



RAPPORT DE PROJET

Projet JEE

Table des matières :

I. Introduction	2
1. Contexte	2
2. Demande du client	2
II. Développement du projet	3
1. Conception	3
2. Implémentation	4
A. Projet initial	5
a) Les pages et leurs servlets, associés à la fonctionnalité de gestion de notes	5
b) Les fonctionnalités et leurs implémentations	5
c) Le respect des exigences et la cohérence des données	10
B. Base de données	11
C. Hibernate	12
D. Spring Boot	12
III. Travail en équipe	14
1. Organisation	14
V. Bibliographie	14

I. Introduction

1. Contexte

Dans le cadre de notre deuxième année de cycle ingénieur informatique au sein de CY Tech, il nous a été demandé de réaliser un projet informatique. Celui-ci fait partie intégrante de la matière « Développement Distribué Java EE » du module informatique. Ce projet est un travail d'équipe.

Notre équipe est constituée de cinq personnes en deuxième année de cycle ingénieur informatique :

- Quentin Fillion (ING2 GSI2), qui souhaite se spécialiser en cybersécurité en dernière année
- Youenn Bogaer (ING2 GSI2), qui souhaite se spécialiser dans l'intelligence artificielle
- Victor André (ING2 GSI2), qui souhaite se spécialiser dans la data
- Wahaj Dilawar (ING2 GSI2), qui souhaite se spécialiser en cybersécurité
- Dawid Malicki (ING2 GSI2), qui souhaite se spécialiser dans l'intelligence artificielle

2. Demande du client

“L'application doit permettre aux étudiants, enseignants et administrateurs de gérer les informations académiques. Les fonctionnalités incluent la gestion des étudiants, des cours, des inscriptions et des résultats.”

Quelques contraintes importantes :

- L'application doit comporter une interface graphique
- L'application doit respecter le modèle MVC
- La gestion des utilisateurs doit inclure un système d'authentification avec rôles (administrateur, enseignant, étudiant)
- Un système de notification par email doit informer les étudiants des changements importants
- Les interactions avec la base de données doivent garantir la cohérence des données
- Les données saisies doivent être vérifiées avant enregistrement dans les classes et la base de données

Cette liste n'est pas exhaustive et l'énoncé complet peut se retrouver dans le dossier 'docs' de notre dépôt GitHub, dans la branche 'README'.

II. Développement du projet

1. Conception

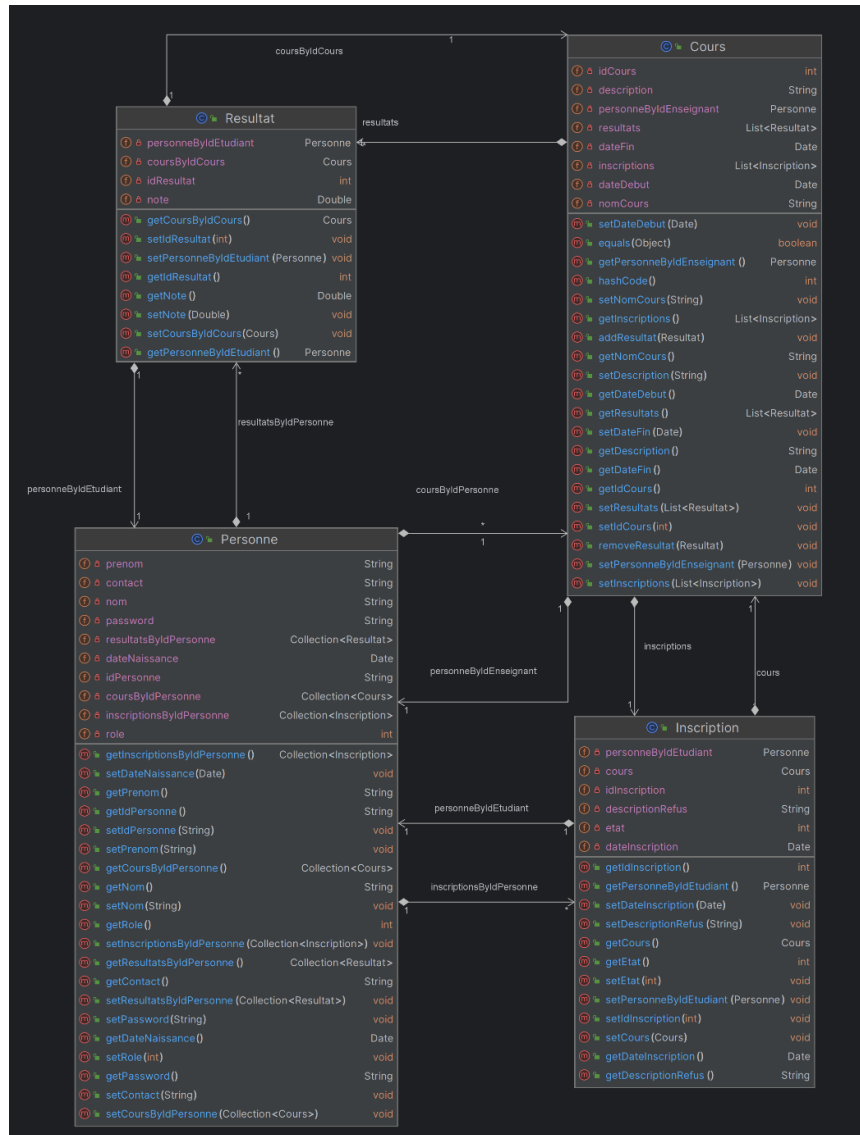


Figure 1 : Diagramme de classes

Les diagrammes sont également disponibles en meilleure qualité sur le dépôt GitHub, dans le dossier 'docs' de la branche 'README'.

2. Implémentation

Nous avons utilisé un [modèle vue-contrôleur](#).



Figure 2 : Modèle MVC avec Modèle (entity), Vue et Contrôleur

Chaque page de l'application, qui fonctionne un peu comme un site web, est gérée par une classe Java qui joue un rôle de contrôleur : à chaque interaction avec l'utilisateur, elle modifie l'affichage en fonction du bouton cliqué, des champs complétés ou des nouvelles données en jeu et gère aussi les transitions vers les autres pages.

La partie graphique se base sur certains principes de conception pour offrir une expérience utilisateur optimale.

A. Projet initial

Illustration du modèle MVC avec les fonctionnalités liées aux gestions des notes des étudiants.

a) Les pages et leurs servlets, associés à la fonctionnalité de gestion de notes

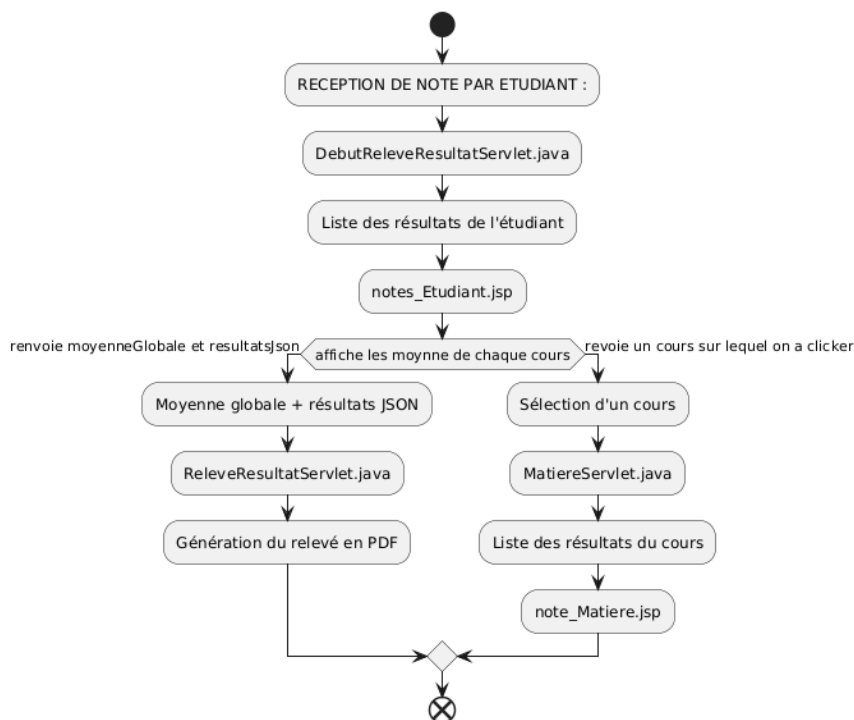


Figure 3 : Diagramme d'activités pour l'affichage des notes

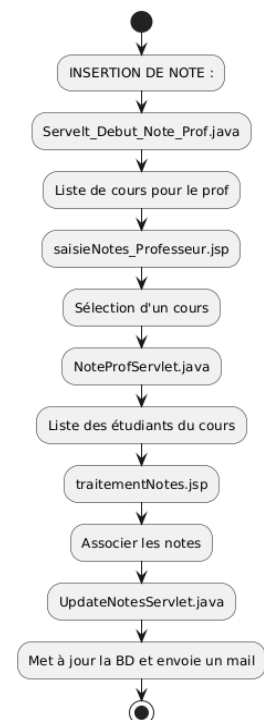


Figure 4 : Diagramme d'activités pour l'insertion des notes

b) Les fonctionnalités et leurs implémentations

Pour chaque cours, le professeur qui l'enseigne doit pouvoir saisir un ensemble de notes pour les étudiants participant au cours. Ces étudiants doivent avoir accès à leurs notes ainsi qu'à leur moyenne pour ce cours. Il doit être possible de générer un bulletin des moyennes d'un étudiant. Un mail doit être envoyé à l'étudiant lorsqu'une nouvelle note lui est attribué.

Choix de conception :

Le choix a été fait que lorsqu'un enseignant saisit une note il doit saisir les notes pour tous les élèves du cours, cela représente la note d'un examen pour ce cours.

Modèle, voir Figure 1 (Diagramme de classes) :

Les notes sont représentées par la classe Résultat. On y retrouve la note (Double), le cours (Cours) et l'étudiant concerné (Personne), ainsi qu'un identifiant (int) pour le résultat. Un résultat est associé à un seul cours, un seul étudiant mais un cours et un étudiant peuvent être associés à plusieurs résultats, respectivement par les biais des attributs Cours.resultats (List<Resultat>) et Personne.resultatsByldPersonne (Collection<Resultat>).

Vue :

Enseignant :

Lorsqu'un enseignant veut saisir une note il doit se connecter sur la page webapp/Vue/login.jsp, la servlet Controllers/Servlets/LoginServlet.java va vérifier que l'utilisateur existe et va autoriser l'accès.

L'utilisateur, étant un enseignant, il va être redirigé vers la page de menu pour les professeurs.



*Figure 5 : Page de menu professeur
webapp/Vue/Professeur/menu_Professeur.jsp*

Comme pour tous les pages de type menu, la connexion est vérifiée.

```
3  <%
4      // Vérifiez si l'utilisateur est connecté et est un prof
5      Personne user = (Personne) session.getAttribute("user");
6      if (user == null || user.getRole() != 2) {
7          // Redirigez vers la page de connexion si l'utilisateur n'est pas connecté ou n'est pas professeur
8          response.sendRedirect(request.getContextPath()+"/Vue/login.jsp");
9          return;
10     }
11 %>
```

*Figure 6 : Vérification de la connexion et de la légitimité de l'utilisateur
webapp/Vue/Professeur/menu_Professeur.jsp*

Tout commence avec la page webapp/Vue/Professeur/saisieNotes_Professeurs.jsp où l'enseignant connecté peut choisir parmi les cours qu'il enseigne lequel il veut noter. Cela passe

par un formulaire html avec des inputs de types “radio” pour sélectionner parmi la liste de cours celui à noter.

```

34     </thead>
35     <tbody >
36     <%
37         List<Cours> coursList = (List<Cours>) request.getAttribute("coursList");
38
39         if (coursList == null || coursList.isEmpty()) {
40             %>
41             <tr><td colspan="2">Pas de cours disponible</td></tr>
42             <%
43             } else {
44                 for (Cours cours : coursList) {
45                     %>
46                     <tr>
47                         <td><input type="radio" name="idCours" value="<%= cours.getIdCours() %>" required></td>
48                         <td><%= cours.getNomCours() %></td>
49                     </tr>
50                     <%=}%>
51                 </tbody>
52             </table>
53             <div style="text-align: center;">
54                 <button type="submit">Choisir ce cours</button>
55             </div>

```

Figure 7 : Contenu du formulaire de choix des cours pour l’enseignant
webapp/Vue/Professeur/saisieNotes_Professeur.jsp

La servlet Controllers/Servlets/Professeur/NoteProfServlet.java récupère l’information sur le cours et affiche sur la prochaine page la liste des étudiants du cours avec un formulaire permettant d’attribuer les notes.

```

74
75         request.setAttribute("students", students);
76
77         request.getRequestDispatcher("s:Vue/Professeur/traitementNotes.jsp").forward(request, response);
78     }

```

Figure 8 : Redirection vers la page de saisie de notes avec la liste des étudiants du cours à noter
Controllers/Servlets/Professeur/NoteProfServlet.java

Sur la page webapp/Vue/Professeur/traitementNotes.jsp on récupère l’information sur le cours et la liste des étudiants.

```

41     <%
42         Object coursObject = request.getAttribute("cours");
43         List<Personne> students = (List<Personne>) request.getAttribute("students");
44     %>

```

Figure 9 : Récupération des données envoyées par le servletModèle
webapp/Vue/Professeur/traitementNotes.jsp

La page webapp/Vue/Professeur/traitementNotes.jsp consiste en un formulaire html, composé d’un tableau des étudiant à noter et d’un input de type “number” restreint entre 0 et 20 et “required”. Dans notre conception un enseignant note toute la classe d’un coup.


```

51 <%
52     if (students != null && !students.isEmpty()) {
53     %>
54     <form action="<%= request.getContextPath() %>/UpdateNotesServlet" method="post" onsubmit="return validateForm();" >
55     <table>
56     <thead>
57     <tr>
58         <th>Nom</th>
59         <th>Prénom</th>
60         <th>Email</th>
61         <th>Note (0-20)</th>
62     </tr>
63     </thead>
64     <tbody>
65     <%
66         for (Personne student : students) {
67         %>
68     <tr>
69         <td><%= student.getNom() %></td>
70         <td><%= student.getPrenom() %></td>
71         <td><%= student.getIdPersonne() %></td>
72         <td>
73             <input type="number"
74                 name="note_<%= student.getIdPersonne() %>"
75                 min="0"
76                 max="20"
77                 required>
78         </td>

```

Figure 10 : Tableau des saisies de notes pour un cours
webapp/Vue/Professeur/traitementNotes.jsp

Comme on peut le voir le formulaire précédent lance la servlet `Controllers/Servlets/Professeur/updateNoteServlet.java` qui se charge de mettre à jour la base de données tout en s'assurant de la cohérence de cette dernière. En cas de réussite l'utilisateur est renvoyer vers la page `webapp/Vue/Professeur/confirmation.jsp` qui lui affiche le message "Les notes ont été soumises avec succès.". L'enseignant peut alors se déconnecter ou retourner au menu.

Etudiant :

Les étudiants voulant consulter leurs notes et moyennes se connectent grâce à la page `login.jsp` et accède ainsi au menu


Étudiant		
 Accueil Inscrire à un cours Générer le relevé de notes Déconnexion		
Inscriptions en attente ou refusées		
Nom du Cours	Professeur	État
Math	dawi malicki	En attente
BDD	dawi malicki	Refusé
Mes Cours :		
Nom du Cours	Professeur	Détails
Ethique	dawi malicki	<button>Voir Détails</button>

Figure 11 : Page menu étudiant
webapp/Vue/Etudiant/menu_Etudiant.jsp

Le menu contient un tableau des cours auquel l'étudiant est inscrit, avec la moyenne pour chacun des cours. Il a aussi la possibilité d'afficher le détail des notes pour un cours.

```

96      <%
97      if (courses != null) {
98          for (Cours cours : courses) {
99              if (cours != null) {
100                  <%
101                  <tr>
102                      <form action="<%= request.getContextPath() %>/MatiereServlet" method="GET">
103                          <td><%= cours.getNomCours() %></td>
104                          <td><%= cours.getPersonneByIdEnseignant().getPrenom() %> <%= cours.getPersonneByIdEnseignant().getNom() %></td>
105                          <td>
106                              <!-- Hidden input to pass the course ID -->
107                              <input type="hidden" name="idCours" value="<%= cours.getIdCours() %>">
108                              <button type="submit">Voir Détails</button>
109                          </td>
110                      </form>
111                  </tr>

```

Figure 12 : Tableau d’affichage des cours et de leur moyenne
src/main/webapp/Vue/Etudiant/menu_Etudiant.jsp

Lorsque l’utilisateur demande à voir le détail pour un cours, la servlet Controllers/Servlets/Etudiant/MatiereServlet.java redirige l’utilisateur vers src/main/webapp/Vue/Etudiant/note_Matiere.jsp avec la liste des résultats correspondant à l’étudiant courant, s’il existe des résultats. Autrement l’utilisateur est redirigé vers le menu étudiant.

La page src/main/webapp/Vue/Etudiant/note_Matiere.jsp affiche la liste des résultats de l’élève courant dans le cours qui a été choisi plus tôt.

```

56 <!-- Results Table -->
57 <div>
58     <h3>Résultats des Étudiants</h3>
59     <table>
60         <thead>
61             <tr>
62                 <th>Note Numéro</th>
63                 <th>Note</th>
64             </tr>
65         </thead>
66         <tbody>
67             <%
68                 if (resultats != null && !resultats.isEmpty()) {
69                     int counter = 1;
70                     for (Resultat resultat : resultats) {
71                         <%
72                         <tr>
73                             <td>Note numéro <%= counter++ %></td>
74                             <td><%= resultat.getNote() != null ? String.format("%.2f", resultat.getNote()) : "N/A" %></td>
75                         </tr>
76                         <%
77                     }
78                 } else {
79                     <%
80                     <tr>
81                         <td colspan="2">Aucun résultat disponible pour ce cours.</td>
82                     </tr>

```

Figure 13 : Tableau d’affichage des resultats de l’étudiant dans le cours
src/main/webapp/Vue/Etudiant/note_Matiere.jsp

La variable “resultats” est une List d’objet Resultat, ce sont les notes de l’étudiant dans le cours.

On remarque ligne 74 que la note est affichée si elle existe ou on affiche “N/A” sinon.

Finalement on peut l’utilisateur étudiant peut aussi générer un relevé de notes au format pdf en cliquant sur “Générer un relevé de notes” dans le header.

Contrôleurs :

La partie Vue détailler le parcours réalisé pour qu’une note choisie par un enseignant arrive jusqu’à un étudiant. La suite va détailler certains points pour ce qui concerne les servlets utilisées.

Lorsqu'un utilisateur tente de se connecter la servlet `Controllers/Servlets/LoginServlet.java` est appelée, les informations entrées dans le formulaire sont récupérées. On vérifie que le mot de passe fournit et l'identifiant correspondent à une personne existante dans la base, grâce à une requête HQL.

```

29  @Override
30  protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
31      String idPersonne = request.getParameter("idPersonne");
32      String password = request.getParameter("password");
33      System.out.println(idPersonne);
34      System.out.println(password);
35
36
37      // Vérifier les informations d'identification via Hibernate
38      Session session = HibernateUtil.getSessionFactory().openSession();
39      Query query = session.createQuery("FROM Personne WHERE idPersonne = :idPersonne AND password = :password");
40      query.setParameter("idPersonne", idPersonne);
41      query.setParameter("password", password);
42
43      Personne personne = (Personne) query.uniqueResult();
44      session.close();
45

```

Figure 14 : Récupération des données en request et vérification de l'existence de l'utilisateur dans la base
Controllers/Servlets/LoginServlet.java

Dans le cas où l'utilisateur existe nous vérifions à quel rôle il appartient afin de le rediriger vers la bonne page menuue. Si la connexion a échoué nous renvoyons l'utilisateurs vers la page de connexion avec un signalement d'erreur en passant par l'URL (voir l.65 et l.70 de la figure suivante).

```

43      Personne personne = (Personne) query.uniqueResult();
44      session.close();
45
46      if (personne != null) {
47          // Authentification réussie
48          HttpSession httpSession = request.getSession();
49          httpSession.setAttribute("user", personne);
50
51          // Redirige en fonction du rôle
52          switch (personne.getRole()) {
53              case 1: // Rôle étudiant
54                  response.sendRedirect(location: "Vue/Admin/menu_admin.jsp");
55                  break;
56              case 2: // Rôle enseignant
57                  System.out.println(personne.getIdPersonne());
58                  response.sendRedirect(location: "Vue/Professeur/menu_Professeur.jsp");
59                  break;
60              case 3: // Rôle administrateur
61                  response.sendRedirect(location: request.getContextPath()+"/MenuEtudiantServlet");
62                  break;
63              default:
64                  // Rôle inconnu
65                  response.sendRedirect(location: "Vue/login.jsp?error=true");
66                  break;
67          }
68      } else {
69          // Authentification échouée
70          response.sendRedirect(location: "Vue/login.jsp?error=true");
71      }

```

Figure 15 : Redirection des utilisateurs vers la page adéquate
Controllers/Servlets/LoginServlet.java

c) Le respect des exigences et la cohérence de données

Les exigences énoncées dans le sujet du projet ont été respectées tout en assurant la cohérence des données entre celles stockées dans les classes du modèle et celles présentes dans les différentes tables de la base de données.

L'exemple de la page de connexion, abordé précédemment (voir ci-dessous), illustre la façon avec laquelle les données rentrées par l'utilisateur sont vérifiées avant d'être traitées et/ou sauvegardées dans les différentes pages de l'application.

Les vérifications du format des données sont essentiellement effectuées grâce à l'ajout d'attributs dans les balises HTML des formulaires, notamment : 'type', 'min', 'max', etc. Cependant, ces contrôles n'empêchent pas l'utilisateur de fournir des données au mauvais format, comme par exemple pour les champs 'Prénom' ou 'Nom' dont l'attribut 'type' est simplement défini comme 'text'.

Pour pallier cette limite, des embranchement conditionnels et des blocs 'try-catch' ont été ajoutés aux servlets et aux contrôleurs, pour que, dans le cas où une opération ne pourrait pas être réalisée à cause d'un mauvais format de données notamment, un message d'erreur soit affiché sur l'interface l'utilisateur. Cette façon de faire est avantageuse car elle nous a permis d'éviter d'écrire des vérifications en se basant sur des motifs d'entrée utilisateur, ce qui aurait été un travail fastidieux et pas avantageux en termes de rapidité d'exécution du code.

```
16      <div class="form-container">
17      <h2>Connexion</h2>
18
19      <form action="<%= request.getContextPath() %>/login" method="POST">
20          <label for="idPersonne">Email</label>
21          <input type="email" id="idPersonne" name="idPersonne" required>
22
23          <label for="password">Mot de passe</label>
24          <input type="password" id="password" name="password" required>
25
26          <div style="text-align: center;">
27              <button type="submit">Se connecter</button>
28          </div>
29      </form>
30
31      <%= if (request.getParameter("error") != null) { %>
32      <p class="error-message">Identifiants invalides. Veuillez réessayer.</p>
33      <%= } %>
34  </div>
```

Figure 16 : Page de connexion
Src/main/webapp/Vue/login.jsp

B. Base de données

```

CREATE TABLE 'Personne' (
    'id_personne' VARCHAR(255) NOT NULL,
    'nom' VARCHAR(255) NOT NULL,
    'prenom' VARCHAR(255) NOT NULL,
    'date_naissance' DATE NULL DEFAULT NULL,
    'contact' VARCHAR(255) NULL DEFAULT NULL,
    'role' INT NULL DEFAULT NULL,
    'password' VARCHAR(255) NULL DEFAULT NULL,
    PRIMARY KEY ('id_personne')
);

CREATE TABLE 'Cours' (
    'id_cours' INT NOT NULL AUTO_INCREMENT,
    'nom_cours' VARCHAR(255) NOT NULL,
    'description' VARCHAR(255) NULL DEFAULT NULL,
    'date_debut' DATE NULL DEFAULT NULL,
    'date_fin' DATE NULL DEFAULT NULL,
    'id_enseignant' VARCHAR(255) NULL DEFAULT NULL,
    PRIMARY KEY ('id_cours'),
    INDEX 'id_enseignant' ('id_enseignant'),
    CONSTRAINT 'cours_ibfk_1'
        FOREIGN KEY ('id_enseignant')
            REFERENCES 'Personne' ('id_personne')
);

CREATE TABLE 'Inscription' (
    'id_inscription' INT NOT NULL AUTO_INCREMENT,
    'date_inscription' DATE NULL DEFAULT NULL,
    'etat' INT NULL DEFAULT NULL,
    'description_refus' VARCHAR(255) NULL DEFAULT NULL,
    'id_etudiant' VARCHAR(255) NULL DEFAULT NULL,
    'id_cours' INT NOT NULL,
    PRIMARY KEY ('id_inscription'),
    INDEX 'FK_cours' ('id_cours'),
    INDEX 'id_etudiant' ('id_etudiant'),
    CONSTRAINT 'inscription_ibfk_1'
        FOREIGN KEY ('id_etudiant')
            REFERENCES 'Personne' ('id_personne'),
    CONSTRAINT 'FK_cours'
        FOREIGN KEY ('id_cours')
            REFERENCES 'Cours' ('id_cours')
);

CREATE TABLE 'Resultat' (
    'id_resultat' INT NOT NULL AUTO_INCREMENT,
    'note' DOUBLE NULL DEFAULT NULL,
    'id_etudiant' VARCHAR(255) NULL DEFAULT NULL,
    'id_cours' INT NULL DEFAULT NULL,
    PRIMARY KEY ('id_resultat'),
    INDEX 'id_etudiant' ('id_etudiant'),
    INDEX 'id_cours' ('id_cours'),
    CONSTRAINT 'resultat_ibfk_1'
        FOREIGN KEY ('id_etudiant')
            REFERENCES 'Personne' ('id_personne'),
    CONSTRAINT 'resultat_ibfk_2'
        FOREIGN KEY ('id_cours')
            REFERENCES 'Cours' ('id_cours') ON DELETE CASCADE
);

```

Figure 17-18 : Table de base de données

C. Hibernate

Hibernate est un framework ORM (Object-Relational Mapping) qui facilite la communication entre les objets Java et les bases de données relationnelles. Dans le cadre de notre projet, il a été utilisé pour automatiser la gestion des entités et simplifier les opérations CRUD (Create, Read, Update, Delete).

Hibernate utilise une configuration ou des annotations pour mapper les classes Java aux tables de la base de données. Il permet de gérer les relations entre les entités (comme les relations One-to-Many ou Many-to-Many) tout en générant automatiquement les requêtes SQL nécessaires. De plus, grâce à ses capacités d'analyse, il garantit la cohérence des données et la gestion des transactions.

```

@ManyToOne 2 usages
@JoinColumn(name = "id_enseignant", referencedColumnName = "id_personne")
@Expose
private Personne personneByIdEnseignant;

@OneToMany(mappedBy = "cours", cascade = CascadeType.ALL, orphanRemoval = true) 2 usages
private List<Inscription> inscriptions;

```

Figure 19 : Exemple de variable hibernate

Src/main/entity/Cours.java

Nous avons configuré Hibernate à l'aide d'un fichier hibernate.cfg.xml où nous avons défini les détails de la connexion à la base de données, notamment l'URL, le nom d'utilisateur et le mot de passe. Une fois configuré, Hibernate a généré automatiquement les modèles (classes entités) à partir de la base de données grâce à sa fonctionnalité de reverse engineering. Cela nous a permis de gagner du temps et d'assurer une correspondance exacte entre la structure des tables et les objets manipulés dans notre application. Puis par la suite nous avons aussi modifié les classes générées par la "persistance" d'hibernate pour qu'il s'adapte parfaitement à notre

projet. Pour s'assurer d'avoir une base de données cohérente nous avons laissé hibernate régénérer les tables.

D. Spring Boot

Pour Spring Boot, la première étape a été de comprendre comment le framework fonctionne. L'utilisation est semblable à ce que nous avons fait auparavant, mais la configuration initiale est différente.

Pour le choix des technologies, nous avons décidé de conserver nos pages d'affichage au format JSP. Bien que nous aurions pu utiliser Thymeleaf, un moteur de templates HTML, nous avons réutilisé les pages JSP déjà existantes pour gagner du temps. Certaines pages ont cependant nécessité des adaptations.

Ensuite, nous avons configuré le projet en suivant les étapes fournies dans le document PDF disponible sur Teams. Nous avons rapidement utilisé la configuration rapide dans IntelliJ, puis ajusté les paramètres dans le fichier `application.properties`.

En ce qui concerne la base de données, la configuration avec Spring Boot s'est avérée beaucoup plus simple. Spring Boot prend en charge de nombreux aspects automatiquement, ce qui nous a permis de réutiliser les entités créées dans la première partie du projet. Nous avons créé un dossier **repository**, avec un fichier par entité, pour gérer les appels à la base de données. Ces fichiers repository ont simplifié nos contrôleurs par rapport à la version sans Spring Boot. Les méthodes de ces classes ont été ajoutées progressivement à mesure que nous développions les contrôleurs. La seule chose qu'il fallait respecter était la convention de nommage des fonctions ce qui a été compliqué à comprendre.



```
16 usages
@Repository
public interface CoursRepository extends JpaRepository<Cours, Integer> {

    2 usages
    List<Cours> findByPersonneByIdEnseignant(Personne enseignant);

    1 usage
    Cours findByNomCours(String nomCours);
}
```

Figure 20: Exemple de repository

`src/main/java/com/example/demo/repository/CoursRepository.java`

Les contrôleurs ont été créés en reprenant les servlets que nous avons développées dans la version sans framework, bien que des ajustements aient été nécessaires pour les adapter à Spring Boot. La logique derrière ces contrôleurs est restée essentiellement la même, c'est pourquoi nous ne nous attarderons pas sur ce point. Cependant, grâce au mécanisme de **mapping** de Spring Boot, certains contrôleurs qui géraient des pages similaires ont pu être regroupés en un seul, comme dans le cas du contrôleur affichant la liste des étudiants ou des professeurs.

Spring Boot a également permis de simplifier certaines logiques, notamment grâce à ses fonctionnalités intégrées. Par exemple, comme évoqué précédemment, les appels à la base de données ont été grandement simplifiés grâce à **Spring Data JPA**. Un autre exemple notable est l'envoi automatique d'e-mails. Nous avons utilisé le composant **JavaMailSender** fourni par Spring Boot, ce qui nous a permis de configurer facilement un serveur SMTP pour l'envoi de mails.

Par la suite, nous avons créé une classe de service dédiée à l'envoi des e-mails, ce qui a considérablement facilité l'intégration par rapport à une solution sans framework.

```
@Service
public class EmailService {

    @Autowired
    private JavaMailSender mailSender;

    3 usages
    public void sendEmail(String to, String subject, String text) {
        SimpleMailMessage message = new SimpleMailMessage();
        message.setTo(to);
        message.setSubject(subject);
        message.setText(text);
        mailSender.send(message);
    }
}
```

Figure 21: Exemple de repository

src/main/java/com/example/demo/Service/EmailService.java

En ce qui concerne les pages JSP, elles ont été développées en parallèle avec les servlets. La plupart du code est resté similaire, mais quelques ajustements ont été nécessaires pour intégrer correctement le Java dans les pages JSP. Dans l'approche sans Spring Boot, il suffisait d'insérer des blocs de code Java directement dans les JSP à l'aide de la syntaxe `<% %>`. Cependant, avec Spring Boot, nous avons utilisé les **tags JSTL** (JavaServer Pages Standard Tag Library), qui permettent d'insérer de la logique de manière plus propre et plus maintenable dans les pages JSP.

Nous avons rencontré quelques difficultés lors de l'adaptation de certaines pages complexes, en particulier pour l'affichage des listes d'étudiants et de professeurs, qui ont nécessité une révision de la logique et aussi l'implémentation d'une page de service pour avoir un appel à la base de données plus complexe que précédemment

```
2 usages
@Service
public class PersonneService {

    @Autowired
    private PersonneRepository personneRepository;

    1 usage
    public List<Personne> getPersonnesByFilter(int role, String id, String nom, String prenom, String contact) {
        // Les paramètres null ou vides doivent être remplacés par une chaîne vide
        return personneRepository.findByRoleAndIdPersonneContainingAndNomContainingAndPrenomContainingAndContactContaining(
            role,
            id != null ? id : "",
            nom != null ? nom : "",
            prenom != null ? prenom : "",
            contact != null ? contact : ""
        );
    }
}
```

Figure 22 : Exemple de service

src/main/java/com/example/demo/Service/PersonneService.java

En conclusion, Spring Boot s'est avéré relativement simple à prendre en main, notamment pour ceux ayant déjà une expérience en JEE. L'application a montré des améliorations notables en termes de performances et de gestion des ressources, notamment lors des appels à la base de données.

III. Travail en équipe

1. Organisation

Nous avons majoritairement travaillé en distanciel, on a désigné un chef de projet pour faciliter la création de tâches, chaque membre du groupe gérant le rythme de ses contributions comme il le souhaitait.

Pour faciliter la communication sur le projet et le partage des fichiers, nous avons travaillé sur un serveur Discord et avons pris avantage du système de salons textuels et vocaux pour mieux nous organiser.

En parallèle, nous avons déposé les avancées du code régulièrement sur un dépôt GitHub, comme demandé dans les consignes. Nous avons mis en place une règle qui consiste à prévenir les autres quand l'on fait un commit important sur la branche principale (plusieurs fichiers affectés, dépendances modifiées, etc.), autrement chacun était libre de déposer son travail sur sa branche individuelle pour rendre accessible son travail.

V. Bibliographie

Voici quelques-unes des ressources qui nous ont aidé à réaliser ce projet :

- MVC : <https://fr.wikipedia.org/wiki/Mod%C3%A8le-vue-contr%C3%B4leur>
- Stack Overflow : <https://stackoverflow.com>
- Maven : <https://maven.apache.org/download.cgi>
- Spring Boot : <https://www.geeksforgeeks.org/spring-boot/>
- Mail : <https://sendgrid.com/>