Compte-rendu

Projet JEE

Clément FORGEARD, Flavien JOURDREN, Alexis LE GAL et Jean-Baptiste DUCHENE

2019-2020

IMR2

INTRODUCTION

Ce rapport traite de la réalisation d'un projet de conception d'une application de gestion de congés. Ce projet a pour but de mettre en pratique les enseignements des matières d'UML et de Java EE. Ce rapport présentera le comportement de notre application avec son architecture, les designs de conception choisis pour notre code et la structure de notre base de données.

TRAVAIL DEMANDÉ

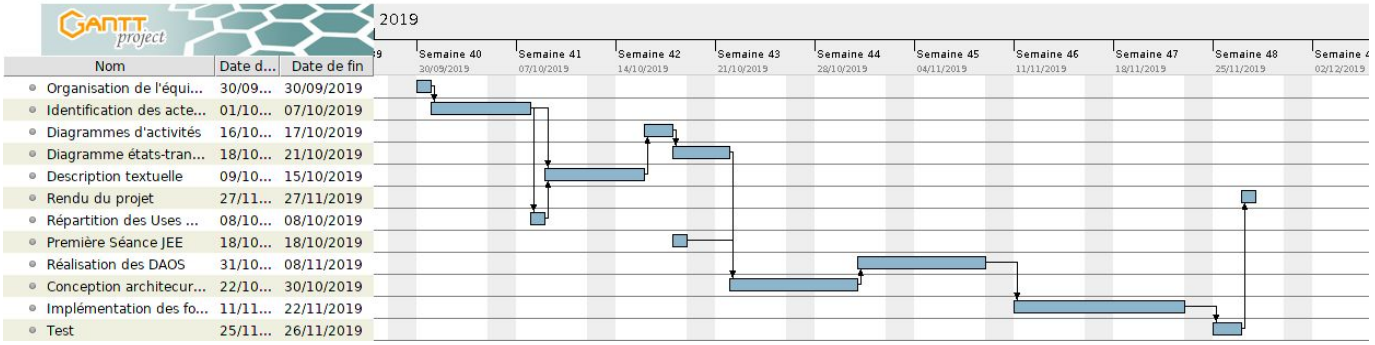
OBJECTIFS ATTENDUS

Ce projet a pour objectif de mener une analyse UML et concevoir en conséquence une application Java EE. Le projet se déroule donc selon une phase d'analyse et de conception. Dans le cadre de la phase d'analyse, il est demandé de suivre les concepts et méthodologies déjà étudiés pour réaliser une application de gestion de congés. Des contraintes sont imposées sur des cas d'utilisations et il est nécessaire d'éclaircir le sujet avec le client pour réaliser une application robuste basée sur notre analyse. Dans le cadre de la phase de conception, nous avons choisi d'utiliser les patterns de conception vus en cours comme le modèle DAO qui permet de créer des objets d'accès aux données sans écrire dans les classes "métiers".

RÉPARTITION DES TÂCHES

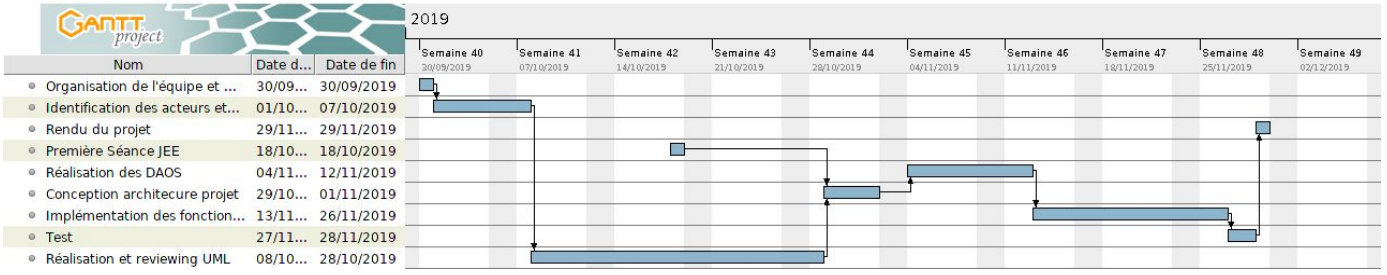
Dans un premier temps, Clément FORGEARD s'est proposé chef de projet. À la suite de cela, le groupe a défini ensemble les différents acteurs et cas d'utilisation de l'application de gestion de congés. Ensuite, les cas d'utilisation ont été répartis en tant que tâches aux membres de l'équipe. Chaque membre a travaillé sur ses tâches pour en obtenir les diagrammes d'activités, de séquences, et d'états-transition correspondants. Grâce à une démarche itérative et incrémentale, les diagrammes ont été revus par d'autres membres avant d'être retravaillés lorsque cela était nécessaire.

PLANNING PREVISIONNEL



Nom	Date d...	Date de fin
Organisation de l'équi...	30/09...	30/09/2019
Identification des acte...	01/10...	07/10/2019
Diagrammes d'activités	16/10...	17/10/2019
Diagramme états-tran...	18/10...	21/10/2019
Description textuelle	09/10...	15/10/2019
Rendu du projet	27/11...	27/11/2019
Répartition des Uses ...	08/10...	08/10/2019
Première Séance JEE	18/10...	18/10/2019
Réalisation des DAOS	31/10...	08/11/2019
Conception architecur...	22/10...	30/10/2019
Implémentation des fo...	11/11...	22/11/2019
Test	25/11...	26/11/2019

PLANNING RÉEL



Nom	Date d...	Date de fin
Organisation de l'équipe et ...	30/09...	30/09/2019
Identification des acteurs et...	01/10...	07/10/2019
Rendu du projet	29/11...	29/11/2019
Première Séance JEE	18/10...	18/10/2019
Réalisation des DAOS	04/11...	12/11/2019
Conception architecure projet	29/10...	01/11/2019
Implémentation des fonction...	13/11...	26/11/2019
Test	27/11...	28/11/2019
Réalisation et reviewing UML	08/10...	28/10/2019

APPLICATION

L'application de gestion de congés doit être réalisée en Java EE. Il s'agit d'une contrainte forte imposée au projet. L'objectif majeur de Java EE (ou Java Enterprise Edition) est de faciliter le développement d'applications web robustes et distribuées, déployées et exécutées sur un serveur d'applications. Pour ce projet, Tomcat est utilisé, un serveur HTTP qui gère les servlets et les JSP. Il a l'avantage d'être open-source et disponible sur les environnements disposant d'une machine virtuelle Java.

L'application a été développé en utilisant Git (avec Github) : https://github.com/fjourdren/projet-conges-JEE. Il contient en README la procédure permettant d'installer l'application sur un environnement vierge contenant un Tomcat et un Mysql.

BESOINS FONCTIONNELS

Voici les fonctionnalités qui seront implémentés dans l'application :

- **Tous les utilisateurs**
  - Authentification
  - Modification du mot de passe
  - Affichage demandes congés de l'employé
  - Création, modification, suppression demandes congés
- **Chef de service**
  - Affichage des (demandes de) congés du service
- **Personnel RH**
  - Affichage des (demandes de) congés de tous les employés
  - Valider ou non les demandes de congés des employés classiques
- **Responsable RH**
  - Affichage, création, modification, suppression employés
  - Affichage, création, suppression types congés
  - Valider ou non les demandes de congés du personnel RH et de ses propres demandes demandes

ARCHITECTURE DU PROJET

Pour assurer une programmation efficace, une architecture du projet a été mise en place :

- **java**
  - **db** : entités et DAOs
  - **servlets**
    - **auth** : servlets pour l'authentification
    - **dayoff** : servlets pour modifier les (demandes de) congés
    - **dayoff_type** : servlets pour modifier les types de congés
    - **employees** : servlets pour modifier les employés
- **webapp**
  - **template**
    - **components** : éléments JSP de base (utilisé par d'autres pages)
    - **Dayoff :** pages JSP pour modifier les (demandes de) congés
    - **DayoffType** : pages JSP pour modifier les types de congés
    - **Employees** : pages JSP pour modifier les employés
    - **includes** : resources JS/CSS/Fonts

GESTION DES DÉPENDANCES

Pour permettre une gestion des dépendances simple, nous avons mis en place Maven. Il permet via dans un fichier *pom.xml* d'installer automatiquement ces dépendances ainsi que toutes les dépendances des dépendances.

Voici la liste des dépendances utilisés :

- Hibernate
- Connecteur MYSQL
- Javax Servlet API

PERSISTENCE DES DONNÉES

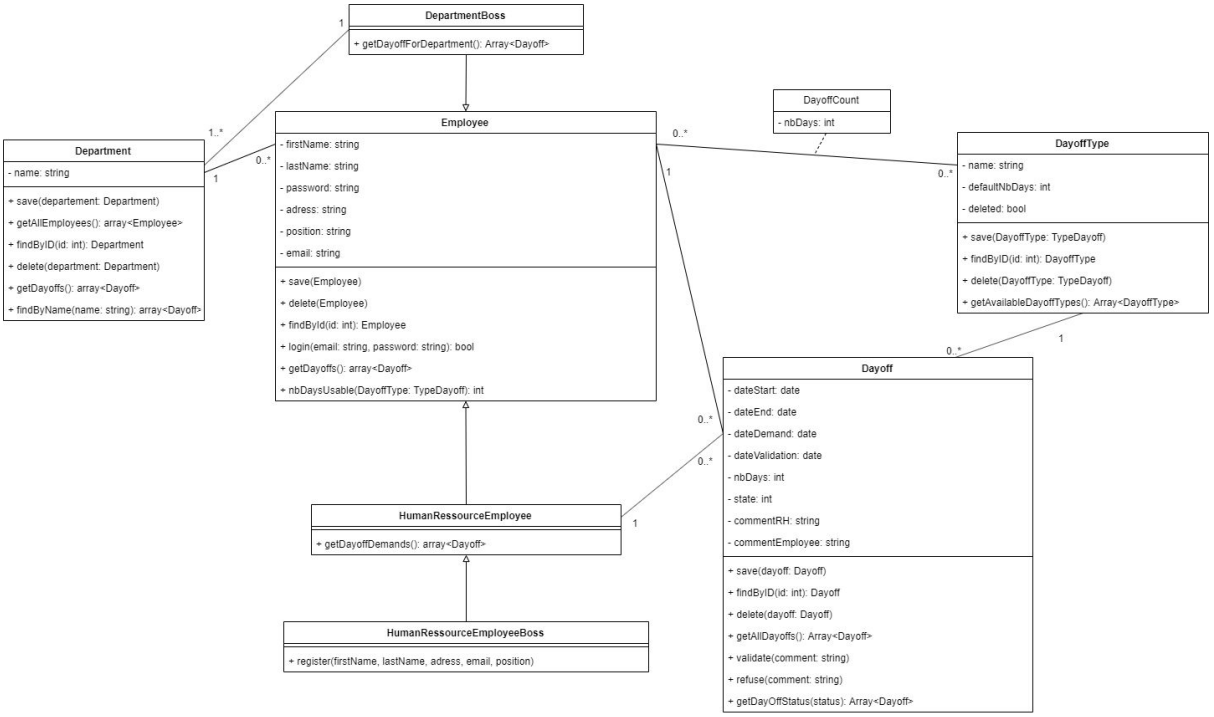
INTRODUCTION

Notre projet contient quatres entités (cinq tables SQL):

- **Dayoff** : (demandes de) congés
- **DayoffType** : types de congés
- **DayoffCount** : nombre de jours de congés restants par type de congé par employé
- **Employee** : employés
- **Department** : service

Nous avons décidé d'utiliser JPA (Java Persistence API), une API permettant de stocker des données via Java sans dépendre d'un type de base de données spécifique. Comme implémentation, nous avons choisi Hibernate, un ORM (object relational management). Cela permet d'avoir une couche d'abstraction entre la base de données et l'application. Notre projet utilise MySQL, mais il est possible d'utiliser n'importe quel type. Un autre avantage d'Hibernate est de ne pas avoir à créer de code SQL, Hibernate gère automatiquement tous les scripts (à part la création de la BDD, via *CREATE DATABASE*).

Voici le diagramme de classes :



Dans notre implémentation, les attributs des classes seront stockés dans des classes entités et les fonctions dans les DAO correspondants.

CONFIGURATION

Afin que JPA fonctionne, il faut créer un fichier de configuration nommé *persistence.xml*, dans le dossier *resources/META-INF*. Il contient :

- l'URL de la base de données
- le login et le mot de passe
- le pilote Java à utiliser (ici un pilote MySQL est utilisé)
- le paramétrage d'Hibernate
    - affichage ou non du SQL généré par Hibernate
    - dialect (type de SQL)
    - détection automatique ou non des classes entités
    - création automatique ou non des tables dans la base

DÉFINITION DES ENTITÉS

Pour l'utiliser, il faut dans un premier temps déclarer via des classes Java. Voici un exemple pour la classe **Department** :

```
@Entity
public class Department implements Serializable {

          private Long id;
          private String name;

          public Department() {}
          public Department(String name) {setName(name);}

          @Id
          @GeneratedValue
          @Column(nullable = false)
          public Long getId() {return id;}

          public void setId(Long id) {this.id = id;}

          @Basic
          @Column(nullable = false, unique = true)
          public String getName() {return name;}

          public void setName(String name) {
              this.name = Objects.requireNonNull(name);
          }
}
```

Lors du lancement de l'application, Hibernate va détecter automatiquement ces classes comme entités.

DÉFINITION DES DAO

Une fois la classe entité réalisée, il faut définir un DAO (Data Access Object), qui permettra de manipuler ces objets (requête, ajout, modification, suppression). Pour chaque DAO trois fichiers sont créés :

- une interface définissant les méthodes du DAO
- une implémentation avec JPA et la base de données
- une implémentation de test, qui fonctionne avec un tableau en mémoire

ACCÈS AUX DAO

Afin d'accéder au DAO, une classe nommé *DAOProvider* a été créée. Elle contient une instance unique (singleton) de chaque DAO. Elle permet également de choisir facilement le mode de fonctionnement (bdd réelle ou mock en mémoire).

DESIGN DE L'APPLICATION

GENERAL

Nous avons opté pour un design d'application sobre et utilisant des ressources open source en raison du temps imparti pour la réalisation du projet (majorité du temps consacré au développement JEE). C'est dans ce contexte que nous avons utilisé Bootstrap combiné à une feuille de style personnalisée pour certains éléments. Cette méthode nous a permis de réaliser rapidement et facilement le design de l'application tout en s'assurant que cette dernière était responsive (compatible avec différentes tailles d'écrans).

MOTEUR DE TEMPLATE

JSP est le composant de JEE qui permet de réaliser du templating de code HTML. Il permet par exemple d'inclure des portions de code depuis d'autres fichiers, d'afficher la valeur d'une variable. Nous nous sommes beaucoup appuyé sur les fonctionnalités de JSP pour avoir un template capable de s'adapter à la page tout en gardant une cohérence. De plus, notre utilisation permet de n'avoir qu'un endroit à modifier pour changer cet élément partout sur l'application.

Il y a plusieurs composants JSP qui sont utilisés sur toutes les pages :

- **head.jsp** : contient l'élément *head* de la page HTML ainsi que les fichiers JS et CSS à charger
- **footer.jsp** : contient le bas de la page
- **menu.jsp** : menu
- **flashMessages.jsp** : composant permettant d'afficher un message flash au début de n'importe quel page si besoin

SYSTÈME DE COMPOSANTS

Grâce à JSP, nous avons mis en place un système qui permet de charger le bon composant en fonction de l'URL demandé. Cela permet de limiter la quantité de code à réaliser et de facilement conserver une homogénéité dans le design de l'application.

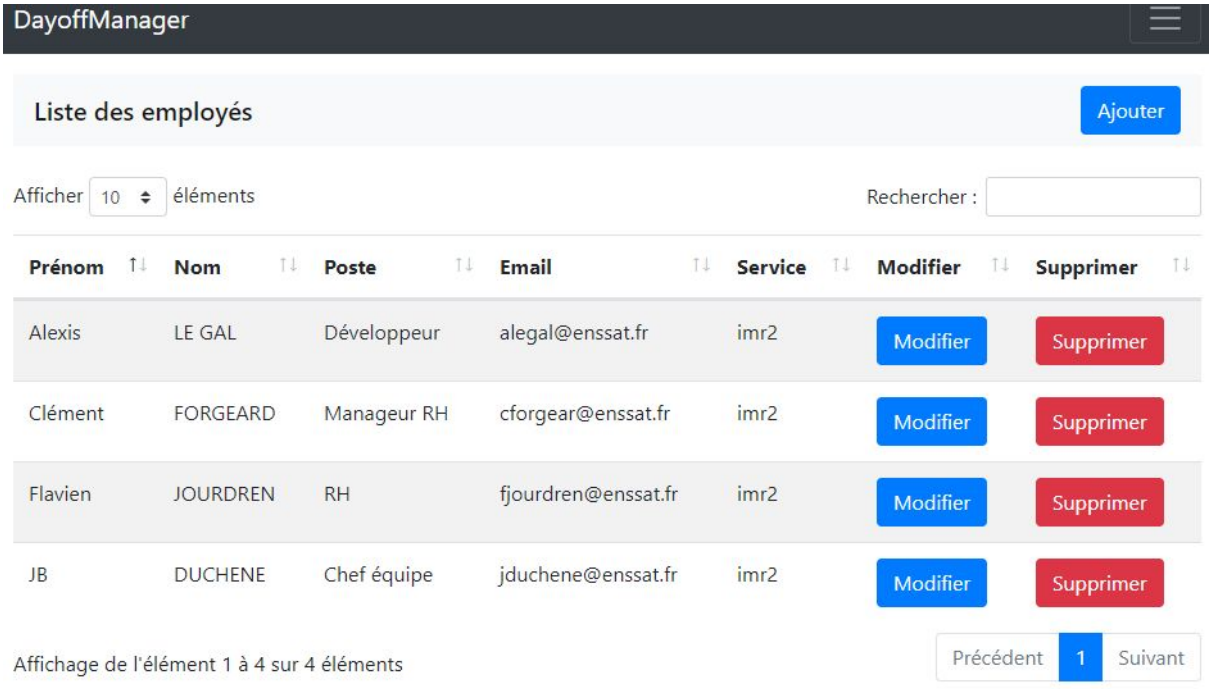
Après un appel sur l'URL, un servlet va être appelé et c'est ce dernier qui va préparer les données à afficher (stockées dans la base de données) puis indiquer le composant qu'il souhaite charger dans la page de l'utilisateur via le paramètre "*componentNeeded*" de l'objet requête. Enfin, le fichier JSP principal "index.jsp" est chargé, contenant le composant demandé.

CRUD

Un CRUD (**C**reate **R**ead **U**pdate **D**elete) est un composant permettant de manipuler les données stockées en BDD. Le composant étant similaire pour les différents CRUD à mettre en place (types de congés, employés), un composant JSP générique nommé *datatable.jsp* a été créé. Il utilise la bibliothèque JS *datatables* (https://www.datatables.net/) permettant d'afficher une liste avec des fonctionnalités de tri, de pagination et de recherche.

Ce composant fonctionne en créant côté servlet deux objets : un contenant les colonnes du tableau et un contenant les données du tableau. Il affiche également un bouton "Ajouter" et deux boutons "Modifier" et "Supprimer" pour chaque entrée.

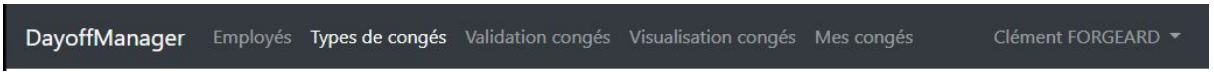
Voici un exemple de CRUD, celui pour les employés :



MENU

Pour la navigation dans l'application, nous avons mis en place un menu qui permet d'accéder à l'ensemble des fonctionnalités métiers de l'application. Le menu affiche une fonctionnalité uniquement si l'utilisateur peut l'utiliser. Il y a également deux options pour se déconnecter et changer son mot de passe. Enfin, si la fenêtre n'est pas assez grande pour afficher les éléments du menu sous forme d'une liste horizontale, il sera affiché verticalement.

Voici un exemple de menu pour un manager RH :



ROUTES

REQUÊTES ET RÉPONSES

JEE utilise comme la plupart des système MVC deux variables pour gérer le HTML affiché dans la navigateur Internet :

- **requête** : ensemble des données envoyées par l'utilisateur, utilisée également pour passer des données à la vue
- **réponse** : donnée envoyée à l'utilisateur (ex: page à afficher)

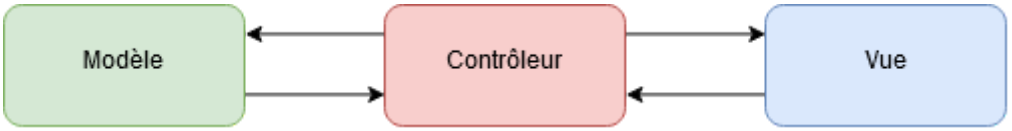
Une servlet génère une réponse en fonction de la requête du client.

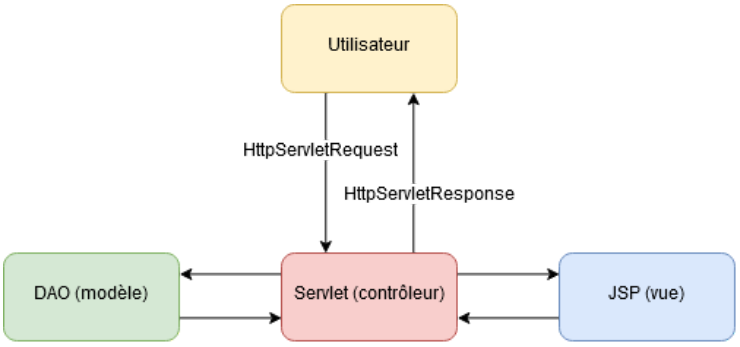
MVC

L'architecture MVC est une architecture d'application qui vise à séparer la partie chargée de gérer les objets (Modèle), la partie qui gère l'affichage (Vue) et la partie qui gère la préparation des données (Contrôleur). Cette architecture permet de bien séparer les différents éléments et de pouvoir ainsi avoir une conception claire et efficace grâce à la séparation des données, un gain de temps de maintenance et une plus grande évolutivité.

Ainsi, dans notre projet JEE, on a :

- les modèles: DAO / classes (qui traitent les données, les enregistrent dans la base de données)
- les vues: Les fichiers JSP (contenant de l'HTML).
- les contrôleurs: Les servlets (qui préparent les données et font le lien entre les modèles et les vues).





INSCRIPTION DES COMPOSANTS

Par défaut, tous les composants JEE créés dans l'application doivent être inscrits dans le fichier */webapp/WEB-INF/web.xml* en définissant la classe utilisée et l'URL associée à cette classe. Cette méthode a un gros défaut : il devient de plus en plus compliqué au fur et à mesure du développement de l'application; il est également une source de conflits avec Git. Après quelques recherches, nous avons vu qu'il était également possible d'utiliser des annotations. Par exemple, pour le servlet listant les employés :

```
@WebServlet(
        name = "EditEmployeeServlet",
        description = "EditEmployeeServlet",
        urlPatterns = {"/employees-add-edit"}
)
public class EditEmployeeServlet extends HttpServlet {
        [...]
}
```

Grâce à cette méthode, le code est plus simple à lire. De plus, l'URL associée au servlet est stockée dans le servlet et non plus dans un fichier externe, rendant cette information plus simple à trouver et à modifier si besoin.

ROUTES IMPLÉMENTÉES

Voici la liste des routes implémentées dans l'application :

URL	Description
`/change-password`	Modification du mot de passe
`/login`	Authentification
`/logout`	Déconnexion
`/dayoff-timeline`	Affichage des congés pour tous les employé
`/dayoff-timeline?teamName=NOM_EQUIPE`	Affichage des congés pour une équipe
`/rh-dayoff-list`	Liste des demandes de congés en attente de validation
`/rh-dayoff-edit?id=ID`	Validation ou non d'une demande de congés
`/manage-my-dayoffs`	Interface pour ajouter/modifier/supprimer/voir ses propres (demandes de) congés
`/dayofftype-list`	Liste des types de congés
`/dayofftype-add-edit` `/dayofftype-add-edit?id=ID`	Ajout/modification d'un type de congés
`/dayofftype-delete?id=ID`	Suppression d'un type de congés
`/employees-list`	Liste des employés
`/employees-add-edit` `/employees-add-edit?id=ID`	Ajout/modification d'un employé
`/employees-delete`	Suppression d'un employé
`/stat-overview`	Affichage de statistiques

AUTHENTIFICATION

L'authentification mise en place est relativement simple. Une fois que l'utilisateur s'est authentifié avec son email et son mot de passe, l'application va stocker dans la session côté serveur l'instance de la classe *Employé* représentant l'utilisateur. Lorsque l'utilisateur tente d'afficher d'une page, un *Filter* nommé *AuthFilter* sera appelé. Si l'utilisateur n'a pas les droits nécessaires, l'appel au *Servlet* sera bloqué.

VALIDATION AUTOMATIQUE DES DEMANDES DE CONGÉS

Pour faire en sorte que les demandes de congés soit validés au bout de deux jours, nous avons mis en place en place un *Listener*. C'est une classe qui contient deux fonctions, appelés lorsque l'application démarre ou s'arrête. Dans notre cas au démarrage de

l'application un Timer sera configuré pour être exécuté toutes les heures et valider les demandes de congés créés il y a plus de deux jours.

PROBLÈMES RENCONTRÉS

- Mise en place de l'environnement de développement sur chacun des postes
- Manque de certaines fonctionnalités dans JSP, comme un système d'include plus avancé qui permet d'inclure un fichier "par le dessus" plutôt que par le dessous, cela permettrait de créer facilement un fichier commun à toutes les pages qui contiendraient par exemple l'entête, les menus et le pied de page.

AMÉLIORATIONS POSSIBLES

- Plateforme multilingue
- Système de récupération de mot de passe
- CRUD pour les services de l'employé

CONCLUSION

D'un point de vue technique, ce projet nous a permis de découvrir de nombreuses technologies : Java EE, les pages JSP, JPA et Hibernate, … Il nous a permis également de mettre à jour nos connaissances concernant le système de gestion de version (GIT).

D'un point du vue gestion de projet, ce projet nous a permis d'améliorer nos compétences concernant le travail en équipe, la répartition des tâches. Nous avons cherché à voir ce projet comme un projet à caractère professionnel, ce qui nous permettra pour notre prochaine période en entreprise d'être plus efficace.