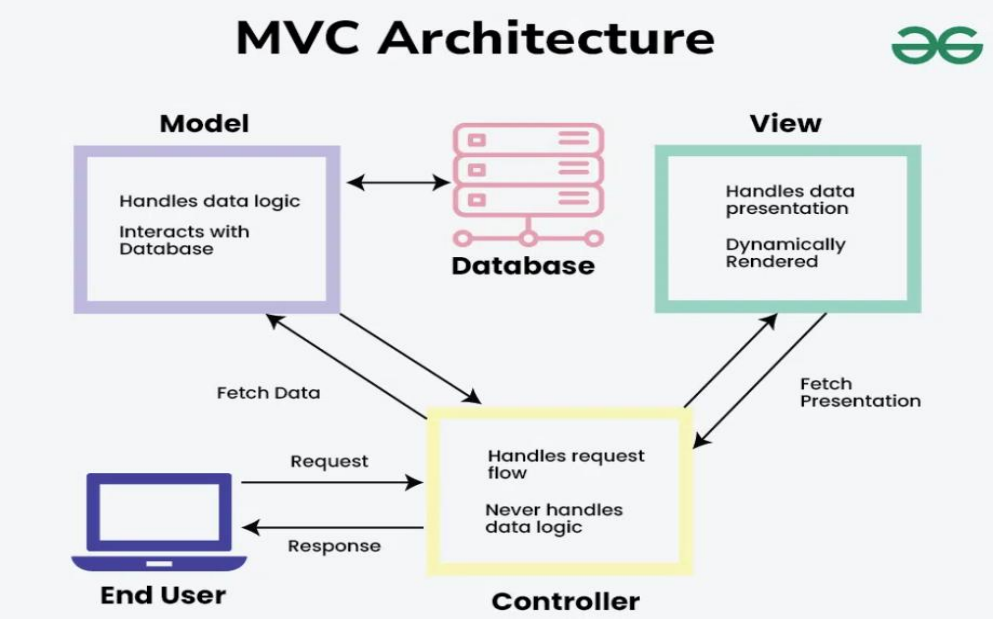
COMPTE RENDU TP J2EE (MVC) - Gestion des notes



Réalisé par :

Yassir Mrabti

Encadré par :

Pr.Benslimane Mohamed

1. Introduction

Ce travail pratique consiste à développer une application web J2EE respectant le modèle MVC (Modèle–Vue–Contrôleur) permettant de saisir le numéro d'inscription d'un étudiant et d'afficher ses notes ainsi que la moyenne correspondante. L'application s'appuie sur une base de données MySQL nommée **SCOLARITE** contenant la table **NOTES**.

---

2. Objectif du TP

- Mettre en pratique l'architecture MVC dans une application web J2EE
- Utiliser JDBC pour accéder à une base MySQL et récupérer les notes d'un étudiant
- Afficher les résultats dans une page JSP avec calcul de la moyenne
- Comprendre le déploiement et l'exécution d'une application web sur Tomcat

---

3. Architecture du projet

Le projet est organisé en trois couches selon le modèle MVC :

3.1 Couche Modèle (Métier / Accès Base de Données)

Classe `Note` : objet métier représentant une note - `idnote` (int) : identifiant unique - `numInscr` (String) : numéro d'inscription de l'étudiant - `matiere` (String) : nom de la matière - `note` (float) : valeur de la note - Getters et setters pour chaque attribut

Classe `Utilitaire` : gère la connexion JDBC à la base MySQL - Bloc static chargant le driver et ouvrant la connexion - Méthode `getConnection()` retournant la connexion

Classe `Scolarite` : logique métier - Méthode `getNotes(String numIns)` récupérant les notes d'un étudiant - Exécute `SELECT * FROM NOTES WHERE NUMINS = ?` - Retourne un `Vector<Note>` avec les résultats.

3.2 Couche Contrôleur

Classe `ControleurServlet` : servlet HTTP héritant de `HttpServlet` - Récupère le paramètre `numIns` du formulaire - Appelle `Scolarite.getNotes(numIns)` pour obtenir les notes - Stocke les résultats dans un objet `NoteForm` en session - Redirige vers la page JSP d'affichage

3.3 Couche Présentation

Classe `NoteForm` : form-bean transportant les données - Attributs : `numIns` (String), `lesNotes` (Vector) - Getters et setters

Page JSP (`Notes.jsp` ou `index.jsp`) - Formulaire pour saisir le numéro d'inscription - Tableau affichant les notes récupérées - Calcul et affichage de la moyenne

---

4. Base de données

Table NOTES

```sql
CREATE TABLE NOTES (
  IDNOTE INT NOT NULL AUTO_INCREMENT,
  NUMINS VARCHAR(12) NOT NULL,
  MATIERE VARCHAR(25) NOT NULL,
  NOTE FLOAT NOT NULL DEFAULT 0,
  PRIMARY KEY (IDNOTE)
);
```

Données de test

```sql
INSERT INTO NOTES (NUMINS, MATIERE, NOTE) VALUES
('HE1', 'math', 12),
('HE1', 'java', 11),
('HE1', 'TCP', 8);
```

---

5. Flux MVC et fonctionnement

Étape 1 : Affichage initial

L'utilisateur accède à `Notes.jsp` et voit un formulaire avec un champ de saisie pour le numéro d'inscription.

Étape 2 : Soumission du formulaire

Après saisie de `numIns` (ex: "HE1") et clic sur OK, le formulaire envoie une requête POST vers `/controleur`.

Étape 3 : Traitement serveur (Servlet)

1. `ControleurServlet.doPost()` reçoit la requête
2. Extrait le paramètre `numIns` via `request.getParameter("numIns")`
3. Crée une instance de `Scolarite` et appelle `getNotes(numIns)`
4. Crée un objet `NoteForm`, affecte `numIns` et la liste des notes
5. Stocke `NoteForm` en session via `session.setAttribute("nf", nf)`
6. Redirige vers `index.jsp`

Étape 4 : Affichage des résultats (JSP)

1. Récupère `NoteForm` depuis la session
2. Parcourt le `Vector<Note>` à l'aide d'un itérateur
3. Affiche chaque note dans un tableau
4. Calcule et affiche la moyenne : `somme / compteur`

---

6. Erreurs rencontrées et solutions

Erreur : HTTP 500 – NullPointerException sur la connexion JDBC

Symptôme : - Page d'erreur HTTP 500 dans le navigateur - Stack trace affichant : `java.lang.NullPointerException: Cannot invoke "java.sql.Connection.prepareStatement(String)" because "conn" is null`

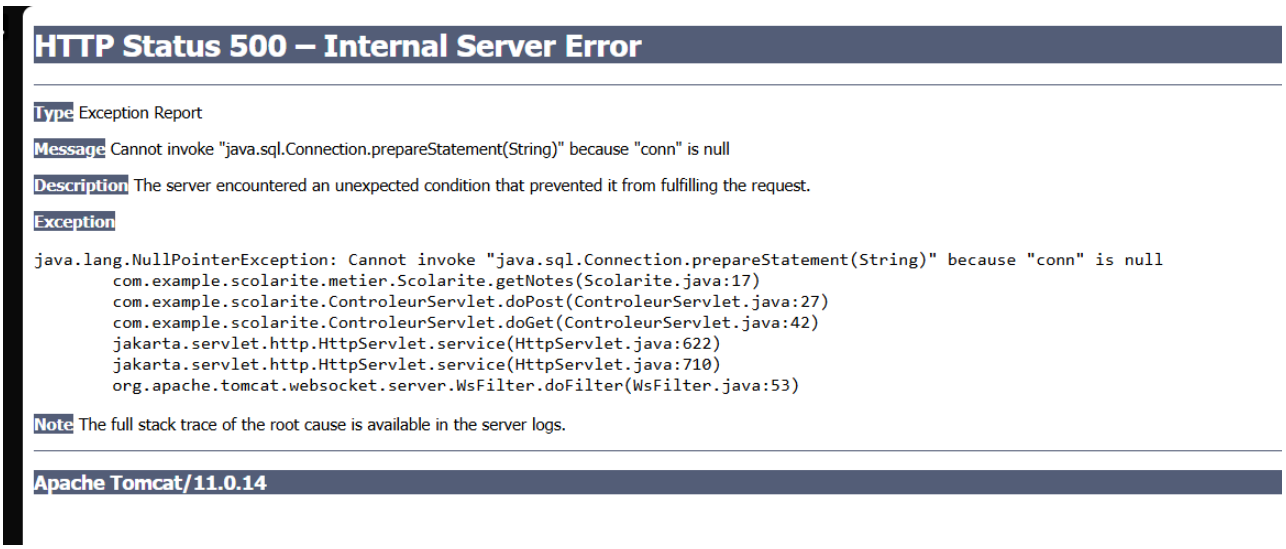
Cause exacte (détaillée) :

La classe `Utilitaire` contient un bloc static chargant le driver MySQL et ouvrant la connexion :

```
Class.forName("com.mysql.cj.jdbc.Driver");
conn = DriverManager.getConnection("jdbc:mysql://localhost:3306/SCOLARITE",
"root", "");
```

Le problème : le driver MySQL (`mysql-connector-j-8.4.0.jar`) n'était **pas visible** dans le classpath de Tomcat au moment du déploiement.

Quand Tomcat charge la classe `Utilitaire`, le bloc static s'exécute : - `Class.forName(...)` tente de charger le driver - Le driver n'est pas trouvé → une `ClassNotFoundException` est levée et capturée silencieusement - `conn` reste `null` - Quand `Scolarite.getNotes()` appelle `conn.prepareStatement()`, on obtient un `NullPointerException`



Observation importante (apprentissage clé) :

La classe `Test` (avec `main()`) fonctionnait correctement, mais pas l'application web sur Tomcat. Pourquoi ?

- **Exécution en mode IDE :** le jar MySQL était ajouté comme library du module, donc trouvable
- **Déploiement sur Tomcat :** Tomcat utilise son propre classpath (`CATALINA_HOME/lib` et `WEB-INF/lib`), où le jar n'était pas présent

Conclusion : un programme Java peut fonctionner localement et échouer sur un serveur en raison des différences de classpath.

Solution appliquée :

Ajout manuel du jar dans le dossier `lib` de Tomcat :

1. Localisation du jar `mysql-connector-j-8.4.0.jar`
2. Copie du jar vers `C:\Program Files\Apache Software Foundation\Tomcat 11.0\lib\`
3. Redémarrage de Tomcat

Pourquoi ça fonctionne : - Tomcat charge automatiquement tous les jars situés dans son dossier `lib` - Le driver devient alors visible au runtime - Le bloc static de `Utilitaire` peut charger le driver avec succès - conn est maintenant une connexion valide

Alternatives (bonnes pratiques) : - Placer le jar dans `WEB-INF/lib` du projet (inclus automatiquement dans le war) - Utiliser Maven et déclarer la dépendance MySQL dans `pom.xml`

Résultat : Après redémarrage de Tomcat, l'application fonctionne sans erreur 500.

---

7. Points clés appris

1. **Différence classpath IDE vs Tomcat** : Un même code peut fonctionner localement mais échouer sur un serveur si les dépendances ne sont pas correctement déployées.

2. **Importance du format web.xml** : Tomcat applique strictement les spécifications Jakarta EE 10. Le `url-pattern` doit commencer par `/`.

3. **Classpath Tomcat** : Placer les drivers JDBC dans `CATALINA_HOME/lib` ou `WEB-INF/lib` est essentiel.

4. **MVC en pratique** : La séparation claire entre Model (métier), View (JSP) et Controller (servlet) facilite le debugging et la maintenance.

---

8. Résultats obtenus

Après correction de toutes les erreurs, l'application fonctionne correctement :

- Saisie du numéro d'inscription dans le formulaire
- Récupération des notes depuis la base MySQL
- Affichage des notes dans un tableau HTML
- Calcul et affichage automatique de la moyenne
- Navigation fluide entre le formulaire et l'affichage des résultats

Exemple d'exécution : - Entrée : HE1 - Résultat affiché : - Matière : math → Note : 12.0 - Matière : java → Note : 11.0 - Matière : TCP → Note : 8.0 - **Moyenne : 10.333333**

---

9. Conclusion

Ce TP a permis de mettre en pratique le modèle MVC en J2EE à travers une application réelle connectée à une base de données MySQL. Les erreurs rencontrées ont renforcé la compréhension de :

- L'architecture d'une application web J2EE et le rôle de chaque composant
- La gestion des dépendances et du classpath dans un environnement serveur
- La différence cruciale entre l'exécution locale et le déploiement sur un conteneur web
- L'importance de la configuration correcte du `web.xml`

Ces compétences acquises seront applicables à des projets web plus complexes utilisant des frameworks comme Spring ou Jakarta EE avancé.

Fin du rapport

Préparé le 12 décembre 2025