Rapport projet JEE

Amâncio Felipe ZANI DA SILVA
Jean Baptiste ANDRAWAS
Malik SMEDA
Shawrov DAS
Aline KIM

[ING2 GSI3]

Professeur encadrant : Mohamed HADDACHE

Soumis le 30 novembre 2025

Sommaire

1) Répartition des tâches

Amâncio Felipe ZANI DA SILVA : gestion des employés
Jean Baptiste ANDRAWAS : Spring Boot
Malik SMEDA : gestion des projets et des départements
Shawrov DAS : fiches de paie
Aline KIM : rédaction du rapport

2) Outils utilisés

Apache Tomcat : Serveur d'applications Servlet/JSP
Serveur web Java léger qui exécute les servlets et JSP (déploie l'archive .war). Utilisé pour héberger l'application Jakarta EE et servir les requêtes HTTP.

GitHub : Système de gestion de versions hébergé (plateforme Git)
Héberge le dépôt Git, gère l'historique, les branches et les pull requests, et fournit CI/CD et collaboration entre développeurs.

Visual Studio Code : Environnement de développement (IDE/éditeur)
Éditeur extensible avec intégrations pour Java, Gradle, débogage, terminal intégré et gestionnaire d'extensions pour le développement et le test.

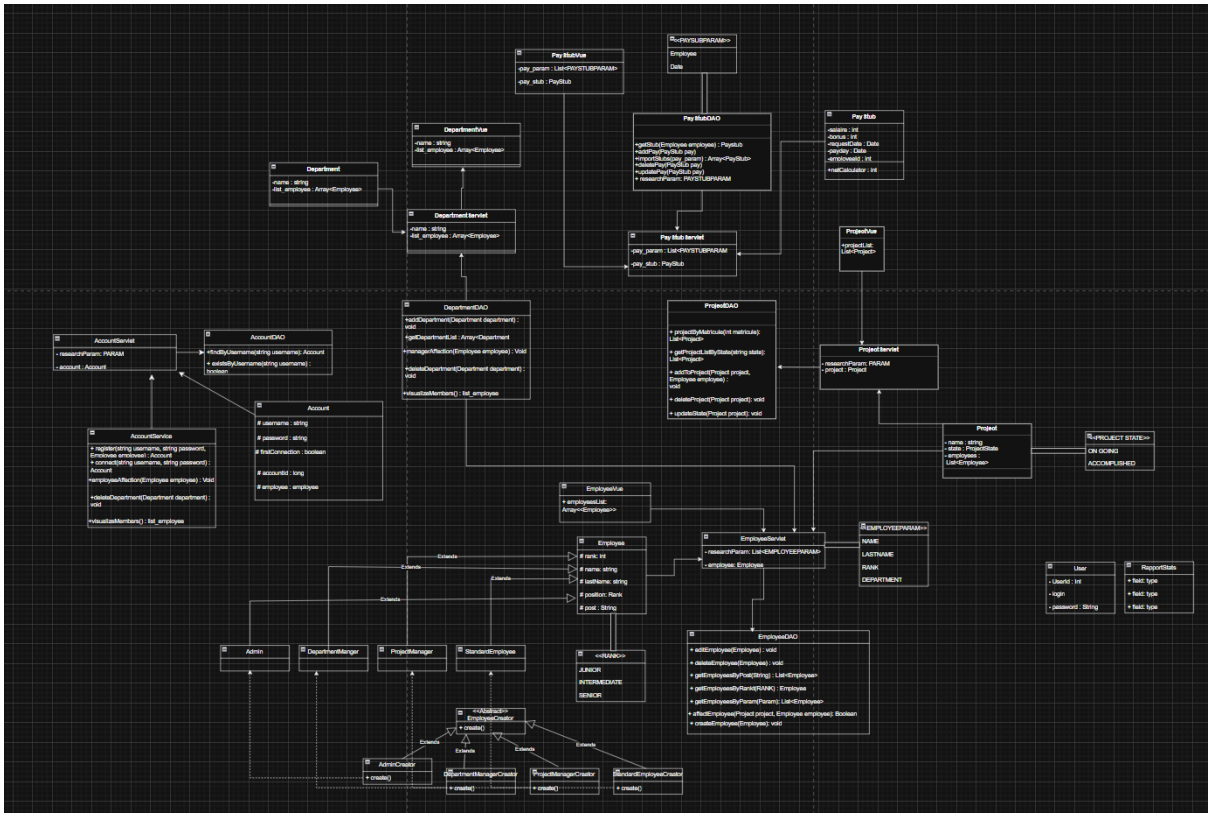
Gradle : Outil de build et de gestion des dépendances
Automatise compilation, packaging, tests et gestion des dépendances; exécute des tâches de build et produit le .war/.jar déployable.

pilotes JDBC MySQL / MariaDB : Drivers JDBC pour SGBD relationnels
Bibliothèques qui implémentent l'API JDBC pour permettre à l'application Java d'établir des connexions, exécuter des requêtes et manipuler les données dans MySQL/MariaDB.

ORM Hibernate : Moteur de mapping objet-relationnel
Traduit les entités Java en tables SQL et inversement, gère la persistance, les transactions et les requêtes HQL/Criteria pour simplifier l'accès à la base de données.

3) MCD

Lien vers le MCD :
https://app.diagrams.net/#G1xUC8TuZkMvy2WZvjWyy-N8OnTc5INAF6#%7B%22pageId%22%3A%22Bo4xSE06YOFsmRl7KkYg%22%7D



4) Fonctionnalités implémentées

Gestion des employés
Fonctionnalités : CRUD (création, lecture, mise à jour, suppression), recherche / filtrage, pagination légère, ajout/modification via formulaires.
UI : pages HTML + fragments HTMX pour interactions partielles (modal / rafraîchissements dynamiques).
Backend : servlets/controllers (ex. EmployeeServlet) qui valident les entrées et orchestrent les opérations.
Persistance : entité Employee + DAO/Repository (Hibernate) pour opérations SQL.
Extras : génération de rapports / export possible via JasperReports.

Gestion des départements
Fonctionnalités : CRUD pour départements, liaisons employés ↔ département, affichage des employés par département.
UI : pages/fragment HTML et actions AJAX/HTMX pour charger listes et formulaires.
Backend : actions/servlets spécialisés (ex. GetDepartmentEmployees) qui retournent vues partielles.
Persistance : entité Department et mapping JPA/Hibernate, gestion des relations (OneToMany / ManyToOne).

Gestion des projets
Fonctionnalités : CRUD projets, association employés ↔ projets (assignations), consultation des ressources par projet.
UI : formulaires et vues listant projets et membres affectés.
Backend : controllers/DAO pour créer/mettre à jour les affectations et récupérer les états.
Persistance : entité Project + tables de jointure gérées par Hibernate.

Gestion des fiches de paie
Fonctionnalités : génération de fiches de paie (calculs de paie basiques ou assemblage des données), stockage / consultation des bulletins, export/impression.
UI : pages de consultation et génération (souvent déclenchées via servlet de reporting).
Backend : logique de collecte des données de paie, génération de documents via JasperReports ou export PDF.
Persistance : entité(s) de paie ou agrégation des données depuis employés, projets et éléments de paie.

Authentification et autorisation
Fonctionnalités : login / logout, sessions utilisateur, contrôle d'accès par rôle (ex. rôles ADMIN / HR / USER).
UI : page de connexion et redirections selon rôle.
Backend : filtre/servlet de sécurité ou vérifications dans controllers qui restreignent l'accès aux opérations sensibles.
Persistance : table utilisateurs / rôles, mots de passe stockés (hashés) et chargés via DAO.
Autorisation : vérification des rôles avant actions CRUD et accès aux pages de gestion ou rapports.

5) Architecture MVC de l'application

Vue d'ensemble
L'application suit le modèle MVC classique : les requêtes HTTP sont traitées par des contrôleurs (Servlets), qui appellent la couche métier/persistance pour manipuler des entités (Modèle) et retournent des vues (JSP/HTML/HTMX/JS) pour l'affichage.

Modèle
Classes Java représentant les entités métier : Employé, Département, Projet, FicheDePaie (JPA/Hibernate entities).
Contient la validation minimale des objets et les mappings JPA (annotations @Entity, relations @OneToMany/@ManyToOne).
Fichiers exemples : app/src/main/java/org/employee/model/Employee.java,
Department.java, Project.java.

Contrôleur
Servlets Java traitant les requêtes HTTP (GET/POST), orchestrant la logique métier, validant les entrées et choisissant la vue.
Gère la navigation, la création/édition/suppression et déclenche la génération de rapports.
Exemples : app/src/main/java/org/employee/servlet/EmployeeServlet.java,
ReportServlet.java, actions comme GetDepartmentEmployees.

Vue
Pages JSP/HTML/CSS/JavaScript + fragments HTMX pour interactions partielles (rafraîchissements asynchrones).
TypeScript compilé en JS pour rich interactions côté client.
Fournit formulaires pour CRUD, listes, modals et visualisation des fiches de paie / rapports.
Exemples : app/src/main/webapp/*.html, *.jsp, app/src/main/webapp/ts/src/…

Couche de persistance
Hibernate (JPA) comme ORM : mappe entités Java ↔ tables SQL, gère sessions, cache et requêtes HQL/Criteria.
DAO / Repository classes encapsulent opérations CRUD et requêtes spécialisées.
JDBC via pilotes MySQL/MariaDB pour la connexion au SGBD (fourni par le driver JDBC).
Exemples : EmployeeDAO.java

Base de données
SGBD relationnel (MySQL/MariaDB) pour stockage des données persistantes : tables utilisateurs/roles, employees, departments, projects, payrolls.
Transactions gérées au niveau Hibernate/JDBC.

Flux typique d’une requête
Client envoie requête HTTP (formulaire/HTMX/JS).
Servlet (Contrôleur) reçoit la requête, extrait paramètres, vérifie l’authentification / autorisation.
Contrôleur appelle la couche service/DAO (Hibernate) pour lire/modifier entités.
La couche persistance exécute les opérations SQL via JDBC driver.
Contrôleur sélectionne une vue (JSP/HTML ou fragment HTMX) et renvoie la réponse HTML ou génère un rapport (JasperReports/PDF).

Aspects transverses
Sécurité : authentification + autorisation (sessions, rôle ADMIN/HR/USER).
Reporting : JasperReports pour export PDF/impression des fiches de paie et rapports.
Build / déploiement : Gradle produit .war, déployé sur Tomcat (servlet container).
Tests : JUnit pour tests unitaires.

Diagramme simplifié (conceptuel)
Client → Tomcat → Servlet (Controller) → Service/DAO → Hibernate/JDBC → Base de données
↳ Vue (JSP/HTML/HTMX/JS) ← réponse

6) Spring Boot

Après une première version du projet développée en Jakarta EE avec Servlets, l'application a été reconstruite une seconde fois à l'aide du framework Spring Boot, afin d'améliorer la maintenabilité du code, la modularité, ainsi que la rapidité de développement.

Spring Boot nous a permis d'adopter une architecture en couches plus claire (Controller → Service → Repository → Model) grâce à l'injection de dépendances Spring. La persistance des données est désormais assurée via Spring Data JPA, ce qui a éliminé la nécessité de créer manuellement des DAO et a simplifié les requêtes grâce à JpaRepository.

Contrairement à Jakarta EE, Spring Boot embarque son propre serveur Tomcat, ce qui nous a permis d'exécuter le projet comme une application standalone, sans déploiement manuel du .war. Cette seconde version est plus évolutive, plus structurée, et plus simple à tester et à maintenir.

Le Spring Boot adopte une architecture proche du MVC (Modèle Vue Controller) avec :

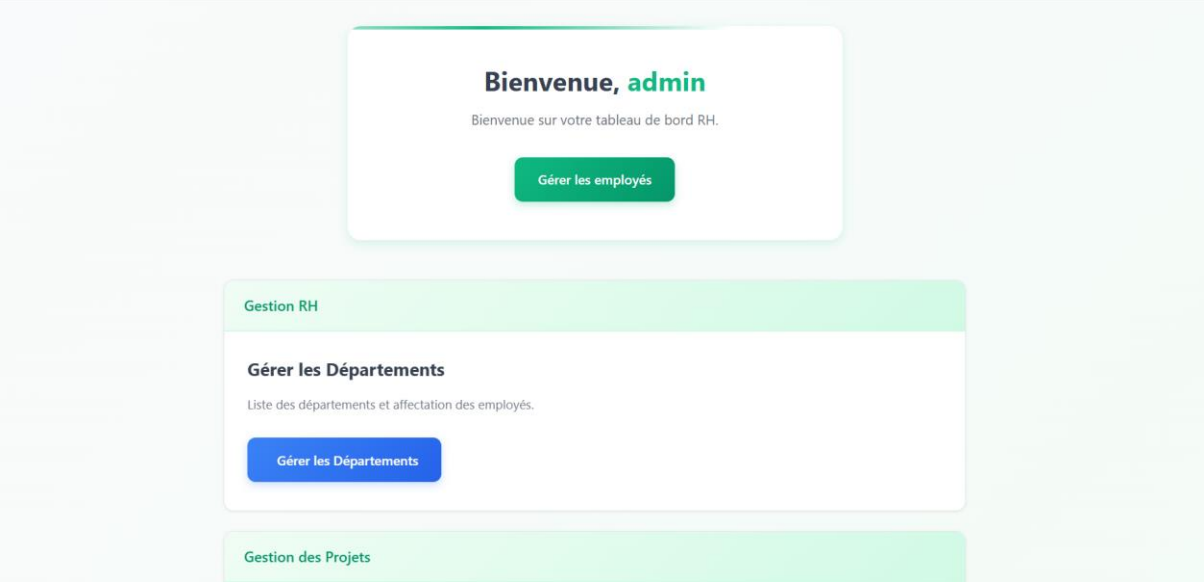
- **Domain :** C'est la couche qui va contenir toutes les entités, qui vont ensuite être mapper dans la base de donnée par le biais de Hibernate.
- **Repository :** Dans cette couche se trouve tous les prototypes de méthodes à utiliser dans les services et liées aux entités, comme des méthodes CRUD ou des méthodes de filtrage dans la base de données.
- **Service :** Ici se trouve toute la logique des méthodes explicitées dans le Repository, regroupées dans une classe Service, centralisant la logique métier.
- **Security :** La couche va gérer tout ce qui touche à la sécurité, comme par exemple les Authorities, le login, le hachage de mot de passe, etc.
- **Controller** : C'est là que sont configurés les endpoints, en appelant le service et en liant tout ça à une requête HTTP pour rendre les méthodes accessibles.

On a ensuite utilisé Thymeleaf, option native de Spring Boot pour pouvoir injecter dans le front des objets Spring Boot, et faciliter l'utilisation des endpoints HTTP. Les fichiers html se retrouvent donc dans le /resources/templates.
Les fichiers dit "statique" de configurations CSS ou JavaScript se retrouvent dans le dossier /resources/static, accessible directement en local depuis localhost:8080/css/… ou localhost:8080/js/…

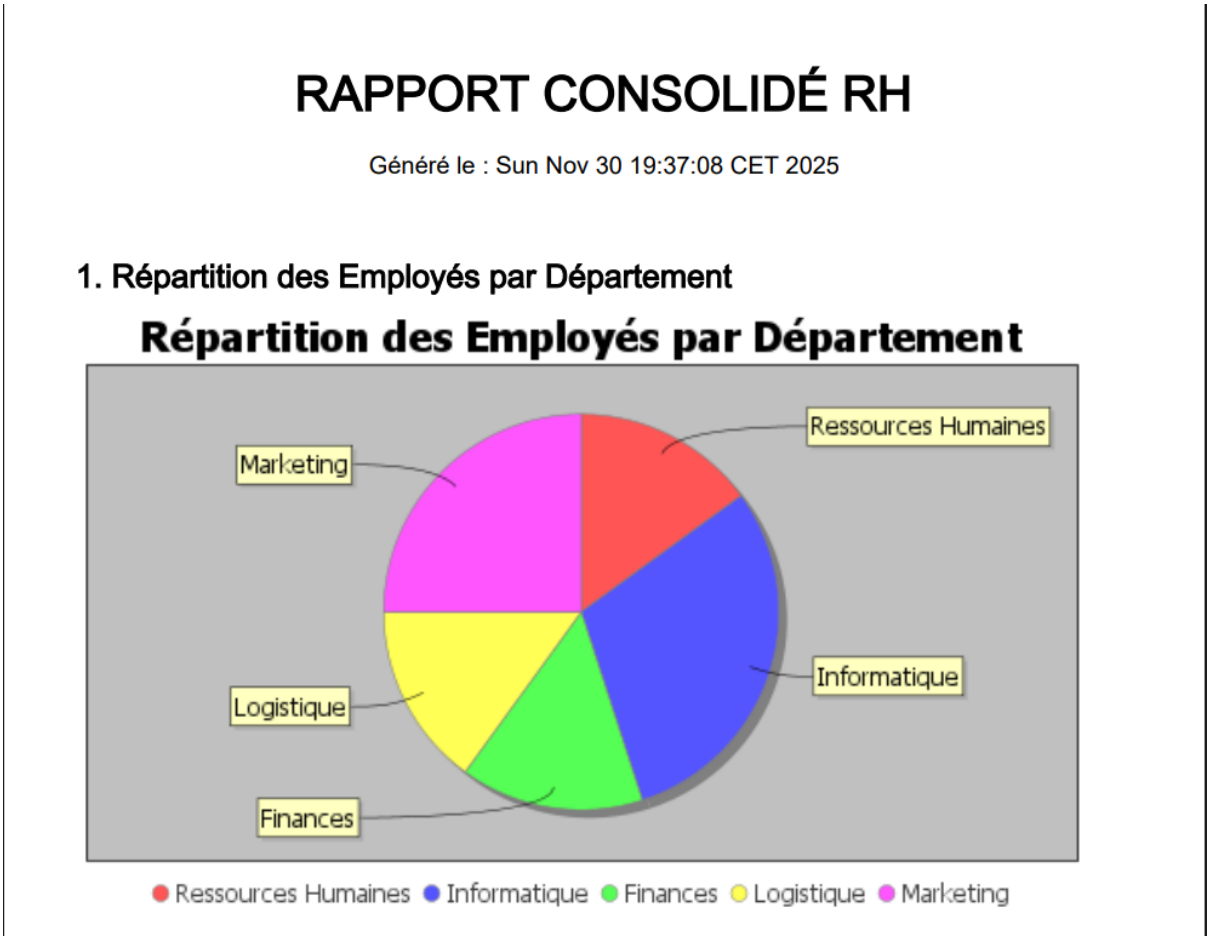
Finalement, les attributs de configurations comme la base de données ou l'e-mail utilisé pour la création de compte sont enregistrés dans le fichier application.yml présent à la racine du dossier resources.

7) Captures d’écran du projet final

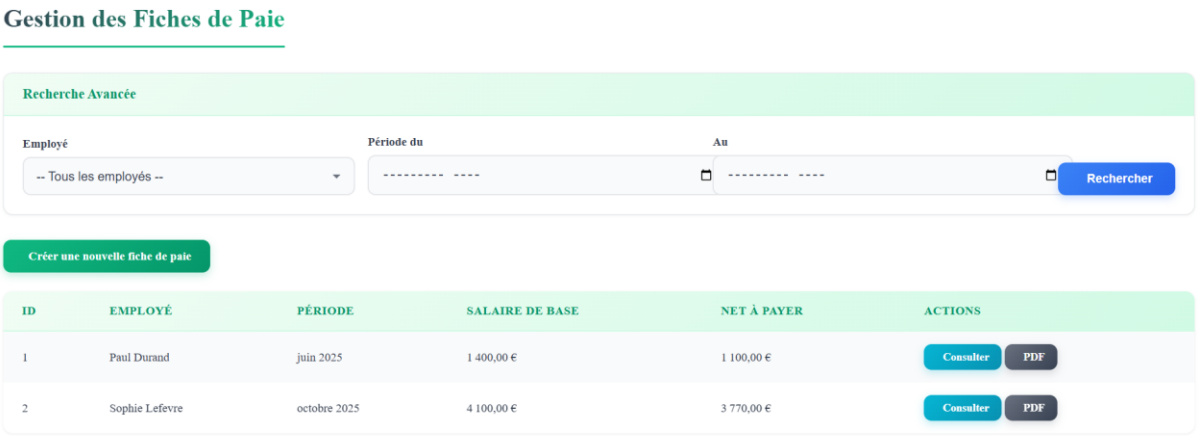
Page d’accueil



Statistiques

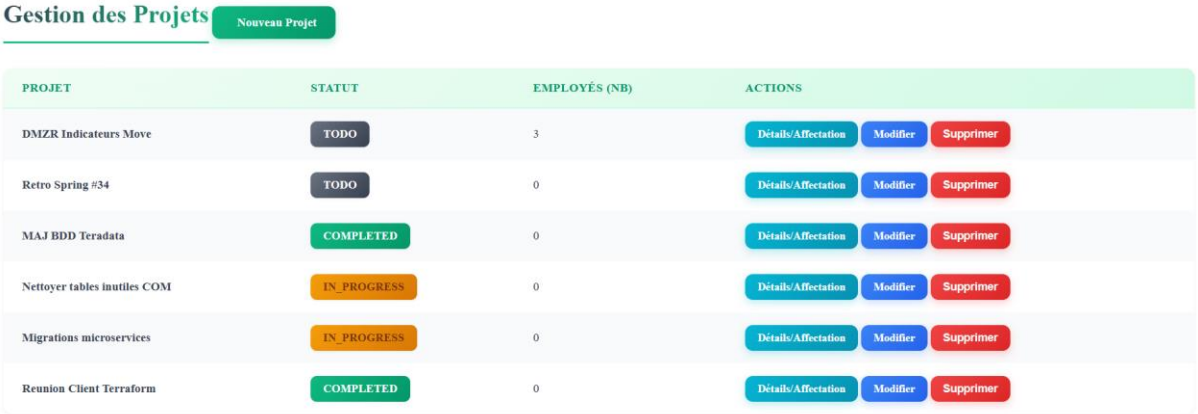


Gestion des Fiches de Paie



Création d’un nouveau compte

Gestion des Projets



8) Conclusion

Ce projet nous a permis de mettre en œuvre une application complète de gestion RH, incluant employés, projets, départements et fiches de paie.

Nous avons d'abord réalisé une version fonctionnelle en Jakarta EE, puis une seconde version modernisée en Spring Boot, ce qui nous a fait découvrir Spring MVC, l'injection de dépendances, Spring Data JPA, et une architecture plus modulaire et maintenable.