CY TECH - 2025-2026- Encadré par M. HADDACHE Mohamed

Rapport Projet JEE

Réalisation d'une Application Ressources Humaines

Orianne COURTADE,
Tom CONTI,
Mathieu ROUET,
Jonathan NGO,
Fantine BODIER

GSI Groupe 1
30/11/2025

Sommaire

Introduction

Dans un contexte où la digitalisation des processus internes devient un enjeu majeur pour les entreprises, la mise en place d'outils de gestion centralisés constitue un levier essentiel d'efficacité, de fiabilité et de modernisation.

Le projet présenté dans ce rapport s'inscrit dans cette dynamique : il vise à concevoir et développer une application web complète destinée à la gestion des employés, des départements, des projets et des fiches de paie au sein d'une organisation.

L'entreprise fictive qui sert de cadre à ce projet exprime un besoin clair : disposer d'un système unique, ergonomique et sécurisé permettant d'automatiser et d'optimiser les tâches courantes liées aux ressources humaines et au suivi opérationnel. Les processus actuellement dispersés — gestion des informations personnelles, affectations, statistiques, suivi des projets ou encore génération des fiches de paie — doivent être regroupés dans une plateforme cohérente et accessible aux utilisateurs autorisés.

La problématique centrale est donc la suivante : Comment concevoir une application robuste, modulaire et évolutive capable de couvrir l'ensemble de ces besoins tout en respectant les principes de l'architecture logicielle moderne ?

Pour y répondre, le projet repose sur une première implémentation en JEE classique, basée sur les technologies Servlets, JSP, JDBC, DAO et Hibernate. Cette architecture en couches permet de structurer clairement l'application selon le modèle MVC.

Dans un second temps, une évolution vers Spring Boot est proposée afin de moderniser l'approche, faciliter l'exposition d'une API REST et améliorer la maintenabilité tout en conservant les fonctionnalités de base.

L'application développée couvre plusieurs grands ensembles fonctionnels :

- Gestion des employés et de leurs informations professionnelles
- Administration des départements et des projets
- Génération automatique et consultation des fiches de paie
- Authentification et gestion des rôles
- Affichage de statistiques et rapports.

Enfin, ce rapport est structuré de manière progressive : il débute par l'analyse du besoin et la conception du système (cas d'usage, diagrammes UML), se poursuit par la présentation de l'architecture JEE et de son implémentation, puis décrit la migration vers Spring Boot et l'API REST associée. Il se conclut par un retour sur les choix techniques, les difficultés rencontrées, les outils utilisés et les perspectives d'amélioration.

Analyse et Conception

Analyse du besoin

1. Description fonctionnelle globale

L'application a pour objectif de fournir un outil de gestion centralisé permettant d'administrer les principales entités d'une entreprise : les employés, les départements, les projets et les fiches de paie. Elle doit offrir une interface simple et intuitive permettant aux utilisateurs autorisés d'effectuer l'ensemble des opérations nécessaires au suivi et à l'organisation interne de la structure.

2. Cas d'usage principaux

Les cas d'usage décrivent les interactions essentielles entre les utilisateurs et le système. Les principaux cas d'usage identifiés sont les suivants :

- Se connecter à l'application
- Gérer les employés
- Gérer les départements
- Gérer les projets
- Gérer les fiches de paie
- Gérer son profil
- Consulter les statistiques

Ces cas d'usage couvrent l'intégralité des opérations réalisables au sein de l'application et décrivent les interactions principales entre le système et ses utilisateurs.

3. Acteurs

Les acteurs représentent les différents types d'utilisateurs ou systèmes externes qui interagissent avec l'application.

Le premier acteur est l'utilisateur authentifié qu'il est le rôle « Employé » ou « Administrateur ». Il s'agit de la personne qui se connecte à l'application pour accéder aux fonctionnalités de gestion. Selon les droits définis, cet acteur peut :

- Gérer les employés, les départements, les projets et les fiches de paie
- Consulter et modifier les données
- Mettre à jour son profil

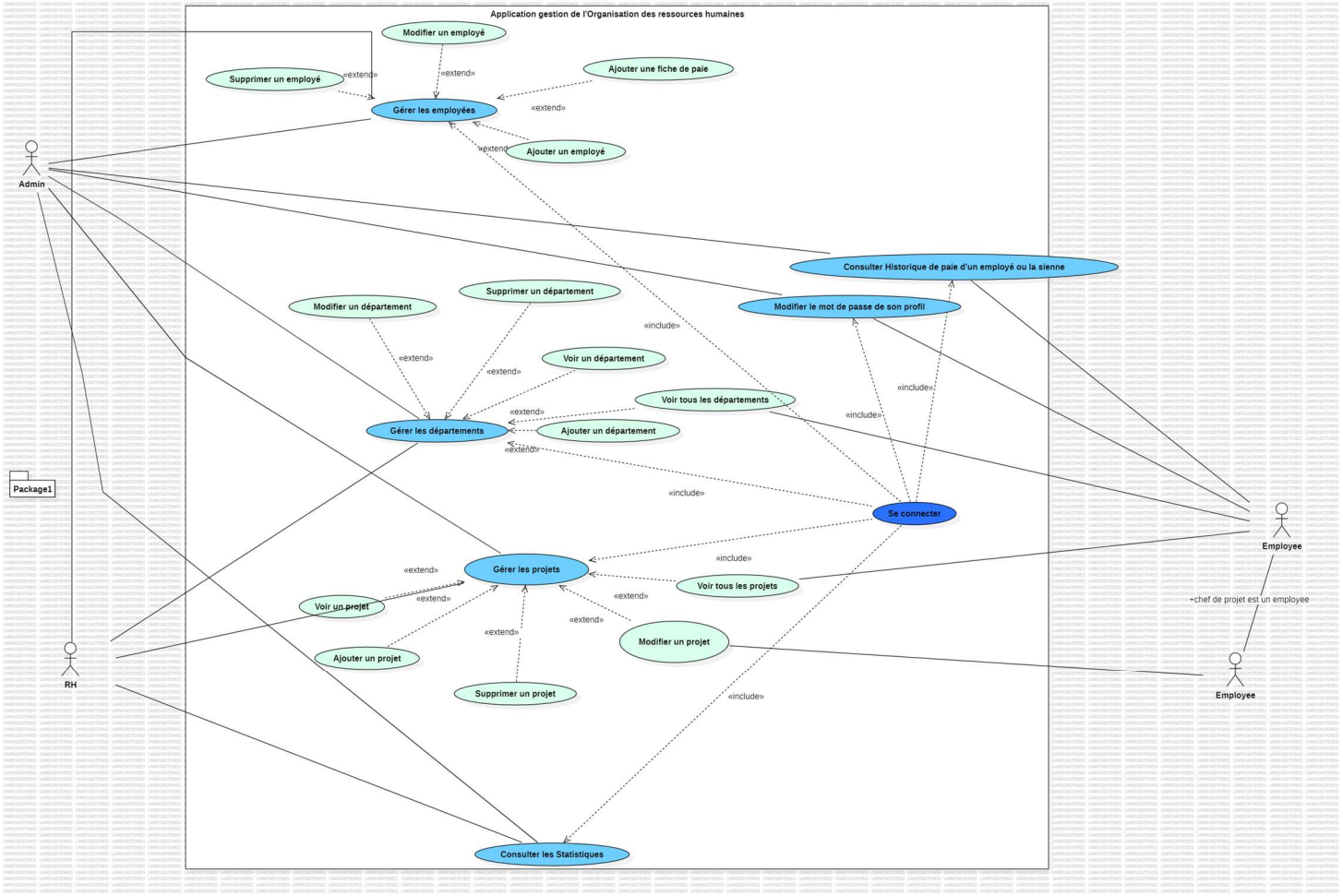
Le deuxième est la base de données, elle agit comme un acteur externe permettant de :

- Stocker des entités
- Récupérer et mettre à jour les informations
- Gérer la cohérence et l'intégrité des données

Dans la seconde partie du projet, une API exposée via Spring Boot peut être considérée comme un acteur externe. Elle fournit :

- Des Endpoint REST pour récupérer ou envoyer des données
- Une interface standardisée permettant la communication entre la partie JEE et la partie backend de Spring Boot

Diagramme des cas d'usage



L'application permet tout d'abord à l'utilisateur de **se connecter** afin d'accéder à ses fonctionnalités. Une fois authentifié, il peut également **modifier le mot de passe de son profil** pour sécuriser son compte.

La gestion interne s'organise autour de plusieurs fonctionnalités centrales. L'usage **Gérer les employés** regroupe l'ensemble des opérations liées au personnel : ajout d'un employé, modification de ses informations, suppression de son profil, ainsi que la création d'une fiche de paie. De manière similaire, le cas d'usage **Gérer les départements** permet d'administrer les services de l'entreprise en ajoutant, modifiant ou supprimant des départements, mais aussi en consultant les informations détaillées d'un département ou la liste complète des départements existants.

La fonctionnalité **Gérer les projets** offre la possibilité d'ajouter de nouveaux projets, d'en modifier les caractéristiques, de les supprimer ou encore d'en consulter les détails ou la liste globale. L'utilisateur dispose aussi d'un accès à **l'historique de paie**, lui permettant de consulter ses propres fiches de salaire ou celles d'un employé, selon ses droits. Enfin, la section **Consulter les statistiques** donne une vue d'ensemble sur l'organisation interne via des indicateurs tels que la répartition des employés, l'état des projets ou les effectifs par département.

L'ensemble de ces cas d'usage couvre toutes les actions essentielles nécessaires à la gestion quotidienne des employés, des structures internes et des activités de l'entreprise.

Architecture générale de l'application

L'application que nous avons développée repose sur une architecture en couches inspirée du modèle MVC (Model-View-Controller). Ce découpage permet de séparer clairement les responsabilités, de faciliter la maintenance et de rendre le code plus évolutif.

1. Présentation des couches

La couche Vue correspond aux pages JSP accessibles depuis l'interface utilisateur. Elle gère l'affichage des données, la mise en forme et la récupération des entrées des utilisateurs notamment via des formulaires. Les pages JPS servent uniquement à présenter les informations et à transmettre les actions de l'utilisateur aux servlets.

Les servlets jouent le rôle de contrôleurs. Elles reçoivent les requêtes envoyées depuis les pages JSP, analysent les actions demandées (connexion, création, modification, etc.) et dirigent les différentes opérations. Les servlets appellent les DAO pour effectuer les traitements nécessaires puis renvoient l'utilisateur vers la JPS appropriée.

La couche service contient la logique métier, elle regroupe l'ensemble des règles appliquée avant de manipuler les données (vérifications, traitements, calculs, contrôles). Cette couche n'est pas toujours explicitement séparée dans les petits projets mais elle reste présente entre les servlets et les DAO pour éviter de surcharger les contrôleurs.

La couche DAO gère l'accès aux données via Hibernate. Chaque entité (Employee, Department, Project, etc.) possède un DAO spécifique permettant la création, la modification, la suppression et la récupération des données dans la base de données. Les interfaces DAO garantissent une structure cohérente et facilitent la maintenance, tandis que leurs implémentations réalisent concrètement les opérations CRUD.

Les données sont stockées dans une base relationnelle, MySQL. La structure est définie via des scripts SQL, puis Hibernate se charge de gérer la liaison entre les modèles Java et les tables SQL.

2. Interactions entre les modules

L'ensemble des couches interagit selon un flux bien définis :

- L'utilisateur effectue une action depuis l'interface.
- La page JSP envoie une requête http à une servlet dédiée.
- La servlet analyse la requête, vérifie les paramètres et délègue la logique au service ou directement au DAO en fonction de l'architecture.
- Le service appelle le DAO qui exécute les opérations nécessaires sur la base de données via Hibernate.
- Le DAO retourne les données au service qui les renvoie ensuite à la servlet.
- La servlet met les données dans la requête ou la session et redirige vers une page JSP qui affiche le résultat à l'utilisateur.

3. Workflow principal de l'application

Le fonctionnement global du site peut être résumé par plusieurs étapes clés.

Étape 1 : Connexion

L'utilisateur arrive sur la page index.jsp avec un accès limité.

- Il accède à la page de connexion FormConnection.jsp via la barre de navigation.
- Il saisit son identifiant et son mot de passe.

- LoginServlet vérifie l'authentification via EmployeeDAO.
- En cas de succès, une session utilisateur est ouverte et il est redirigé vers la page index.jsp sans limitation.

Étape 2 : Accès à la gestion

Sur Gestion.jsp, l'utilisateur accède aux différentes sections du site :

- Gestion des employés
- Gestion des départements
- Gestion des projets
- Gestion des fiches de paie (via la gestion des employés)

Chaque bouton redirige vers une servlet spécifique qui charge les données à afficher via les DAO.

Étape 3 : Opérations CRUD

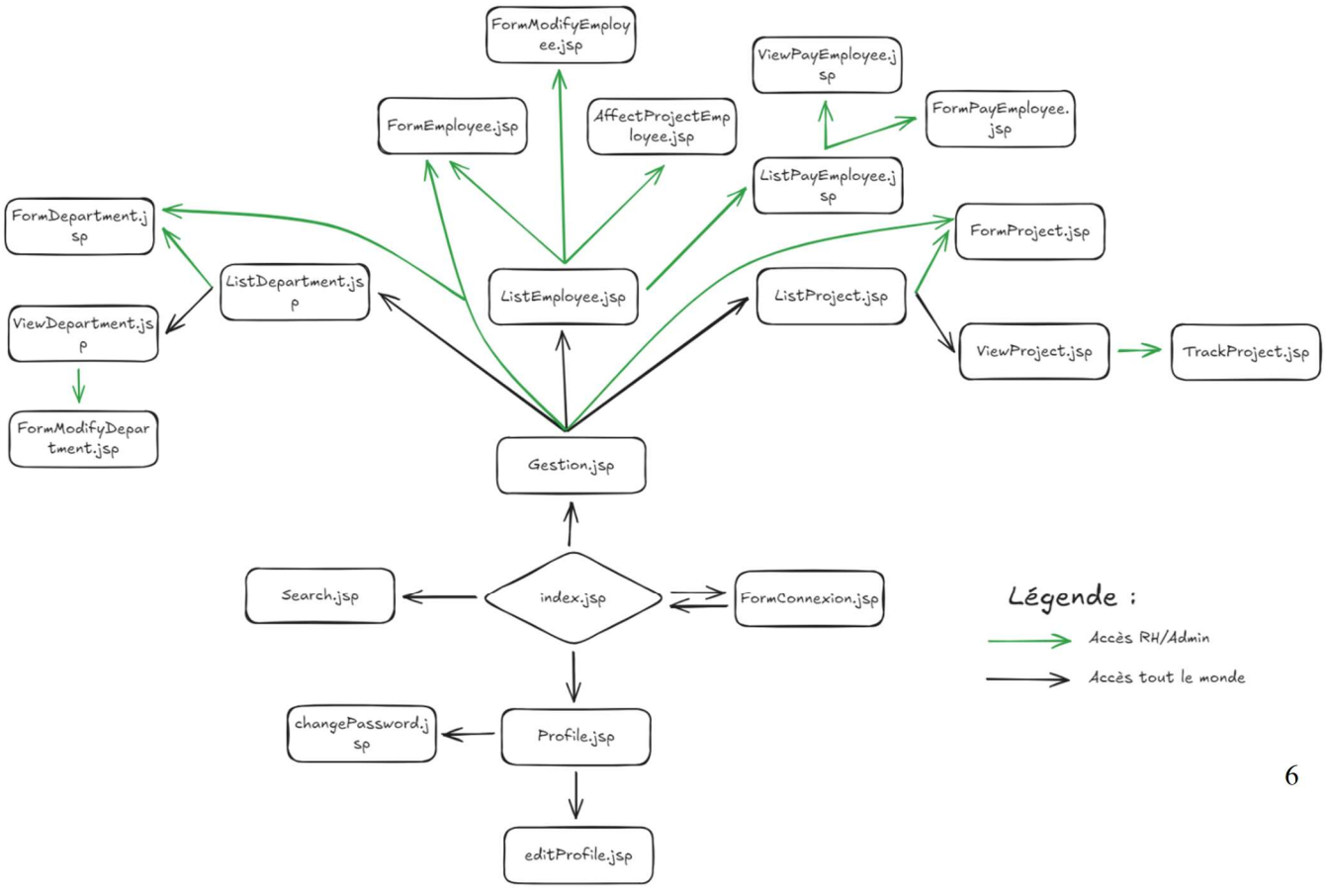
Selon la fonctionnalité choisie, l'utilisateur peut :

- Créer un élément (Exemple : Employee)
- Lire les informations d'un élément (Exemple : ViewEployee.jsp)
- Modifier les données (Exemple : FormModifyEmployee.jsp)
- Supprimer une entité

Chaque opération suit le même schéma :

- La page JSP envoie un formulaire à la servlet concernée.
- La servlet traite les données et fait appel au DAO.
- Le DAO exécute l'action via Hibernate.
- L'utilisateur est redirigé vers la page mise à jour.

Le schéma suivant illustre l'architecture générale du site que nous avons développé. Chaque flèche représente un lien de navigation entre les pages.



Modélisation UML du domaine

Le modèle de domaine est organisé autour de l'entité centrale Employee, qui représente un employé de l'entreprise avec ses informations personnelles (nom, prénom, grade, poste, identifiant de connexion, etc.). Chaque employé est associé à un Grade (enum) qui caractérise son niveau hiérarchique.

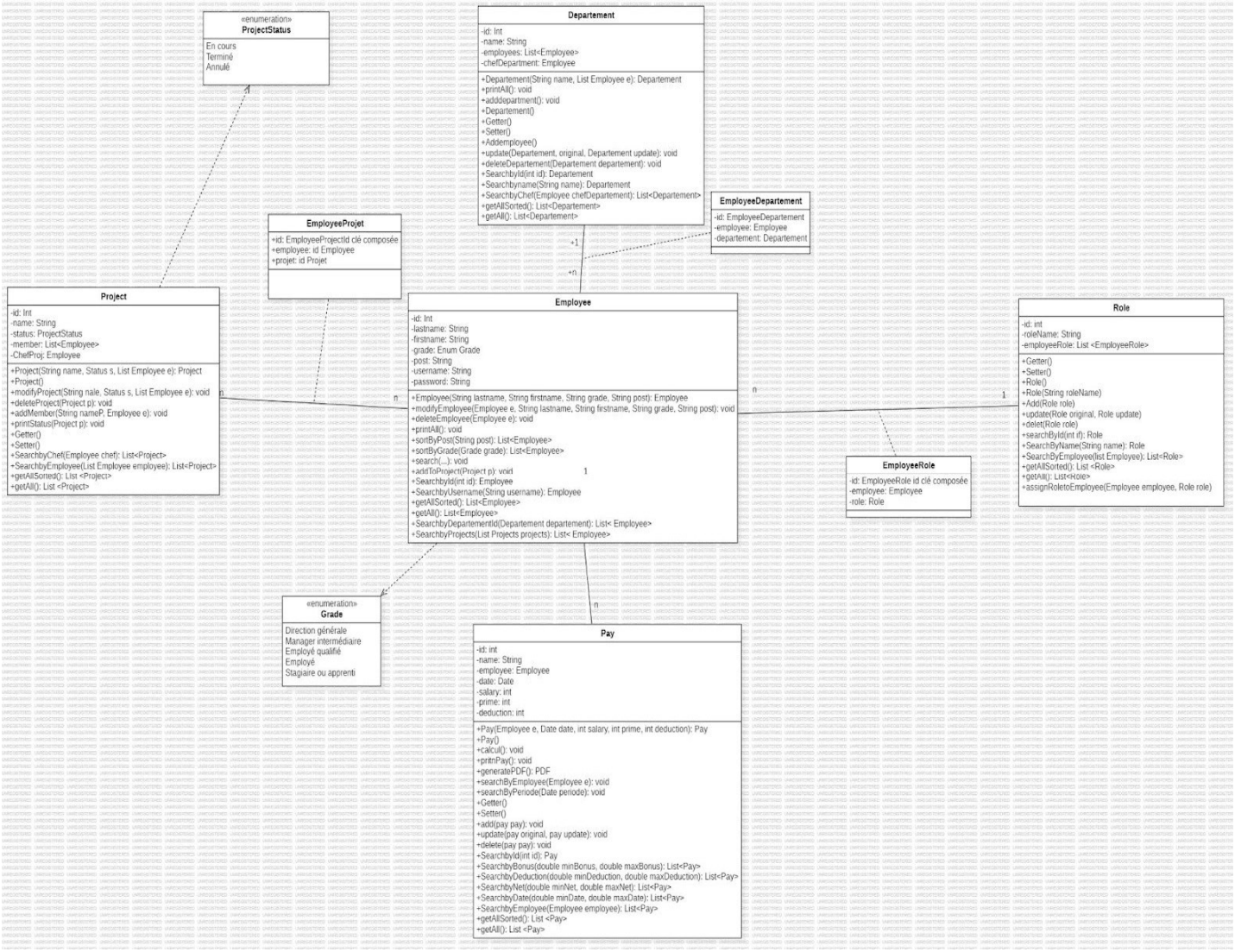
L'entité Department représente les différents services de l'organisation. Un département regroupe plusieurs employés, ce qui correspond à une relation 1–N (un département, plusieurs employés). Cette relation est matérialisée par la liste employees côté Department et par la référence au département côté Employee.

L'entité Project modélise les projets internes de l'entreprise : nom, statut, équipe, etc. Le statut d'un projet est représenté par l'énumération ProjectStatus (En cours, Terminé, Annulé…). La relation entre Employee et Project est de type N–N, car un employé peut participer à plusieurs projets et un projet peut impliquer plusieurs employés. Cette association est gérée par la classe intermédiaire EmployeeProjet, qui contient la clé composée et les références vers un Employee et un Project.

La gestion des rôles applicatifs est assurée par l'entité Role, qui décrit les droits d'accès (ex. : ADMIN, MANAGER, EMPLOYEE). La relation entre Employee et Role est également N–N, implémentée via la classe d'association EmployeeRole. Un employé peut donc posséder plusieurs rôles, et un même rôle peut être attribué à plusieurs employés.

Enfin, l'entité Pay représente les fiches de paie : salaire de base, primes, retenues et net à payer. Chaque fiche de paie est liée à un seul employé, tandis qu'un employé peut avoir plusieurs fiches au cours du temps, ce qui correspond à une relation 1–N entre Employee et Pay.

Dans l'ensemble, ce diagramme de classes met en évidence un modèle cohérent où Employee est au cœur des relations avec les départements, les projets, les rôles et les fiches de paie, tandis que les classes d'association permettent de gérer proprement les relations multiples (N–N) entre les différentes entités.



Implémentation

Structure du projet JEE

Afin de mieux comprendre l'organisation interne de notre projet, le schéma ci-dessous présente l'arborescence complète de notre application JEE, en détaillant la structure des packages, des ressources et des fichiers principaux.

- .idea
- .mvn
- Data : Dossier lié à la base de données
    - CreateDataBase.sql : Création de la base de données et insertion de plusieurs données
    - CreateUser.sql : Création de l'identifiant et du mot de passe permettant de se connecter à la base de données
- src
    - main
        - java
            - com.example.projetjeegroupeq
                - controller : Dossier lié aux contrôlleurs
                    - AuthFilter.java : Accès des pages si l'utilisateur est connecté
                    - DepartmentServlet.java : Visualisation, ajout, modification, suppression des départements
                    - EmployeeServlet.java : Visualisation, ajout, modification, suppression des employées
                    - ListProjectServlet.java : Visualisation, ajout, modification, suppression des projets
                    - LoginServlet.java : Connexion
                    - LogoutServlet.java : Déconnexion
                    - PayServlet.java : Visualisation, ajout, modification, suppression des paies
                    - ProjectServlet.java : Visualisation, ajout, modification, suppression des projet
                    - ReportServlet.java : gere le mappage des statistiques
                - dao : Dossier lié aux DAO
                    - implementation
                        - DepartmentDAO.java : Gère les entités 'department'
                        - EmployeeDAO.java : Gère les entités 'employee'
                        - PayDAO.java : Gère les entités 'pay'
                        - ProjectDAO.java : Gère les entités 'project'
                        - ReportDAO.java : Gère les entités 'report'
                        - RoleDAO.java : Gère les entités 'role'
                    - interfaces
                        - DepartmentDAOI.java : Interface DAO de 'departement'
                        - EmployeeDAOI.java : Interface DAO de 'employee'
                        - PayDAOI.java : Interface DAO de 'pay'
                        - ProjectDAOI.java : Interface DAO de 'project'
                        - ReportDAOI.java : Interface DAO de 'report'
                        - RoleDAOI.java : Interface DAO de 'role'
                    - sortingType : Dossier spécifiant les façon de trier les résultats lors de requêtes sur la BDD
                        - DepartmentSortingType.java : Enumération listant les façons possibles de trier un résultat contenant plusieurs entités 'department'
                        - EmployeeSortingType.java : Enumération listant les façons possibles de trier un résultats contenant plusieurs entités 'employee'
                        - PaySortingType.java : Enumération listant les façons possibles de trier un résultats contenant plusieurs entités 'pay'
                        - ProjectSortingType.java : Enumération listant les façons possibles de trier un résultats contenant plusieurs entités 'project'
                    - testDAO
                        - TestDAO.java : tests divers sur les DAO afin de vérifier leur fonctionnement sans relancer le serveur
                - model : Dossier lié aux modèles
                    - embeddedId
                        - EmployeeProjectId.java : Clé enrichie pour la jointure
                        - EmployeeRoleId.java : Clé enrichie pour la jointure
                    - Department.java : Modèle de la table "department"
                    - Employee.java : Modèle de la table "employee"
                    - EmployeeProject.java : Modèle de la table "employeeProject"
                    - EmployeeRole.java : Modèle de la table "employeeRole"
                    - Grade.java : Enumération des grades d'un employé
                    - Pay.java : Modèle de la table "pay"
                    - Project.java : Modèle de la table "project"
                    - ProjectStatus.java : Enumération du statut d'un projet
                    - Role.java : Modèle de la table "role"
                - util : Dossier lié à Hiebernate
                    - HibernateUtil.java : code pour simplifier l'initialisation des objets manipulant la base de données (entity manager factory)
                    - PayPdfGenerator.java :
        - ressources
            - META-INF

```
                - persistence.xml
        - templates :
                - PayslipTemplate.html : Apparence de la fiche de paie
    - webapp : Dossier lié aux pages JSP et CSS
        - CSS : Dossier lié au CSS
                - style.css : Fichier contenant tout le style du site
        - WEB-INF
                - web.xml
        - AffectEmployeeProject.jsp : Page d'affectation d'un projet à un employé
        - changePassword.jsp : Page de modification du mot de passe
        - FormConnection.jsp : Page de connexion au site
        - FormDepartment.jsp : Page de création d'un département
        - FormEmployee.jsp : Page de création d'un employé
        - FormModifyDepartment.jsp : Page de modification d'un département
        - FormModifyEmployee.jsp : Page de modification d'un employé
        - FormPay.jsp : Page de création d'une fiche de paie
        - FormProject.jsp : Page de création d'un projet
        - Gestion.jsp : Page de gestion des employés, départements et projets
        - index.jsp : Page d'accueil
        - ListDepartment.jsp : Page de listing des départements
        - ListEmployee.jsp : Page de listing des employés
        - ListPay.jsp : Page de listing des fiches de paie
        - ListProject.jsp : Page de listing des projets
        - Profile.jsp : Page du profil de l'utilisateur
        - Report.jsp : Page des statistiques de l'entreprise
        - Search.jsp : Page de recherche
        - TrackProject.jsp : Page de modification de l'avancement d'un projet
        - ViewDepartment.jsp : Page de visualisation d'un département
        - ViewEmployee.jsp : Page de visualisation d'un employé
        - ViewPay.jsp : Page de visualisation d'une fiche de paie
        - ViewProject.jsp : Page de visualisation d'un projet
- target
- .gitignore
- mvnw
- mvnw.cmd
- pom.xml
- README.md
- SujetProjet.pdf
```

Implémentation JEE

La première version de l'application a été développée en utilisant les technologies Java EE classiques : Servlets, JSP, DAO, JDBC/Hibernate, et une base de données relationnelle MySQL. Le projet est organisé selon une structure claire inspirée du modèle MVC (Model–View–Controller), avec plusieurs packages spécialisés :

- model : contient les entités métier (Employee, Department, Project, Pay, Role…) ainsi que les classes d'identifiants composites.
- DAO et DAO.implementation : regroupent les interfaces DAO et leurs implémentations pour l'accès aux données via Hibernate.
- controller / servlet : contient les différentes servlets responsables de la gestion des requêtes HTTP.
- cutil : inclut la configuration Hibernate (SessionFactory).
- webapp : comporte les pages JSP, les fichiers CSS, ainsi que les ressources front-end nécessaires.

Cette organisation assure une séparation nette entre la logique métier, la gestion des requêtes et l'affichage des données.

Avant toute insertion, la servlet effectue des contrôles afin de garantir :

- La présence de toutes les informations obligatoires,
- La validité des formats (ex. : salaire numérique),
- La cohérence de l'attribution du département ou du grade.

En cas d'erreur, un message explicatif est renvoyé à la JSP afin d'informer l'utilisateur.

Pour la partie Accès et authentification nous avons fait comme suit. AuthFilter gère les pages accessibles en fonction de si l'utilisateur est connecté, s'il est déconnecté alors il a accès à l'accueil et à la page de connexion

Les permissions se font selon les rôles suivants :

Employé :
- Accès aux listes d'employés, de projets, de département
- Accès uniquement à ses propres fiches de paie

RH :
- Même chose qu'employé
- Accès à la liste générale des fiches de paies
- Droit de modification et de création
- Accès aux statistiques

Admin :
- Tous les accès dont la suppression

Focus sur le cas d'usage : Ajouter un employé

Le scénario Ajouter un employé illustre parfaitement le fonctionnement de l'ensemble des couches JEE.

1. Saisie du formulaire (JSP)

L'utilisateur accède à une page FormEmployee.jsp, contenant un formulaire où il renseigne les informations du nouvel employé : nom, prénom, grade, département, salaire, etc. Lorsque l'utilisateur valide le formulaire, une requête HTTP POST est envoyée à la servlet EmployeeServlet avec l'action add.

2. Traitement par la servlet

EmployeeServlet identifie l'action demandée et appelle la méthode dédiée. La servlet :

- Récupère les données du formulaire,
- Vérifie que les champs requis sont bien remplis,
- Crée un objet Employee avec les informations saisies,
- Fait appel au DAO pour enregistrer l'employé dans la base de données.

En cas de champ vide ou invalide, un message d'erreur est renvoyé vers la JSP.

3. Enregistrement en base via le DAO

La couche DAO utilise Hibernate pour ajouter l'employé dans la base. Les annotations JPA permettent à Hibernate de générer automatiquement les requêtes SQL associées aux opérations CRUD. Le DAO ouvre une transaction, persiste l'objet Employee, puis valide la transaction.

4. Redirection vers la liste des employés

Après l'ajout réussi, la servlet recharge la liste des employés et redirige l'utilisateur vers la page ListEmployee.jsp. L'utilisateur voit alors apparaître le nouvel employé dans la liste.

Implémentation Spring Boot

1. Pourquoi Spring Boot ?

Notre projet a démarré avec une architecture JEE classique (servlets, JSP, JDBC) mais nous avons migré vers Spring Boot pour les plusieurs raisons :

Simplification du développement

Spring Boot embarque une configuration automatique (« auto-configuration ») qui évité la création manuelle de fichier XML. Il détecte automatiquement les dépendances présentes dans le projet et initialise les composants nécessaires.

Structure claire et modulaire

En séparent explicitement Controllers, Services, Repositories et Entities, le code devient plus simple à maintenir, tester et faire évoluer.

Intégration facile avec JPA et une base SQL

Grâce à Spring Data JPA, les opérations CRUD sont gérées de manière transparente. Il suffit de déclarer une interface Repository pour obtenir ls fonctions de base.

2. Architecture REST

L'architecture REST est une manière d'organiser les interactions entre client et serveur en utilisant des ressources identifiées par des URLs. Dans le projet, chaque entité correspond à une ressource REST, les opérations CRUD sont réalisées via des méthodes HTTP.

3. Structure du module Spring Boot

Afin de mieux comprendre l'organisation interne de notre projet, le schéma ci-dessous présente l'arborescence complète de notre application SpringBoot.

```
- .idea/
- .mvn/
- .smarttomcat/
- src/
    - main/
        - java/
            - org.example.projetjeegroupeqspringboot/
                - config/ : Configuration de sécurité, JPA, conversion enums, beans
                    - GlobalControllerAdvide.java : Rend l'utilisateur accessible de n'importe quelle page
                    - SecurityConfig.java : Configuration de la connexion automatique à la BDD
                - controller/
                    - DepartmentController.java : Contrôleur REST/MVC pour department
                    - EmployeeController.java : Contrôleur REST/MVC pour employee
                    - GestionController.java : Contrôleur REST/MVC pour gestion
                    - HomeController.java : Contrôleur REST/MVC pour la page index
                    - LoginController.java : Contrôleur REST/MVC pour la connexion
                    - PayController.java : Contrôleur REST/MVC pour pay
                    - ProfileController.java : Gestion de connexion/déconnexion et changement de mot de passe
                    - ProjectController.java : Contrôleur REST/MVC pour les projets
                    - ReportController.java : Contrôleur REST/MVC pour le rapport
                - entity/
                    - embededId/
                        - EmployeeProjectId.java : Clé composite (id_employee, id_project) pour employee_project
                        - EmployeeRoleId.java : Clé composite (id_employee, id_role) pour employee_role
                    - enumeration/
                        - Grade.java : Enum des grades employés
                        - ProjectStatus.java : Enum des statuts projet
                    - Department.java : Entité JPA pour la table department
```

```
                                        - Employee.java : Entité JPA pour la table employee
                                        - EmployeeProject.java : Entité JPA de jointure
                                        - EmployeeRole.java : Entité JPA de jointure
                                        - Pay.java : Entité JPA pour la table pay
                                        - Project.java : Entité JPA pour la table project
                                        - Role.java : Entité JPA pour la table role
                                - repository/
                                        - DepartmentRepository.java : Interface Spring Data JPA pour Department
                                        - EmployeeProjectRepository.java : Interface Spring Data JPA pour employee-project
                                        - EmployeeRepository.java : Interface Spring Data JPA pour Employee
                                        - EmployeeRoleRepository.java : Interface Spring Data JPA pour employee-role

                                        - ProjectRepository.java : Interface Spring Data JPA pour Project
                                        - PayRepository.java : Interface Spring Data JPA pour Pay
                                        - RoleRepository.java : Interface Spring Data JPA pour Role
                                - service/
                                        - AssignService.java : Service d'affectation employee-project
                                        - DepartmentService.java : Logique métier département
                                        - EmployeeService.java : Logique métier employé
                                        - ProjectService.java : Logique métier projet
                        - ReportService.java : Service de statistiques
                                        - UserService.java : Service de l'utilisateur
                                - util/
                                        - PayPdfGenerator.java Génération PDF des fiches de paie
                                - package-info.java : Métadonnées de package (Javadoc ou annotations).
                                - ProjetJeeGroupeQSpringBootApplication.java : Configure le contexte et lance le serveur embedded Tomcat
                                - ServletInit.java : Initialisation si nécessaire entre filtres/servlets et Spring MVC
        - resources/
                - Data/
                        - fix_grade_values.sql : Correctifs de valeurs d'enum grade dans la BDD
                        - user.sql : Création utilisateur BDD ou données utilisateur seed
        - static/
                - css/
                        - style.css : Style global pour les pages Thymeleaf
                - templates/
                        - fragments/
                                - navbar.html :  Fragment de navbar inclus dans toutes les pages
                        - AssignEmployeeProject.html : Page d'affectation employee-project
                        - ChangePassword.html : Page de changement de mot de passe
                        - FormDepartment.html : Formulaire création/modification département
                        - FormEmployee.html : Formulaire création/modification employé
                        - FormPay.html : Formulaire création/modification fiche de paie
                        - FormProject.html : Formulaire création/modification projet
                        - Gestion.html : Tableau de bord
                        - index.html : Page d'accueil
                        - ListDepartment.html : Liste des départements
                        - ListEmployee.html : Liste des employés (tri, actions)
                        - ListPay.html : Liste des fiches de paie (global ou par employé)
                        - ListProject.html : Liste des projets
                        - Login.html : Page de connexion
                        - PayslipTemplate.html : Page de génération des fiches de paie
                        - Profile.html : Page profil utilisateur
                        - Report.html : Page statistiques
                        - TrackProject.html : Page de suivi/avancement projet
                        - ViewDepartment.html : Détail département (chef, membres)
                        - ViewEmployee.html : Détail employé (département, projets, paies)
                        - ViewPay.html : Détail fiche de paie (net à payer, impression)
                        - ViewProject.html : Détail projet (statut, chef, membres)
                - application.properties : Configuration BDD, JPA/Hibernate, Spring MVC/Thymeleaf, sécurité.
                - data.sql : Script d'initialisation des données à l'amorçage
        - test/
                - java/
                        - org.example.projetjeegroupeqspringboot
                                - ProjetJeeGroupeQSpringBootApplicationTests.java
- target/
- mvnw
- mvnw.cmd
- pom.xml
- README.md
```

4. Contrôleurs REST

Les contrôleurs sont le point d'entrée de l'application. Ils reçoivent les requêtes HTTP, appellent les services appropriés et retournent les réponses (JSON ou vues HTML).

Voici un exemple d'un contrôleur pour les vues HTML.

```
@Controller
@RequestMapping("/employees")
public class EmployeeController {
   @GetMapping
   public String listEmployees(Model model) {
      List<Employee> employees = employeeService.findAll();
model.addAttribute("employees", employees);       return "ListEmployee";
```

@Controller renvoie du JSON automatiquement
@RequestMapping définit la route principale
@GetMapping, @PostMapping définissent les endpoints REST
@Autowired définit une injection de dépendance du service

5. Services et Repositiries

Spring Boot fonctionne avec une architecture en couche : Controller → Service → Repository → Database

Le rôle de la couche Service est d'orchestrer les appels aux repositories et de gérer les transactions avec @Transactional. Elle contient la logique métier du projet. La couche Repository sert d'interface d'accès pour les données, étend JpaRpository pour hériter des méthodes CRUD et définit des requêtes personnalisées.

6. Interaction avec le Frontend

Notre application utilise une approche hybride car nous utilisons Thymeleaf pour générer des pages HTML du côté serveur en fusionnant un Template HTML (avec attributs th :*) et des données du modèle mais nous n'avons pas d'API REST JSON qui permet une architecture monolithique.

Interaction JEE/Spring Boot

Après la migration vers Spring Boot, la partie « Front-End JSP » existe toujours mais ne dialogue plus directement avec les servlets. A la place, elle utilise l'API REST Spring Boot. Les pages JSP servent uniquement à afficher l'interface utilisateur, envoyer des requêtes http, et récupérer les données JSON fournies par l'API Spring Boot. Les servlets sont remplacés par des Controllers REST et du JavaScript côté JSP. Le front n'accède plus aux objets Java directement dans l'ancienne version on fait « <%= employee.getFirstName() %> », mais avec Spring Boot ce n'est plus possible car les données viennent d'un appel http et non d'un attribut de requête.

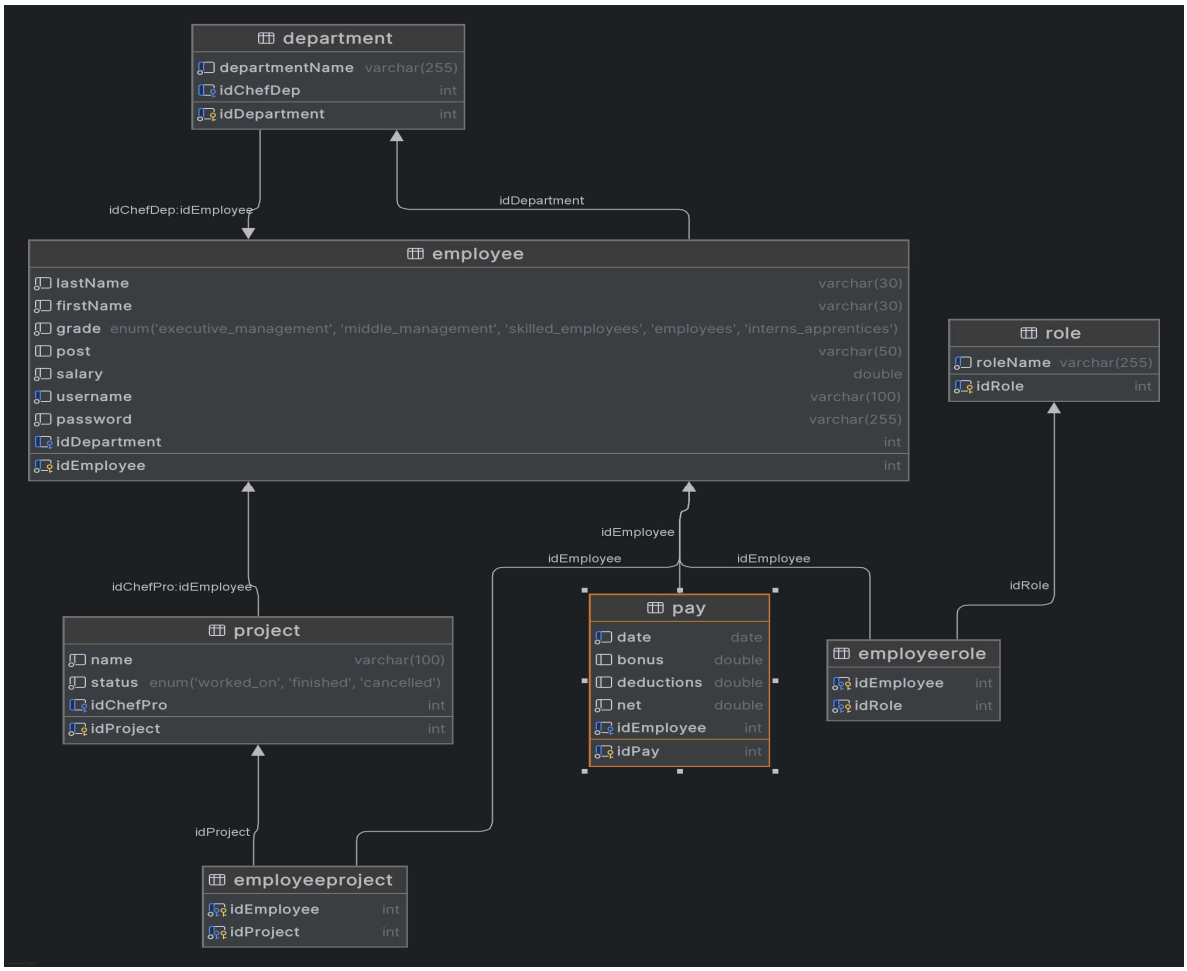
Voici un exemple d'un appel REST pour récupérer la liste des employés.

```
@GetMapping("/employees")
public List<Employee> getAllEmployees() {
   return employeeService.findAll();
}
```

L'URL de l'API pour effectuer cet appel est «http://localhost:8080/employee ».

Base de données

Schéma physique de BD



Choix techniques

Nous avons choisi **MySQL** pour sa facilité d'utilisation, sa compatibilité avec Hibernate et Spring Boot, et sa bonne performance pour une application de gestion. La base respecte les **contraintes d'intégrité** via des clés étrangères garantissant la cohérence des données. Enfin, la structure suit les principes de **normalisation**, ce qui évite les redondances et facilite la maintenance.

Développement, outils et méthodologie

Environnement de développement

Technologies utilisées

- **Spring Boot 4.0.0** : Framework principal
- **Spring Data JPA** : Accès aux données
- **Spring Security** : Authentification et autorisation
- **Thymeleaf** : Moteur de templates
- **MySQL** : Base de données
- **Hibernate** : ORM (Object-Relational Mapping)

- **Maven** : Gestion des dépendances

Outils utilisés

- Git / GitHub (gestion version)
- Trello / Jira (suivi des tâches)
- Postman (tests API)
- Figma / Whimsical/StarUML (maquettes, UML)

Déploiement

Les consignes de déploiement se trouvent dans le READ ME du projet.

Répartition des tâches

PERSONNE EN CHARGE	TACHE	PROJET
ORIANNE	Création de la structure du projet	JEE
JONATHAN	Création de la structure du projet	SpringBoot
ORIANNE	Création des pages JSP/CSS	JEE
ORIANNE	Création de la BDD	JEE
FANTINE	Création des models	JEE
MATHIEU	Création des DAO	JEE
JONATHAN	Gestion du module Employee	JEE
FANTINE	Gestion du module Department	JEE
MATHIEU	Gestion du module Project	JEE
JONATHAN	Gestion du module Pay	JEE
JONATHAN	Gestion du module Profile	JEE
FANTINE	Gestion du module Report	JEE
JONATHAN	Gestion des accès/Role	JEE
MATHIEU	Gestion de la recherche	JEE
JONATHAN	Remplissage de la BDD	JEE
ORIANNE	Création des pages HTML	SpringBoot
TOM	Création des entities	SpringBoot
TOM	Création des repositories	SpringBoot
ORIANNE	Création des services	SpringBoot
ORIANNE	Remplissage de la BDD	SpringBoot
ORIANNE	Gestion du module Employee	SpringBoot
ORIANNE	Gestion du module Department	SpringBoot
ORIANNE	Gestion du module Project	SpringBoot
ORIANNE	Gestion du module Pay	SpringBoot
ORIANNE	Gestion du module Profile	SpringBoot
ORIANNE	Gestion du module Report	SpringBoot
ORIANNE	Gestion des accès	SpringBoot
TOM	Gestion de la recherche	SpringBoot
TOUTE L'EQUIPE	Rédaction du rapport	JEE/SB

Difficultés rencontrées & solutions

Nous avons rencontré plusieurs difficultés au cours du projet. La première concernait la **gestion des accès et de l'authentification**, car les rôles comme chef de projet ou chef de département n'étaient pas directement liés à la table Employee. Nous avons résolu ce problème en établissant une table des roles qui ne sont pas liés au chef departement, chef projet mais en fonction de Admin, RH, ou employé permettant d'attribuer facilement différents rôles à chaque utilisateur.

Une autre difficulté importante a été la **suppression des employés associés à d'autres entités**. Lorsqu'un employé était lié à un projet, un département ou un rôle, la suppression était bloquée par les contraintes de la

base. La solution a consisté à **supprimer d'abord tous les liens associés**, puis à supprimer l'employé, ce qui garantit la cohérence des données.

Enfin, nous avons aussi dû améliorer notre organisation interne. Au début, la répartition des tâches n'était pas optimale. Nous avons finalement adopté une division du travail par modules (employés, départements, projets…) et selon les couches MVC, ce qui a permis de travailler plus efficacement et sans chevauchement.

Conclusion

Le projet nous a permis de concevoir et de développer une application complète de gestion des employés, des départements, des projets et des fiches de paie, en utilisant successivement une architecture JEE classique puis une version améliorée en Spring Boot. L'ensemble des fonctionnalités essentielles est opérationnel et conforme aux objectifs du cahier des charges.

Sur le plan technique, ce travail nous a permis d'aborder concrètement les technologies JEE (Servlets, JSP, DAO, Hibernate) et d'explorer Spring Boot pour structurer une API REST plus moderne. Nous avons également renforcé nos compétences en conception UML, en organisation d'un projet collaboratif et en gestion de la persistance des données.

Cependant, plusieurs **améliorations restent possibles**. La gestion des accès, rudimentaire et pas assez cohérente dans la version JEE et absente dans la version Spring Boot, pourrait être totalement repensée pour intégrer une authentification robuste et une vraie gestion des rôles. De même, la partie statistique gagnerait à être modernisée, avec des graphiques plus esthétiques et interactifs. Enfin, certaines pages pourraient être rendues plus ergonomiques afin d'améliorer l'expérience utilisateur.

L'application constitue ainsi une base solide, fonctionnelle et extensible, sur laquelle il serait pertinent de poursuivre le développement pour proposer une solution plus complète, sécurisée et agréable à utiliser.