Java Enterprise édition ING2 GSI

RAPPORT PROJET : RHxManager

CY TECH 2025-2026

Maxime D. Lucas J. Pierre G. Armand P. Gabriel B.

29/11/2025

Table des Matières:

1. Introduction et Contexte

Dans le cadre de la modernisation des systèmes d'information d'entreprise et de la digitalisation des processus administratifs, le projet **RHxManager** a pour objectif la mise en œuvre d'une solution web robuste basée sur les standards **Jakarta EE**.

Ce projet vise à répondre à une problématique métier concrète : la centralisation et l'automatisation de la gestion des ressources humaines, incluant le suivi des carrières, l'organisation départementale, l'affectation aux projets et le traitement de la paie. Ce rapport détaille la démarche de conception, l'architecture technique retenue ainsi que les choix d'implémentation opérés pour garantir la scalabilité, la sécurité et la maintenabilité de l'application.

2. Analyse et Conception

Besoins Fonctionnels :

Gestion des employés :

L'application permet de :

- Créer un employé avec ses informations personnelles et professionnelles, ainsi que ses rôles.
- Modifier les données d'un employé (poste, salaire, grade, projets associés).
- Supprimer un employé tout en respectant les règles d'intégrité référentielle.
- Lister l'ensemble des employés sous forme de tableau.
- Rechercher un employé par nom, prénom, poste, grade ou identifiant.
- Affecter un employé à un département et à un ou plusieurs projets.

Gestion des départements :

Ce module assure la structuration de l'organisation :

- création de nouveaux départements,
- consultation de la liste des départements,
- affectation d'employés à un département,
- visualisation des membres d'un département et de son manager.

Gestion des projets :

Le module projet permet de :

- créer, modifier ou supprimer un projet ;
- assigner des employés à un projet ;
- suivre l'état d'avancement (en cours, terminé, annulé).

Gestion des fiches de paie :

La génération de la paie est automatisée :

- création d'une fiche de paie pour un employé et une période ;
- calcul automatique du net à payer ;
- consultation de l'historique des fiches ;
- recherche par employé ou par période ;
- affichage d'une fiche imprimable.

Authentification et autorisation :

- authentification via un formulaire de connexion,
- vérification du mot de passe haché,
- gestion des rôles utilisateur (ADMIN, EMPLOYE),
- filtrage des accès via un filtre de sécurité bloquant les pages sensibles.

Reporting et statistiques :

- nombre d'employés par département ;
- nombre d'employés par projet ;
- état des projets ;
- répartition des employés par grade.

Ces informations alimentent le tableau de bord.

Besoins Non Fonctionnels :

Architecture et modularité :

L'application adopte une architecture MVC, permettant une séparation claire des responsabilités :

- **Modèle** : entités JPA, DAO et logique métier ;
- **Vue** : pages JSP associées à la bibliothèque JSTL ;
- **Contrôleur** : servlets gérant les interactions et le routage.

Intégrité et cohérence des données :

- une validation côté serveur pour éviter les erreurs de saisie et les données invalides ;
- une gestion des relations assurée par Hibernate, réduisant les risques d'incohérence entre les objets Java et les tables SQL.

Sécurité :

- une authentification obligatoire avant l'accès aux fonctionnalités ;
- un contrôle d'accès basé sur les rôles utilisateurs (ADMIN, EMPLOYE) ;
- un stockage sécurisé des mots de passe (hachage), empêchant la lecture en clair.

Ergonomie :

L'interface utilisateur est conçue pour être simple et intuitive :

- des pages organisées par modules (employés, projets, départements, paie) ;
- des formulaires clairs et faciles à utiliser ;
- des messages d'erreur ou de validation explicites pour guider l'utilisateur.

Structure générale de la base de données :

La structure de la base de données reprend directement le modèle conceptuel défini dans l'application et correspond parfaitement aux entités JPA qui seront présentées dans la troisième partie. Chaque table reflète une entité métier (Employé, Département, Projet, Rôle, Fiche de paie).

Contraintes :

- **Contraintes d'unicité :**
  Le nom du département et le nom du rôle doivent être uniques pour éviter les doublons.
  De plus, le couple *(employe_id, month, year)* est unique afin d'empêcher la création de plusieurs fiches de paie pour un même employé sur la même période.

- **Clés étrangères :**
  Certaines relations utilisent une suppression en cascade (comme pour les rôles ou les fiches de paie), ce qui garantit qu'aucune donnée orpheline ne reste en base.

D'autres relations sensibles, comme le chef d'un département ou d'un projet, utilisent une mise à *NULL* lorsque l'employé référencé est supprimé. Cela permet de conserver la structure sans supprimer inutilement d'autres informations.

- **Tables de jonction Many-to-Many :**
  Les associations entre employés et projets, ou entre employés et rôles, sont gérées via des tables de jonction à clés composites. Ce mécanisme permet de représenter correctement les relations multiples.

Ces différents mécanismes assurent une base de données propre, cohérente et fidèle aux règles métier du système.

Architecture et Conception Spécifique (Version sans Spring Boot)

La phase de conception a suivi une approche **Orientée Objet**, formalisée par le langage de modélisation unifié (UML). L'objectif était de traduire les règles métiers complexes en un modèle de données cohérent et normalisé.
Le système d'information repose sur cinq entités pivots, dont la persistance est assurée par le standard JPA (Java Persistence API) :

- **L'Employé (`Employe`)**
  - *Description :* Acteur central du système, cette entité agrège les données civiles, contractuelles (Salaire, Grade) et les informations d'authentification.
  - *Sécurité :* Les données sensibles, telles que le mot de passe, ne sont jamais stockées en clair (hachage cryptographique).
- **Le Département (`Department`)**
  - *Structure :* Représente l'unité hiérarchique de l'organisation.
  - *Règle de gestion :* La relation hiérarchique est modélisée par une association 1-1 spécifique pour le Manager (Chef de département), distincte de l'association 1-N liant les membres du département.
- **Le Projet (`Project`)**
  - *Pilotage :* Cette entité permet le suivi opérationnel des missions.
  - *Cycle de vie :* L'intégrité de l'état du projet est garantie par une énumération stricte (ProjectState : WORKED_ON, FINISHED, CANCELED), interdisant les états incohérents en base de données.
- **La Fiche de Paie (`Payslip`)**
  - *Intégrité des données :* Une contrainte d'unicité composite (UniqueConstraint) a été implémentée sur le trio {employe_id, month, year}. Cette contrainte technique prévient physiquement les erreurs de doublons de paie.

    - *Logique métier :* Le calcul du salaire net (salaire + bonus - déductions) est encapsulé côté serveur avant la persistance.
- **Le Rôle (`Role`)**
    - *Sécurité :* Implémentation d'un contrôle d'accès basé sur les rôles (**RBAC**). Cette abstraction permet de découpler l'identité de l'utilisateur de ses privilèges (Admin, Chef de Projet... vs Employé).

Architecture et Conception Spécifique (Version Spring Boot)

Alors que la structure des données reste inchangée, la migration vers le framework **Spring Boot** introduit un changement de paradigme architectural majeur. La conception ne repose plus sur une instanciation manuelle des classes, mais sur le principe de l'**Inversion de Contrôle (IoC)**.

L'Inversion de Contrôle (IoC) et Injection de Dépendances (DI) :

Contrairement à la version Jakarta EE standard où le développeur gère le cycle de vie des objets (via des Singletons comme `JpaUtil`), Spring Boot utilise un **Conteneur IoC** (ApplicationContext).

- **Conception :** Les composants de l'application (Repositories, Services, Contrôleurs) sont définis comme des "Beans" gérés par le conteneur.
- **Mécanisme :** Les dépendances sont injectées dynamiquement (via l'annotation `@Autowired`), ce qui découple fortement les classes et facilite la testabilité.

Évolution du Pattern DAO vers le Pattern Repository :

La couche d'accès aux données a été repensée pour utiliser **Spring Data JPA**, remplaçant le modèle DAO traditionnel.

- **Abstraction :** Au lieu d'écrire manuellement les requêtes CRUD dans des classes d'implémentation (`GenericDao`), nous définissons des **Interfaces** étendant `JpaRepository`.
- **Avantage :** Spring génère automatiquement l'implémentation des méthodes standard (`save`, `findById`, `delete`) et permet de définir des requêtes complexes par simple déclaration de signature de méthode (ex: `findByUsername`).

Architecture MVC avec Spring Web et Thymeleaf :

La gestion des requêtes HTTP et de l'affichage évolue :

- **Contrôleur (`@Controller`) :** Remplace les Servlets. Une classe Java annotée gère plusieurs routes (endpoints) et prépare le modèle de données (`Model`).

- **Vue (Thymeleaf) :** Remplace les JSP. Thymeleaf est un moteur de template moderne qui permet un rendu serveur (Server-Side Rendering) tout en maintenant des fichiers HTML valides et lisibles, facilitant la collaboration avec les designers web.

Sécurité Déclarative (Spring Security):

La sécurité n'est plus gérée par un filtre Servlet manuel (`SecurityFilter`), mais par une chaîne de filtres configurable (`SecurityFilterChain`).

- **Authentification :** Utilisation d'un `DaoAuthenticationProvider` couplé à un service utilisateur (`JpaUserDetailsService`) pour charger les utilisateurs depuis la base de données.
- **Autorisation :** Les règles d'accès aux URL (ex: `/admin/**`) sont définies de manière déclarative dans une classe de configuration (`SecurityConfig`), renforçant la robustesse de l'application.

3. Architecture Technique et Implémentation

L'application repose sur une architecture **N-tiers** stricte, respectant le patron de conception **MVC (Modèle-Vue-Contrôleur)**. Ce choix garantit une séparation claire des responsabilités, facilitant la maintenance et l'évolution future du code.

La Couche Modèle (Persistance et Données)

La couche de données est isolée dans le package `com.rhxmanager.model`.

- **Technologie :** Hibernate Core 7.1.3 (Implémentation de référence de JPA).
- **Mapping ORM :** L'utilisation des annotations JPA (`@Entity`, `@Table`, `@OneToMany`) permet une correspondance transparente entre le monde objet (Java) et le monde relationnel (SQL), éliminant la nécessité d'écrire des requêtes SQL manuelles pour la définition du schéma (DDL).

La Couche Contrôleur (Logique Métier)

Située dans `com.rhxmanager.servlets`, cette couche orchestre les interactions.

- **Cycle de vie :** Les Servlets interceptent les requêtes HTTP, valident les entrées utilisateurs, invoquent la couche de service/DAO, et sélectionnent la vue appropriée.
- **Mécanisme :** Utilisation intensive du RequestDispatcher pour le transfert de données (DTO/POJO) vers la vue via les attributs de requête.

La Couche Vue (Interface Utilisateur)

L'interface, située dans `/WEB-INF/jsp`, est générée côté serveur.

- **Technologie :** JSP (JavaServer Pages) couplée à la JSTL (Jakarta Standard Tag Library).
- **Principe :** Aucune logique métier Java ("Scriptlet") n'est présente dans les vues. L'affichage repose exclusivement sur l'**Expression Language (EL)** pour garantir la lisibilité et prévenir les failles de type XSS.
- **Expérience Utilisateur :** Design responsive assuré par CSS3 (Flexbox/Grid), garantissant l'accessibilité sur différents terminaux.

Patrons de Conception (Design Patterns)

Pour assurer la robustesse du code, plusieurs patrons de conception ont été appliqués :

- **DAO (Data Access Object) :** Découple la logique métier de l'accès aux données. Une classe `GenericDao<T, ID>` centralise les opérations CRUD communes, favorisant la réutilisabilité du code (DRY - Don't Repeat Yourself).
- **Singleton :** Utilisé via la classe `JpaUtil` pour garantir une instance unique de l'EntityManagerFactory, optimisant ainsi la consommation de ressources système.
- **Front Controller (Adapté) :** Le filtre de sécurité agit comme un point d'entrée unique pour la vérification des droits avant d'atteindre les contrôleurs.

4. Outils et Environnement de Développement

Le projet s'inscrit dans un écosystème technologique moderne, conforme aux spécifications **Jakarta EE 10**.

Chaîne de Production (Build & Dependency)

- **Apache Maven :** Outil central pour la gestion du cycle de vie du projet. Il assure la résolution transitive des dépendances, la compilation, l'exécution des tests unitaires et le packaging de l'artefact final (.war).

Environnement d'Exécution (Runtime)

- **Serveur d'Application :** Apache Tomcat 10.1.
  - *Justification :* Ce choix est imposé par la migration de l'espace de noms `javax.*` vers `jakarta.*` inhérente à Jakarta EE 9/10.
- **SGBD :** MySQL 8.0.
  - *Connecteur :* MySQL Connector/J 9.4.0 pour la liaison JDBC.

Sécurité et Qualité Logicielle

- **Authentification :** Mise en œuvre d'un filtre Servlet (`SecurityFilter`) interceptant toutes les requêtes pour valider la session utilisateur.
- **Cryptographie :** Sécurisation des mots de passe via l'algorithme de dérivation de clé **PBKDF2WithHmacSHA256**. L'ajout d'un sel ("salt") unique par utilisateur protège contre les attaques par Rainbow Tables.

5. Conclusion et Perspectives

Le projet **RHxManager** a permis de mettre en œuvre une solution complète de gestion RH, respectant les contraintes architecturales J2EE (MVC, DAO, JPA). L'application assure avec succès la gestion du cycle de vie des employés, des départements, des projets et la génération automatisée des fiches de paie.

Perspectives d'évolution

Dans une optique d'amélioration continue ("Future Works"), plusieurs axes pourraient être explorés :

1. **API RESTful :** Exposition des services via JAX-RS pour permettre le développement d'une application mobile ou d'un frontend SPA (React/Angular).
2. **Reporting Avancé :** Intégration d'une bibliothèque comme JasperReports pour la génération de fiches de paie au format PDF certifié.
3. **Tests d'Intégration :** Mise en place de tests automatisés (JUnit 5, Mockito) pour renforcer la couverture de code et la fiabilité lors des régressions.