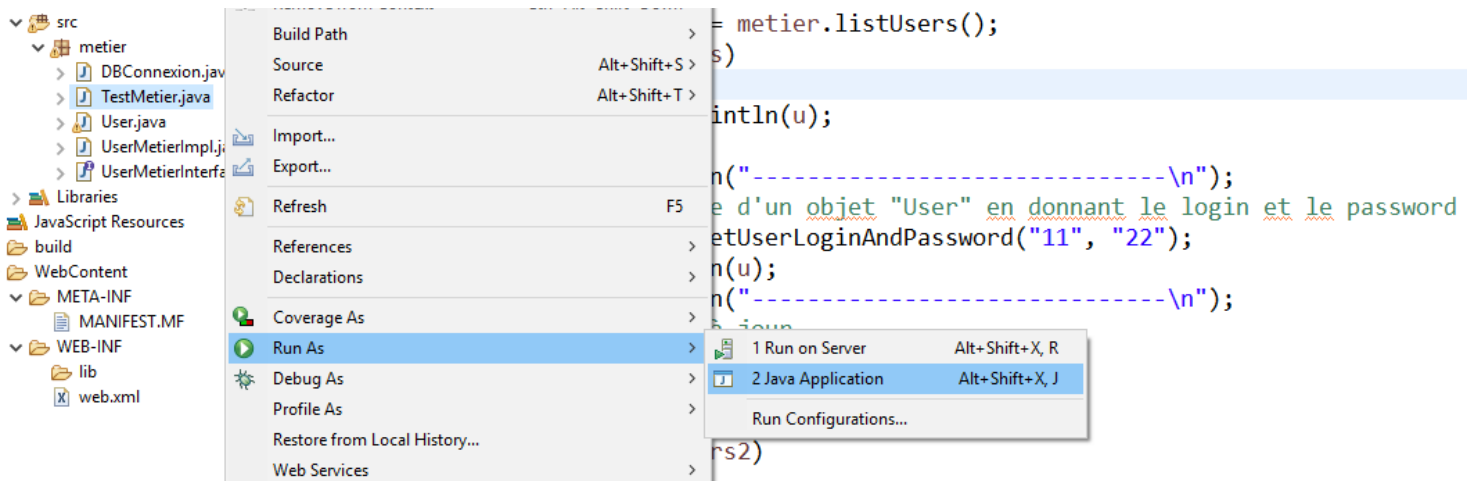
        //Test d'affichage d'un objet "User" en donnant le login et le password
        User u = metier.getUserByLoginAndPassword("11", "22");
        System.out.println(u);
        System.out.println("-----\n");

        //Tester la mise à jour
        u.setNom("Sallemi");
        metier.updateUser(u);

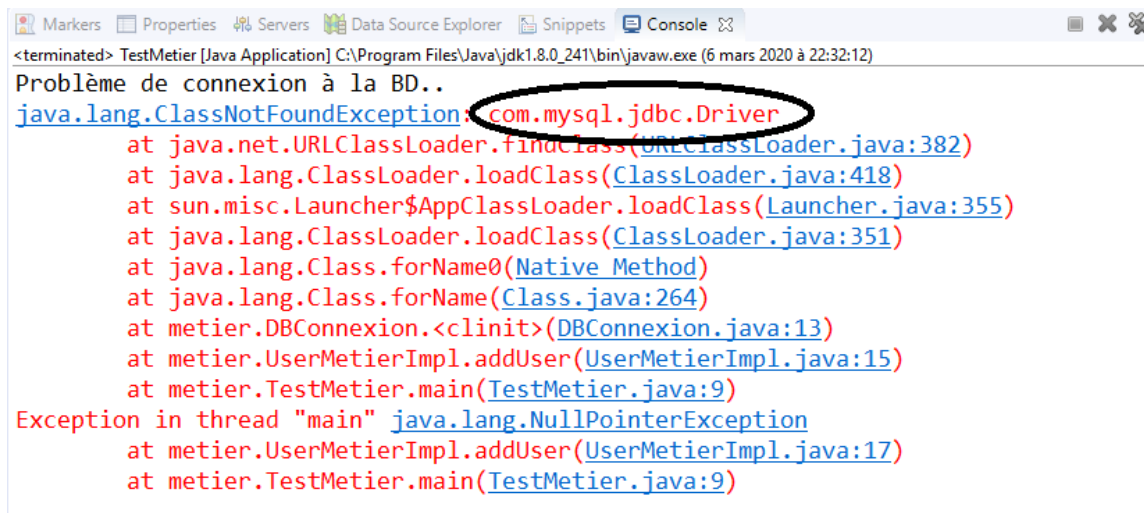
        List<User> users2 = metier.listUsers();
        for (User u2: users2)
        {
            System.out.println(u2);
        }
        System.out.println("-----\n");
    }
}

```

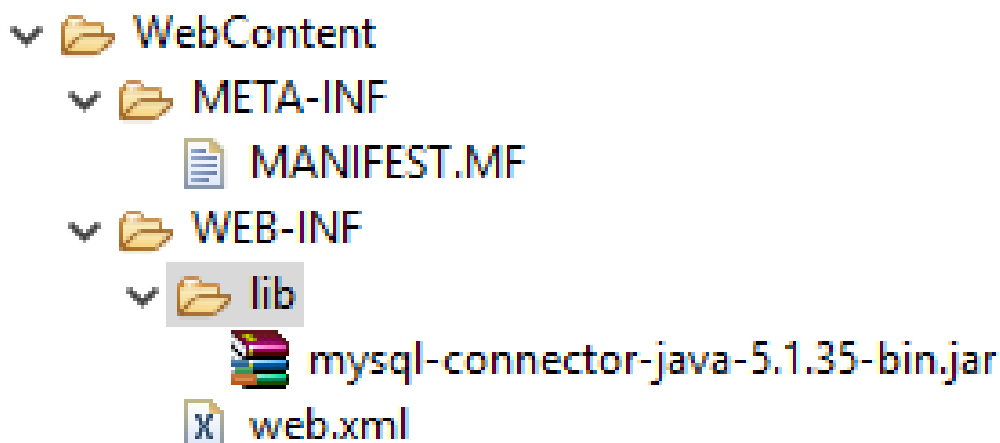
17. Sélectionner la classe « **TestMetier** » et lancer son exécution



18. Remarquer, dans la console, une exception qui indique l'absence du pilote de « MySQL » (« **com.mysql.jdbc.Driver** »)



19. Placer le pilote « **mysql-connector-java-5.1.35-bin.jar** » (donné en pièce jointe) dans le dossier « **lib** » de « **WEB-INF** ».



20. Relancer l'exécution de la classe « **TestMetier** » :

Voici le résultat dans la console et au niveau de la base de données:











```
Markers Properties Servers Data Source Explorer Snippets Console
<terminated> TestMetier [Java Application] C:\Program Files\Java\jdk1.8.0_241\bin\javaw.exe (6 mars 2020 à 23:33:17)

-----

User [id=1, nom=Ayyadi, prenom=Slim, login=aa, password=bb]
User [id=2, nom=Ben Omar, prenom=Ali, login=zz, password=yy]
User [id=7, nom=Ben Saleh, prenom=Mohamed, login=11, password=22]
-----

User [id=7, nom=Ben Saleh, prenom=Mohamed, login=11, password=22]
-----

User [id=1, nom=Ayyadi, prenom=Slim, login=aa, password=bb]
User [id=2, nom=Ben Omar, prenom=Ali, login=zz, password=yy]
User [id=7, nom=Salleme, prenom=Mohamed, login=11, password=22]
-----
```

				id	nom	prenom	login	password
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	1	Ayyadi	Slim	aa	bb
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	2	Ben Omar	Ali	zz	yy
<input type="checkbox"/>	 Éditer	 Copier	 Supprimer	7	Salleme	Mohamed	11	22

21. Exercice : Tester la méthode « **deleteUser (int id)** »

C. Gérer la partie « web »

22. Créer sous « **src** » un package « **web** » dans lequel ajouter tous les contrôleurs données en pièce jointe (simple copier/coller):

- a. **UserController**
- b. **UserEditionController**
- c. **UserListController**
- d. **UserDeconnexionController**

NB : Remarque que la servlet « **UserEditionController** » utilise une méthode « **getUserById(int id)** » de la classe « **UserMetierImpl** ».

Voici le code de déclaration de cette méthode dans l'interface « **UserMetierInterface** » (A ajouter):

```
//Retourner un objet "User" par id
public User getUserById (int id);
```

et voici le code de son implémentation dans la classe « **UserMetierImpl** » :

```
@Override
public User getUserById(int id) {
    //récupérer une connexion à la BD
    Connection conn= DBConnexion.getConnection();
    User u= null;
    try {
        // préparer la requête SQL
        PreparedStatement ps = conn.prepareStatement(" select
* from User where id = ?");
        ps.setInt(1,id);

        ResultSet rs= ps.executeQuery();
        if (rs!=null)
        {
            while(rs.next())
            {
                u = new User();
                u.setId(rs.getInt("id"));
                u.setNom(rs.getString("nom"));
                u.setPrenom(rs.getString("prenom"));
                u.setLogin(rs.getString("login"));
                u.setPassword(rs.getString("password"));
            }
        }
        ps.close();
    } catch (SQLException e) {
        e.printStackTrace();
    }
    return u;
}
```

23. Dans le dossier « **WebContent** », créer un dossier « **css** ». Dans ce dossier placer le fichier « **style.css** » (donné en pièce-jointe)

24. Dans le dossier « **WebContent** », copier les vues suivantes (données en pièce-jointe) :

- a. **UserConnexion.jsp**
- b. **accueil.jsp**
- c. **entete.jsp**
- d. **UserForm.jsp**
- e. **UserList.jsp**

25. La vue « **UserConnexion.jsp** » constitue le formulaire de connexion. Cette vue englobe un formulaire à être traité par le contrôleur « **UserController** » (Servlet) en mode « **POST** » :

- a. S'il existe un problème de saisie, l'affichage est redirigé vers la vue « **UserConnexion.jsp** »
- b. Sinon, la servlet appelle la méthode « **getUserByLoginAndPassword(String l , String p)** » d'un objet « **metier** » et vérifie l'existence de tel utilisateur :

- Si l'utilisateur n'est pas reconnu, l'affichage est redirigé vers la vue « **UserConnexion.jsp** ».
- Sinon, l'utilisateur sera enregistré dans la session et l'affichage est redirigé vers la vue « **accueil.jsp** » (qui inclut le fichier « **entete.jsp** »)

26. La vue « **accueil.jsp** » présente la page de menu : Elle donne l'accès aux deux fonctionnalités (deux liens « hypertexte »):

- a. **Ajouter un utilisateur**
- b. **liste des utilisateurs**

27. La page « **entete.jsp** » affiche une barre de navigation globale et qui contrôle les droits d'accès en mode connecté et présente un lien pour se déconnecter (Cette page est incluse dans toutes les pages en mode connecté)

28. Les deux liens de la page d'accueil fonctionnent comme suit :

- a. Le lien « **Ajouter un utilisateur** » mène au formulaire « **UserForm.jsp** »
- b. Le lien « **Liste des utilisateurs** » mène à la servlet « **UserListController** » en mode « **GET** ». Cette servlet charge la liste des utilisateurs existants (à travers un appel à un objet « **metier** ») puis elle transfère à page « **UserList.jsp** » sous forme d'attribut dans la session.

29. La vue « **UserList.jsp** » présente un tableau d'utilisateurs et un ensemble d'actions à réaliser pour les utilisateurs

30. Lancer l'exécution des composants web et interpréter le résultat.

Exercice

31. Etablir les modifications nécessaires pour ajouter dans la vue « **UserList** » un formulaire de recherche (par nom ou par prénom) pour un affichage par filtre des utilisateurs.