Application de connexion utilisant Java Swing + JDBC + MySQL

1. Introduction et Objectifs du Projet

Le projet a consisté à développer une application d'authentification utilisateur simple en utilisant **Java Swing** pour l'interface graphique , et en connectant celle-ci à une base de données **MySQL** via **JDBC**.

Les objectifs fonctionnels du TP étaient:

- L'authentification de l'utilisateur.
- La déconnexion de l'utilisateur.
- Le changement du mot de passe de l'utilisateur.

2. Réalisation de l'Interface Homme-Machine (IHM)

L'application est composée de trois interfaces principales, toutes développées avec les API Java Swing :

2.1. IHM d'Authentification (`AuthentificationUtilisateur`)

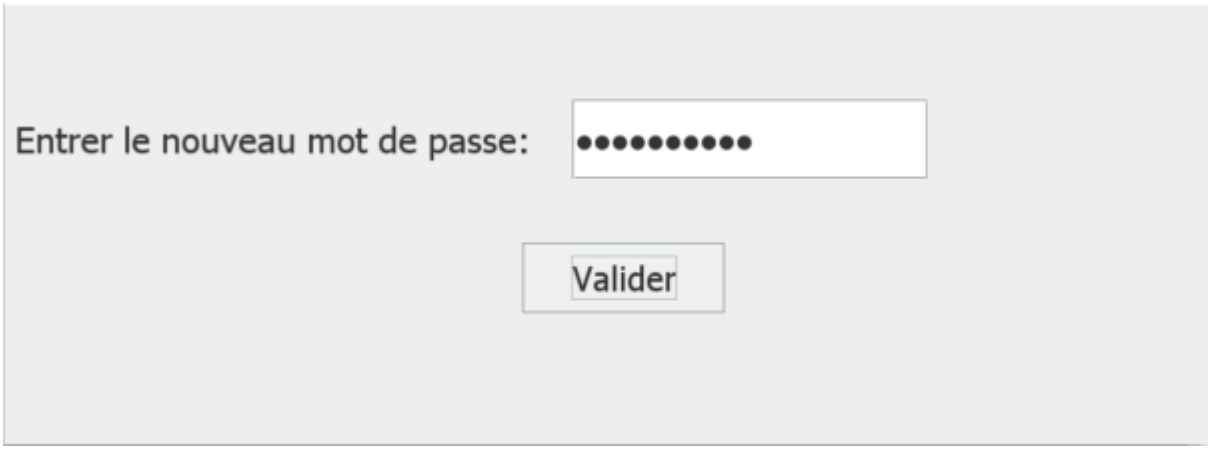
- **Rôle :** Permet à l'utilisateur de saisir son **identifiant** et son **mot de passe**.
- **Composants :** Un `JTextField` pour l'identifiant, un `JPasswordField` pour le mot de passe , et un bouton "Valider".
- **Logique de Connexion :** Lors du clic sur "Valider" , une connexion JDBC est établie avec la base de données `swing_demo`. Une requête préparée (`PreparedStatement`) est utilisée pour vérifier si la paire `nom` et `mdp` existe dans la table `etudiant`.
- **Résultat :** En cas de succès, la fenêtre d'authentification est fermée (`dispose()`) et l'IHM d'accueil (`AccueilUtilisateur`) est affichée. En cas d'échec, un message d'erreur est affiché.

2.2. IHM d'Accueil (`AccueilUtilisateur`)

- **Rôle :** S'affiche après une connexion réussie.
- **Fonctionnalités :** Comporte les boutons "Déconnexion" et "Changer le mot de passe".
- **Déconnexion :** Le clic sur "Déconnexion" affiche une boîte de dialogue de confirmation. En cas de confirmation, la fenêtre actuelle est fermée et la fenêtre d'authentification est rouverte.

2.3. IHM de Changement de Mot de Passe (`ChangerMdp`)

- **Rôle :** Permet à l'utilisateur de modifier son mot de passe.
- **Logique :** Saisit le nouveau mot de passe (via un `JPasswordField`). L'action du bouton "Valider" déclenche une requête `UPDATE` en JDBC pour mettre à jour le champ `mdp` de l'étudiant concerné dans la base de données. Un message de succès est affiché après la mise à jour.



3. Conclusion

Le projet a permis de mettre en pratique l'interfaçage entre l'IHM Java Swing et une base de données MySQL via JDBC. Les objectifs fonctionnels d'authentification, de déconnexion et de mise à jour du mot de passe ont été atteints, démontrant la capacité à lier efficacement la logique front-end (Swing) et la persistance des données (JDBC/MySQL).

TP 2 SWING- JTable-ORM:

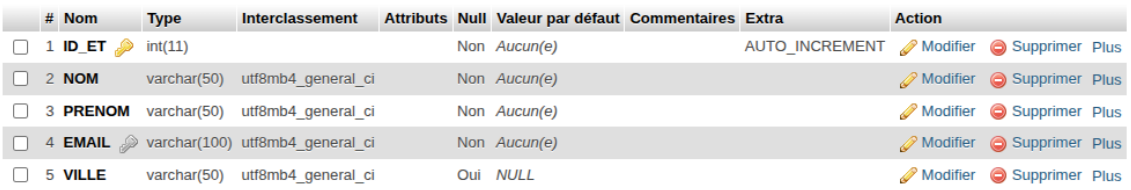
1. Introduction et Objectifs du Projet

Le projet visait à développer une application Java pour la gestion et la recherche d'étudiants. L'objectif principal était de concevoir une Interface Homme-Machine (IHM) en **Java Swing** pour interagir avec une base de données MySQL. Le projet imposait une architecture stricte de séparation des couches, mettant en œuvre le **Mapping Objet-Relationnel (ORM)** manuel via JDBC.

2. Architecture Logicielle et Technique

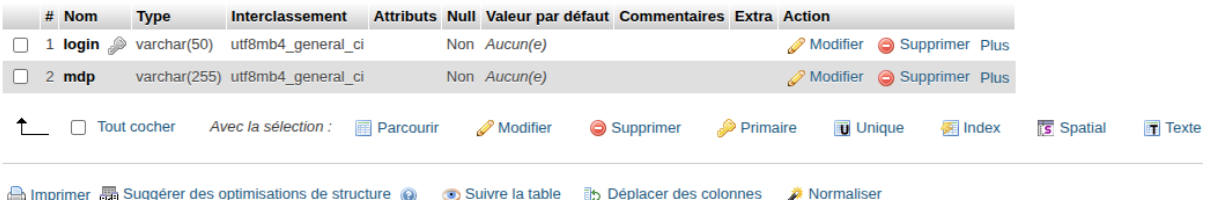
2.1. Couche Données (Persistance)

- **Base de Données :** MySQL, nommée Scolarité.
- **Table :** ETUDIANT, comprenant les champs ID_ET, NOM, PRENOM, EMAIL, et VILLE.



#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action
1	ID_ET	int(11)			Non	*Aucun(e)*		AUTO_INCREMENT	Modifier Supprimer Plus
2	NOM	varchar(50)	utf8mb4_general_ci		Non	*Aucun(e)*			Modifier Supprimer Plus
3	PRENOM	varchar(50)	utf8mb4_general_ci		Non	*Aucun(e)*			Modifier Supprimer Plus
4	EMAIL	varchar(100)	utf8mb4_general_ci		Non	*Aucun(e)*			Modifier Supprimer Plus
5	VILLE	varchar(50)	utf8mb4_general_ci		Oui	*NULL*			Modifier Supprimer Plus

- **Table :** USERS, comprenant les champs Login et mdp.



#	Nom	Type	Interclassement	Attributs	Null	Valeur par défaut	Commentaires	Extra	Action
1	login	varchar(50)	utf8mb4_general_ci		Non	*Aucun(e)*			Modifier Supprimer Plus
2	mdp	varchar(255)	utf8mb4_general_ci		Non	*Aucun(e)*			Modifier Supprimer Plus

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

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

2.2. Couche Métier (package metier)

- **Etudiant.java :** Classe persistante (Java Bean) modélisant un enregistrement de la table ETUDIANT. Elle contient les attributs privés (id_ET, nom, prenom, email, ville) et les accesseurs/mutateurs (getters/setters) correspondants.

```
package Metier;
public class Etudiant {
	private int ID_ET;
	private String NOM,PRENOM,EMAIL,VILLE;

	public Etudiant(int iD_ET, String nOM, String pRENOM, String eMAIL,
String vILLE) {
		super();
		ID_ET = iD_ET;
		NOM = nOM;
```

```
		PRENOM = pRENOM;
		EMAIL = eMAIL;
		VILLE = vILLE;
	}
	public Etudiant() {
		super();
	}
	public int getID_ET() {
		return ID_ET;
	}
	public void setID_ET(int iD_ET) {
		ID_ET = iD_ET;
	}
	public String getNOM() {
		return NOM;
	}
	public void setNOM(String nOM) {
		NOM = nOM;
	}
	public String getPRENOM() {
		return PRENOM;
	}
	public void setPRENOM(String pRENOM) {
		PRENOM = pRENOM;
	}
	public String getEMAIL() {
		return EMAIL;
	}
	public void setEMAIL(String eMAIL) {
		EMAIL = eMAIL;
	}
	public String getVILLE() {
		return VILLE;
	}
	public void setVILLE(String vILLE) {
		VILLE = vILLE;
	}
	@Override
	public String toString() {
		return "Etudiant [ID_ET=" + ID_ET + ", NOM=" + NOM + ", PRENOM="
+ PRENOM + ", EMAIL=" + EMAIL + ", VILLE="
				+ VILLE + "]";
	}



}
```

- **Scolarite.java :** Classe non persistante implémentant la logique métier et la gestion de la persistance (ORM manuel).
    - **Méthode Clé :** getEtudiants(String mc) qui exécute une requête SQL et retourne une Liste d'objets Etudiant dont le nom contient le mot-clé (mc).

```
    public List<Etudiant> getEtudiants(String mc) {
        List<Etudiant> etudiants = new ArrayList<>();
        Connection conn = SingletonConnection.getConnection();
        try {
            String sql = "SELECT * FROM etudiants WHERE NOM LIKE ?";
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setString(1, "%" + mc + "%");
            ResultSet rs = ps.executeQuery();
            while (rs.next()) {
                Etudiant e = new Etudiant();
                e.setID_ET(rs.getInt("ID_ET"));
                e.setNOM(rs.getString("NOM"));
                e.setPRENOM(rs.getString("PRENOM"));
                e.setEMAIL(rs.getString("EMAIL"));
                e.setVILLE(rs.getString("VILLE"));
                etudiants.add(e);
            }
            rs.close();
            ps.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return etudiants;
    }
```

- **AjoutEtudiant :** addEtudiants(etudiant e) qui exécute une requête SQL pour ajouter un étudiant à la base de donnée.

```
    public void addEtudiant(Etudiant e) {
        Connection conn = SingletonConnection.getConnection();
        try {
            String sql = "INSERT INTO etudiants (NOM, PRENOM, EMAIL, VILLE)
VALUES (?, ?, ?, ?)";
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setString(1, e.getNOM());
            ps.setString(2, e.getPRENOM());
            ps.setString(3, e.getEMAIL());
            ps.setString(4, e.getVILLE());
            ps.executeUpdate();
            ps.close();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
```

- **updateEtudiant :** updateEtudiants(etudiant e) qui exécute une requête SQL pour modifier un étudiant de la base de donnée.

```java
    public void updateEtudiant(Etudiant e) {
        Connection conn = SingletonConnection.getConnection();
        try {
            String sql = "UPDATE etudiants SET NOM=?, PRENOM=?, EMAIL=?,
VILLE=? WHERE ID_ET=?";
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setString(1, e.getNOM());
            ps.setString(2, e.getPRENOM());
            ps.setString(3, e.getEMAIL());
            ps.setString(4, e.getVILLE());
            ps.setInt(5, e.getID_ET());
            ps.executeUpdate();
            ps.close();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
```

- **deleteEtudiant :** addEtudiants(int id) qui exécute une requête SQL pour supprimer un étudiant de la base de donnée grâce a son id.

```java
    public void deleteEtudiant(int id) {
        Connection conn = SingletonConnection.getConnection();
        try {
            String sql = "DELETE FROM etudiants WHERE ID_ET=?";
            PreparedStatement ps = conn.prepareStatement(sql);
            ps.setInt(1, id);
            ps.executeUpdate();
            ps.close();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
    }
```

3. Couche Présentation (IHM - Java Swing)

Le développement de l'interface graphique a été réalisé en **Java Swing** (package pres).

3.1. Fonctionnalité de Recherche

- **Interface Principale :** Conçue pour permettre la saisie d'un **mot-clé** et l'affichage des étudiants correspondants.
- **Composants :** Un champ de saisie pour le mot-clé, un bouton de validation ("OK"), et un composant **JTable** pour afficher les résultats de la recherche (CODE ET, NOM, PRENOM, EMAIL).
- **Gestion des Événements :** L'événement (clic sur "OK") déclenche l'appel à la méthode métier getEtudiants(String mc) et met à jour dynamiquement le modèle de données de la JTable.

3.2. Autres Interfaces Développées

Le projet a inclus la création d'interfaces supplémentaires essentielles à la gestion:

- Une IHM pour l'**authentification**.

- Une IHM pour les opérations **CRUD** (Ajouter, Modifier et Supprimer) un étudiant.

4. Conclusion et Perspectives

Le projet m'a permis de maîtriser la chaîne complète du développement d'une application de gestion : conception de la base de données, implémentation des classes métier (Java Beans et ORM via JDBC) et réalisation d'une IHM fonctionnelle en Swing.

Application Java SWING en architecture MVC Sous Eclipse & DAO

1. Introduction et Objectifs du Projet

Le projet a consisté à développer une application simple de gestion d'annuaire, en mettant l'accent sur la structuration du code selon une architecture **Modèle-Vue-Contrôleur (MVC)**. L'objectif principal était de garantir une séparation stricte des préoccupations, notamment entre la couche d'accès aux données et la logique métier, en utilisant le motif **Data Access Object (DAO)**.

2. Architecture Logicielle : DAO et MVC

L'application est décomposée en trois couches principales, correspondant au modèle DAO/MVC :

2.1. Couche Modèle (Métier et Données)

Cette couche comprend la logique de l'application et l'accès à la persistance :

- **Entité (Modèle métier) :** La classe `annuaire` représente l'objet métier (l'enregistrement de l'annuaire).

```java
package annuaire.model;
/**
*
* @author balbali
*/
public class annuaire {
	private Integer id;
	private String num;
	private String nom;
	private String adresse;
	/**
	 *
	 * @return
	 */
	public String getAdresse() {
		return adresse;
	}
	/**
	 *
	 * @param adresse
	 */
	public void setAdresse(String adresse) {
		this.adresse = adresse;
	}
	/**
	 *
```

```
	 * @return
	 */
	public Integer getId() {
		return id;
	}
	/**
	 *
	 * @param id
	 */
	public void setId(Integer id) {
		this.id = id;
	}
	/**
	 *
	 * @return
	 */
	public String getNom() {
		return nom;
	}
	/**
	 *
	 * @param nom
	 */
	public void setNom(String nom) {
		this.nom = nom;
	}
	/**
	 *
	 * @return
	 */
	public String getNum() {
		return num;
	}
	/**
	 *
	 * @param num
	 */
	public void setNum(String num) {
		this.num = num;
	}
}
```

- **DAO (Data Access Object) :** Des classes et interfaces spécifiques sont utilisées pour encapsuler la logique d'accès à la base de données (JDBC/MySQL). Le DAO fournit les méthodes CRUD (Création, Lecture, Mise à jour, Suppression) pour l'objet `annuaire`, isolant le reste de l'application des détails de la base de données.

```
public class daoAnnuaire implements implementAnnuaire {
	Connection connection;
	final String insert = "INSERT INTO annuaire (num, nom, adresse) VALUES
(?, ?, ?);";
```

```
	final String update = "UPDATE annuaire set num=?, nom=?, adresse=?
where id=? ;";
	final String delete = "DELETE FROM annuaire where id=? ;";
	final String select = "SELECT * FROM annuaire;";
	final String recherchenom = "SELECT * FROM annuaire where nom like ?";
	/**
	 *
	 */
	public daoAnnuaire() {
		connection = SingletonConnection.getConnection();
	}
	/**
	 *
	 * @param b
	 */
	public void ajouter(annuaire b) {
		PreparedStatement statement = null;
		try {
			statement = connection.prepareStatement(insert);
			statement.setString(1, b.getNum());
			statement.setString(2, b.getNom());
			statement.setString(3, b.getAdresse());
			statement.executeUpdate();
			ResultSet rs = statement.getGeneratedKeys();
			while (rs.next()) {
				b.setId(rs.getInt(1));
			}
		} catch (SQLException ex) {
		} finally {
			try {
				statement.close();
			} catch (SQLException ex) {
				ex.printStackTrace();
			}
		}
	}
	/**
	 *
	 * @param b
	 */
	public void update(annuaire b) {
		PreparedStatement statement = null;
		try {
			statement = connection.prepareStatement(update);
			statement.setString(1, b.getNum());
			statement.setString(2, b.getNom());
			statement.setString(3, b.getAdresse());
			statement.setInt(4, b.getId());
			statement.executeUpdate();
		} catch (SQLException ex) {
			ex.printStackTrace();
		} finally {
			try {
```

```
                        statement.close();
                  } catch (SQLException ex) {
                        ex.printStackTrace();
                  }
            }
      }
      /**
       *
       * @param id
       */
      public void supprimer(int id) {
            PreparedStatement statement = null;
            try {
                  statement = connection.prepareStatement(delete);
                  statement.setInt(1, id);
                  statement.executeUpdate();
            } catch (SQLException ex) {
                  ex.printStackTrace();
            } finally {
                  try {
                        statement.close();
                  } catch (SQLException ex) {
                        ex.printStackTrace();
                  }
            }
      }
      /**
       *
       * @return
       */
      public List<annuaire> getALL() {
            List<annuaire> lb = null;
            try {
                  lb = new ArrayList<annuaire>();
                  Statement st = connection.createStatement();
                  ResultSet rs = st.executeQuery(select);
                  while (rs.next()) {
                        annuaire b = new annuaire();
                        b.setId(rs.getInt("id"));
                        b.setNum(rs.getString("num"));
                        b.setNom(rs.getString("nom"));
                        b.setAdresse(rs.getString("adresse"));
                        lb.add(b);
                  }
            } catch (SQLException ex) {

Logger.getLogger(daoAnnuaire.class.getName()).log(Level.SEVERE,
                              null, ex);
            }
            return lb;
      }
      /**
       *
```

```
         * @param nom
         * @return
         */
        public List<annuaire> getRechercheNom(String nom) {
                List<annuaire> lb = null;
                try {
                        lb = new ArrayList<annuaire>();
                        PreparedStatement st =
connection.prepareStatement(recherchenom);
                        st.setString(1, "%" + nom + "%");
                        ResultSet rs = st.executeQuery();
                        while (rs.next()) {
                                annuaire b = new annuaire();
                                b.setId(rs.getInt("id"));
                                b.setNum(rs.getString("num"));
                                b.setNom(rs.getString("nom"));
                                b.setAdresse(rs.getString("adresse"));
                                lb.add(b);
                        }
                } catch (SQLException ex) {

Logger.getLogger(daoAnnuaire.class.getName()).log(Level.SEVERE,
                                        null, ex);
                }
                return lb;
        }
}
```

- **Classe de Service/Contrôleur (Modèle du MVC) :** Une classe peut être présente pour gérer la logique de haut niveau et orchestrer les interactions entre le DAO et l'IHM.

```
public class controleurAnnuaire {
        FrameTelephone frame;
        implementAnnuaire implAnnuaire;
        List<annuaire> lb;
        /**
         *
         * @param frame
         */
        public controleurAnnuaire(FrameTelephone frame) {
                this.frame = frame;
                implAnnuaire = new daoAnnuaire();
                lb = implAnnuaire.getALL();
        }
        // Vider les champs
        /**
         *
         */
        public void reset() {
                frame.getTxtID().setText("");
```

```
            frame.getTxtNum().setText("");
            frame.getTxtNom().setText("");
            frame.getTxtAdresse().setText("");
      }
      // Afficher les données de la table
      /**
       *
       */
      public void isiTable() {
            lb = implAnnuaire.getALL();
            tableModelAnnuaire tmb = new tableModelAnnuaire(lb);
            frame.getTabelData().setModel(tmb);
      }
      //cette fonction pour afficher les données sélectionnées à partir de
la grille
      /**
A.F.BALBALI 20
       *
       * @param row
       */
      public void isiField(int row) {
            frame.getTxtID().setText(lb.get(row).getId().toString());
            frame.getTxtNum().setText(lb.get(row).getNum());
            frame.getTxtNom().setText(lb.get(row).getNom());
            frame.getTxtAdresse().setText(lb.get(row).getAdresse());
      }
      //Fonction permettant insérer des données en fonction de l'entrée
utilisateur du champ de texte dans le cadre
      /**
       *
       */
      public void insert() {
            annuaire b = new annuaire();
            b.setNum(frame.getTxtNum().getText());
            b.setNom(frame.getTxtNom().getText());
            b.setAdresse(frame.getTxtAdresse().getText());
            implAnnuaire.ajouter(b);
      }
      //mettre à jour les données sur la base de l'entrée d'utilisateur de
la textfield dans le cadre
      /**
       *
       */
      public void update() {
            try {
                  annuaire b = new annuaire();
                  b.setId(b.getId());
                  b.setNum(frame.getTxtNum().getText());
                  b.setNom(frame.getTxtNom().getText());
                  b.setAdresse(frame.getTxtAdresse().getText());
                  b.setId(Integer.parseInt(frame.getTxtID().getText()));
                  implAnnuaire.update(b);
            } catch (NumberFormatException numberFormatException) {
```

```
                  JOptionPane.showMessageDialog(frame, "Sélectionnez les
donnéesà mettre à jour");
            }
      }
      //fonction pour supprimer les données sélectionnées
      /**
       *
       */
      public void delete() {
            if (!frame.getTxtID().getText().trim().isEmpty()) {
                  int id = Integer.parseInt(frame.getTxtID().getText());
                  implAnnuaire.supprimer(id);
            } else {
                  JOptionPane.showMessageDialog(frame, "Sélectionnez les
données à supprimer");
            }
      }
      /**
       *
       */
      public void isiTableRechercheNom() {
            lb =

implAnnuaire.getRechercheNom(frame.getTxtRechercheNom().getText());
            tableModelAnnuaire tmb = new tableModelAnnuaire(lb);
            frame.getTabelData().setModel(tmb);
      }
      /**
       *
       */
      public void recherchenom() {
            if (!frame.getTxtRechercheNom().getText().trim().isEmpty()) {

implAnnuaire.getRechercheNom(frame.getTxtRechercheNom().getText());
                  isiTableRechercheNom();
            } else {
                  JOptionPane.showMessageDialog(frame, "Sélectionner des
données");
            }
      }
}
```

2.2. Couche Vue (Présentation)

- Développée en **Java Swing**, la Vue est responsable de l'affichage de l'annuaire et de la capture des actions utilisateur (clics, saisie).

3. Bilan et Apprentissages

Ce projet a été fondamental pour l'apprentissage de l'**ingénierie logicielle**. L'adoption du modèle DAO/MVC a permis de :

- **Faciliter la Maintenance :** Toute modification de la base de données (ex: changement de SGBD) n'impacte que le code dans la couche DAO.
- **Améliorer la Testabilité :** Les différentes couches peuvent être testées indépendamment.