RAPPORT DE PROJET

J2EE (ING2, GSI)

2025-2026

Membres du groupe :

Ivan CHANTEPIE

Iskander DRIDI

Gédéon MUYA LABA

Thomas PISANESCHI

Nicolas LAGRANGE

Date de rendu : 30/11/2025

Sommaire :

1. Introduction

1.1 Contexte du projet

La transformation numérique des entreprises nécessite des outils performants pour gérer le capital humain. Dans le cadre du module J2EE de notre cursus d'ingénieur, nous avons reçu pour mission de concevoir une application Web de gestion des Ressources Humaines. Ce projet a pour double ambition de répondre à un besoin métier concret (administrer des employés et leur paie) et de nous faire acquérir une maîtrise technique approfondie des technologies Java Web, depuis les fondamentaux (Servlets/JSP) jusqu'aux frameworks modernes (Spring Boot).

1.2 Objectifs et périmètre fonctionnel

L'application "Gestion RH" centralise quatre domaines fonctionnels majeurs :

1. **Administration du Personnel :** Création et mise à jour des dossiers employés (état civil, poste, coordonnées).

2. **Organisation Structurelle :** Gestion des départements et affectation des collaborateurs.

3. **Gestion de Projets :** Suivi des missions, constitution des équipes et monitoring des états d'avancement.

4. **Gestion de la Paie :** Calcul automatisé des salaires nets mensuels et édition de fiches de paie.

5. **Gestion des Départements :** Vue et gestion globale des départements.

6. **Rapports et Statistiques :** Rapport détaillé ainsi qu’une utilisation de graph afin de représenter les différentes informations concernant la population dans l’entreprise.

2. Partie 1 : Conception et Analyse

2.1 Identification des Acteurs et Rôles

La sécurité et l'accès aux données sont régis par un système de rôles hiérarchiques (RBAC - Role Based Access Control), défini dans l'énumération Role.java :

- **Administrateur (Niveau 4) :** Possède tous les droits. Il est le seul à accéder aux rapports statistiques globaux et à pouvoir supprimer des données sensibles.
- **Chef de Département (Niveau 3) :** Peut gérer les employés de son périmètre et valider les fiches de paie.
- **Chef de Projet (Niveau 2) :** Responsable de la création des projets et de l'affectation des ressources. Il n'a pas accès aux données salariales.
- **Employé (Niveau 1) :** Accès en lecture seule à son profil et ses propres fiches de paie.

2.2 Règles de Gestion Métier

Lors de la phase d'analyse, nous avons établi des règles strictes pour garantir la cohérence du système :

- **RG-01 :** Un employé ne peut appartenir qu'à un seul département à la fois.
- **RG-02 :** Le calcul du salaire net suit la formule : Net = Salaire de Base + Primes - Déductions.

- **RG-03 :** Un projet ne peut avoir que trois états : "En cours" (*in process*), "Terminé" (*finished*) ou "Annulé" (*canceled*).

- **RG-04 :** La suppression d'un employé est logique (soft delete via le champ isActive) ou physique, mais doit être interdite s'il possède des fiches de paie liées (contrainte d'intégrité).

2.3 Modélisation des Données (Conception Technique)

Notre schéma de base de données a été conçu pour être robuste et performant.

[Diagramme de Classes UML]

Choix de conception : La Gestion des Relations N-N Une particularité de notre modèle réside dans la gestion des relations *Many-to-Many* entre Employe et Project. Au lieu d'utiliser une table de jointure classique (employee_project), nous avons opté pour une approche "NoSQL-like" au sein du relationnel :

- Les IDs des employés assignés à un projet sont stockés sous forme de chaîne CSV (ex: "102,105,108") dans la colonne employees de la table Projects.

- **Justification :** Ce choix simplifie considérablement les requêtes de lecture (pas de jointures SQL coûteuses) et facilite le mapping objet-relationnel manuel dans le DAO. Le parsing est géré efficacement par la couche applicative Java.

3. Partie 2 : Implémentation J2EE (Standard)

Cette phase constitue le cœur technique du projet, réalisé sans framework externe pour comprendre les mécanismes bas niveau du Web Java.

3.1 Architecture MVC et Pattern "Front Controller"

Nous avons structuré l'application selon le modèle MVC (Modèle-Vue-Contrôleur). Pour éviter la multiplication des fichiers Servlets, nous avons implémenté un **Front Controller par module**.

Exemple du ProjectManagementServlet : Cette Servlet unique intercepte toutes les requêtes liées aux projets (/projects/*). Elle agit comme un aiguilleur :

1. Elle analyse l'URL via request.getPathInfo().

2. Elle délègue le traitement à une méthode privée spécifique (createProject, assignEmployees, viewProject).

3. Elle redirige vers la vue appropriée (JSP).

Ce mécanisme centralise la logique de navigation et de gestion des erreurs, rendant le code plus maintenable.

3.2 La Couche Vue : Stratégie "Server-Side Rendering"

L'interface utilisateur est générée côté serveur via des pages JSP (JavaServer Pages).

- **JSTL (JavaServer Pages Standard Tag Library) :** Nous utilisons intensivement la JSTL pour la logique de présentation, éliminant ainsi le code Java (Scriptlets <% %>) des vues.

    - *Exemple :* L'affichage conditionnel des boutons d'administration se fait via <c:if test="${currentUser.role == 'ADMINISTRATEUR'}">.

- **Composants Réutilisables :** Un fichier index.jsp ou des fragments d'en-tête/pied de page sont inclus pour garantir une charte graphique cohérente sur toutes les pages.

3.3 La Couche Modèle : Persistance JDBC et Intégrité

L'accès aux données est encapsulé dans une classe RHDAO (Data Access Object).

- **Sécurité des données (SQL) :** Contrairement à une validation uniquement logicielle, nous avons renforcé la base de données avec des contraintes CHECK. Par exemple, la colonne project_state refuse toute valeur autre que 'in process', 'finished' ou 'canceled', protégeant l'intégrité des données même en cas de bug applicatif.

- **Transactions :** Les opérations critiques (comme la création d'un département avec affectation massive d'employés) sont gérées de manière atomique.

3.4 Focus : Algorithmes et Reporting

Le module de statistiques (ReportsServlet) démontre notre capacité à traiter des données complexes en Java. Plutôt que de surcharger la base de données avec des requêtes d'agrégation (GROUP BY), nous récupérons les données brutes et utilisons l'**API Stream de Java 8**.

Algorithme de répartition par grade :

```
Map<Integer, Long> rankDistribution = deptEmployees.stream()

    .collect(Collectors.groupingBy(Employe::getEmploye_rank, Collectors.counting()));
```

Cette approche fonctionnelle permet de générer des statistiques dynamiques très performantes. Côté affichage, nous avons développé un système de graphiques en CSS pur (reports.jsp), où la largeur des barres est calculée par le serveur (width: ${value}%), évitant l'usage de lourdes librairies JavaScript externes.

3.5 Sécurité et Authentification

La sécurité a été une priorité dès la conception :

1. **Hachage des Mots de Passe :** Utilisation de l'algorithme **BCrypt** (avec un coût de 12, visible via $2a$12... en base) pour qu'aucun mot de passe ne soit stocké en clair.

2. **Gestion de Session :** Le LoginServlet place l'objet Employe en session HTTP. Un filtre ou une vérification en début de chaque JSP (<c:if test="${empty sessionScope.currentUser}">) empêche l'accès aux pages protégées sans authentification.

3. **Cookies Persistants :** Implémentation d'une fonctionnalité "Se souvenir de moi" via un cookie sécurisé d'une durée de 7 jours.

4. Partie 3 : Refonte et Modernisation avec Spring Boot

La seconde phase du projet consistait à migrer l'application vers le framework Spring Boot. Cette transition a mis en lumière la puissance de l'inversion de contrôle.

4.1 Transition Architecturale

Le passage à Spring Boot a transformé notre code "impératif" (nous disons à la machine *comment* faire : ouvrir connexion, créer statement, exécuter) en code "déclaratif" (nous disons *quoi* faire).

- **Spring Data JPA :** Remplacement complet du DAO manuel. Les interfaces Repository génèrent automatiquement les implémentations des méthodes CRUD courantes (findAll, save, deleteById), réduisant le code d'accès aux données de 80%.

- **Service Layer :** Introduction d'une couche de service (@Service) pour isoler la logique métier (calculs de paie) des contrôleurs, améliorant la testabilité.

4.2 Gains de Productivité et Robustesse

- **Configuration Automatique :** La suppression du fichier web.xml et de la configuration manuelle de la DataSource (gérée via application.properties) a éliminé une source fréquente d'erreurs de déploiement.

- **Injection de Dépendances :** L'usage de l'annotation @Autowired permet de coupler les composants de manière souple, facilitant l'évolution du code.

4.3 Comparatif Technique (Avant / Après)

Fonctionnalité	Implémentation J2EE Standard	Implémentation Spring Boot
Point d'entrée	Servlets multiples + web.xml	Classe Application.java (@SpringBootApplication)
Routing	Manuel (if/else sur pathInfo)	Annotations (@GetMapping, @PostMapping)
Base de Données	JDBC pur (Connection, PreparedStatement)	Spring Data JPA (JpaRepository)
Serveur	Installation et config de Tomcat externe	Tomcat embarqué ("Embedded Server")

5. Retour d'Expérience et Difficultés

Ce projet ne s'est pas déroulé sans défis techniques, qui ont été formateurs :

1. **Gestion des dépendances (JARs) :** Au début du projet J2EE, nous avons rencontré des problèmes de ClassNotFoundException car les drivers MySQL n'étaient pas correctement inclus dans le WAR. L'adoption de Maven a résolu ce problème.

2. **Mapping Relationnel Manuel :** La gestion des clés étrangères et la reconstruction des objets (ex: lier un Employé à son Département) en JDBC pur a nécessité une rigueur importante pour éviter les NullPointerException.

3. **Débogage des JSPs :** Les erreurs dans les pages JSP sont parfois difficiles à tracer. L'implémentation de notre DiagnosticServlet nous a permis de vérifier rapidement si les vues étaient bien déployées au bon endroit.

6. Conclusion

Ce projet de semestre a été une expérience complète de développement logiciel. Nous avons non seulement produit une application fonctionnelle respectant le cahier des charges (RH, Paie, Projets), mais nous avons surtout compris l'évolution historique de la plateforme Java.

Maîtriser les Servlets et JDBC nous donne aujourd'hui une compréhension fine de ce qui se passe "sous le capot" de Spring Boot. Nous sommes désormais capables de justifier les choix d'architecture et de mesurer la valeur ajoutée des frameworks modernes en entreprise : robustesse, maintenabilité et rapidité de développement.