IMR2

Gestion de Congés
RAPPORT DU PROJET JEE

—

DADIE Marie-Danielle
DELVOYE Nicolhas
MORIN-COZANNET Maeg
TRAN Sovanny

ENSSAT
LANNION

Introduction

L'objectif du projet était de concevoir une application web de gestion de congés. Ce logiciel fait intervenir plusieurs acteurs à savoir des employés standards, des employés des ressources humaines et des chefs d'équipe. Chacun d'eux a accès à des fonctionnalités de l'application spécifiques à leur fonction.

Pour réaliser ce projet, nous avions pour obligation d'utiliser JavaEE, un serveur Tomcat ainsi qu'une base de donnée MySQL. Pour réaliser la persistance en base, nous avons mis en place Hibernate et différents DAO.

Sommaire

Fonctionnalités de l'application

1. Les demandes

Chaque employé a accès au formulaire de demande de congés.

Une demande est constituée d'un identifiant qui représente la clé primaire dans la base de données, d'un type, d'un état, d'une date de début, d'une date de fin, du courriel du demandeur et d'une date de création.

Une demande possède un état:

- S'il passe à **1** : la demande est considéré comme **acceptée** on parle alors de Congé validé.
- S'il passe à **-1** : la demande est considérée comme **refusée** on parle alors de Congé refusé.

Si un congé en cours de validation reste 48h sans réponse de la part du service RH, il est considéré comme **Acceptée automatiquement**. S'en suit alors la mise à jour de l'état à **1** et un commentaire spécifique.

2. Employé standard

L'application doit permettre l'affichage de la liste des demandes d'un employé standard ainsi que l'affichage des détails de chacune de ses demandes. Via l'application, il doit également être possible de modifier et supprimer une demande et ce même si elle est en cours de validation.

Un employé est constitué d'un courriel professionnel qui représente la clé primaire dans la base, d'un nom, d'un prénom, d'une adresse postale, d'un service, d'une fonction, d'une équipe et d'une date de recrutement.

3. RH

Un RH hérite des différentes fonctionnalités que possède un employé. Il possède d'autres fonctionnalités telles que : L'affichage de la liste des demandes de tous les

employés. Il peut valider ou refuser la demande : dans le cas d'une Validation, il n'est pas obligé de laisser un commentaire. À contrario, dans le cas d'un Refus, il est dans l'obligation de laisser un commentaire. Ce commentaire est visible par l'employé ayant émis la demande.

Un RH peut modifier le type d'un congé avant de le valider. Enfin, il peut accéder à une page de statistiques où il pourra voir la proportion de type de congé pris ou encore le nombre de congés pris par tel ou tel employé.

4. DRH

Un DRH hérite des différentes fonctionnalités que possède un RH. Il possède néanmoins de nombreuses autres fonctionnalités : Il peut afficher les différents employés de l'entreprise. Le DRH peut ajouter un nouvel employé en base mais également modifier ses informations, ou bien le supprimer de la base.

5. Chef d'équipe

Un chef d'équipe hérite des différentes fonctionnalités que possède un employé. La seule particularité réside en le fait de pouvoir afficher la liste des congés qui ont été posés par les membres de son équipe.

6. Authentification

L'accès à l'application n'est possible que si l'utilisateur s'est authentifié au préalable via une page de connexion nécessitant son adresse mail et un mot de passe. La liste des utilisateurs du site correspond à la liste des employés.

Une fois connecté, un utilisateur peut modifier librement son mot de passe au sein de la page "Mon Profil".

7. Compteur

Un compteur est constitué de l'email du demandeur et de son solde de congés pour chaque type. Si une demande de congés est acceptée de type X est acceptée, son solde de congés pour ce type se voit modifié.

Une unité de congé correspond à une demi-journée de congé dans la vraie vie (Exemple : si un employé possède 10 RTT, il peut poser une semaine entière de congé).

Spécifications techniques

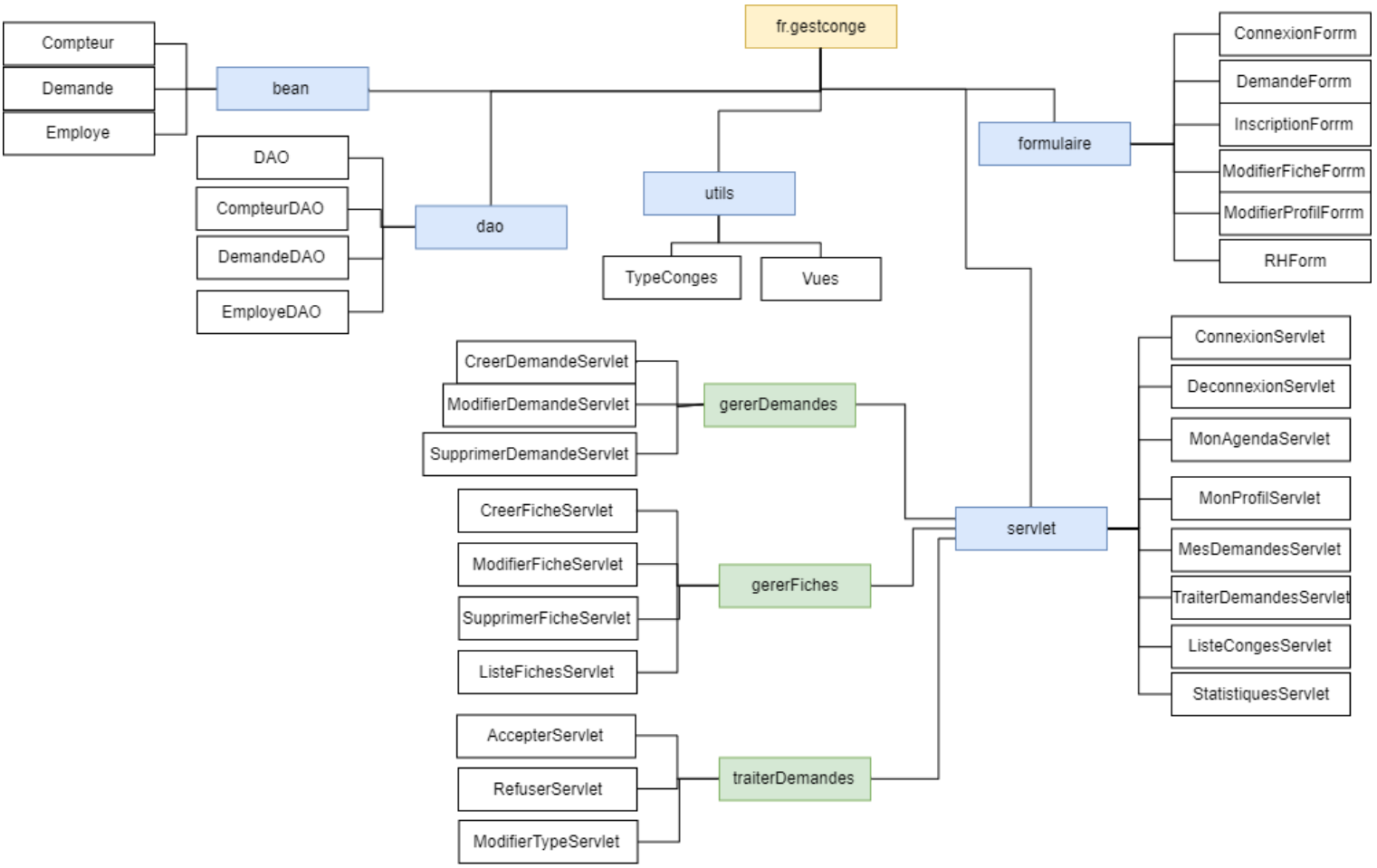
Lors de ce projet, nous avions pour obligation de développer notre application web en JavaEE. Nous devions également utiliser un serveur Tomcat (version 9), mais également un serveur MySQL

Nous avons décidé de mettre en place le framework Hibernate afin de réaliser la persistance de nos données très facilement.

Pour configurer le mapping entre les classes Servlet et les routes associées, nous n'utilisions pas le fichier `web.xml` comme nous avions pu voir en TP mais avons opté pour l'utilisation d'annotations `@WebServlet("/route")`. Ces annotations nous permettaient de retrouver la route pour telle ou telle Servlet très rapidement, sans avoir à aller lire dans un fichier où de nombreuses lignes sont présentes.

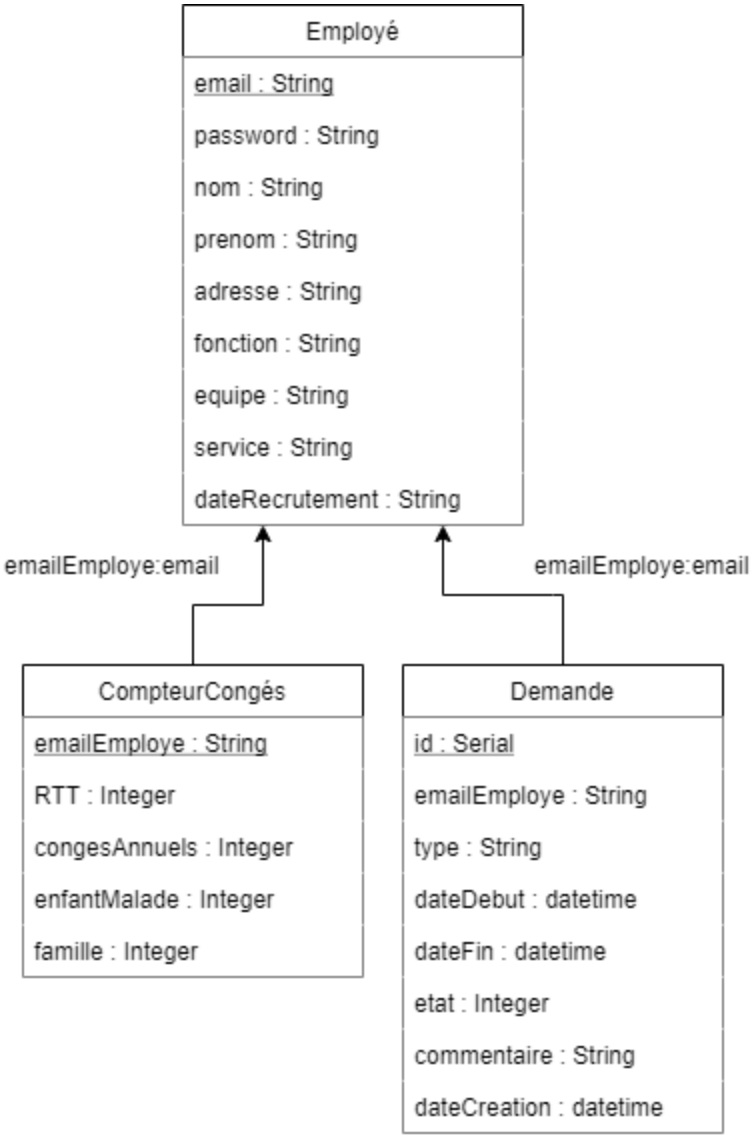
Structure de l'application

Figure 1 : Diagramme de classe



Modèle de données

Figure 2 : Structure et relations entre tables de la base de données



Implémentation des fonctionnalités

L'application utilise le modèle MVC :

- Les modèles (classes Java générées via Hibernate) sont stockés dans le dossier **bean**
- Les vues (JSP) sont stockés dans le dossier **web**
- Les contrôleurs (Servlet) sont stockés dans le dossier **servlet**.

Pour insérer ou modifier des données en base de données, nous avons mis en place divers formulaires (création de fiche employé, connexion, demande, ...). Les servlets associés sont appelés via des requêtes **POST**. L'objet **request** comporte ainsi les différents paramètres du formulaire appelant la servlet.

Les objets métiers étaient récupérés et mis à jour en base de donnée à l'aide du pattern DAO.

Afin de garantir un certain niveau de sécurité au sein de notre application, nous effectuons des conditions sur l'objet Employe stocké au sein de notre session. Si un utilisateur cherche à accéder à la page "MesDemandes", il doit obligatoirement se connecter via la page de login. Si aucun utilisateur n'est stocké au sein de la Session, il est redirigé vers le formulaire de connexion.

Problèmes rencontrées et axes d'amélioration

1. Choix des outils de développement

Pour des raisons pratiques, nous avons décidé de développer notre application sur Intellij IDEA. En effet, cet IDE comporte de nombreux avantages que ses concurrents (Eclipse notamment) :

- une interface graphique plus agréable/ergonomique
- le moteur de recherche est plus efficace

Nous avons également fait ce choix dans le but de découvrir/de perfectionner un outil actuel puisqu'il est énormément utilisé par les entreprises de nos jours.

Nous pouvions rapidement télécharger les librairies et les placer dans le dossier **libs** (et la création de l'artifact afin de le lancer dans le serveur tomcat) sans avoir à perdre du temps.

Toutefois, nous avons pu perdre quelques jours de travail à cause de la gestion des dépendances suite au déploiement d'Hibernate. En effet, pour pallier à un problème que nous avions, il suffisait de marquer le répertoire resources (où se situe notre persistence.xml) en tant que répertoire "Ressources" (Mark directory as… Resources root).

Mais les différents avantages permettent de dire qu'IntelliJ était un choix tout à fait légitime de notre part.

2. Gestion de la base de données

La mise en place de la persistance au sein de l'application a été l'étape la plus compliquée de notre projet. Nous sommes passés par différents moyens de persistance :

Dans un premier temps nous avons suivi l'exemple du TP de JEE qui nous permettait d'aller chercher des valeurs de la base de données. Cependant, nous avons rencontré des difficultés au moment où nous cherchions à écrire, modifier ou supprimer des valeurs au sein de notre base.

Le temps de trouver une solution, nous avons mis en place un Mock pour tester chacune des fonctionnalités.

Finalement, nous avons choisi d'intégrer Hibernate qui nous a permis de faciliter la persistance et la recherche de données dans la base. On réalisait ainsi la création des objets et les traitements de remplissage de ceux ci en accédant directement à la base de données.

Là aussi, la prise en main a été progressive vu que nous n'avions jamais utilisé ce framework.

Par ailleurs, dans la table Employe, la clé primaire est représentée par son email (De même pour Compteur). Ainsi, un axe d'amélioration afin d'optimiser les différentes requêtes appels en base de données aurait été d'utiliser le type Serial.

3. Factorisation des formulaires

Lorsqu'un utilisateur saisit un formulaire, il peut arriver qu'il commette des erreurs de différents type (Exemple : Lors de la création d'une demande de congé, il peut mettre une date de fin avant une date de début).

Pour lui permettre de savoir ce qui a pu poser problème lors du traitement, on utilise une classe Formulaire qui est composée d'une Map associant l'id d'un champ "input" (String) à un message d'erreur (String).

Cependant, lors du projet, nous n'avons pas mis en place une classe Formulaire dont hériteront tous nos formulaires. Cette classe aurait pu comporter un message "global" de succès ainsi que la Map décrite ci-dessus.

4. Ergonomie

Les données affichées sur chacune des vues correspondent directement à celles récupérées dans la BDD. Ce qui souvent était inapproprié. En effet, afin d'optimiser notre base de donnée, nous avions choisi de mettre des codes de "TypeCongés" (Exemple : Le code ENFMALAD corrrespond à un congé de type "Enfant malade").

D'un point de vue ergonomie, nous aurions pu appeler une fonction CongesUtils afin de récupérer le libellé associé au codeTypeCongé.

Mettre en place notre gestionnaire de congés

Pour mettre en place le gestionnaire de congés, il faudra au préalable avoir mis en place le serveur Tomcat ainsi que le serveur MySQL.

Vous devez également télécharger les sources de l'application qui sont disponible sur notre git : https://github.com/ndelvoye/PROJET_JEE

Sur celui-ci, vous retrouverez les sources du projet ainsi que le fichier qui nous intéresse ici : `gestionConges.war`.

Avant de pouvoir accéder à l'application, il faut dans un premier temps mettre en place notre base de données MySQL.

Pour cela, on modifie le mot de passe root afin qu'il soit cohérent avec la configuration Hibernate :

- mysql -u root -p
    - Action utilisateur : Rentrer le mot de passe MySQL(tonystark sur la VM du TP)
- > SET PASSWORD FOR root@localhost=PASSWORD(root);
- > FLUSH PRIVILEGES;

Et on créé la base :

- mysql -u root -p
    - Action utilisateur : Rentrer le mot de passe MySQL (root)
- CREATE DATABASE gestionconges;

Puis, on exécute les différents scripts contenus au sein du dossier **SQL** :

- mysql -u root -p gestionconges < [repertoire_SQL]/createTables.sql
- mysql -u root -p gestionconges < [repertoire_SQL]/insertValues.sql
- mysql -u root -p gestionconges < [repertoire_SQL]/acceptAutoDemandes.sql

Un problème rencontré lors de l'export du .war sur l'environnement Unix nous oblige à modifier le nom des tables (dans un souci de respect de la casse) :

- RENAME TABLE employe TO Employe;
- RENAME TABLE compteur TO Compteur;
- RENAME TABLE demande TO Demande;

Ensuite, il suffit d'installer le .war sur un serveur Apache Tomcat comme vu lors du TP :

1. Démarrer Tomcat : `sudo sh /opt/tomcat/latest/bin/startup.sh &`
2. Se rendre sur l'interface Web Tomcat : http://localhost:8080/manager (admin/admin)
3. Uploader et déployer le war "WAR file to deploy"

L'application est désormais lancée et vous y accéder sur le lien suivant : http://localhost:8080/gestionConges

Vous pouvez maintenant essayer notre application. Pour cela, nous avons configuré un compte DRH de base :

- **email** : mdadie@jee.fr
- **password** : mdadie

Conclusion

Ce projet nous a permis de monter en compétences en matière de développement Java. Nous avons également pu découvrir l'utilisation du modèle MVC en JavaEE mais également le pattern DAO ainsi que la persistance en base de données. Nous avons notamment pu mettre en place le framework Hibernate.

Bien que nous ayons rencontré des difficultés sur certains axes au cours de ce projet, il a permis à l'ensemble de l'équipe de monter en compétences d'un point de vue technique, fonctionnel mais également d'un point de vue de gestion de projet.