PAIMBA-SAIL Owen

DIFFO Glenn-Airton

SADIQI Mohamed ElGhali

MILLET Christelle

Groupe : *GSI 3*

Rapport de projet Gestion RH :

Professeur référent : *Haddache Mohamed*

Année scolaire : *2025 - 2026*

Matière : *JEE*

CY TECH

Sommaire :

Table des matières

1. Introduction :

Notre projet de gestion RH s'inscrit dans le cadre de notre module de JEE.
Il a pour objectif de fournir une application Web permettant aux entreprises de gérer efficacement leurs employés, leurs départements, leurs projets ainsi que leurs fiches de paie. Conçu comme un outil professionnel et évolutif, notre application JEE met à disposition une interface claire et intuitive permettant de suivre l'ensemble des processus RH facilement.

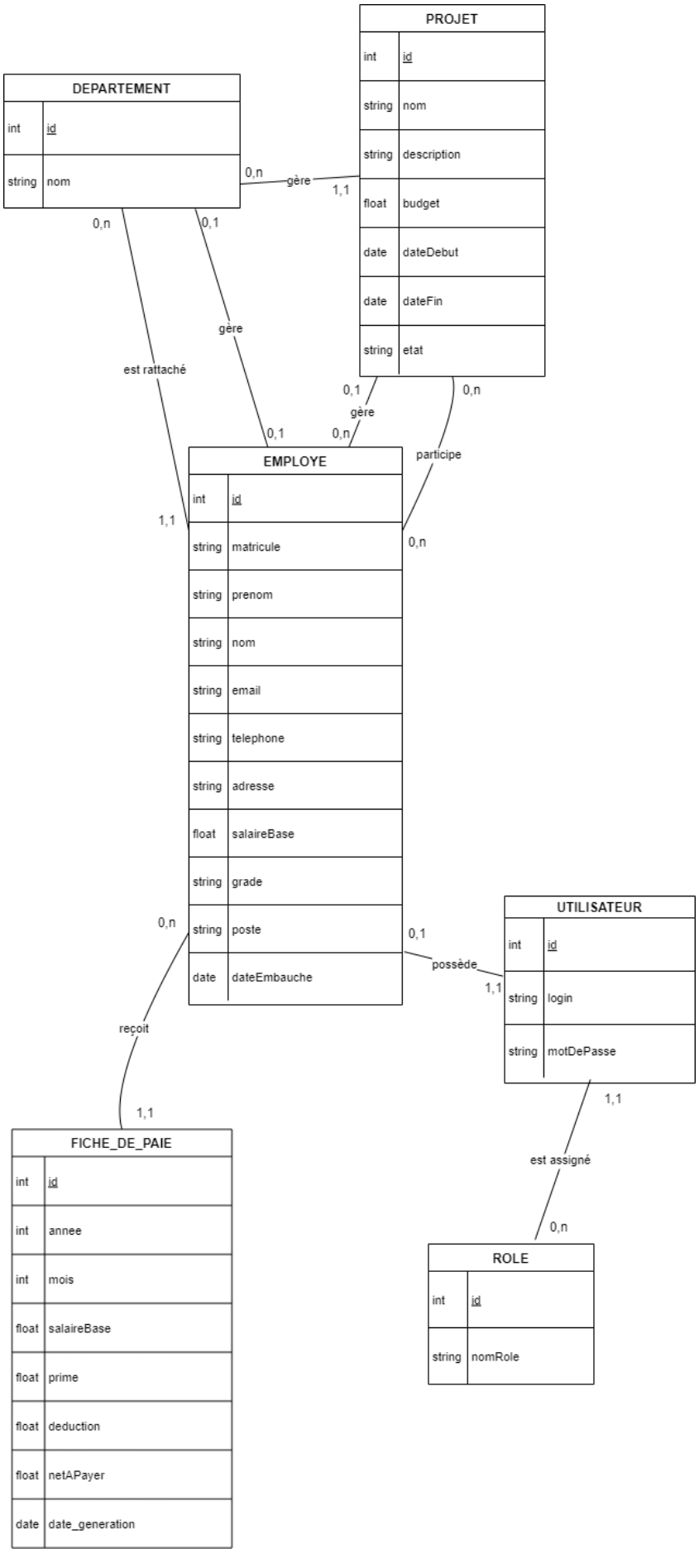
Notre application repose sur une organisation structurée autour de plusieurs profils utilisateurs : l'administrateur (le RH), le(s) chef(s) de département, le(s) chef(s) de projet(s) et les employés. Chaque rôle possède des droits spécifiques lui permettant d'accéder ou non à différentes fonctionnalités, que ce soit la gestion des employés, la supervision des projets, ou encore la création de fiches de paie. L'authentification est centralisée via un système de comptes utilisateurs liés aux employés, avec gestion des rôles et sécurisation des accès.

Dans la première version, notre projet est développé en Java/Jakarta EE, en s'appuyant sur une architecture MVC : Servlets pour le contrôle, JSP pour la vue, et Hibernate/JPA pour la persistance des données.

La seconde version repose sur le Framework Spring Boot. Dans cette version, nous avons légèrement modifié l'application JEE afin de respecter les normes SpringBoot.Cela nous donne une configuration simplifiée, des contrôleurs plus structurés, un moteur de templating (Thymeleaf) mieux intégré et une gestion de la persistance largement facilitée par Spring Data JPA.

Grâce à l'ensemble de ces fonctionnalités, notre projet de gestion RH constitue un outil fiable et accessible pour superviser les ressources humaines d'une entreprise, qu'il s'agisse de gérer les employés, d'administrer les projets en cours, ou de générer des fiches de paie de manière automatisée et conforme.

2. MCD :


PROJET
int id
string nom
string description
float budget
date dateDebut
date dateFin
string etat
DEPARTEMENT
int id
string nom
0,n
gère
1,1
0,n
0,1
gère
est rattaché
0,1
0,n
gère
0,n
participe
EMPLOYE
int id
1,1
string matricule
0,n
string prenom
string nom
string email
string telephone
string adresse
float salaireBase
string grade
0,n
string poste
0,1
date dateEmbauche
possède
1,1
UTILISATEUR
int id
string login
string motDePasse
reçoit
1,1
1,1
est assigné
0,n
FICHE_DE_PAIE
int id
int annee
int mois
float salaireBase
float prime
float deduction
float netAPayer
date date_generation
ROLE
int id
string nomRole

Notre modèle assure une cohérence globale entre les informations relatives aux employés, aux projets, aux départements, aux utilisateurs et aux fiches de paie.

L'entité Employé occupe une place centrale dans le MCD. Elle contient l'ensemble des informations personnelles et professionnelles (nom, prénom, matricule, email, poste, grade, salaire, ...). Chaque employé est rattaché à un département (relation 0,n vers 1,1), ce qui reflète la structure hiérarchique classique d'une entreprise. Un employé peut également participer à plusieurs projets, tandis qu'un projet peut inclure plusieurs employés (relation 0,n ⇔ 0,n), matérialisant une collaboration flexible entre équipes.

L'entité Projet regroupe toutes les informations liées aux projets en cours : nom, description, budget, dates de début et de fin, ainsi que l'état d'avancement. Chaque projet est géré par un chef de projet, qui est lui-même un employé (relation 0,1 vers 1,1).

L'entité Fiche_de_paie est associée à un employé selon une relation 0,n vers 1,1. Elle contient les éléments essentiels de la rémunération : salaire de base, primes, déductions et net à payer. Chaque fiche correspond à un mois donné et à une année précise, garantissant un historique clair et structuré.

L'entité Utilisateur permet la gestion des accès à la plateforme. Chaque employé possède au maximum un compte utilisateur (relation 0,1 vers 1,1), lequel contient le login et le mot de passe. Un utilisateur est associé à un rôle, défini dans l'entité Role, permettant ainsi de déterminer ses droits d'accès dans l'application : administrateur, chef de département, chef de projet ou simple employé.

Ce MCD assure une vision claire de l'organisation et des interactions du système. Il garantit également la cohérence de la base de données et facilite la mise en œuvre des fonctionnalités de l'application, telles que la gestion des employés, l'administration des projets ou encore la génération des fiches de paie.

a) Organisation :

Afin d'assurer une bonne coordination tout au long du développement, notre groupe a mis en place une organisation simple mais efficace. Nous organisions régulièrement des réunions d'équipe afin de présenter les fonctionnalités et nos avancées. En complément, nous échangions quotidiennement via Whatsapp, ce qui nous permettait de partager rapidement les modifications effectuées et de suivre l'évolution du projet en temps réel.

Les différentes tâches ont été réparties entre les membres du groupe de manière équilibrée, en fonction des compétences et des préférences de chacun. Voici la répartition :

Christelle :

- Gestion de la base de données et construction du MCD
- Création des entités JPA (Employé, Projet, Département, Fiche de paie, Utilisateur, Rôle)

- Mise en place des relations entre les tables (OneToMany, ManyToMany, etc.)
- Aide à la partie Hibernate / DAO
- Participation à la rédaction du rapport

Glenn :

- Développement de la gestion des employés (CRUD complet)
- Implémentation des pages JSP côté administrateur
- Connexion avec Hibernate pour la récupération des données
- Mise en place des pages liées aux projets (ajout, affectation employés, état)
- Support sur la gestion des rôles pendant la connexion

Ghali :

- Mise en place du système de connexion et d'authentification (JEE puis Spring Boot)
- Création du système d'inscription via l'administrateur
- Génération automatique du login et du mot de passe temporaire
- Système d'envoi d'email (identifiants de connexion)
- Intégration Spring Boot : contrôleurs, routes, sécurisation des pages

Owen :

- Réalisation de la navigation (navbar, redirections selon le rôle)
- Page d'accueil (index.jsp) personnalisée selon l'utilisateur connecté
- Gestion de la partie Fiches de paie (création, calculs, affichage)
- Migration des vues JSP → Thymeleaf dans la version Spring Boot
- Structure du rapport + relecture et mise en forme finale

3. Code architecture MVC :

Notre application de gestion RH repose sur une architecture MVC (Modèle Vue Contrôleur).
Cette organisation permet de séparer clairement :

- Les données métier (employés, départements, projets, fiches de paie, utilisateurs, rôles) ;
- La logique de traitement (contrôle des actions, règles applicatives, redirections) ;
- Et la présentation (pages JSP affichées à l'utilisateur).

Cette séparation rend le code plus lisible et plus facile à faire évoluer.

a) Couche Modèle :

La couche Modèle est composée :

- Des entités JPA : Employe, Departement, Projet, FicheDePaie, Utilisateur, Role
- Et des DAO (Data Access Objects) qui encapsulent l'accès à la base via Hibernate : EmployeDAO, UtilisateurDAO, etc.

Les DAO s'occupent de toutes les opérations CRUD (Create, Read, Update, Delete) ainsi que des recherches un peu plus complexes (par login, par département, par mot-clé...).

Extrait 1 : Vérification des identifiants (UtilisateurDAO) :

```
public Utilisateur getByLogin(String login, String mdp) {  1 usage
    try (Session session = HibernateUtil.getSessionFactory().openSession()) {
        return session.createQuery( s: "from Utilisateur where login = :login and motDePasse = :mdp", Utilisateur.class)
                .setParameter( s: "login", login)
                .setParameter( s: "mdp", mdp)
                .uniqueResult();
    }
}
```

Cette méthode de la classe UtilisateurDAO interroge la base de données pour récupérer un utilisateur à partir du couple (login, motDePasse).
Elle utilise Hibernate pour exécuter la requête.
C'est cette partie de code qui est appelée lors de la connexion : si elle renvoie null, les identifiants sont invalides, sinon l'utilisateur est authentifié et stocké en session.

b) Couche Contôleur (Servlets) :

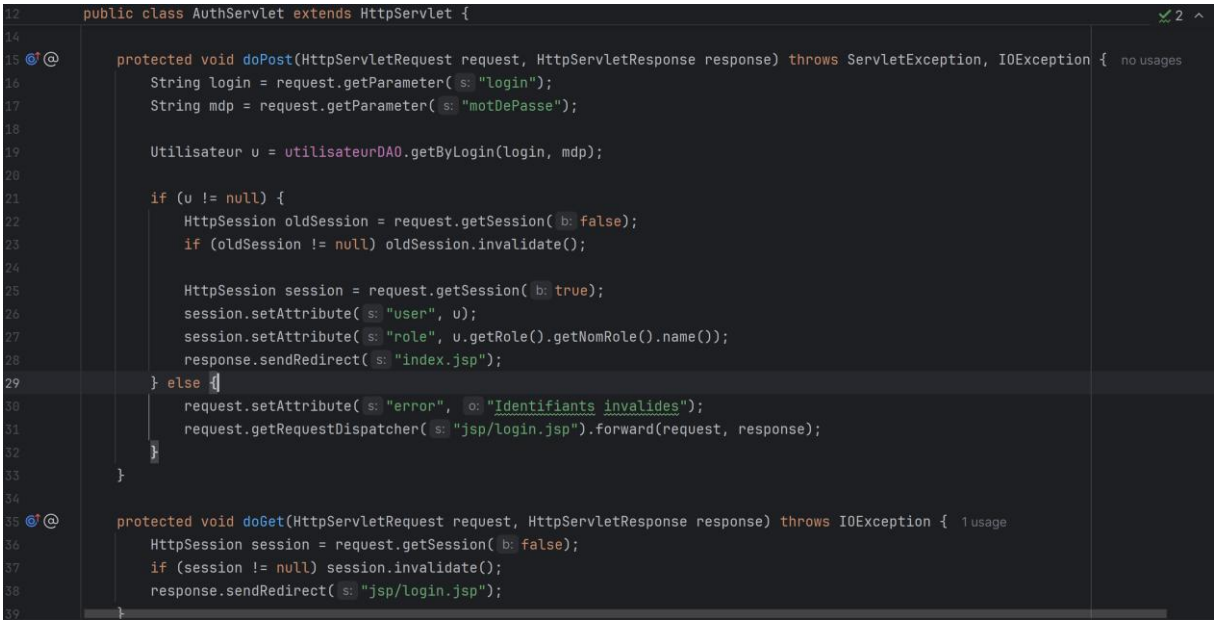
La couche Contrôleur est implémentée à l'aide de Servlets Jakarta EE.
Chaque servlet correspond à une partie fonctionnelle du site :

- LoginServlet pour la connexion,
- EmployeServlet pour la gestion des employés,
- DepartementServlet pour les départements,

- ProjetServlet pour les projets,
- FichePaieServlet pour les fiches de paie,
- …

Les servlets reçoivent les requêtes HTTP (GET, POST…), appellent les DAO, appliquent les règles de gestion (droits d'accès selon le rôle) puis redirigent vers la JSP appropriée.

Extrait 2 : Servlet d'authentification :



```java
public class AuthServlet extends HttpServlet {

    protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException {
        String login = request.getParameter( s: "login");
        String mdp = request.getParameter( s: "motDePasse");

        Utilisateur u = utilisateurDAO.getByLogin(login, mdp);

        if (u != null) {
            HttpSession oldSession = request.getSession( b: false);
            if (oldSession != null) oldSession.invalidate();

            HttpSession session = request.getSession( b: true);
            session.setAttribute( s: "user", u);
            session.setAttribute( s: "role", u.getRole().getNomRole().name());
            response.sendRedirect( s: "index.jsp");
        } else {
            request.setAttribute( s: "error",  o: "Identifiants invalides");
            request.getRequestDispatcher( s: "jsp/login.jsp").forward(request, response);
        }
    }

    protected void doGet(HttpServletRequest request, HttpServletResponse response) throws IOException {
        HttpSession session = request.getSession( b: false);
        if (session != null) session.invalidate();
        response.sendRedirect( s: "jsp/login.jsp");
    }
```

Cette servlet « AuthServlet » illustre le rôle de la couche Contrôleur dans notre architecture MVC :

- Elle récupère les paramètres login et motDePasse envoyés par le formulaire de connexion.
- Elle appelle la couche Modèle via UtilisateurDAO.getByLogin(login, mdp) pour vérifier si le couple identifiant/mot de passe existe en base.
- En cas de succès, elle :
    - Invalide une éventuelle ancienne session ;
    - Crée une nouvelle session et y stocke l'objet Utilisateur ainsi que son rôle (role) ;
    - Redirige l'utilisateur vers la page d'accueil index.jsp.
- En cas d'échec, elle renvoie vers login.jsp en ajoutant un message d'erreur (error) qui sera affiché par la JSP.

c) Couche Vue :

La couche Vue repose sur des pages JSP enrichies avec JSTL (<c:forEach>, <c:if>, ...) et du HTML/CSS (Bootstrap).
Les JSP ne contiennent pas de logique métier : elles se contentent d'afficher les données fournies par les servlets via les attributs de requête ou de session

Extrait 3 – Formulaire de connexion (login.jsp) :

```jsp
<form action="${pageContext.request.contextPath}/auth" method="post">
    <div class="mb-3">
        <label class="form-label fw-semibold">Nom d'utilisateur</label>
        <input type="text" name="login" class="form-control" placeholder="Entrez votre identifiant" required>
    </div>

    <div class="mb-3">
        <label class="form-label fw-semibold">Mot de passe</label>
        <input type="password" name="motDePasse" class="form-control" placeholder="••••••••" required>
    </div>

    <c:if test="${not empty error}">
        <div class="alert alert-danger text-center py-2">${error}</div>
    </c:if>

    <div class="text-center mt-4">
        <button type="submit" class="btn btn-primary w-100">Se connecter</button>
    </div>
</form>
```

Cette vue illustre le rôle des JSP dans notre architecture :

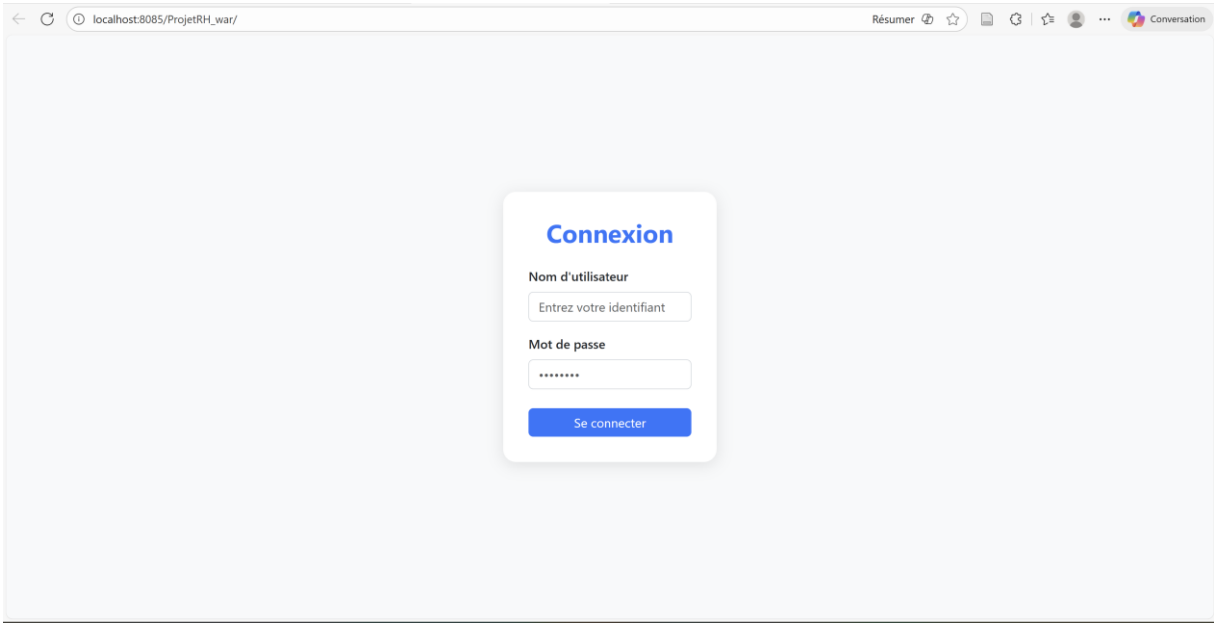
- Elle affiche le formulaire de connexion (HTML + Bootstrap).
- Elle utilise JSTL (<c:if>) pour afficher un message d'erreur si l'attribut error a été placé par la servlet.
- Elle ne fait aucune requête SQL et ne contient aucune règle métier : toute la logique est déléguée au contrôleur.

En résumé, l'architecture MVC de notre projet JEE permet une séparation claire entre la gestion des données, la logique et l'affichage. Cette structuration garantit une application plus stable et facile à tester, tout en offrant une base solide pour la transition vers Spring Boot qui reprendra ces mêmes principes.

4. Présentation du site (partie JEE) :

a) Connexion et Inscription :

Dès le lancement de l'application, l'utilisateur est dirigé vers une interface de connexion



Cette page constitue la porte d'entrée du système. Deux champs sont demandés : le nom d'utilisateur (qui est en fait votre identifiant) et un mot de passe.

```
protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException,
    String login = request.getParameter( s: "login");
    String mdp = request.getParameter( s: "motDePasse");
```

Cette identifiant est unique, généré automatiquement par le système lorsque l'administrateur (le RH) vous inscrit sur le site, ce mot de passe et cet identifiant sont, bien-sûr, modifiable ensuite sur le site dans notre page profil.

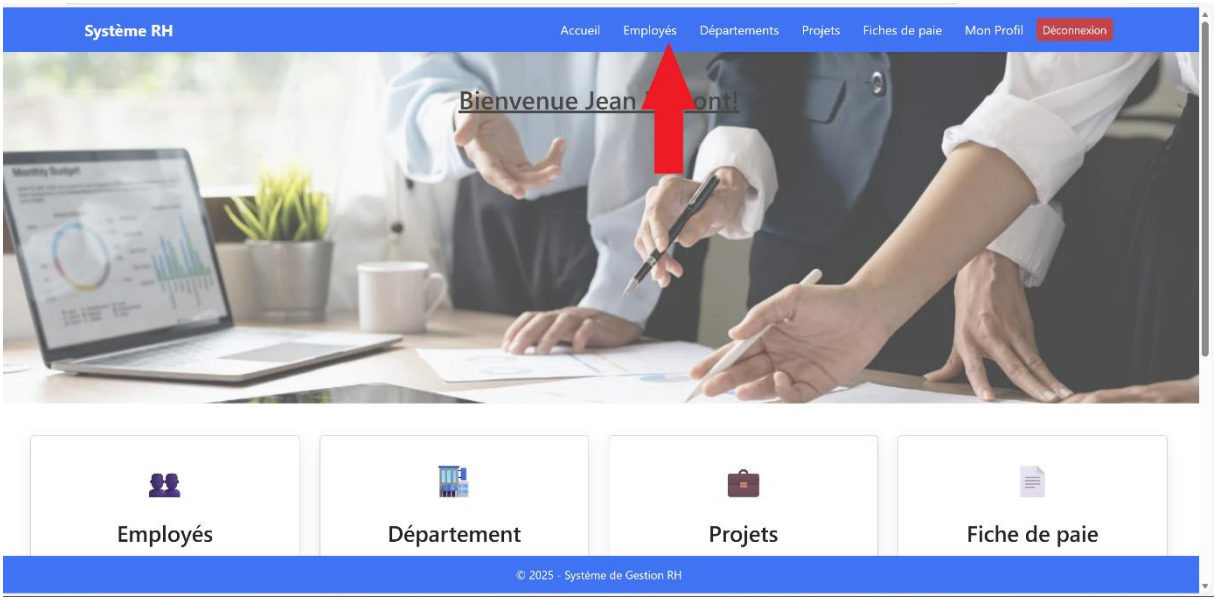
En cas de mot de passe incorrect ou d'identifiant inexistant, l'accès est bloqué.

Ainsi, si l'utilisateur n'a pas de compte, il se doit de demander au RH de l'inscrire sur le site. Grâce à ça, seulement les employés de l'entreprise peuvent avoir accès à notre application.

Pour vérifier les identifiants lors de la connexion, nous utilisons un DAO (UtilisateurDAO) qui interroge la base de données via Hibernate. La méthode getByLogin récupère l'utilisateur correspondant au couple (login, mot de passe).

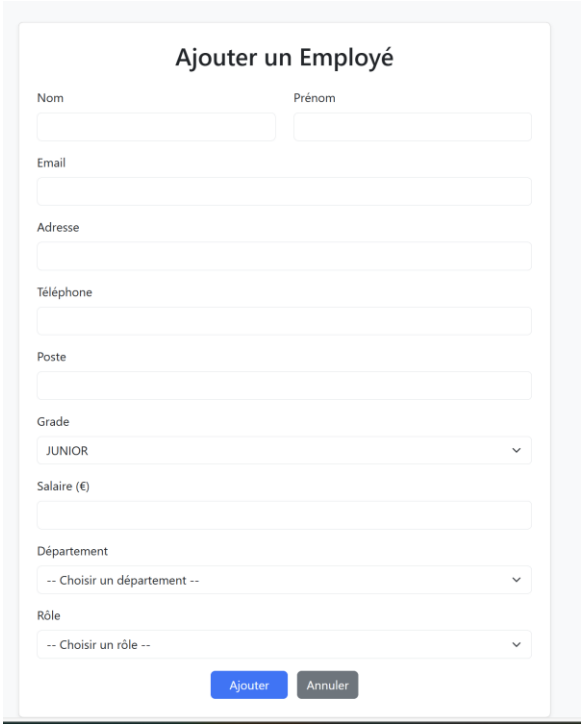
```
public Utilisateur getByLogin(String login, String mdp) {  1 usage
    try (Session session = HibernateUtil.getSessionFactory().openSession()) {
        return session.createQuery( s: "from Utilisateur where login = :login and motDePasse = :mdp", Utilisateur.class)
                .setParameter( s: "login", login)
                .setParameter( s: "mdp", mdp)
                .uniqueResult();
    }
}
```

Pour ajouter un utilisateur, l'administrateur doit se rendre dans la section dédiée à la gestion des employés :



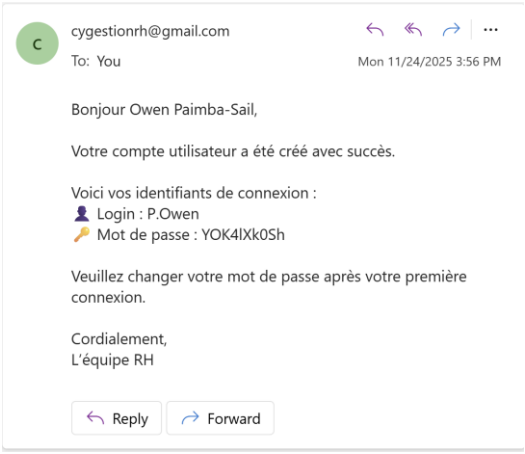
Depuis cette interface, il peut créer un nouvel employé en renseignant les informations personnelles essentielles (nom, prénom, poste, département, etc.). À la validation du formulaire, le système génère automatiquement :

- Un identifiant de connexion unique,
- Un mot de passe (temporaire),
- Ainsi que le rôle attribué à l'utilisateur (Employé, Chef de département, Chef de projet, Admin).



Une fois le compte créé, ces informations sont enregistrées dans la base via Hibernate, et l'utilisateur est désormais autorisé à se connecter.

Un message est ensuite envoyé sur l'adresse mail renseigné contenant l'identifiant ainsi que le mot de passe de l'utilisateur.



Ce fonctionnement garantit une gestion centralisée et contrôlée des accès, permettant à l'entreprise de maîtriser entièrement les utilisateurs autorisés à consulter ou modifier les données du système.

b) Gestion rôle :

L'application inclut un système de gestion des utilisateurs par un administrateur, garantissant la sécurité et la validité des informations enregistrées sur le site.

Lors de l'inscription faite par l'admin, un utilisateur se voit attribuer soit le rôle « Employé », « Chef de département », « Chef de projet », ou « Administrateur », ce qui lui donne accès à certaines fonctionnalités ou non (cf. chapitre suivant). Ce champ « nomRole » est donc utilisé à la connexion pour rediriger automatiquement l'utilisateur vers son espace/interface correspondant.

c) Les différentes fonctionnalités :

Comme dit au-dessus, notre site repose sur un système de rôles hiérarchisés, chacun disposant d'un ensemble de fonctionnalités adaptées à ses responsabilités.
Le rôle ayant le plus haut niveau d'accès est celui d'Administrateur (RH). Celui-ci possède une

vision complète du système et peut gérer l'ensemble des données : employés, départements, projets, fiches de paie et comptes utilisateurs.

1. Fonctionnalités Administrateur (RH) :

Il dispose de tous les droits, sans restriction.

Gestion des employés :

L'admin peut créer un nouvel employé, mettre à jour ses informations ou le supprimer si nécessaire. Il a également la possibilité d'effectuer des recherches ciblées à partir du nom, du prénom, du matricule ou du département. Depuis la fiche d'un employé, l'administrateur peut consulter les projets auxquels il participe ainsi que l'ensemble de ses fiches de paie.

Gestion des départements :

L'administrateur peut également gérer entièrement les départements. Il a la possibilité d'en créer de nouveaux, d'en modifier le nom ou le chef, ou encore de les supprimer. Depuis chaque département, il peut aussi consulter l'ensemble des employés qui y sont rattachés.

Gestion des projets :

De plus, il dispose d'un contrôle complet sur les projets. Il peut en créer de nouveaux, les modifier ou les supprimer selon les besoins de l'entreprise. Il est aussi en mesure de désigner un chef de projet, d'ajouter ou de retirer des employés impliqués, et de consulter à tout moment la liste complète des projets en cours.

Gestion des fiches de paie :

Enfin, le RH possède un accès complet à la gestion des fiches de paie. Il peut en créer une nouvelle pour n'importe quel employé, modifier une fiche déjà enregistrée, consulter l'ensemble des fiches disponibles dans le système ou encore en supprimer lorsqu'elles ne sont plus nécessaires.

2. Fonctionnalités des autres rôles :

Après l'administrateur, les autres profils ont accès à une version limitée de l'application. L'objectif est de respecter la hiérarchie et les responsabilités internes à l'entreprise :

Chef de Département

Le chef de département possède des droits intermédiaires :

```
<c:if test="${role == 'ADMINISTRATEUR' || role == 'CHEF_DE_DEPARTEMENT'}">
```

Ses droits sont centrés uniquement sur la gestion de son propre service. Il peut ainsi consulter les informations de son département et les modifier si nécessaire. Il a également la possibilité d'ajouter un nouvel employé au sein de son équipe, mais aussi de modifier ou supprimer ceux déjà présents dans son département. Enfin, il peut visualiser l'ensemble des projets auxquels participent les employés qui lui sont rattachés, ce qui lui permet de suivre l'activité de son service sans pour autant accéder aux données des autres départements.

En revanche, comme dit au-dessus, son champ d'action reste limité à son propre périmètre. Le chef de département n'a pas la possibilité d'accéder aux autres services de l'entreprise ni d'intervenir sur la gestion globale des projets. Il ne peut pas non plus administrer les utilisateurs ou consulter toutes les fiches de paie du système, étant restreint à celles concernant uniquement les employés de son département

Chef de Projet :

Le chef de projet n'a accès qu'aux fonctionnalités de son propre projet.

```
- Chef de projet : peut modifier seulement ses projets -->
if test="${role == 'CHEF_DE_PROJET' && projet.chefProjet != null && meId == projet.chefProjet.id}
```

Ainsi, de la même manière que le chef de département, le chef de projet peut consulter la liste de ses propres projets, en modifier les informations, ajouter ou retirer des employés et mettre à jour l'état d'avancement. En revanche, son rôle reste circonscrit à ce périmètre : il ne peut en aucun cas gérer des employés extérieurs à ses projets, ni accéder ou modifier les départements. Il n'a pas non plus la possibilité de créer de nouveaux projets, ni d'intervenir sur la gestion des fiches de paie.

Employé :

L'employé est le rôle le plus restreint du système, puisqu'il ne dispose que d'un accès consultatif. Il peut consulter sa propre fiche personnelle, visualiser les projets auxquels il participe et accéder à ses fiches de paie. En revanche, il ne peut modifier aucune information et n'a accès à aucune fonctionnalité de gestion : il ne peut ni administrer les employés, ni intervenir sur les projets ou les départements, et ne peut évidemment pas accéder aux pages réservées à l'administration.

Grâce à ce système de rôles hiérarchisés et contrôlés, l'application garantit une clarté organisationnelle, une sécurité renforcée, une gestion adaptée à chaque profil, et une lisibilité des responsabilités selon la structure d'une entreprise réelle.

5. Présentation du site (partie SpringBoot) :

Dans une seconde étape, nous avons migré notre application de gestion RH vers le framework Spring Boot.
L'objectif n'était pas de réécrire entièrement le projet, mais de reprendre la même logique métier (employés, projets, fiches de paie, rôles…) dans un nouvel environnement

L'architecture reste MVC :

- Modèle : mêmes entités (Employe, Departement, Projet, FicheDePaie, Utilisateur, Role), mais cette fois gérées par Spring Data JPA.
- Contrôleur : les anciennes Servlets sont remplacées par des classes annotées @Controller.
- Vue : les pages JSP sont migrées vers des templates Thymeleaf (.html).

Spring Boot nous évite aussi beaucoup de configuration : plus de web.xml, plus de HibernateUtil, le serveur Tomcat est intégré, et la connexion à la base se fait simplement via le fichier application.properties.

a) Configuration :

Le projet Spring Boot est créé à partir de Spring Initializer avec les dépendances principales :

- spring-boot-starter-web (Spring MVC + Tomcat intégré),
- spring-boot-starter-thymeleaf (moteur de template),
- spring-boot-starter-data-jpa (JPA + Hibernate),
- mysql-connector-j (connexion MySQL).

La classe principale lance l'application :

Extrait Spring 1 – Classe Application (Spring Boot) :

Extrait Spring 2 – Configuration application.properties :

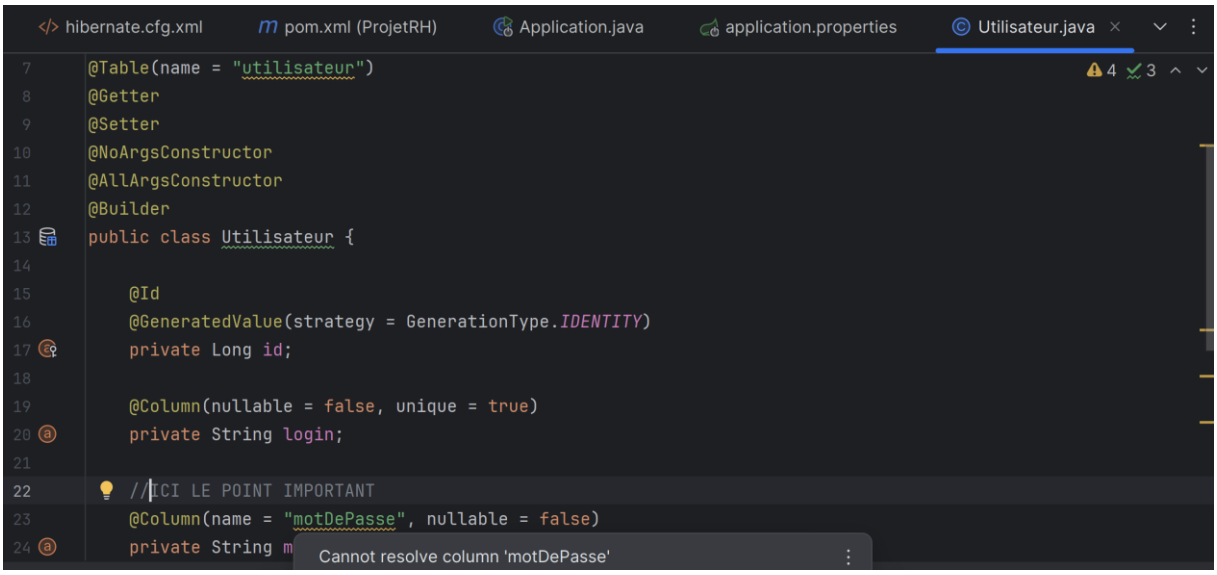
```
# =========  CONFIG BDD =========
spring.datasource.url=jdbc:mysql://localhost:3306/gestion_rh?useSSL=false&serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

# On NE TOUCHE PAS au schéma
spring.jpa.hibernate.ddl-auto=none
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect
```

Nous avons choisi d'utiliser la configuration spring.jpa.hibernate.ddl-auto=none, car la base de données était déjà entièrement créée dans la version JEE du projet. Ainsi, Spring Boot n'effectue aucune modification automatique du schéma

b) MVC Spring :

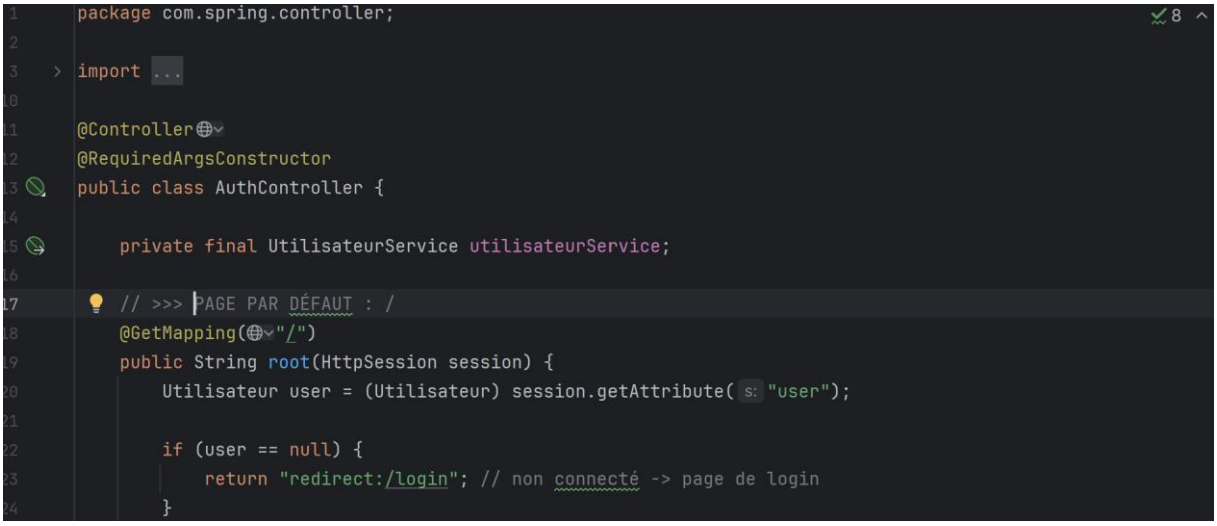
Les entités restent pratiquement identiques à celles de la version JEE : mêmes tables, mêmes champs, mêmes relations. La principale différence est que, dans Spring Boot, elles sont gérées directement par Spring Data JPA au travers d'interfaces Repository.
On n'écrit plus de DAO « à la main » comme UtilisateurDAO : Spring génère pour nous les opérations CRUD.

Extrait Spring 3 – Entité Utilisateur (Spring Boot) :



Ensuite, pour la partie controller, dans la version JEE, la connexion était assurée par la servlet AuthServlet.
En Spring, nous avons créé un contrôleur MVC AuthController qui joue exactement le même rôle, mais en s'appuyant sur les annotations Spring.

Extrait Spring 4 – Contrôleur d'authentification :



On retrouve exactement la même logique qu'avec AuthServlet :

- récupération des paramètres login et motDePasse
- appel à la couche modèle pour vérifier les identifiants,
- gestion de la session (user, role),
- redirection vers la page d'accueil ou retour sur l'écran de connexion avec un message d'erreur.

Enfin, pour le Vue, les pages de la version JEE (par exemple login.jsp ou index.jsp) ont été migrées vers des pages Thymeleaf (login.html, index.html) dans src/main/resources/templates. Le contenu graphique reste quasiment identique (HTML + Bootstrap), seule la syntaxe des balises change.

Extrait Spring 5 – login.html (Thymeleaf) :

```
<!DOCTYPE html>
<html lang="fr" xmlns:th="http://www.thymeleaf.org">
<head>
    <meta charset="UTF-8">
    <title>Connexion - Système RH</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">

    <style>
        body {
            background-color: #f8f9fa;
        }
        .login-card {
            max-width: 420px;
            margin: auto;
            padding: 2rem;
            border-radius: 1rem;
            background-color: #fff;
            box-shadow: 0 4px 20px rgba(0,0,0,0.1);
```

c) Bilan Spring :

La migration vers Spring Boot nous a permis de conserver toute la logique métier développée en JEE (MCD, entités, rôles, fonctionnalités), tout en profitant :

- D'une configuration simplifiée (plus de web.xml, gestion de la base centralisée dans application.properties) ;
- D'un accès aux données plus concis grâce à Spring Data JPA (DAO remplacés par des Repository) ;
- De contrôleurs plus lisibles avec les annotations @Controller, @GetMapping, @PostMapping ;
- D'un moteur de templates moderne, Thymeleaf, qui s'intègre naturellement à Spring.

En résumé, la version Spring Boot de notre application de gestion RH reprend la même architecture MVC que la version Java/Jakarta EE, mais avec une grammaire Spring plus simple et une quantité de code technique réduite. Cela rend l'application plus facile à maintenir et ouvre la voie à des évolutions futures (Spring Security, API REST, tests unitaires, ...).

6. Conclusion :

Ce projet de gestion RH nous a permis d'aborder de manière concrète le développement d'une application Web complète, d'abord avec l'approche Jakarta/JEE, puis à travers une migration vers Spring Boot. Travailler avec ces deux environnements nous a aidés à mieux comprendre les bases du modèle MVC, la gestion des données avec Hibernate/JPA et l'importance d'une architecture claire pour maintenir une application.

La réalisation du projet nous a également poussés à nous organiser efficacement en équipe. Chacun a pu contribuer selon ses compétences, partager ses avancées et résoudre les difficultés au fur et à mesure. Cette dynamique nous a permis d'obtenir une application cohérente, et qui répond à l'énoncé de départ.

La transition vers Spring Boot a représenté une étape importante : elle nous a montré comment moderniser et modifier un projet existant. Cette expérience nous a donné une meilleure vision des pratiques dont nous serons sujets dans nos vie professionnelle.

En résumé, ce projet nous a fait progresser autant sur le plan technique que sur le travail en groupe. Il nous a permis de manipuler des notions essentielles du développement Java tout en construisant une application structurée et concrète.