Contexte : nous avons développé par adaptation une application de suivi de stage. Les utilisateurs peuvent être des élèves, entreprises proposant des stages, professeur ou visiteur, qui se connectent via un navigateur Web.

Les fonctionnalités sont :

- dépôt de propositions de stage

- consultation d'offres de stages

- chat interactif entre utilisateurs (étudiants/étudiants, professeur / étudiant ou entreprise / étudiant …)

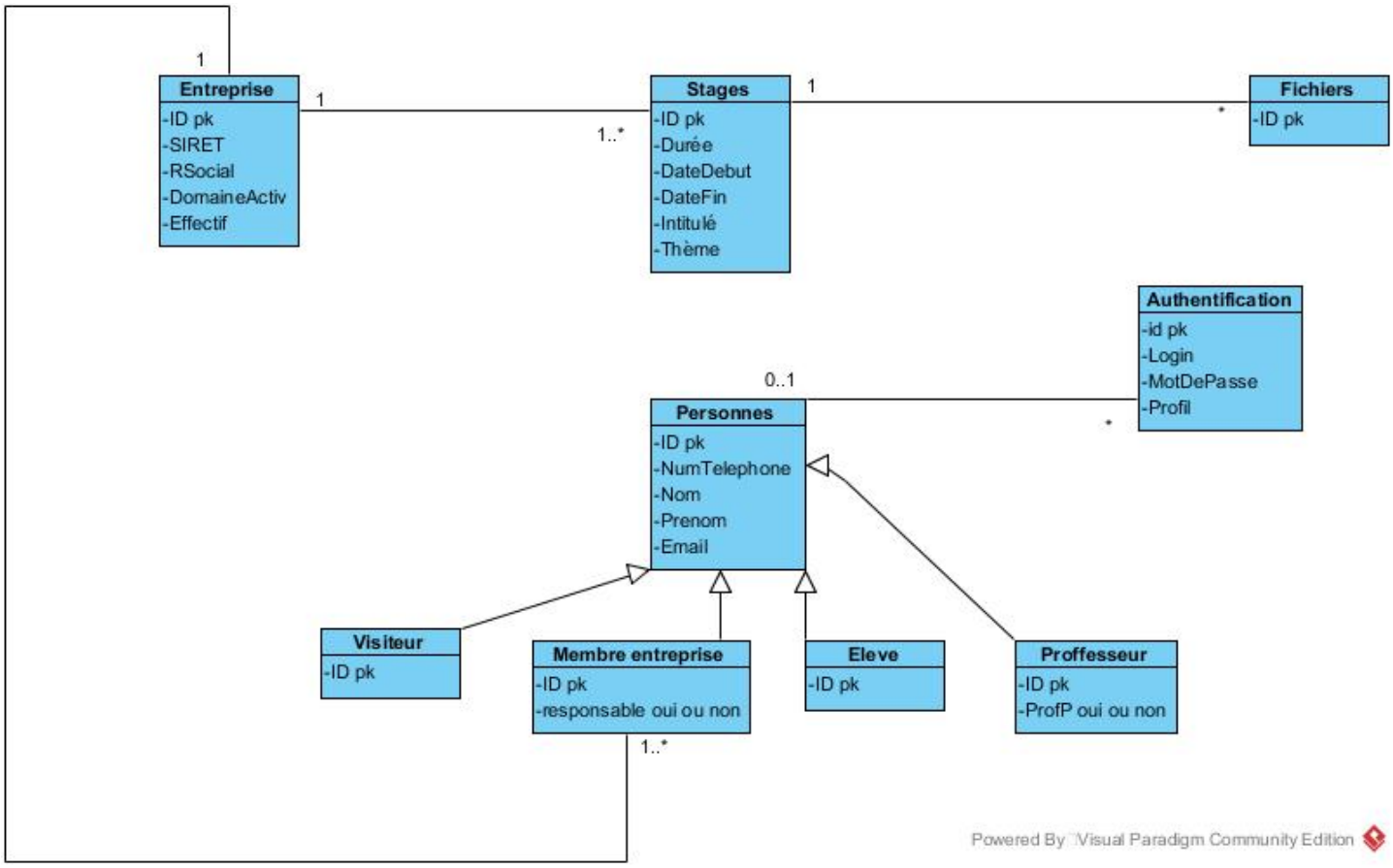
- logging des utilisateurs connectés

Le contexte est un environnement J2EE avec Servlets, JSP et utilisation de WebSockets

Modélisation de l'application

Ci-dessous, la modélisation conceptuelle effectuée par Mr xxx, qui donne le scope cible de notre contexte.

Modèle conceptuel de données



Modèle physique de données

Pour l'ensemble des étudiants il y a une table disponible dans le modèle physique permettant de s'authentifier à l'application : LOGIN_USER

Certains d'entre vous ont implémenté davantage de tables ou avec un nom différent. Il faut le documenter dans votre situation professionnelle.

Exemple :

```
CREATE TABLE LOGIN_USER
(
NOM VARCHAR(40),
PRENOM VARCHAR(40),
UTILISATEUR VARCHAR(40),
MDP VARCHAR(40)
);

INSERT INTO LOGIN_USER VALUES ('ROBERT','Timothee','Timothee','Robert') ;
INSERT INTO LOGIN_USER VALUES ('DA SILVA','Daniel','daniel','daniel') ;
```

Annexes

Les annexes doivent vous aider et vous permettre de :

- compléter la documentation de votre situation professionnelle
- vérifier que vous êtes à jour pour votre contexte, en particulier sur la partie technique

Annexe 0 : exemple d'instructions pour créer un bean utilisateur, qui servira à faire l'interface entre les données de l'application et les données persistantes

Remarque préliminaire: toutes ces modifications sont déjà présentes (sauf catastrophe) dans l'application présentée par les étudiants. Elles ont été implémentées lors de PPE précédents (16 Mars – 23 Mars – 30 Mars)

1. Création d'une classe Bean utilisateur dans le projet

Pour ce faire, nous allons effectuer les tâches suivantes :

Création d'un package dao

Création d'une classe UserBean dans ce package.

Déclaration des variables suivantes : username, password, nom, prenom comme String et valid comme booléen.

Ajout des méthodes get et set correspondantes à ces variables

Pour la variable valid, on nommera la méthode isValid() plutôt que getValid() pour plus de clarté.

2. Création d'une classe ConnectionManager dans le package dao

Dans le projet, dans le package dao créé à l'étape précédente, vous devez ajouter maintenant une nouvelle classe java, nommée par exemple ConnectionManager, qui va permettre de créer la connexion JDBC à la base de données. Cf en **annexe 4** un exemple de classe à **adapter** (modifier le serveur, login SQL, mot de passe SQL …).

3. Création d'une classe DAO dans le package dao

Nous devons maintenant créer une classe effectuant la requête, en passant les paramètre login et mot de passe, de valider l'existence d'un compte associé, et de retourner le résultat à l'application.

Dans cette classe, nommée par exemple UserDAO, nous devons :

Exécutez la requête

Effectuer le mapping des données SQL Server avec les variables java (notre bean)

Vérifier la présence d'un couple login / mdp

Positionner la valeur de la variable valid en fonction de l'étape de vérification précédente

Un exemple de classe est fourni en **Annexe 5**

4. Modification de la phase de login dans l'application

Cette dernière étape consiste à modifier la validation du login dans l'application.

Les données de connexion saisies par l'utilisateur sont fournies à l'application via un formulaire Web. Il s'agit de modifier dans notre classe les informations et le processus de validation.

Les étapes à modifier sont :

- Dans la méthode qui traite les données de formulaire via un POST, créer un objet de type UserBean (cf étape 3) et l'intialiser
- Associer à ce « bean » les paramètres passés par l'utilisateur
- Valider le couple login / mdp en utilisant la classe UserDAO
- Cf un exemple en **Annexe 6**
- Enfin, en conclusion, **il reste à modifier les conditions de test pour poursuivre ou revenir à la page de login**, en utilisant la méthode isValid() appliquée à notre UserBean crée et alimenté auparavant au moyen de notre DAO

5. Tester et débugger l'application

Annexe 1 : rappel des étapes nécessaires pour déployer l'application

Remarque : cette annexe vise à rappeler les grandes étapes d'installation d'une application J2EE pour ceux qui voudraient installer l'application chez eux (et qui ne l'auraient pas encore fait …).

Etape 0 : lancer Eclipse (EE)

Etape 1 : copier le projet dans le workspace Eclipse EE

Etape 2 : effectuer un **import maven (File → Import → Maven → import existing maven project)**

Etape 3 : (optionnel) : si 1ère utilisation de maven, rajouter le fichier de définition du proxy dans le répertoire maven, puis effectuer un Run As → Maven → Maven Install

Z:\TSSIO2015_Commun\PPE\20170316\settings.xml

Répertoire de maven : répertoire .m2 dans le dossier utilisateur

Etape 4 : (optionnel) : si pas de serveur d'application installé

- Installer serveur d'application : Tomcat 8.5 ou supérieur (pour bénéficier du support des WebSockets).
- Copier le sqljdbc4.2 (driver jdbc de Microsoft) dans le répertoire lib de Tomcat.

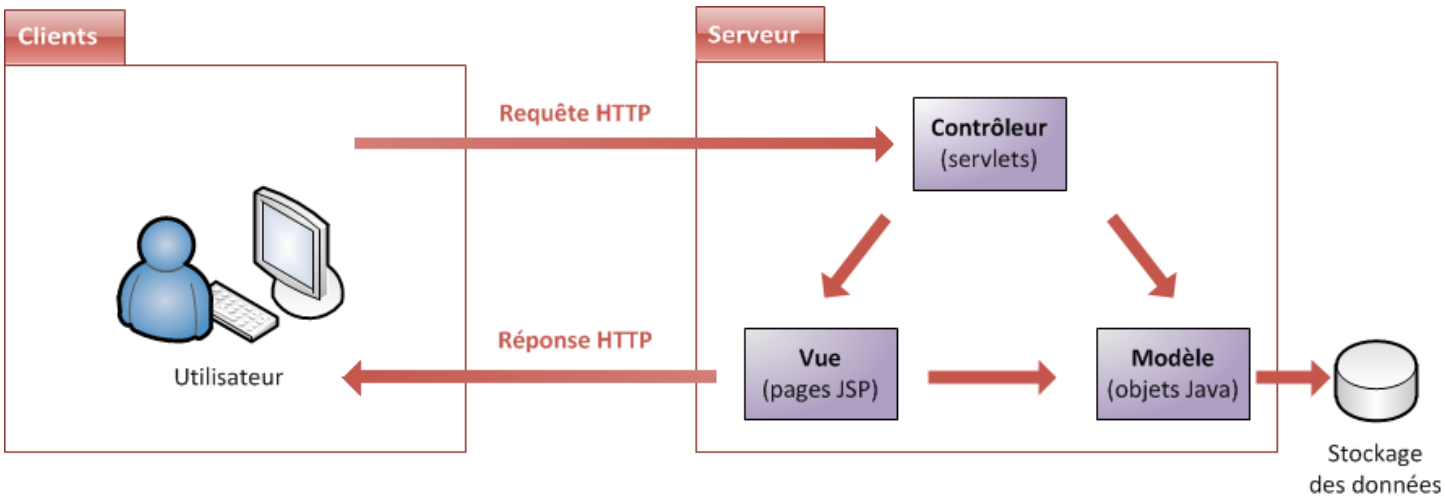
Pour Server Location, préférez le répertorie d'installation de Tomcat (pour éviter d'aller fouiller pour regarder ce qui est déployé).

Etape 5 : Run As → Run on Server

Etape 6 (optionnel) : Windows → Preferences → General → Web Browser pour avoir le navigateur dans uen fenêtre externe

Annexe 2 : architecture MVC Servlet / J2EE

Schéma simplifié issu de Open Class Rooms



Annexe 3 : rappel de la déclaration de Servlet

Il y a 2 façons de déclarer des Servlets.

1. Dans le fichier descripteur de déploiement web.xml.
2. Via des annotations.

C'est cette dernière méthode que nous utilisons.

Syntaxe :

Déclaration d'une classe de type Servlet avec quelques informations additionnelles

```
import javax.servlet.annotation.WebServlet
import javax.servlet.http.HttpServlet;

@WebServlet(
        name = "MaServlet",
        description = "Voici une Servlet declare avec une annotation",
        urlPatterns = "/processServlet"
)
public class MaServlet extends HttpServlet {
    // implement servlet doPost() and doGet()...
}
```

Annexe 4 : classe ConnectionManager du professeur

Remarque : cette classe contient le code nécessaire pour se connecter à VOTRE base de données et table. Vous avez besoin de connaître :

- Votre serveur SQL
- Votre base de données
- Vos identifiants SQL de connexion (user, password)

```
package dao;

import java.sql.*;
import java.util.*;
public class ConnectionManager {
	static Connection con;
	static String url;
	public static Connection getConnection() {
		try {
			// String url = "jdbc:odbc:" + "DataSource";
			// hypothese "DataSource" est le nom de la Datasource
			// ci dessous on passe directement la chaine en parametre ...
			Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
			try {
				con = DriverManager.getConnection("jdbc:sqlserver://SERVITEUR:1433;" +
						"databaseName=STAGES;user=sa;password=Madrid2020");
				// assuming your SQL Server's username is "username"
				// and password is "password"
				}
			catch (SQLException ex) {
				ex.printStackTrace();
				}
			}
		catch(ClassNotFoundException e) {
```

```
                System.out.println(e);
                }
            return con;
            }
      }
```

Annexe 5 : classe UserDAO du professeur

```
package dao;

import java.text.*;
import java.util.*;
import java.sql.*;

public class UserDAO {
        static Connection currentCon = null;
        static ResultSet rs = null;
        public static UserBean login(UserBean bean) {
              //preparing some objects for connection
              Statement stmt = null;
              String username = bean.getUsername();
              String password = bean.getPassword();
              String searchQuery = "select * from LOGIN_USER where UTILISATEUR='" + username + "'
AND MDP='" + password + "'";
              System.out.println("Votre nom est " + username);
              System.out.println("Votre mot de passe est " + password);
              System.out.println("La requete soumise est: "+searchQuery);
              try { //connect to DB
                    currentCon = ConnectionManager.getConnection();
                    stmt=currentCon.createStatement();
                    rs = stmt.executeQuery(searchQuery);
                    boolean more = rs.next();
                    // si l'utilisateur n'existe pas
                    if (!more) {
                          System.out.println("Desole, pas de login / mdp associé");
                          bean.setValid(false); }
                    //si l'utilisateur existe, on positionne la variable valid à vrai
                    else if (more) {
                          String nom = rs.getString("nom");
                          String prenom = rs.getString("prenom");
                          System.out.println("Bonjour " + nom);
                          bean.setNom(nom);
                          bean.setPrenom(prenom);
                          bean.setValid(true);
                          }
                    }
              catch (Exception ex) {
                    System.out.println("Echec du login: une exception a été levée! " + ex);
                    }
              //some exception handling
              finally { if (rs != null) {
                    try {
                          rs.close();
                          }
                    catch (Exception e) {}
                    rs = null; }
              if (stmt != null) {
```

```
                try {
                        stmt.close(); }
                catch (Exception e) {}
                stmt = null; }
        if (currentCon != null) {
                try {
                        currentCon.close(); }
                catch (Exception e) { }
                currentCon = null;
                }
        }
        return bean;
        }
}
```

Annexe 6 : exemple initialisation du Bean et validation de l'utilisateur

Ce code est ajouté dans la classe LoginServlet pour :

- Récupérer les informations en provenance du formulaire Web (présent dans la page HTML générée par le fichier login.jsp)
- Les mapper au bean
- Le bean permettra ensuite d'assurer la persistance des données dans la base SQL Server

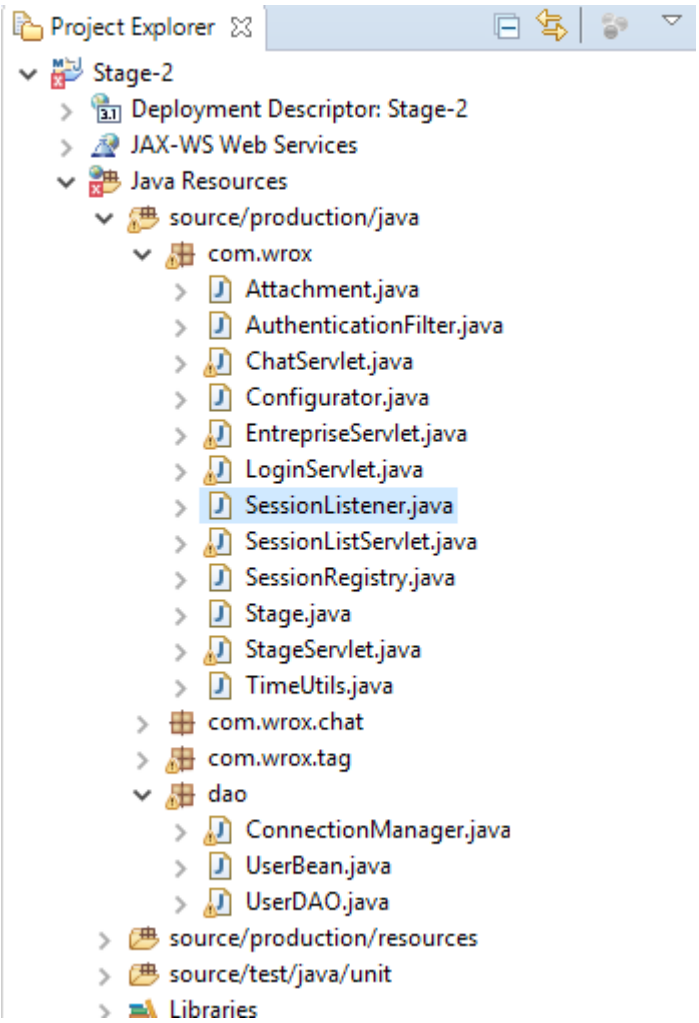
…..

```
UserBean user = new UserBean();
user.setUserName(request.getParameter("username"));
user.setPassword(request.getParameter("password"));
user = UserDAO.login(user);
String username = request.getParameter("username");
```

……

Annexe 7 : arborescence de l'application

Partie Servlets et DAO



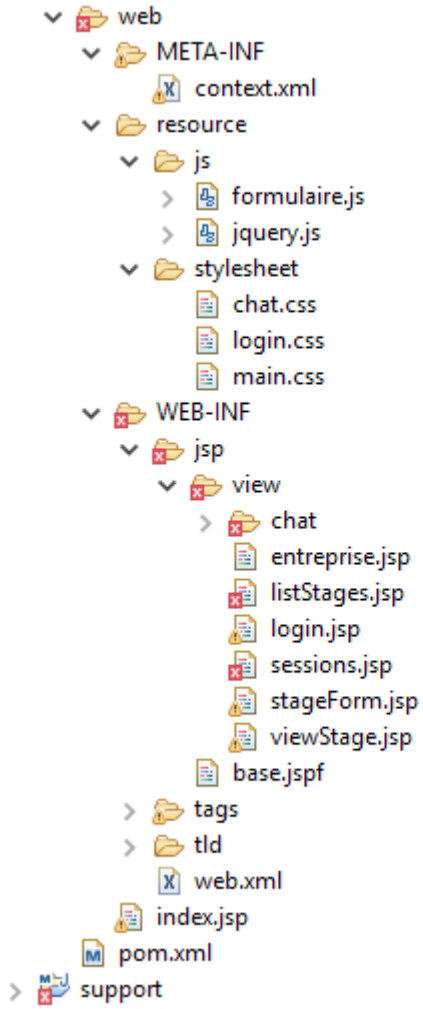
La Servlet LoginServlet contient le code d'authentification des utilisateurs et de redirection vers la servlet stages en cas de succès.

Initialement, l'authentification était faite directement dans le code via une HashTable (qui peut subsister dans certains projets). Lors d'un PPE, l'objectif était de migrer l'authentification vers la base de données SQL Server. Chaque étudiant dispose de sa propre base de données SQL Server sur un serveur SQL.

L'authentification via la base de données se fait grace aux 3 classes DAO (ConnectionManager déclare le driver et les paramètres de connexion, UserBean est le bean utilisateur, voire plus haut, et UserDAO contient le code de validation + du code lié à une procédure stockée développée lors d'un autre PPE).

EntrepriseServlet n'est pas une servlet initiale du projet mais a été développée comme proposition de solution à l'expression de besoin (voir **annexes 8 et 9**).

Partie JSP



Annexe 8 : exemple d'expression de besoin par rapport au contexte

Au delà de l'écran d'accueil (login), on souhaite différencier le traitement des utilisateurs entreprises par rapport aux autres utilisateurs. Pour se faire, on souhaite :

- distinguer les entreprises (des autres utilisateurs) sur l'écran d'accueil
- si l'utilisateur s'authentifie en tant qu'entreprise, il doit être dirigé vers une page dédiée (succincte), qui respecte l'architecture de l'application
- ajouter à cette page un retour vers le menu login de l'application
- proposez une feuille de style personnalisée pour les entreprises

Vous proposerez dans un 1er temps une méthodologie et une démarche d'implémentation. Une fois validée et étayée, vous la mettrez en œuvre.

Annexe 9 : proposition de démarche de résolution et éléments de résolution

1ère partie : distinction des entreprises sur l'écran d'accueil

Cette partie a déjà été faite en PPE par certains. On peut ici se contenter d'ajouter un Radio Bouton dans le formulaire de la page login.jsp.

Par exemple :

```
        <p>Etes vous une entreprise ?</p>
         <input type="radio" name="entreprise" value="oui" id="entreprise_oui" /><label
for="entreprise_oui">Oui</label>
         <input type="radio" name="entreprise" value="non" id="entreprise_non" /><label
for="entreprise_non">Non</label>
```

Remarque n°1 : si vous voulez utiliser du JavaScript ici, il faut s'assurer:

-répertoire dédie au JS dans web\resource\js (répertoire à créer pour des étudiants « en retard » de PPE)

- utilisation de tag JSP <c :url> pour avoir une url propre contextuelle à l'application

Exemple : `<script type="text/javascript" src="<c:url value="/resource/js/formulaire.js" />"></script>`

Remarque n°2 : certains étudiants peuvent proposer autre chose. Par exemple une listbox dans le formulaire, marche très bien également.

2ème partie : ajout de la Servlet / JSP et traitement conditionnel

a. Dans Configurator.java

Ajout d'une Url /entreprises correspondant à la nouvelle Servlet

b. Ajout nouvelle classe / servlet nommée EntrepriseServlet

Remarque n°1 : dans notre application, les Servlet sont configurées par des annotations (et non pas dans le fichier web.xml comme cela se faisait avant en J2EE)

Exemple ici :

```
@WebServlet(
        name = "entrepriseServlet",
        urlPatterns = {"/entreprises"},
        loadOnStartup = 1
)
```

Remarque n°2 :

Le seul code nécessaire est une redirection vers la JSP (ici entreprise.jsp) dans la méthode GET.

```
request.getRequestDispatcher("/WEB-INF/jsp/view/entreprise.jsp")
               .forward(request, response);
```

c. Créez la JSP entreprise.jsp

Il faudra également pour répondre aux points 3 et 4 ajouter le code de retour vers l'application de login.

d. Modification du Bean UserBean pour inclure la présence d'une entreprise.

Dans UserBean. Java, rajouter un champ String estEntreprise avec ses getter et setter

e. Modification du code de la Servlet login.

Modifiez le code de la classe LoginServlet pour orienter vers la page entreprise si on se connecte comme entreprise

```
else if (user.isValid() && entreprise.equals("oui"))
   {
       session.setAttribute("username", username);
       request.changeSessionId();
       response.sendRedirect("entreprises");
```

Annexe 10 : 2ème exemple d'expression de besoin par rapport au contexte et solution

Au delà de l'écran d'accueil (login), on souhaite logger les connexions des utilisateurs entreprise :

- distinguer les entreprises (des autres utilisateurs) sur l'écran d'accueil
- on souhaite également faire persister dans la base de données de l'application existante le fait qu'une entreprise se soit connectée, avec la date et l'heure de la connexion

1ème partie : ajout d'un logging entreprise

Idem Annexe 9

2ème partie : ajout d'un logging entreprise

Une solution partielle est ici donnée (on fusionne la notion entreprise et connexion).

a. Modification du modèle de données

On ajout une table ENTREPRISE_CONNEXION, avec un ID_CONNEXION, ID_ENTREPRISE et date / heure de connexion.

```
CREATE TABLE ENTREPRISE_CONNEXION
(
ID_CONNEXION int identity(1,1),
ID_ENTREPRISE int,
DATE_CONNEXION DATETIME
);
```

Remarque : IDENTITY est pour l'auto incrémentation sous SQL Server

b. On ajoute un champ ID_ENTREPRISE dans la table LOGIN_USER. On l'alimente via Management Studio pour le besoin

```
ALTER TABLE LOGIN_USER ADD ID_ENTREPRISE INT;
```

ou alors via Microsoft SQL Management Studio

c. On ajoute une variable num_entreprise dans le bean (et les méthodes get et set associées)
d. On alimente ce bean à partir de la valeur d'entreprise récupérée dans le SELECT (méthode login dans UserDAO)

```
int numEntreprise = rs.getInt("ID_ENTREPRISE") ;

bean.setNumEntreprise(numEntreprise);
```

On ajoute une méthode d'insertion dans le DAO : **exemple**

```
        public static void executeInsert(int num_entreprise) {
                try {
                     currentCon = ConnectionManager.getConnection();
Calendar calendar = Calendar.getInstance();
java.sql.Timestamp dateDuJour = new java.sql.Timestamp(calendar.getTime().getTime());


                        String insertion =
                                 "INSERT INTO ENTREPRISE_CONNEXION(ID_ENTREPRISE,DATE_CONNEXION)
VALUES (?,?)";
                        PreparedStatement preparedStatement =
currentCon.prepareStatement(insertion);
                        preparedStatement.setInt(1, num_entreprise);
                        preparedStatement.setTimestamp(2, dateDuJour);

                        // (4) execute the sql timestamp insert statement, then shut everything
down
                        preparedStatement.executeUpdate();
                        preparedStatement.close();

                }
                catch (Exception e) {
                   e.printStackTrace();
                }
            }
```

Remarque : cette solution complète éviter ce qu'on appelle l'injection SQL. Une solution simplifiée est possible.

e. On appelle cette méthode dans LoginServlet

```
UserDAO.executeInsert(user.getNumEntreprise());
```