Alvariane CUPESSALA
Arthur JIN
Lohan BRIARD
Sarusan NITHIYARAJAN
Yussuf RUMELI

Rapport Projet JEE ING 2

Introduction

Les applications sont des éléments clés dans l'utilisation d'outils techniques et en tant qu'ingénieurs, il n'y a rien de mieux que pouvoir comprendre et développer des programmes si fascinant du côté de l'utilisateur. Ainsi nous allons voir comment Alvariane Cupessala, Arthur Jin, Lohan Briard, Sarusan Nithiyarajan et Yusuf Rumeli ont développé une application développée pour gérer une entreprise. Notre équipe s'est divisé en 2 groupes: le premier groupe, constituée d'Alvariane Cupessala, Lohan Briard et Arthur Jin, s'est occupé de la conceptualisation et l'implémentation de l'application Web Dynamique; le deuxième groupe, constituée de Sarusan Nithiyarajan et Yusuf Rumeli, s'est occupé de la partie d'implémentation en refaisant l'application en utilisant SpringBoot et l'implémentation de la base de données.

1. Conception

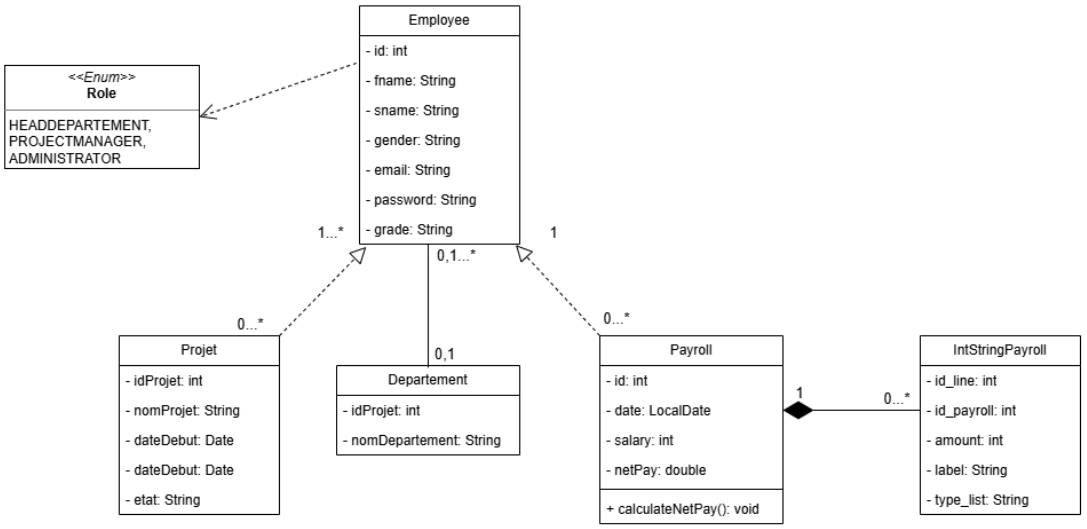
La conception de l'application a commencé le 5 Novembre, par le développement de notre diagramme de classe où on a conceptualisé chaque classe nécessaire pour le développement de notre application et création de la base de données. Les principaux points à développer étaient la gestion des employés, la gestion des départements, la gestion des projets et la gestion des fiches de paie. Nous avons choisi le l'architecture MVC (Modèle-Vue Contrôleur) conformément au cahier des charges et car c'est une architecture efficace.

Commençons par la gestion des employés, sachant qu'on aurait besoin d'une manière de gérer les comptes des employés, en premier on a commencé par la conception d'une classe pour les comptes et un autre pour le rôle et caractéristiques de l'employé associé au compte x. Ainsi on avait les informations associées à l'utilisateur et les informations de l'employé dans deux classes séparées, au début cela semblait la meilleure solution pour la gestion des employés. Pourtant avec le temps, on s'est rendu compte que cela était inutile car un compte d'employé peut regrouper toutes ces données et ça nous a évité de créer et gérer une classe supplémentaire. Pourtant un problème persistait, la gestion des rôles et l'authentification.

On devait faire en sorte qu'on puisse catégoriser les employés, ainsi nous avons décidé de faire une énumération nommée "Rôle" constituée de trois options : administrateur(ADMINISTRATOR), chef de projet(PROJECTMANAGER) et chef de département(HEADDEPARTEMENT). L'employé n'existe pas sans son rôle, ainsi on a établi une liaison de dépendance entre les deux classes, ce qui nous a permis d'englober toutes les demandes concernant la gestion des employés, l'authentification et l'autorisation.

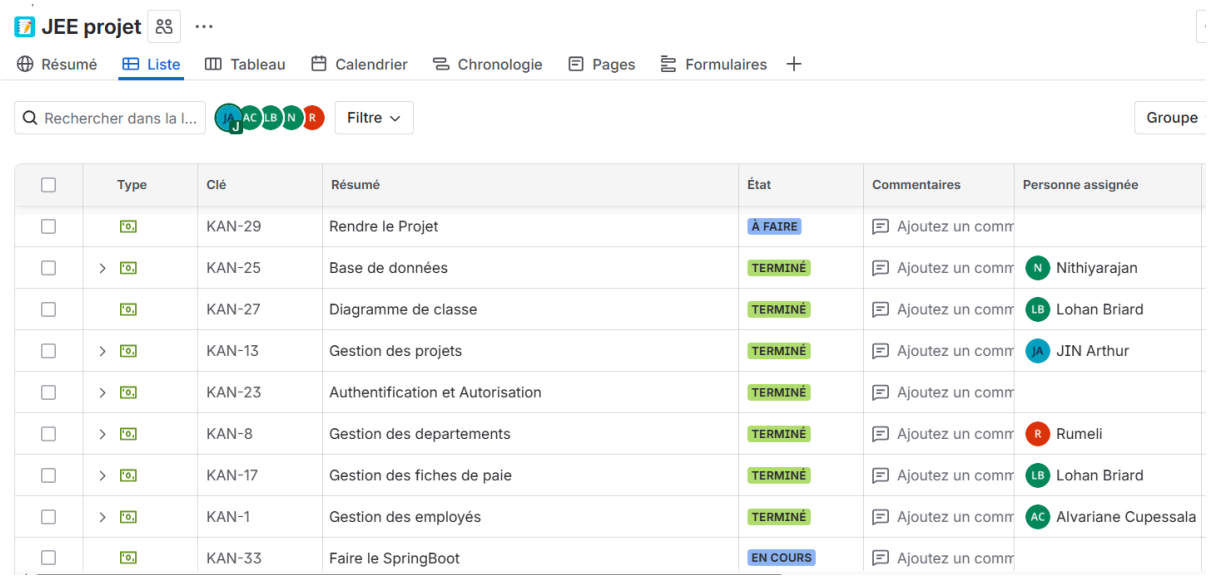
Ensuite, nous avons attaqué la gestion des départements et des projets, une fois qu'ils fonctionnent de la même manière. La gestion de département et des projets similaires car ils ont la même relation avec la classe employé et ont quasiment la même structure de données. Ces deux classes ont le même concept dans le sens où elles gardent principalement les données d'un seul employé, qui est le chef du département ou projet et les données respectives à l'entité. Le reste des employés, ayant une relation différente de chef de projet, gardent les identifiants de leur entité respective.

Finalement pour la gestion de fiche de paie, on s'est mis d'accord sur le fait qu'on aurait deux classes pour gérer les fiches de paie. Une classe de fiche de paie ne possède que les informations essentielles de la fiche de paie, les données supplémentaires (Primes et Déductions) qui aurait la clé primaire d'une fiche de paie et une colonne de type option avec les seules options étant Primes et Déductions.


Employee
- id: int
- fname: String
- sname: String
- gender: String
- email: String
- password: String
- grade: String
<<Enum>>
Role
HEADDEPARTEMENT,
PROJECTMANAGER,
ADMINISTRATOR
1...*
0,1...*
1
0...*
0...*
0,1
1
0...*
Projet
- idProjet: int
- nomProjet: String
- dateDebut: Date
- dateDebut: Date
- etat: String
Departement
- idProjet: int
- nomDepartement: String
Payroll
- id: int
- date: LocalDate
- salary: int
- netPay: double
+ calculateNetPay(): void
IntStringPayroll
- id_line: int
- id_payroll: int
- amount: int
- label: String
- type_list: String

2. Implémentation

Tout d'abord pour nous organiser, nous avons créé dès le début du projet un groupe sur l'application Discord, afin de pouvoir communiquer rapidement et planifier des réunions pour assurer le bon progrès du projet. De plus, nous avons mis en place un projet sur Jira, un site permettant de créer des tickets en fonction des tâches afin de rendre le développement de l'application en équipe plus facile. Enfin, on a créé un dépôt GitHub commun, qui nous permettait de toujours être synchronisé dans notre travail, sans avoir besoin de s'envoyer des fichiers partout, un lien pour accéder à ce dépôt est mis sur le fichier d'inscription des équipes pour le projet.



Screenshot du Jira

2.1 Version classique

Nous nous sommes d'abord consacré à la version "classique" du projet, en utilisant Maven et Hibernate, déployés sur un serveur Tomcat. Comme base de données, nous avons choisi MySQL, pour sa facilité d'utilisation et ses fonctionnalités qui correspondaient à ce dont on avait besoin. À partir du MCD (Modèle Conceptuel des Données), on a pu identifier les classes qu'il fallait implémenter et ainsi nous avons

commencé par modéliser la base de données et la remplir avec quelques valeurs temporaires pour pouvoir tester si nos implémentations de code fonctionne.

Une fois la base de données opérationnelle, nous avons configuré les fichiers essentiels au projet. Cela a nécessité la création du fichier de configuration hibernate.cfg.xml pour définir les paramètres de connexion. Nous avons ensuite développé les classes d'entités Java correspondant à nos tables dans la base de données en utilisant les annotations JPA (@Entity, @Id, @OneToMany..). Pour manipuler ces objets, nous avons implémenté le pattern DAO (Data Access Object) à l'architecture MVC. Cet ajout nous a permis de centraliser toutes les opérations de lecture et d'écriture (CRUD) en isolant le code technique lié à Hibernate du reste de la logique métier de l'application.

Après tout cela, nous avons enfin commencé à implémenter du code, commençant par les classes Java des entités correspondant à notre base de données (Employés, Départements, Projets..), nous avons fait cela pour le 9 Novembre car c'est la partie essentiel pour pouvoir progresser sur les fonctionnalités sans ces entités, on ne peut pas ajouter les fonctionnalités attendus. Ensuite, entre le 9 et le 15 Novembre, nous avons travaillé pour ajouter toutes les fonctionnalités attendues dans les parties : Gestion des Employés, Gestion des départements, Gestion des projets et Authentification et Autorisation. La partie pour les fiches de paie est implémentée pour le 18 Novembre et jusqu'au 20 Novembre, nous avons consacré notre temps à la correction de bug et au "clean up" du code et du style de la page web.

2.2 Version Spring Boot

Après avoir fait les fondations du projet version classique, nous avons travaillé en parallèle sur le Spring Boot ce qui nous a permis de se passer de la configuration lourde du serveur Tomcat externe et des fichiers XML. Nous avons utilisé à nouveau notre base de données MySQL existante en réutilisant nos classes d'entités annotées JPA, telles que la classe Departement, assurant ainsi une continuité dans le modèle de données.

L'évolution majeure de cette version réside dans la suppression du pattern DAO manuel au profit de Spring Data JPA. Comme illustré par notre interface DepartementRepository, il nous a suffi d'étendre JpaRepository pour bénéficier automatiquement des opérations CRUD. Pour les besoins plus complexes, comme les statistiques, nous avons intégré des requêtes personnalisées via l'annotation @Query (ex : comptage des employés par département), éliminant ainsi une grande quantité de code.

Sur le plan architectural, les Servlets ont été remplacés par des classes annotées @Controller, profitant de l'injection de dépendances pour appeler les repositories. Notre DashboardController centralise ainsi la logique de navigation et d'administration.

Concernant la sécurité, nous avons implémenté un système d'authentification hybride : la gestion de session reste manuelle via HttpSession pour conserver la maîtrise du flux de connexion, mais nous avons renforcé la sécurité des données en intégrant PasswordEncoder pour le hachage et la vérification des mots de passe lors du login et de la mise à jour du profil.

Le problème majeur qu'on a rencontré sur cette partie Spring Boot était lors de l'ajout de plusieurs pages dont la réelle page de connexion, la page de profil ainsi que la page de statistiques qui a été ajoutée environ 1 semaine après le début du projet Spring Boot. On avait prévu de continuer à créer un controller.java pour chaque nouvelle page mais on tombait sur une erreur qu'on a pas réussi à résoudre. On a donc décidé de rajouter ces nouveaux controller en les rassemblant dans le DashboardController qui fonctionne.