CY TECH

ANNÉE 2025 – 2026

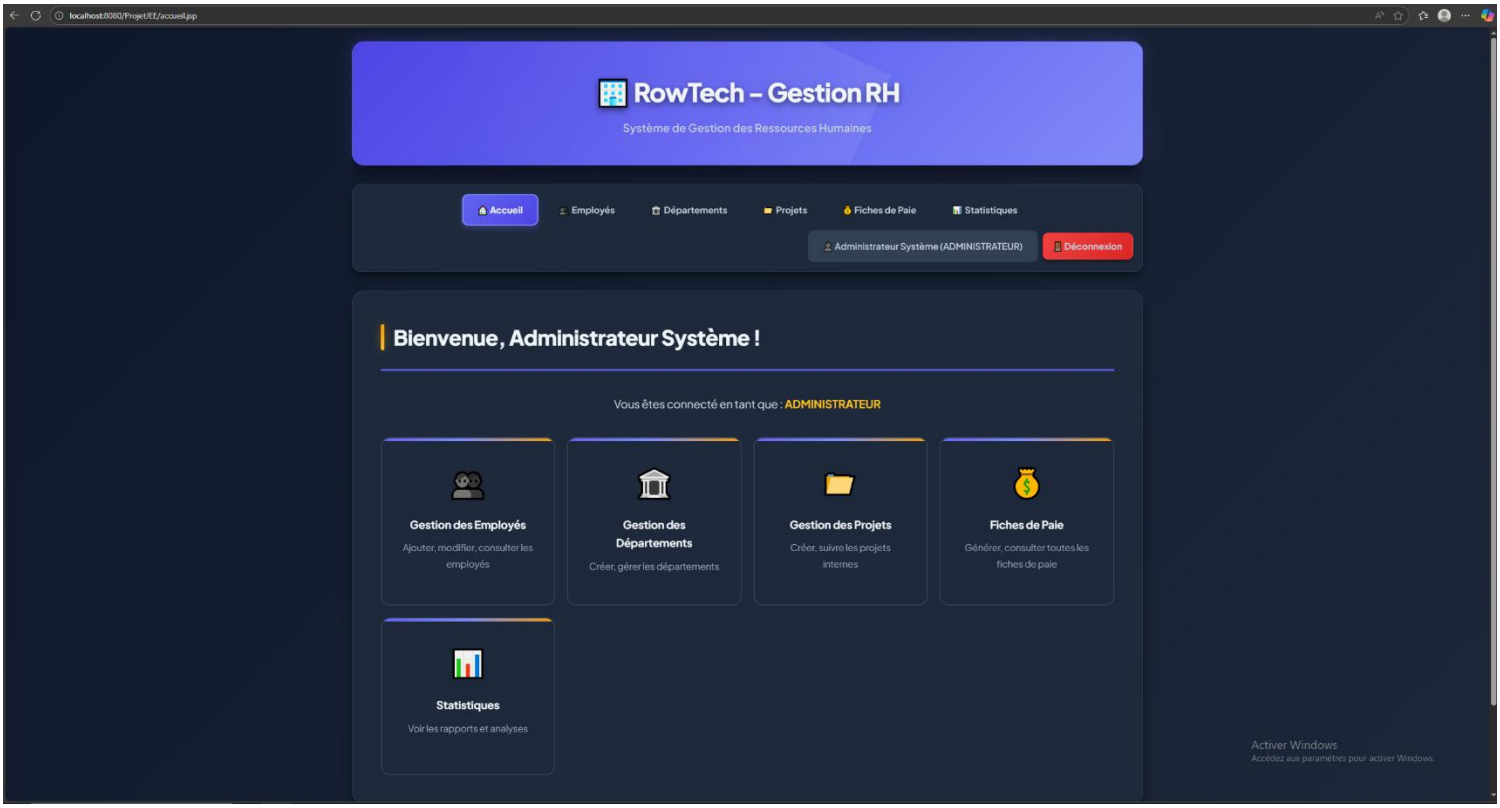
ING2 GSI

JEE

Mohamed HADACHE

Rapport du Projet ROW TECH

Riyad ZALEGH, Yassir EL JARRARI, David LE MER, Ahmed, Rayane SAIGHI, Vincent DELB



Plan détaillé du rapport :

1.Introduction

1.1 Contexte et Objectifs

- **Présentation de l'entreprise (RowTech) et du contexte de digitalisation RH/Projet.**
- **Synthèse des objectifs principaux : CRUD complet, Calcul de la Paie, Gestion des Rôles, Reporting.**

1.2 Technologies Utilisées

- **Environnement technique : Java/JEE, Tomcat, Maven.**
- **Architecture : Modèle MVC (Modèle-Vue-Contrôleur).**
- **Persistance : Hibernate et MySQL.**

2. Analyse et Conception

2.1 Analyse des Besoins et Rôles

- **Détail des fonctionnalités par module (Employés, Départements, Projets, Paie).**
- **Définition des acteurs et des rôles (ADMINISTRATEUR, CHEF_DEPARTEMENT, CHEF_PROJET, EMPLOYE) et des règles d'autorisation associées.**

2.2 Structure des Données et Relations

- **Description des Entités et Associations : Description des entités Employe, Departement, Projet, FicheDePaie, User.**
    - **Explication des relations (Many-to-One, Many-to-Many via table de jointure) et de la manière dont les rôles de Chef sont gérés dans la structure des données.**

2.3 Modélisation du Processus de la Paie

- **Formule de Calcul : Explication de la logique de calcul du salaire net (Salaire Brut, Cotisations, Déductions).**
- **Composantes : Détails sur l'implémentation du calcul et les taux sociaux simulés (basé sur la logique de FicheDePaie.java).**

3. Implémentation en J2EE / Architecture MVC

3.1 Architecture MVC et Flux Applicatif

- **Flux de Requête : Description textuelle du cheminement de la requête (Filtre de Sécurité $\rightarrow$ Contrôleur $\rightarrow$ Couche d'Accès aux Données $\rightarrow$ Affichage des Vues).**
- **Contrôleurs et Dispatching : Rôle des Servlets (ex: EmployeServlet, FicheDePaieServlet) dans le traitement des requêtes et la redirection.**
- **Validation : Implémentation des mécanismes de validation côté serveur et client (JavaScript).**

3.2 Couche de Persistance : Hibernate et DAO

- **Configuration : Présentation du fichier hibernate.cfg.xml et de HibernateUtil.java pour l'accès à la base de données MySQL.**
- **Logique DAO : Implémentation des classes DAO et gestion des transactions pour les opérations CRUD.**
- **Cohérence des Données : Mécanismes pour garantir l'intégrité des relations complexes (Many-to-Many).**

3.3 Mise en Œuvre des Fonctionnalités Spécifiques

- **Sécurité et Rôles : Fonctionnement du filtre d'authentification (AuthFilter.java) et du hachage de mots de passe (BCrypt).**
- **Rapports et Statistiques : Description de la collecte des données agrégées (StatistiquesDAO) et de leur présentation.**
- **Génération PDF : Description de l'utilisation de iText 7 pour l'export des bulletins de paie.**

4. Implémentation Alternative Spring Boot

4.1 Justification de la Migration et Avantages

- **Comparaison des gains (simplification de la configuration, serveur embarqué, Spring Data JPA) par rapport à l'approche JEE/Servlet.**

4.2 Architecture Spring Boot Proposée

- **Nouvelle Structure : Description des couches (Controller $\rightarrow$ Service $\rightarrow$ Repository).**
- **Mise en Parallèle : Tableau de conversion des composants JEE/Servlet vers Spring Boot.**

5. Outils, Déploiement et Conclusion

5.1 Outils de Développement et Déploiement

- **Fichier séparé : Mention de l'exigence de fournir un fichier décrivant les outils.**
- **Outils Clés : Java 17, Maven, Tomcat, MySQL, Git/GitHub.**

5.2 Bilan Critique du Projet

- **Forces de l'implémentation et respect des contraintes.**
- **Points de vigilance et difficultés techniques.**
- **Perspectives d'amélioration.**

Conclusion

- **Résumé de l'atteinte des objectifs initiaux.**

1. Introduction

1.1 Contexte et Objectifs

Dans le cadre du cours de J2EE, le projet RowTech vise à développer une solution d'entreprise pour la gestion intégrée des processus internes. Le contexte de ce projet est la digitalisation et l'optimisation des services RH et de suivi de projet d'une société fictive. L'implémentation a été réalisée en respectant les spécifications d'une architecture multicouche et les contraintes techniques imposées par l'environnement JEE.

Les objectifs fonctionnels principaux atteints par cette application sont :

1. Gestion des Entités de Base (CRUD) : Permettre l'ajout, la modification, la consultation et la suppression des fiches Employés, des Départements, des Projets et des Fiches de Paie.
2. Calcul Salarial Automatique : Mettre en œuvre une logique de calcul robuste pour déterminer le salaire net à payer, intégrant le salaire de base, les primes, les heures supplémentaires, les cotisations sociales simulées et les déductions pour absences.
3. Sécurité et Autorisation : Établir un système d'authentification sécurisé (via hachage de mots de passe) et une gestion des rôles (Administrateur, Chefs de Département, Chefs de Projet, Employé) pour contrôler l'accès aux différentes fonctionnalités.
4. Reporting et Rapports : Générer des statistiques agrégées sur la structure de l'entreprise et permettre l'export des fiches de paie individuelles au format PDF.

1.2 Architecture et Technologies Utilisées

L'application a été développée en adoptant l'architecture classique Modèle-Vue-Contrôleur (MVC), en utilisant les technologies J2EE :

A. Couches Architecturales

- Contrôleur (Servlets Jakarta) : Les Servlets sont le point d'entrée de l'application. Elles gèrent le flux de la requête, les validations initiales, le contrôle d'accès (AuthFilter.java) et l'orchestration des appels aux services de données avant de rediriger vers la vue appropriée (Dispatcher).
- Modèle (Entités JPA, Logique Métier, DAO) : Cette couche contient les entités Java (POJO) mappées à la base de données (JPA), la logique métier complexe (telle que le calcul de la paie) et les classes DAO (Data Access Object) qui encapsulent les interactions avec la persistance.
- Vue (JSP) : Les JavaServer Pages (JSP) sont utilisées pour générer dynamiquement l'interface utilisateur (HTML, CSS custom, JavaScript).

B. Piliers Technologiques

Catégorie	Technologie	Rôle
Back-end	Java (JDK17)	Langage d'exécution du Contrôleur et du Modèle
Persistence	Hibernate ORM	Framework de mappage Objet-Relationnel (JPA), configuré via hibernate.cfg.xml
Base de donn	MySQL	SGBD relationnel utilisé pour la persistance des données.
Build	Maven	Outil de gestion des dépendances (Hibernate, iText, BCrypt) et de compilation.
Sécurité	Bcrypt	Algorithme de hachage des mots de passe pour l'authentification.
Rapports	iText 7	Librairie Java utilisée pour la génération dynamique de documents PDF (fiche de paie etc)

2. Analyse et Conception

2.1 Analyse des Besoins et Rôles

- Gestion des Employés :
    - Fonctionnalités CRUD (Création, Lecture, Mise à jour, Suppression) complètes des fiches employés.
    - Fonctionnalités de recherche avancée par nom, prénom, matricule ou département, et de filtrage par grade et poste.
    - Capacité d'affecter un employé à un département et à plusieurs projets.
- Gestion des Départements :
    - Fonctionnalités de base pour l'ajout et le listage des départements.
    - Attribution et modification d'un Chef de Département responsable.
    - Visualisation de la liste des employés membres de chaque département.
- Gestion des Projets :
    - Fonctionnalités CRUD pour les projets avec suivi de l'état d'avancement (EN_COURS, TERMINE, ANNULE).
    - Attribution d'un Chef de Projet responsable du suivi.
    - Gestion de l'équipe : affectation et retrait des employés du projet.
- Gestion des Fiches de Paie :
    - Création d'une fiche pour une période donnée (mois et année).
    - Calcul automatique du net à payer basé sur le brut et les déductions.
    - Consultation et export en PDF (Exigence clé).
- Gestion des Rôles (Sécurité) :
    - Le rôle ADMINISTRATEUR dispose d'un accès illimité à toutes les fonctions de gestion et de consultation.
    - Les rôles CHEF_DEPARTEMENT et CHEF_PROJET disposent de droits étendus limités à leur périmètre de responsabilité (leur département ou leurs projets) et peuvent consulter les fiches de paie de leur équipe.

    - Le rôle EMPLOYE (et tous les utilisateurs connectés) peut consulter ses propres informations, sa fiche de paie et ses projets.

2.2 Structure des Données et Relations

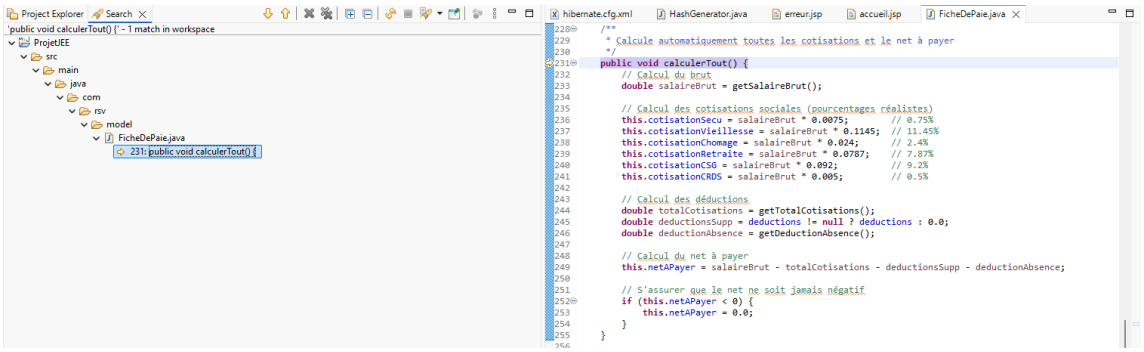
La structure relationnelle du système est définie par les entités suivantes, qui assurent la Cohérence des Données :

- Entité Employe : Représente le noyau RH. Elle est liée à un département (relation 1 vers N) et à plusieurs projets (relation N vers N).
- Entité Departement : La relation d'un département à son chef est modélisée par un lien direct optionnel vers l'entité Employe.
- Entité Projet : Similaire au Département, le projet a un lien direct optionnel vers l'Employe qui est son chef.
- Relation Employe-Projet : L'affectation d'un employé à un ou plusieurs projets est gérée par une relation de type Many-to-Many.
- Entité FicheDePaie : L'intégrité est garantie par une clé étrangère obligatoire vers l'Employe concerné, et par l'unicité de la combinaison employe-mois-année.
- Entité User : Utilisée pour l'authentification, cette entité est associée à un Employe et stocke le username, le password haché et le role de l'utilisateur dans le système.

2.3 Modélisation du Processus de la Paie

Le modèle de paie est implémenté dans la classe FicheDePaie.java via la méthode calculerTout(), garantissant que le calcul est effectué de manière cohérente à chaque création ou modification de fiche.

- Calcul du Salaire Brut : Le brut est la somme du salaire de base de l'employé, des primes exceptionnelles et de la valorisation des heures supplémentaires saisies.
- Calcul des Cotisations Sociales :
    - Six types de cotisations (Sécu, Vieillesse, Chômage, Retraite Complémentaire, CSG, CRDS) sont calculés en appliquant des taux simulés sur le Salaire Brut.
    - Le total des cotisations est la somme de ces montants déduite du brut.
- Déductions :
    - Les déductions supplémentaires (mutuelle, acomptes, etc.) sont prises en compte comme un montant forfaitaire.
    - L'impact des jours d'absence est calculé en déduisant une fraction du salaire de base proportionnelle au nombre de jours saisis (Salaire Base / 30 jours).
- Résultat Final : Le Net à Payer est la valeur finale après toutes les déductions. Ce montant est stocké en base de données avec les détails des cotisations.

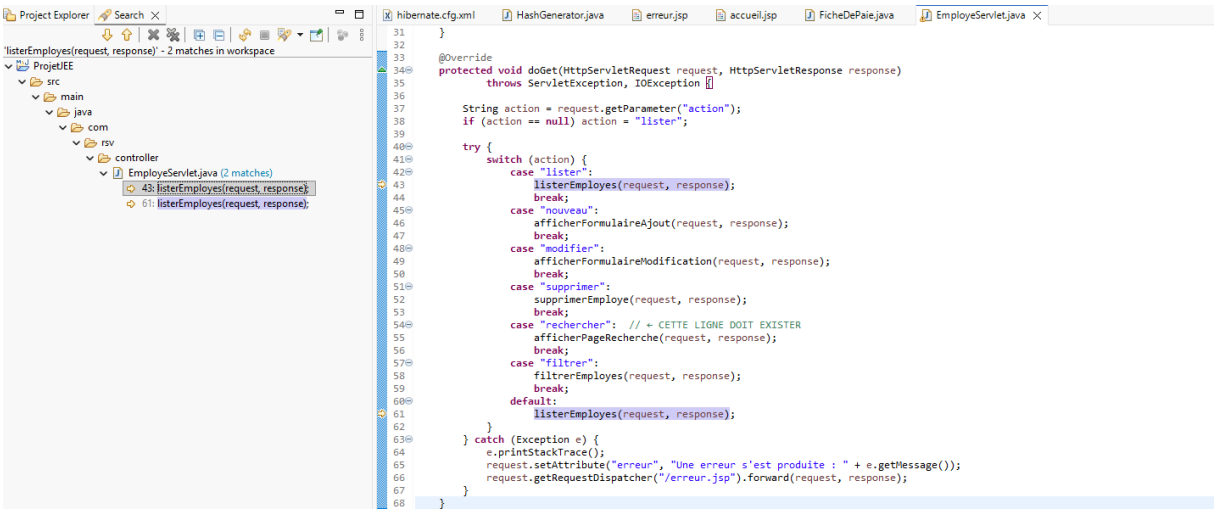


3. Implémentation en J2EE / Architecture MVC

3.1 Architecture MVC et Flux Applicatif

L'implémentation respecte rigoureusement le patron **Modèle-Vue-Contrôleur (MVC)**, assurant la séparation des préoccupations :

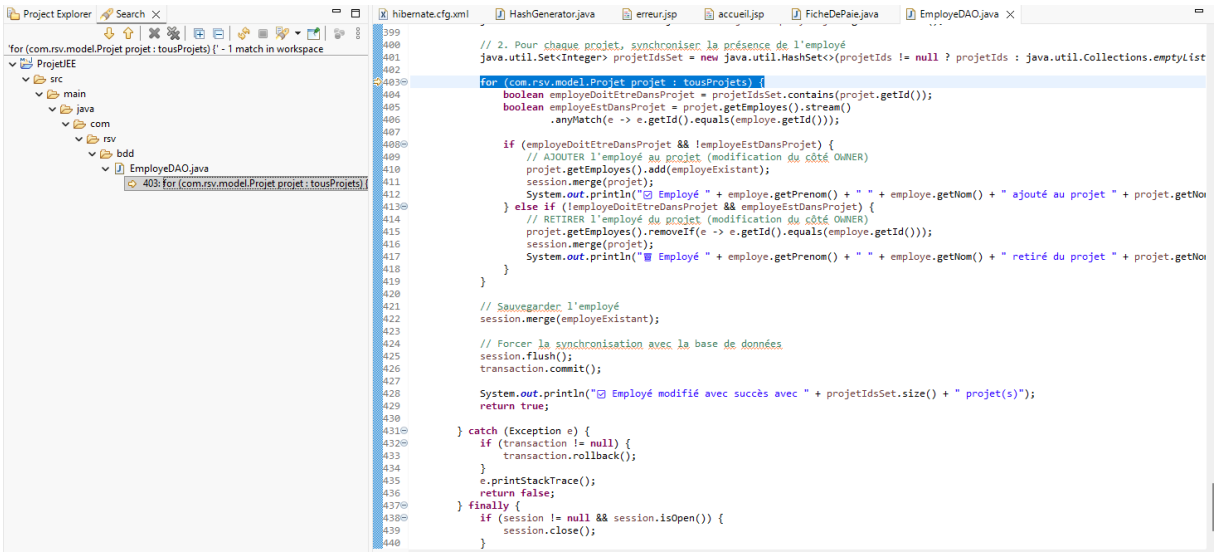
- **Point d'Entrée et Filtrage :**
    - Toutes les requêtes HTTP entrent dans le système et passent par le AuthFilter.java (filtre d'authentification).
    - Ce filtre vérifie l'existence d'une session valide et du userId. Si l'utilisateur n'est pas connecté et que la ressource demandée n'est pas publique (/auth.jsp, /inscription, /css, etc.), l'accès est immédiatement refusé et l'utilisateur est redirigé vers la page de connexion.
- **Contrôleurs (Servlets) :**
    - Chaque fonctionnalité majeure (Employés, Départements, Projets, Fiches de Paie) est gérée par une Servlet dédiée (ex: EmployeServlet, ServletProjet).
    - La Servlet analyse le paramètre action (doGet ou doPost) et délègue la tâche à une méthode privée appropriée (ex: listerEmployes, ajouterEmploye).
    - Le contrôleur est responsable de la **Validation des Données** (via ValidationUtil.java) avant l'appel à la couche de persistance.
    - Après traitement, le Contrôleur utilise l'objet RequestDispatcher pour transmettre les données au Modèle et effectuer un forward vers la JSP (Vue).
- **Validation :**
    - La validation est implémentée à la fois côté client (via validation.js pour une meilleure expérience utilisateur) et, de manière obligatoire, **côté serveur** (via ValidationUtil.java dans la Servlet) pour des raisons de sécurité et de cohérence des données.



3.2 Couche de Persistance : Hibernate et DAO

L'accès et la manipulation des données sont gérés par la couche DAO (Data Access Object) utilisant le framework **Hibernate** (implémentation de JPA).

- **Configuration Hibernate :**
  - Le fichier hibernate.cfg.xml centralise la configuration de la connexion à la base de données (Driver MySQL, URL projetjeeapp, identifiants) et mappe toutes les entités (Employe, Departement, etc.) à la session factory.
  - La propriété hbm2ddl.auto est réglée sur update pour gérer automatiquement l'évolution du schéma de la base de données.
- **Utilitaire de Session (HibernateUtil.java) :**
  - La classe statique HibernateUtil.java est responsable de la création unique de la SessionFactory lors du démarrage de l'application, assurant la gestion des ressources.
  - Chaque classe DAO utilise HibernateUtil.getSessionFactory().openSession() pour obtenir une session de travail, garantissant l'isolation des transactions.
- **Logique DAO et Transactions :**
  - Les classes DAO (ex: EmployeDAO.java) implémentent les méthodes CRUD et les requêtes spécifiques (recherche, statistiques).
  - Chaque opération de modification (ajouter*, modifier*, supprimer*) est encapsulée dans une **Transaction Hibernate** (début, commit ou rollback en cas d'erreur) afin de garantir l'atomicité et la cohérence des données.
- **Cohérence des Relations Complexes :**
  - Pour la relation **Many-to-Many** (Employe - Projet), le code du DAO (ex: EmployeDAO.modifierEmployeAvecProjets) gère explicitement l'ajout et le retrait des employés des projets en manipulant la collection employes du côté *Owner* (Projet), assurant la bonne mise à jour de la table de jointure employe_projet.



9

3.3 Mise en Œuvre des Fonctionnalités Spécifiques

- **Sécurité et Hachage :**
    - Lors de l'inscription ou de la création d'un utilisateur, le mot de passe est haché via PasswordUtil.hashPassword() (utilisant **BCrypt**) avant d'être stocké en base de données.
    - L'authentification (AuthServlet.java) vérifie la correspondance entre le mot de passe en clair et le hash stocké via PasswordUtil.checkPassword().
- **Rapports et Statistiques :**
    - La classe StatistiquesDAO.java exécute des requêtes agrégées complexes (utilisant COUNT, SUM, AVG, MIN, MAX et GROUP BY dans HQL) pour collecter les totaux, la masse salariale et la répartition des employés.
    - Ces données sont transmises à statistiques.jsp où elles sont présentées sous forme de tableau de bord et affichées graphiquement via **Chart.js** (côté client).
- **Génération PDF des Fiches de Paie :**
    - L'exigence de génération de fiche de paie imprimable est satisfaite par le GeneratePdfServlet.java.
    - Ce servlet utilise la librairie **iText 7** pour créer un document PDF structuré. Il récupère les montants finaux de la fiche depuis l'objet FicheDePaie (qui contient déjà tous les montants de cotisations pré-calculés) et les insère dans le layout du bulletin de paie.

4. Implémentation Alternative Spring Boot

(Cette partie répond à l'exigence 4.3 du cahier des charges : refaire l'application en utilisant Spring Boot.)

4.1 Justification de la Migration et Avantages

La proposition de migration de l'application vers **Spring Boot** répond à l'exigence d'explorer une architecture moderne, tirant parti de ses capacités pour simplifier le développement et le déploiement. Les avantages sont multiples et se manifestent à tous les niveaux du projet :

- **Avantages Techniques Clés :**
    - **Serveur Embarqué (Stand-alone) :** L'intégration native d'un serveur Tomcat ou Jetty élimine le processus fastidieux de création et de déploiement d'un fichier .war sur un serveur d'applications externe. L'application devient directement exécutable.
    - **Auto-configuration :** Spring Boot simplifie drastiquement la configuration. Le framework configure automatiquement la source de données (DataSource) et la couche JPA, réduisant ainsi la dépendance à des fichiers de configuration XML complexes comme web.xml ou le hibernate.cfg.xml.
    - **Gestion des Dépendances :** L'utilisation des *Starters* Maven permet d'importer des ensembles de dépendances cohérents et testés (ex: spring-boot-starter-web, spring-boot-starter-data-jpa), simplifiant la gestion du pom.xml par rapport à l'approche JEE classique.
    - **Accès à Spring Data JPA :** Cette abstraction remplace la nécessité d'écrire des classes DAO entières. Les opérations CRUD sont gérées par des interfaces (Repositories), améliorant la productivité et la lisibilité du code de persistance.
- **Bénéfices Opérationnels et d'Architecture :**
    - Le découplage des composants devient plus net grâce à l'**Injection de Dépendances** native de Spring. Les contrôleurs n'ont plus besoin d'instancier manuellement les objets DAO ou utilitaires (via new), car ces dépendances sont injectées automatiquement par le conteneur Spring (IoC).
    - Le modèle Service est naturellement encouragé par l'architecture Spring, forçant une séparation plus stricte entre la couche de Contrôle (Web) et la Logique Métier (Service), ce qui manquait à l'architecture Servlet/DAO couplée actuelle.

4.2 Architecture Spring Boot Proposée

La refonte architecturale s'articulerait autour d'un modèle multicouche Spring standard, plus robuste et orienté microservices, même pour une application monolithique.

- **Structure de Couches Remplaçante :**
    - **Couche Web (Controllers) :** Les Servlets sont remplacées par des classes **@RestController** (pour les APIs REST) ou **@Controller** (pour les vues JSP ou Thymeleaf/FreeMarker). Ces classes se concentrent uniquement sur la réception des requêtes, la validation légère et la réponse, déléguant toute la logique métier à la couche Service.
    - **Couche Service (Logique Métier) :** Une couche Service (@Service), comme FicheDePaieService ou EmployeService, est introduite pour encapsuler et gérer la logique métier complexe (par exemple, le calcul du salaire, les validations métier, l'orchestration des données provenant de différents Repositories). Cette couche assure la modularité.

    - **Couche Persistance (Repositories) :** Les classes DAO manuelles (ex: EmployeDAO.java) sont remplacées par des interfaces **Spring Data JPA Repositories**. Ces interfaces étendent JpaRepository, permettant à Spring de générer automatiquement les méthodes d'accès aux données (CRUD, requêtes par nom de méthode, etc.), réduisant considérablement la quantité de code "boilerplate".

- **Parallèle des Composants :** La migration permet de distribuer les responsabilités des composants J2EE couplés vers des composants Spring plus modulaires, comme illustré par les équivalences suivantes :
    - Le AuthFilter.java actuel serait avantageusement remplacé par **Spring Security**, offrant une solution standard, puissante et hautement configurable pour la gestion des rôles et des autorisations.
    - Les Servlets de Contrôle (ex: EmployeServlet.java) sont remplacées par des Contrôleurs annotés (EmployeController.java).
    - Le EmployeDAO.java est remplacé par une interface EmployeRepository.java étendant JpaRepository.
    - Le rôle de HibernateUtil.java (démarrage et gestion des sessions) est repris par l'auto-configuration de Spring Boot.

5. Outils, Déploiement et Conclusion

5.1 Outils de Développement et Déploiement

Le développement de l'application RowTech a été structuré autour d'un ensemble d'outils éprouvés, essentiels pour garantir la stabilité, la reproductibilité et la collaboration au sein de l'équipe. Le cœur de la construction et de l'orchestration du projet repose sur **Maven**, qui ne se contente pas de gérer les dépendances (Hibernate, iText, Servlets, BCrypt), mais formalise également le cycle de vie du projet. Ce rôle est crucial pour compiler l'application, exécuter les tests unitaires et, finalement, conditionner le code source et les ressources statiques dans un fichier WAR (Web Archive).

Le déploiement, conformément à la pratique standard des applications Java EE, s'effectue sur une infrastructure largement utilisée :

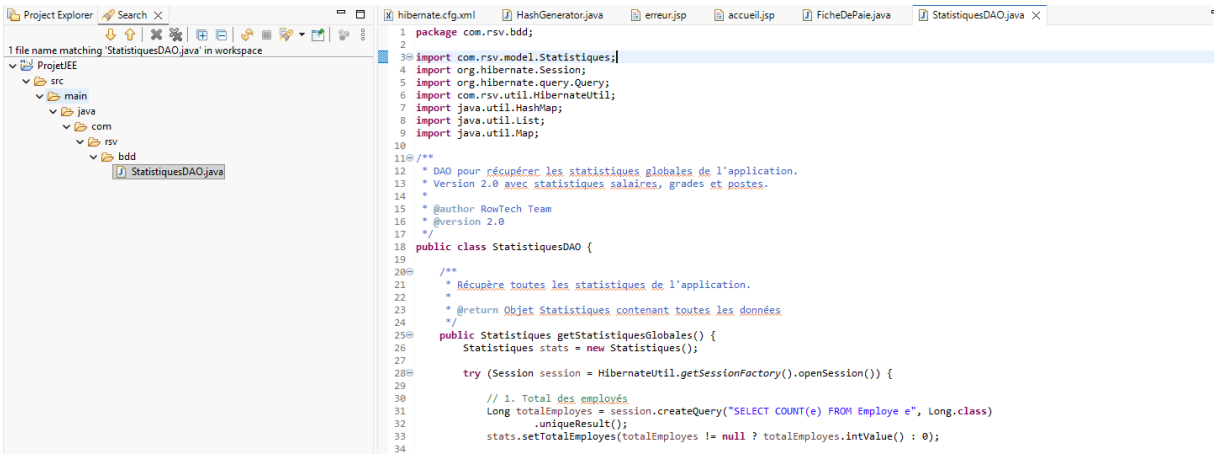
- Le fichier WAR généré par Maven est déposé sur le serveur d'applications **Apache Tomcat v10.1**. Ce serveur est indispensable, car il assure l'environnement d'exécution des Servlets et des JSPs, agissant comme le conteneur fondamental de l'application. La configuration des Servlets et du filtre d'authentification est gérée via le fichier web.xml et les annotations dédiées.
- L'application utilise **MySQL** comme SGBD relationnel. Ce choix offre une solution mature et fiable pour la persistance des données structurées (Employé, Projet, Fiche de Paie), configurée via la couche Hibernate pour un accès optimisé et sécurisé.
- La gestion des versions a été collaborative et transparente grâce à l'utilisation de **Git** et à un dépôt **GitHub**. Cette démarche a permis un suivi rigoureux des modifications, l'intégration continue des contributions des membres de l'équipe, et la traçabilité des évolutions du code source.

5.2 Bilan Critique du Projet

Forces de l'Implémentation

L'architecture JEE, bien que demandant une gestion fine des ressources, a prouvé son efficacité, aboutissant à une application fonctionnelle et robuste.

- **Séparation des Couches (MVC) :** La distinction claire entre les Servlets agissant comme Contrôleurs, les JSP servant de Vues, et les DAO/Entités formant le Modèle, a grandement facilité le développement modulaire et la maintenance future. Cette séparation permet à chaque développeur de se concentrer sur une couche spécifique sans impacter directement les autres.
- **Sécurité Rigoureuse et Proactive :** L'implémentation de l'authentification est basée sur **BCrypt**, un algorithme de hachage reconnu et résistant au *brute-force* grâce à son facteur de travail paramétrable. De plus, le filtre central AuthFilter garantit qu'aucune ressource protégée ne peut être accédée sans un userId valide en session, ce qui est fondamental pour la protection des données RH sensibles.
- **Fonctionnalités Métier Avancées :** Le projet a réussi à intégrer des fonctionnalités complexes qui dépassent le simple CRUD. On note notamment le mécanisme de calcul automatique et détaillé des fiches de paie (FicheDePaie.java), qui gère les multiples composantes (prudence dans le calcul des cotisations, gestion des jours d'absence) et la **génération de rapports PDF officiels** (via iText 7), assurant une conformité documentaire essentielle pour la fonction Paie.



Points de Vigilance et Difficultés

Plusieurs défis inhérents à l'architecture JEE/Hibernate ont été rencontrés et ont nécessité une expertise particulière pour garantir la robustesse du système.

- **Gestion Manuelle des Sessions et Transactions Hibernate :** L'absence d'un conteneur gérant la persistance (comme Spring) a imposé une gestion manuelle stricte des sessions Hibernate et des transactions au sein de chaque DAO. Le non-respect des blocs session.openSession(), beginTransaction(), commit() et session.close() dans la

clause finally est une source fréquente d'erreurs. Des problèmes récurrents comme la LazyInitializationException ont été observés, exigeant l'utilisation stratégique de l'opération FETCH dans les requêtes HQL pour charger les relations essentielles de manière anticipée.

- **Cohérence des Relations M:N :** La gestion de la relation Employé-Projet, bien que correctement modélisée, a nécessité une logique explicite dans le EmployeDAO pour synchroniser la table de jointure. Toute modification de l'employé (ex: changement de département) devait être soigneusement suivie d'une mise à jour de ses affectations aux projets pour éviter l'incohérence des données orphelines.
- **Validation des Données :** L'implémentation de la validation côté serveur via ValidationUtil.java et son invocation manuelle dans chaque Servlet augmente la quantité de code de contrôle à maintenir. Comparée à l'approche déclarative de solutions modernes basées sur des annotations JPA/Bean Validation, cette méthode est plus lourde et plus sujette à l'oubli de vérifications dans certaines requêtes.

Perspectives d'Amélioration

Afin de faire évoluer l'application vers un standard industriel plus actuel et d'améliorer la maintenabilité à long terme, l'équipe envisage les pistes suivantes :

- **Finalisation de la Migration Spring Boot :** L'adoption complète de Spring Boot permettrait d'éliminer la gestion manuelle des ressources JEE et de basculer vers **Spring Data JPA**. Cette simplification éliminerait la majeure partie du code de la couche DAO, les remplaceraient par des interfaces claires, et confierait la gestion transactionnelle au framework, rendant l'application plus légère et plus rapide à déployer.
- **Découplage Front-end via API REST :** Il est crucial de remplacer les JSPs par une véritable API RESTful (exposée par les Contrôleurs Spring Boot) et d'utiliser un framework Front-end moderne (tel que React ou Angular). Ce découplage permettrait une expérience utilisateur plus dynamique et une meilleure évolutivité de l'interface indépendamment de la logique serveur.
- **Optimisation de la Sécurité :** L'intégration d'un framework de sécurité dédié tel que **Spring Security** est la prochaine étape logique. Cela permettrait de gérer les droits d'accès (hasRole('ADMIN'), etc.) de manière déclarative et centralisée au niveau de la couche Service ou du Contrôleur, offrant ainsi une granularité et une robustesse bien supérieures au filtre d'authentification actuel.

Conclusion Générale et Bilan (Synthèse du Projet)

Le projet RowTech s'achève sur la livraison d'une application de gestion des ressources humaines, des départements et des projets, entièrement fonctionnelle et bâtie sur une architecture J2EE MVC éprouvée. Ce travail a permis d'atteindre la totalité des objectifs fixés dans le cahier des charges initial, en allant au-delà du simple développement CRUD pour intégrer des processus métier critiques et des mécanismes de sécurité robustes.

Atteinte des Objectifs Fonctionnels et Techniques

L'application valide avec succès l'ensemble du périmètre fonctionnel en offrant une solution intégrée pour la gestion administrative et opérationnelle de l'entreprise :

- **Gestion Complète du Cycle de Vie :** Les fonctionnalités CRUD sur les Employés, Départements, et Projets sont opérationnelles. La gestion des relations complexes, notamment l'affectation multiple des employés aux projets et l'intégration du rôle de Chef (Département ou Projet) au niveau des entités, est assurée par une logique métier rigoureuse dans les classes DAO.
- **Fonctionnalité Critique de la Paie :** Le mécanisme de calcul des fiches de paie est l'une des réussites majeures. Le système gère les entrées variables (primes, heures supplémentaires, jours d'absence) et applique des taux de cotisations sociales simulés pour déterminer un **Net à Payer** cohérent. Cette donnée est historisée et est rendue consultable pour l'employé et le chef d'équipe, assurant la transparence du processus.
- **Sécurité et Autorisation Basées sur les Rôles :** Le système de sécurité est pleinement fonctionnel grâce au filtre d'authentification (AuthFilter) et à l'utilisation de l'algorithme **BCrypt** pour le stockage sécurisé des mots de passe. Les autorisations (Administrateur, Chef, Employé) sont appliquées pour restreindre l'accès aux Servlets et aux vues, garantissant que chaque utilisateur n'interagit qu'avec les données relevant de son périmètre de responsabilité.
- **Reporting Essentiel :** Le projet a intégré des capacités de reporting vitales. L'utilisation de **iText 7** pour générer les bulletins de paie au format PDF garantit une conformité documentaire essentielle. Parallèlement, le tableau de bord des statistiques (alimenté par StatistiquesDAO) offre une visibilité immédiate sur les indicateurs clés de performance des ressources humaines (répartition par grade, masse salariale, suivi des projets).

Maîtrise Architecturale et Compétences Acquises

La réalisation de ce projet en environnement J2EE a permis à l'équipe de valider une compréhension approfondie de l'architecture logicielle d'entreprise :

- **Maîtrise de l'Architecture MVC :** Le respect strict du modèle MVC a été un pilier, démontrant la capacité à séparer la logique de contrôle (Servlets), le modèle de données et la présentation (JSP), facilitant le développement parallèle et la mise au point des fonctionnalités.
- **Persistance Avancée avec Hibernate :** Le projet a permis d'acquérir une expertise concrète dans l'utilisation d'Hibernate comme couche de persistance, impliquant :
    - La gestion manuelle et rigoureuse des sessions et des transactions pour garantir l'intégrité des opérations.
    - Le traitement des *LazyInitializationExceptions* via l'utilisation stratégique des FETCH JOIN en HQL.
    - La gestion des contraintes d'intégrité référentielle, notamment ON DELETE SET NULL pour les départements et l'utilisation de tables de jointure pour les relations Many-to-Many.
- **Développement Full Stack J2EE :** L'équipe a géré l'ensemble de la pile technique, depuis la conception de la base de données MySQL et le mappage ORM (JPA), jusqu'à l'implémentation de la logique métier en Java, l'exposition des contrôleurs Servlets, et la construction de l'interface utilisateur dynamique via les JSPs.

Conclusion :

Bien que l'architecture Servlets/JSP/Hibernate ait servi de base solide et ait permis d'atteindre tous les objectifs, une évaluation critique révèle des opportunités d'optimisation :

- **Nécessité de Modernisation :** L'approche JEE traditionnelle a démontré sa rigidité, notamment dans la lourdeur de la configuration (XML) et la gestion manuelle des transactions DAO. C'est pourquoi l'étude de l'implémentation Spring Boot a été un exercice précieux.
- **Feuille de Route pour l'Avenir :** La migration vers **Spring Boot** représente la prochaine étape logique. En adoptant une architecture basée sur les microservices ou, à défaut, une application monolithique Spring, l'équipe pourra :
    - Déléguer la gestion des ressources et la couche transactionnelle au framework, réduisant le code "boilerplate".
    - Remplacer les Servlets par des Contrôleurs REST, permettant un découplage total du Front-end.
    - Intégrer **Spring Security** pour une gestion des autorisations plus modulaire et déclarative.

Finalement, le projet RowTech est une réussite technique et fonctionnelle. Il représente une validation complète de la chaîne de développement Java/JEE et sert de tremplin solide pour l'exploration de l'écosystème Spring, préparant l'équipe aux exigences des architectures logicielles modernes.