



DOSSIER-PROJET

Nom de naissance ▶ COURIOL
Prénom ▶ Fred
Adresse ▶ 6, Av. de Villeneuve Saint-Georges
94600 CHOISY LE ROI

Titre professionnel visé

Développeur-se logiciel - Niveau III

Sommaire

Remerciements	4
Résumé du projet en anglais	5
Liste des compétences du référentiel	6
Cahier des charges ou expressions des besoins de l'application à développer	9
a) Gestion des utilisateurs	12
b) Gestion des absences	13
c) Circuit de validation des demandes	14
d) Echanges par mail	16
Spécifications fonctionnelles	18
Spécifications techniques	27
Réalisations	29
a) L'étude	29
b) La réalisation	31
1 - Conception générale	32
2 - Préparation	35
3 - Mise en place de la base de données	36
4 - Réalisation du Back End	36
5 - Réalisation du Front End	44
6 - Conditionnement et packaging	48
Conclusions	51
Annexes	53

REMERCIEMENTS

Je tiens à remercier Nicolas HOVART et Emmanuel LE PEVEDIC qui, tout au long de notre cursus, nous ont formés, accompagnés et nous ont apportés un très bon capital de connaissances et de méthodes.

Remerciements à Simplon et à tous ses employés, particulièrement à :

- Laëtitia AMOUROUS pour son implication et son dévouement à nous mettre dans les meilleures conditions.
- Sarah GUERRIC qui a contribué à nous faire découvrir ReactJS en un temps court tout en nous captivant.
- Elise FRAISSE pour son accompagnement et sa bonne humeur communicative.
- Erwan KEZZAR pour ses nombreux conseils et encouragements et ce, depuis le jour des entretiens de sélection.
- David OSTERMANN, Eric SIBER, Yvan SRAKA, Mehdi MAIZATE, Enzo ESTEVES, Norbert NADIR, Victor HANSEMANN qui ont aussi, chacun à leur manière, amené leurs pierres à notre édifice.

Merci à tous les simploniens car les échanges sur Slack sont une source d'informations et d'inspirations.

Remerciements à La Poste qui a permis que cette aventure ait lieu en particulier à Charles et Nelly qui nous ont chaperonnés.

Mention spéciale à la Promo 1, « les précurseurs », nous avons, par nos différents échanges et entraides, su avancer et grandir ensemble, merci pour cette belle aventure humaine.

Un remerciement particulier à mes collaborateurs de projet, Jean LEFRANÇOIS et Mokhtar KHIDER, avec lesquels j'ai eu plaisir à travailler.

RESUME DU PROJET EN ANGLAIS

The goal of our project is to realize an application allowing the management of the absences in a company based on a workflow of e-mail validation.

To insure the functioning, 5 types of different actors intervene each having a distinct role, basically after authentication:

- Any employee can formulate a requirement of absence.
- His team leader can validate or refuse the demand. He can also indicate irregular absences of any employee of his team.
- The Human Resources verify the validity of the demand.
- The administrator manages the database of the employees.
- The department of pay which receives an extraction allowing it to establish pay sheet.

Every employee can consult his calendar as well as that of his team. The graphical representation permits to distinguish every type of absence according to its color.

- The highlighted absences are mainly the paid leaves, RTT, the rest days and the sick leaves.
- The not validated demands will be materialized by a shaded off these colors.

To realize this project, our team of 3 persons developed the design-pattern Model View Controller. We chose to develop mainly in Java for the Back End and ReactJS for the Front End and in parallel, we established our database in MySQL.

Please note: The Back End is built in API REST and the Front End is using Single Page Application principle.

LISTE DES COMPETENCES DU REFERENTIEL

❖ Compétences du référentiel

Dans le cadre de notre formation, nous avons eu à pratiquer dans différents domaines afin d'atteindre le but recherché, l'acquisition de chacune des compétences suivantes ce qui nous a permis de les mettre en œuvre dans le cadre de notre projet.

✓ **Maquetter une application**

Consiste à modéliser un projet et de ce fait, permettre une meilleure approche lors du développement en découlant. Pour mener à bien l'opération, nous avons appris à réaliser différents diagrammes UML tels que :

- Les **Cas d'utilisation** relatifs au Qui fait quoi ? C'est-à-dire l'aspect fonctionnel lié aux différents acteurs et rôles.
- Le diagramme de **Séquences** qui propose le déroulement des actions entre les acteurs et les objets du projet.
- Le diagramme de **Classes** qui décrit le comportement et les relations entre les objets du projet.
- Le diagramme d'**Activité** qui représente le déroulement des actions en fonction des événements ou interactions.
- Les **Mockups** qui sont une représentation grossière de l'ensemble des vues de l'interface utilisateur (voir exemples en annexe).

Pour l'ensemble de ces diagrammes, nous avons utilisé le logiciel StarUML pour les 3 premiers diagrammes et Balsamiq pour les 2 autres.

✓ **Concevoir une base de données**

Dans le prolongement du maquetage d'application, nous avons appris à réaliser le **modèle de données** (ou diagramme de données) en méthode Merise permettant ainsi de représenter la maquette de la future base de données du projet. Nous avons opté pour le langage SQL et de ce fait, inspiré du diagramme de Classes et à l'aide du logiciel MySQL Workbench nous avons pu maquetter notre base de données.

✓ **Mettre en place une base de données**

Afin de mettre en œuvre, il a fallu au préalable installer le serveur SQL, MySQL Server (ou PHPMyAdmin) puis à l'aide de MySQL Workbench, nous avons exporté le modèle de données vers la requête en langage SQL permettant la création de la base.

Concernant les données, nous avons créé 3 requêtes d'ajout de données exploitables permettant de tester le rendu de l'application et de tester son comportement selon la charge (Les 3 requêtes créées permettent la création de la base simple, la base à mi charge et la base chargée).

A noter : Les dates de notre base sont au format ISO pour rester conforme au standard c'est-à-dire au format YYYY-MM-DD, nous les avons adaptés à notre convenance dans notre Back End.

✓ **Développer une interface utilisateur**

Nous avons choisi de créer une interface Web conçu en ReactJS qui permet facilement la réalisation de Single Page Application grâce à son DOM virtuel.

A noter : ReactJS s'appuie sur un serveur Node JS.

✓ **Développer des composants d'accès aux données**

Durant notre formation, nous avons procédé à l'accès aux données (objet Model associé à un DAO) par le biais de JDBC et des Statements mais par la suite et c'est aussi le cas dans notre projet, nous avons opté pour une persistance par le biais d'un ORM (Object-Relational Mapping) qui réalise les relations entre les objets et la table relationnel. JPA (Java Persistence API) est l'ORM choisi, il s'appuie sur Hibernate et repose essentiellement sur des annotations.

✓ **Développer des pages web en lien avec une base de données**

Il s'agit d'un lien indirect qui se décompose plutôt en un lien Front End/Back End et un lien Back End/Data. En complément des composants d'accès aux données vu précédemment, matérialisant le lien Back End/Data, pour compléter cette compétence, il faut ajouter au Front End une API REST établissant le lien Front End/Back End. L'ensemble permet le lien avec une base

de données.

Par exemple dans le cas de notre projet en design-pattern MVC, le Front End formule des requêtes au contrôleur qui, au travers du modèle, les transmet à la base de données.

✓ **Développer une application simple de mobilité numérique**

Nous avons étudié et pratiqué le principe des pages Responsive, l'orientation mobile adaptive et le concept Mobile first. L'usage de Bootstrap nous a été très utile dans ce contexte. Dans le cadre de notre projet, nous nous sommes efforcés à rendre Responsive l'ensemble des pages de notre Front End.

✓ **Utiliser l'anglais dans son activité professionnelle en informatique**

Yes of course.

✓ **Compétences transversales de l'emploi type**

- ✓ Actualiser et partager ses compétences en développement informatique.
- ✓ Organiser son activité en développement informatique.

Nous avons tout au long de notre cursus employé régulièrement différents canaux de partages de compétences comme les restitutions, le live-coding, les échanges et l'entraide. Concernant notre projet, nous avons privilégié les canaux suivants :

- Trello, notre kanban pour l'avancement et le suivi du projet (voir en annexe).
- Slack à 3 participants pour la communication.
- Lync et Lync attendee pour nos réunions hebdomadaires.
- Github pour le travail collaboratif (une branche master et une branche par développeur, voir en annexe).
- Echanges inter-projets, le but étant de s'imprégner voire de s'inspirer du travail accompli par les autres groupes de projet.

CAHIER DES CHARGES OU EXPRESSIONS DES BESOINS DE L'APPLICATION A DEVELOPPER

L'expression des besoins d'après l'énoncé fournie par notre équipe de formation :

La société Simplon veut optimiser sa gestion du personnel, et notamment la gestion des absences. Pour cela, elle veut mettre en place un circuit de validation ainsi qu'un logiciel permettant de valider ce circuit. L'objectif est de mieux gérer son personnel et éviter des sous effectifs.

Le salarié se connectera avec son adresse Gmail et un mot de passe. Un premier mot de passe lui sera attribué, qu'il devra modifier à la 1° connexion. Il remplira un formulaire pour sa demande de congés. Un mail sera automatiquement envoyé à son N+1, avec un lien de validation ou de refus des congés. Le salarié sera averti par mail de la réponse. S'il y a refus, la demande sera enregistrée dans la base avec la mention refus et le circuit s'arrête. Si la demande est acceptée, elle sera affichée dans Google Calendar. Le N+1 aura la possibilité de visualiser les congés de ses subalternes sur une période donnée. Le N+1 aura aussi la possibilité de saisir des congés maladies ainsi que des absences irrégulières.

Un récapitulatif mensuelle sera générée pour le calcul de la paye.

Un mail sera aussi envoyer à la responsable RH pour information et vérification des règles RH.

Un SysAdm gèrera les différentes bases de données.

Sa page d'accueil sera composé d'un récapitulatif de ses congés restant à prendre ainsi que d'un formulaire de demande de congés.

Il aura aussi accès à ses informations personnelles qu'il pourra demander à faire modifier par la responsable RH, en cas d'erreurs et à un récapitulatif de ses absences depuis le 1° janvier.

Identification des acteurs :

- **Employé** : La personne qui fait sa demande de congés. Elle peut aussi visualiser le solde de ses différentes absences. Niveau 0.
- **N+1 (Valideur)** : Généralement le responsable d'équipe, c'est la personne qui valide la demande de congés des personnes sous ses ordres. Elle saisit aussi certaines absences et peut faire une demande de congés. Niveau 1.
- **Secrétaire RH (Valideur RH)** : La personne qui vérifie d'un point de vue RH, la validité des congés. Elle peut les modifier ou les annulés. Elle peut aussi faire une demande de congés.
- **Administrateur de la base de personnes et d'équipes** : Il saisit les nouveaux arrivants et gère les problèmes pouvant survenir.
- **Service paye** : C'est un acteur externe qui reçoit un état mensuel des congés pour établir la paye.

Liste des absences prise en compte :

- Congé payé
- Congé maladie
- RTT
- Repos compensateur
- Autres cas d'absence
 - Absence irrégulière
 - Congé exceptionnel (raison familiale, fête religieuse, etc...)
 - Congé sabbatique
 - Autres

Liste des fonctionnalités :

- *Identification*
- *Administration des employés*
- *Création de la chaîne de validation*
- *Demande de congé*
- *Suivi de la demande*
- *Validation de la demande à chaque niveau*
- *Consultation des demandes pour une équipe*
- *Envoi de mail*
- *Maj du calendrier google agenda*
- *Création d'un rapport pour la RH chaque mois.*

Circuit de validation des demandes d'absence :

- *Le demandeur formule une demande.*
- *L'enregistrement de la demande entraîne l'envoi d'un mail de demande contenant un lien sur la décision vers le valideur avec son backup en copie.*
- *Le valideur ou son backup refuse, un mail de refus est transmis au demandeur et le workflow s'arrête là.*
- *Le valideur accepte, un autre mail est transmis au RH pour prise en compte.*

Circuit de validation des déclarations d'absence :

- *Le responsable signale une absence inopinée.*
- *L'enregistrement de la demande entraîne l'envoi d'un mail de déclaration d'absence à son service RH pour prise en compte.*
- *Le service RH refuse, un mail de refus est transmis au responsable et le workflow s'arrête là.*
- *Le service RH accepte, un autre mail est transmis au responsable (avec l'employé en copie) pour prise en compte.*

a) Gestion des utilisateurs

Les utilisateurs de l'application doivent s'authentifier à l'aide de leur adresse mail ainsi que de leur mot de passe donnant accès aux environnements auxquels ils ont droit.

Il revient à l'administrateur de l'application de gérer la base des personnes. Il peut ainsi créer, modifier ou supprimer des utilisateurs.

Un utilisateur se définit par son nom, son prénom, son matricule, son adresse mail, son ou ses rôles, l'équipe et le service RH de validation dont il dépend.

En cas de création, le mot de passe et le compteur de congés sont définis à leurs valeurs par défaut. A noter que tous les utilisateurs ont à minima le rôle de demandeur.

L'administrateur a aussi à sa charge la gestion des équipes ainsi que des responsables, essentiel pour permettre l'affichage du calendrier d'équipe.

A noter : Le rôle d'administrateur ne sous-entend pas avoir des privilèges supérieurs aux autres, il n'est, par exemple, pas censé valider des demandes sauf si il cumule le rôle de valideur.

b) Gestion des absences

Différents type d'absences sont gérés par l'application, ainsi, un employé peut formuler une demande d'absence selon le type parmi les congés payés, les RTT, les repos compensateur (récupération suite à heures supplémentaires) ou les congés exceptionnels (maternité, paternité, fête religieuse, évènement familial,...).

Un responsable d'équipe ou le service RH peut également déclarer l'absence inopinée d'un employé dont il a la charge. Dans le cas où un employé est en absence irrégulière ou en congé maladie, son responsable ou son service Rh peut signaler l'absence dans l'application pour prise en compte. Il est aussi possible de signaler une absence légitime dans le cadre d'un oubli de l'employé ou d'une régularisation.

Remarque : Les compteurs de congés ou de RTT sont exprimés en jour ce qui n'est pas le cas pour le repos compensateur qui est exprimé en heure.

La pose de congé engendre un décompte en fonction de la durée tronquée des jours non travaillés puisque l'application sait faire la différence entre les jours travaillés et ceux non travaillés que sont les jours fériés et les jours de repos hebdomadaire.

A ce propos, les jours reconnus comme fériés, au nombre de 13 (11 officiels + Pâques + Pentecôte), sont par défaut des jours non travaillés mais selon les conventions propre à une entreprise, il est possible de définir un ou plusieurs jours fériés comme jours travaillés.

De même, par défaut, les 2 jours du week-end sont considérés comme jours de repos hebdomadaire mais il est possible de redéfinir ces jours de repos hebdomadaire.

c) Circuit de validation des demandes

Un employé soumet une demande d'absence par le biais du formulaire prévu à cet effet.

La prise en compte de la demande entraîne automatiquement l'envoi d'un mail destiné au valideur et à son backup en copie. Ce mail contient un lien permettant de se prononcer sur l'opportunité de la demande. En parallèle, un mail notifiant la prise en compte de la demande est envoyé au demandeur.

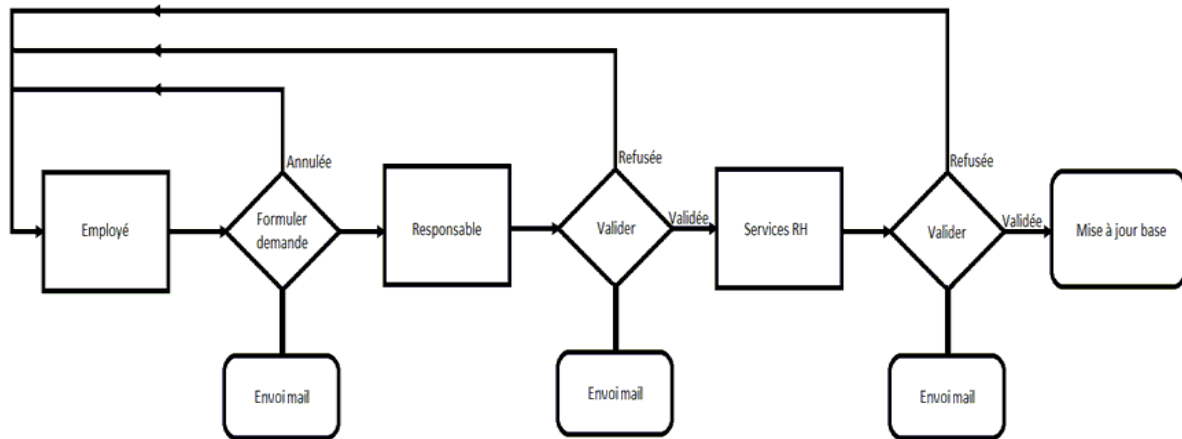
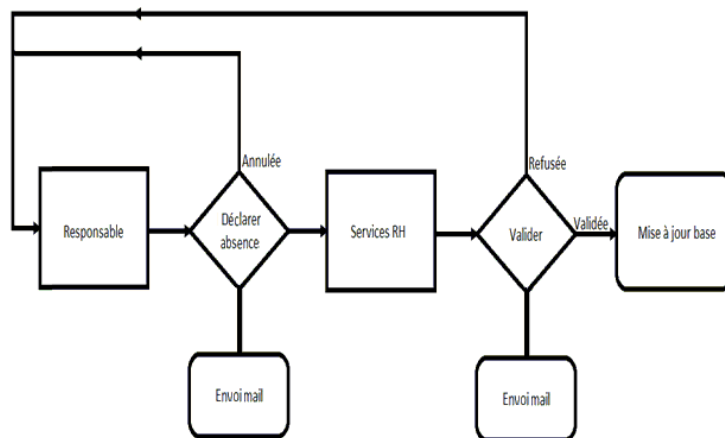
Si la demande est validée, deux mails sont transmis, le premier au demandeur pour lui rendre compte de la validation et le deuxième, destiné au service RH pour prise de décision. De même que pour le responsable hiérarchique, le mail destiné au service RH contient un lien lui permettant de se prononcer.

En cas de validation du service RH, le demandeur, son responsable et son backup en sont notifiés par mail, la base de données est actualisée en conséquence. Une tâche est envoyée à l'agenda du demandeur afin d'y faire figurer la période d'absences.

Dans le cas où le valideur ou son backup ou les RH oppose un refus, un mail signifiant le refus et mentionnant le motif du refus est transmis au demandeur. Cela entraîne, par la même occasion, l'arrêt du workflow.

Le demandeur peut à tout moment suivre le workflow de sa demande. Il a aussi la possibilité de relancer sa demande restée en attente de validation.

Il peut aussi annuler les demandes en attente de décision ou celles validées à date de début non échue.

Circuit de validation par le schéma :**Demande d'absence****Déclaration d'absence**

Remarque : Le demandeur est notifié par email à chaque changement d'état de sa demande. Il peut aussi suivre l'état de sa demande en consultant le tableau de ses dernières demandes.

d) Echanges par mail

Exemple de mail transmis par l'application :

De : Gestion des absences <gestion.absences75@gmail.com>

Date : 02/06/2017 13:30 (GMT+01:00)

À : stephanie.lambert@gmail.com

Cc : lionel.rouviere@gmail.com

Objet : Demande d'absence N° DEM012768.

Bonjour,

L'employé(e) DUPONT Arnaud dont vous êtes le ou la responsable, demande à bénéficier de congés payés du 12/06/2017 au 16/06/2017.

En qualité de responsable, merci de bien vouloir étudier la demande référencée sous le N° [DEM012768](#) et d'y apporter votre validation ou votre refus **en cliquant sur le lien**.

Nous attirons votre vigilance sur l'importance et votre engagement concernant votre validation ou votre refus.

Cordialement,
Service de gestion des absences.

Ce message vous a été envoyé automatiquement, merci de ne pas répondre.

L'exemple ci-dessus concerne une demande d'absence formulée, soumise à décision du responsable hiérarchique ou de son backup. Le responsable ou son backup, en cliquant sur le lien contenu dans le mail, accède au formulaire de décision (sous couvert de l'authentification du décideur concerné).

Nota bene : Le lien du mail propose un accès direct au formulaire de décision, il reste, pour autant, la possibilité d'y accéder par le biais du lien prévu à cet effet dans la liste des dernières demandes.

Tous les mails transmis par l'application sont émis à partir de l'adresse mail « gestion.absences75@gmail.com » qui a été créée pour le projet.

Voir récapitulatif des échanges par mail en annexe.

SPECIFICATIONS FONCTIONNELLES

L'application de notre projet doit être réalisée sous la forme d'un site internet proposant des pages web interagissant avec une base de données.

L'accès à l'application doit être sécurisé, chaque intervenant doit s'authentifier pour accéder aux pages auxquelles il a droit.

Tous les intervenants de l'application auront, à tout moment, la possibilité de consulter l'aide adaptée en fonction de son rôle.

L'application doit également pouvoir intégrer des fonctionnalités d'envoi de mails dans le cadre du workflow ainsi que d'ajout de planifications dans l'agenda des demandeurs dans le cadre du suivi sur leur mobile.

Le rôle particulier du valideur backup peut être utile en cas de défection du responsable. En l'absence du responsable ou par délégation de celui-ci, le valideur backup peut se substituer au valideur attitré afin de se prononcer sur la décision à donner à la demande dont il a la charge.

Chaque employé a la possibilité de visualiser son calendrier ainsi que celui de son équipe. Les services RH peuvent visualiser le calendrier de l'ensemble des employés de l'entreprise.

Dans le cas particulier où l'employé est responsable, en guise de calendrier d'équipe, il visualise à la fois celui de l'équipe qu'il encadre, celui de ses collaborateurs et celui de son propre responsable.

L'application propose une visibilité sur 3 mois pour le calendrier individuel et d'un mois et demi pour celui de l'équipe. L'utilisateur peut faire usage des boutons de navigation permettant d'avancer ou de reculer d'un mois.

Chaque type d'absence est distingué par une couleur (qui est estompée pour les demandes en cours de validation et qui le restent tant que la décision finale n'a pas eu lieu).

Les jours de repos hebdomadaire ainsi que les jours fériés sont aussi matérialisés par une couleur distinctive.

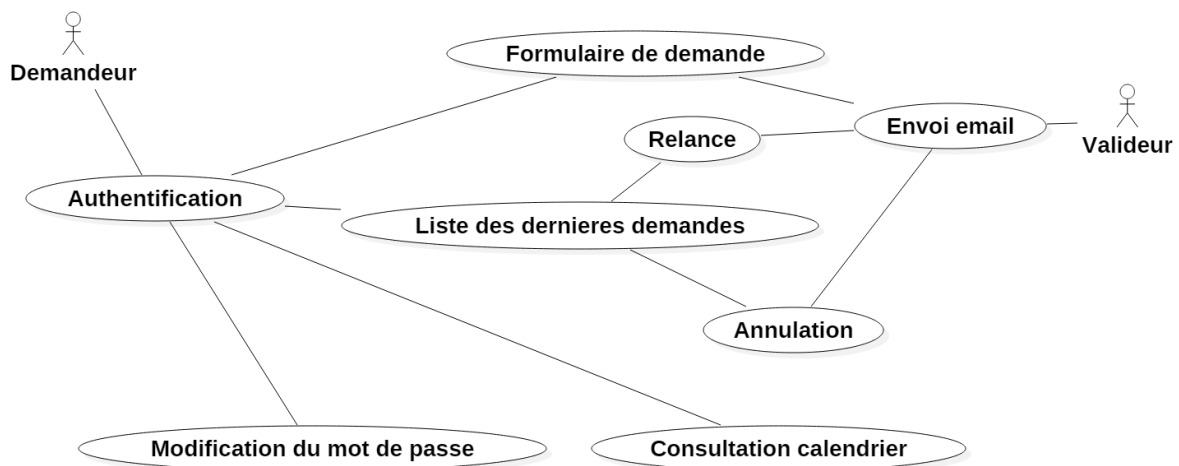
Un employé a la possibilité de consulter son calendrier ainsi que celui de son équipe comprenant ses collaborateurs et son responsable. Le but du calendrier d'équipe est de fournir aux employés une visibilité sur les forces en présence avant de formuler une demande.

Remarque : L'application est en mesure de tenir compte des jours fériés et des jours de repos hebdomadaire permettant ainsi le décompte uniquement des jours travaillés lors de la prise en compte d'absences.

Diagrammes des cas d'utilisation

Les cas d'utilisation permettent de représenter le comportement fonctionnel par utilisateur.

- Cas d'utilisation du demandeur :



- Cas d'utilisation du valideur (partie décision) :

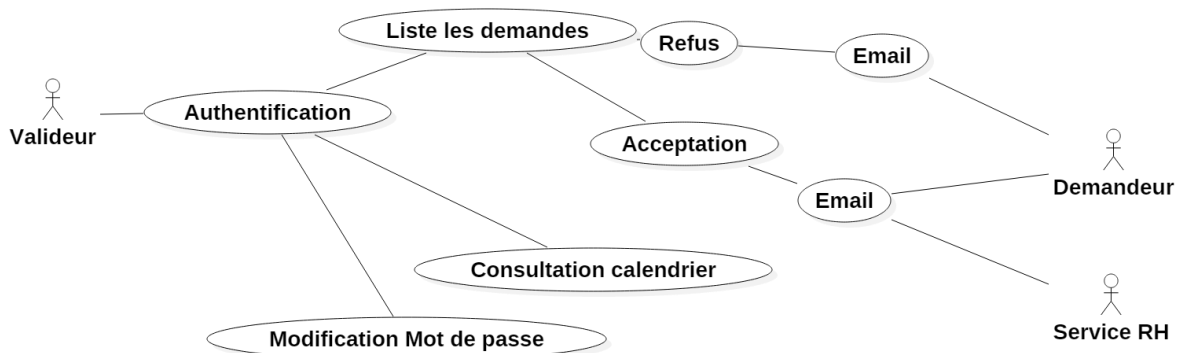
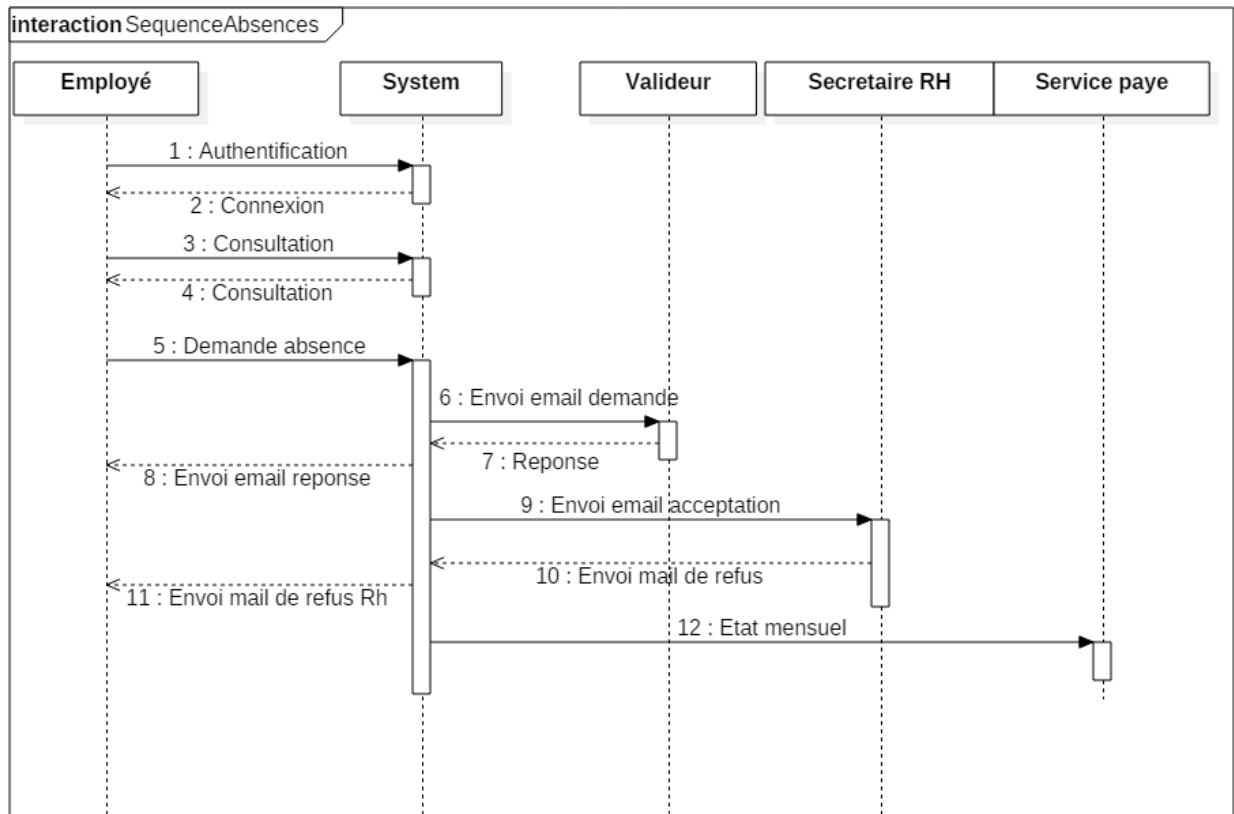


Diagramme de séquence

Permet de représenter les interactions entre les acteurs et le système et de visualiser l'enchaînement des actions chronologiquement.

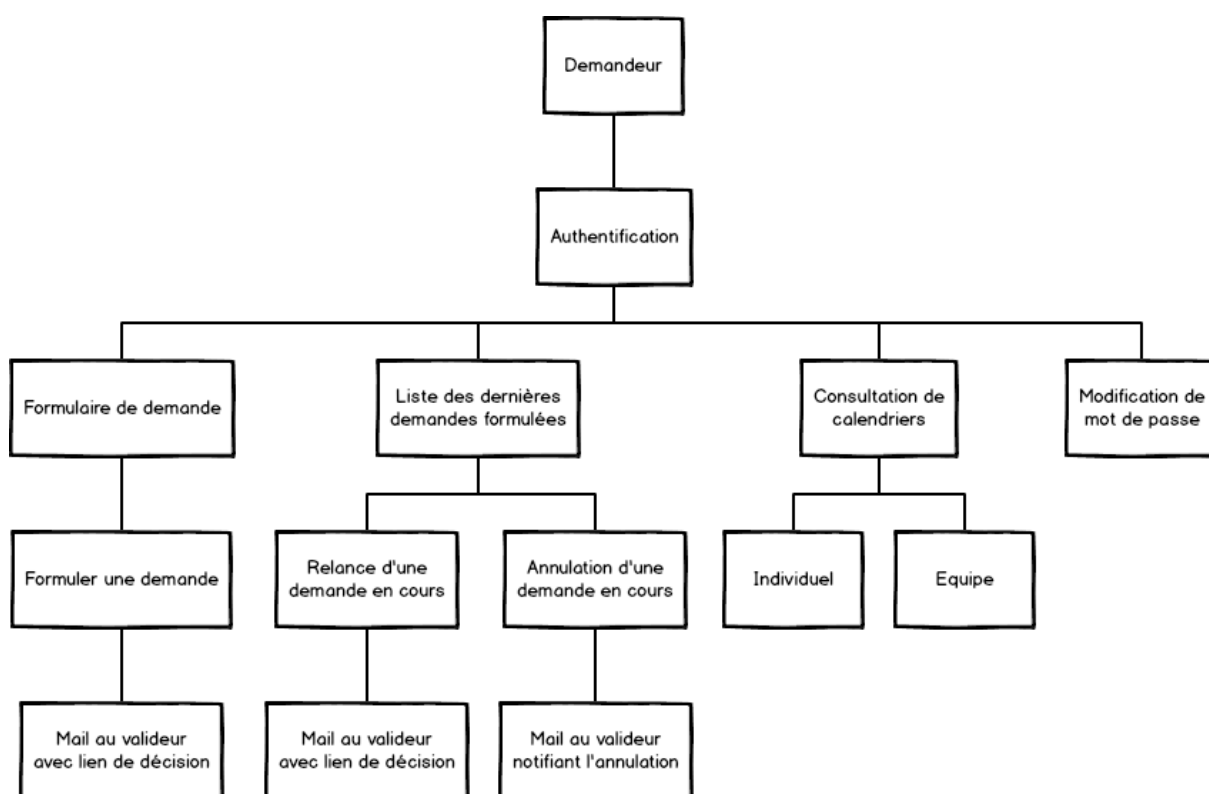


Dans le cas de notre projet, le diagramme fait ressortir l'enchaînement des actions des différents acteurs et les opérations déclenchées par le système (particulièrement les transmissions d'emails).

Diagrammes d'activité

Les diagrammes d'activité permettent de modéliser les possibilités de navigation et champs d'action que peuvent avoir les utilisateurs en fonction de leur rôle.

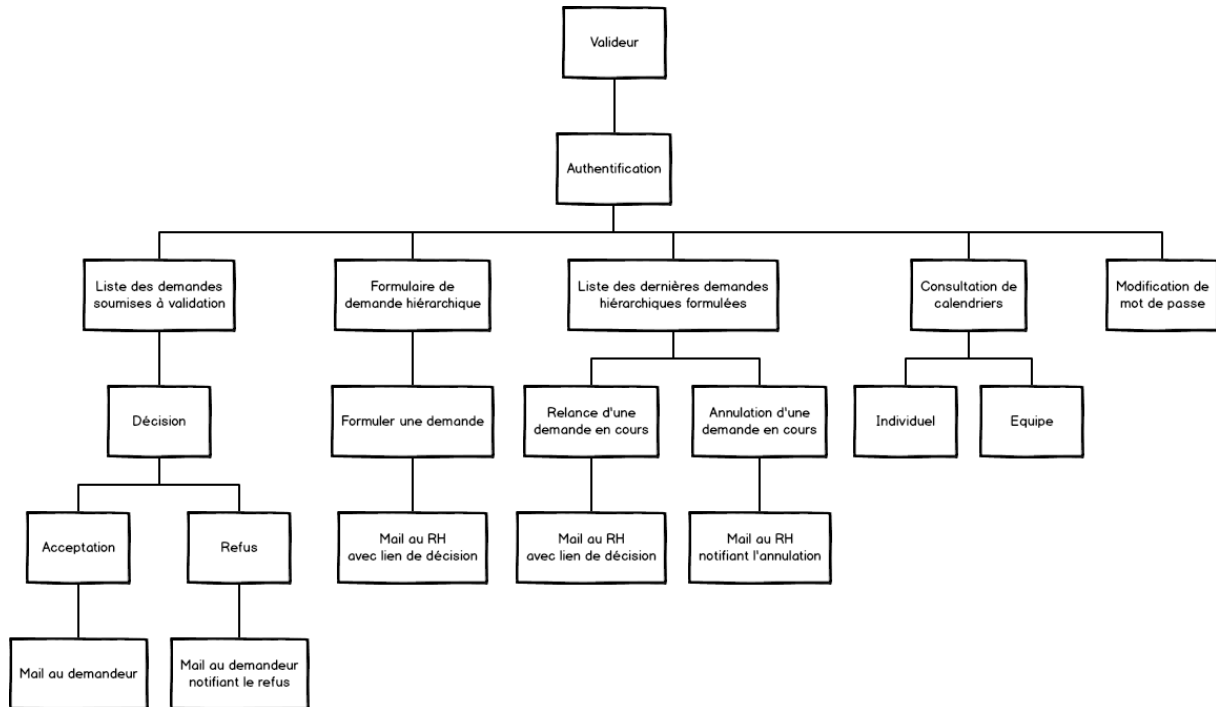
- Demandeur :



Les opérations liées au demandeur concernent tous les employés de l'entreprise. Les utilisateurs qui disposent d'autres privilèges les cumulent avec le rôle de l'employé.

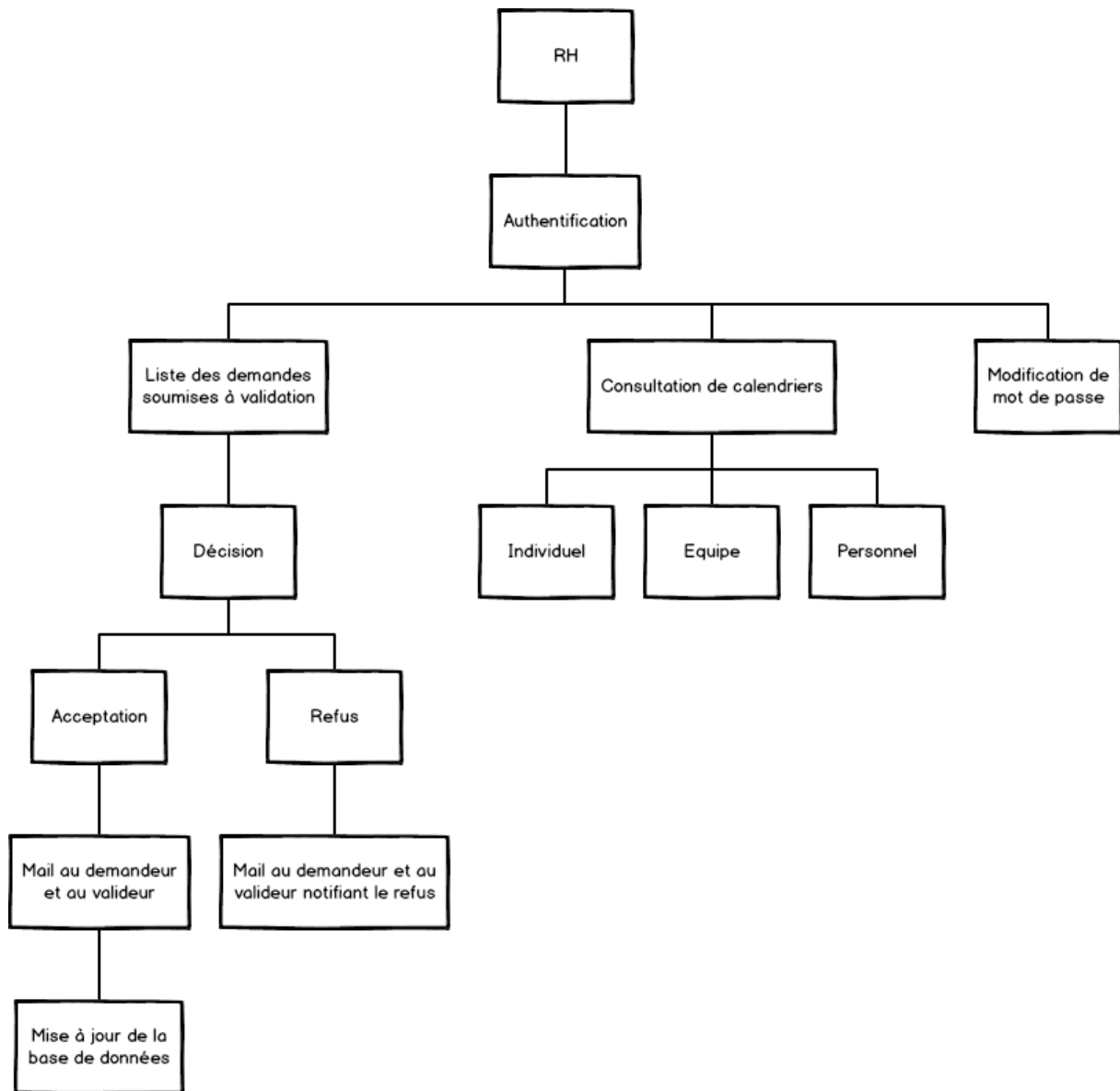
A noter : Un utilisateur peut cumuler plusieurs rôles ainsi, un utilisateur peut à la fois être Valideur, Valideur RH, Administrateur et de facto Employé à condition qu'il ne soit pas son propre valideur.

- **Valideur :**



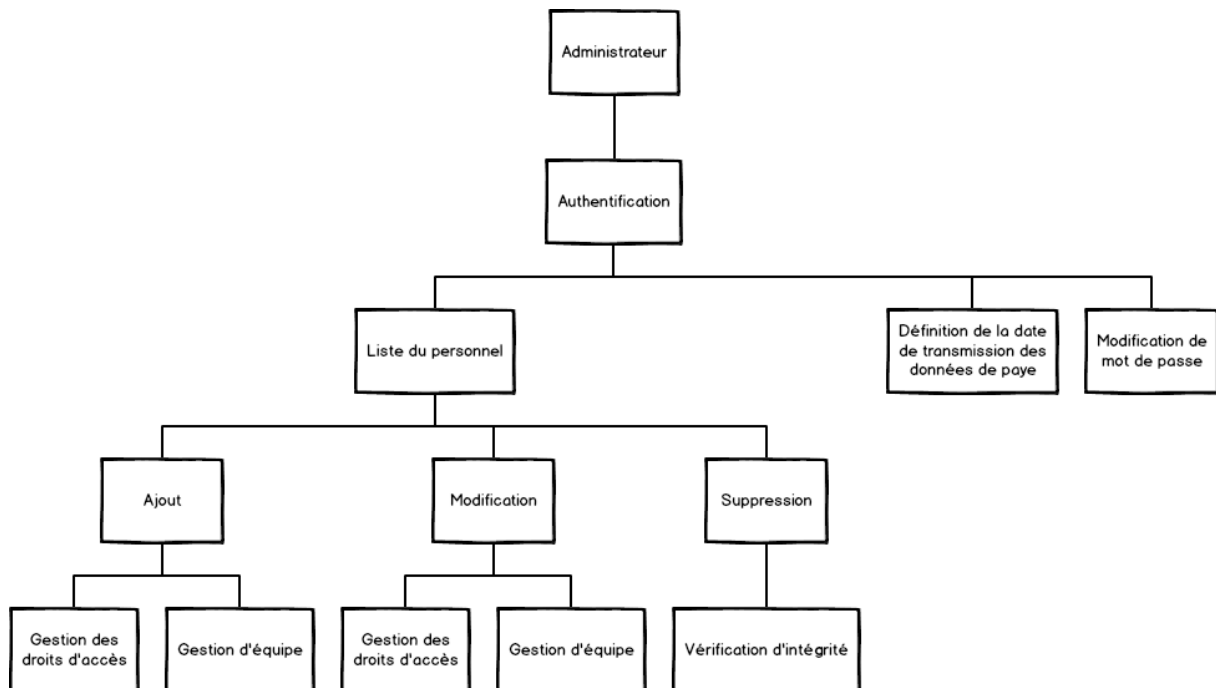
Le diagramme du valideur (ou responsable hiérarchique) est aussi valable pour son Backup. Sous-entendu que le backup a le même rôle et privilèges que le valideur attribué.

- **Valideur RH :**



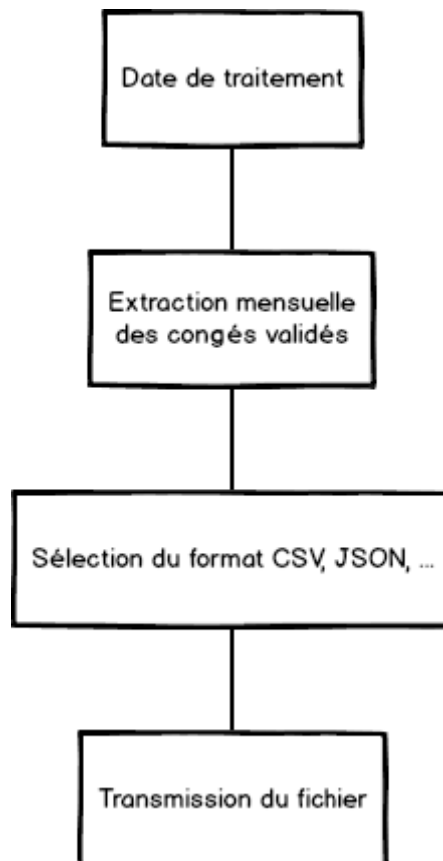
Le Valideur RH ne désigne pas directement un utilisateur de l'application mais un service ou un pôle d'activité même si au final c'est l'un des utilisateurs de ce service qui réalisera l'action.

- **Administrateur :**



Le rôle d'administrateur correspond au gestionnaire de la base de personnes et celle d'équipes. Le fait d'avoir le rôle d'Administrateur ne sous-entend pas un privilège supérieur, par exemple, un administrateur ne peut pas valider de demande (sauf si il cumule ce rôle).

- **Service de paye :**



Ce rôle n'engendre pas d'action d'un utilisateur dans l'application, les opérations sont automatisées et à déclenchement à date fixée. L'utilisateur concerné n'est que destinataire de l'état produit par le système.

Diagrammes de classes

Voir ci-après dans la partie Conception générale.

Modèle de données

Voir ci-après dans la partie Conception générale.

SPECIFICATIONS TECHNIQUES

Chaque employé doit pouvoir accéder à l'application à partir d'un ordinateur ou d'un mobile, le choix se porte sur un site internet proposant des pages web interagissant avec une base de données.

Les utilisateurs ont accès uniquement aux pages auxquelles ils ont droit parmi les pages suivantes :

- | | |
|--------------------------------------|---------------------------|
| - Liste des dernières demandes | (tout le monde) |
| - Nouvelle demande d'absence | (tout le monde) |
| - Déclaration d'absence | (le responsable d'équipe) |
| - Décision du responsable | (le responsable d'équipe) |
| - Décision du service RH | (service RH) |
| - Reliquat de congés dans une équipe | (le responsable d'équipe) |
| - Calendrier personnel | (tout le monde) |
| - Calendrier d'équipe | (tout le monde) |
| - Calendrier de l'entreprise | (service RH) |
| - Gestion du personnel | (administrateur) |
| - Gestion des équipes | (administrateur) |
| - Aide | (tout le monde) |

Dans cette liste figurent le rôle des utilisateurs habilités à accéder à chacune des pages web citées (voir colonne de droite). Il convient que le champ d'action de certaines pages est limité à l'utilisateur ou à l'équipe de l'utilisateur.

A noter qu'un utilisateur authentifié peut cumuler plusieurs rôles et peut accéder aux pages auxquels ces rôles lui donnent droit. En effet, l'administrateur a la possibilité de configurer en ce sens si cela s'avère nécessaire.

Ces pages interagissent avec la liste des entités Merise suivantes :

Entités fortement évolutives :

- EMPLOYE recensant les employés de l'entreprise.
- ABSENCE consignant la liste des absences.
- UTILISATEUR contenant les informations d'authentification.

Entités moyennement évolutives :

- EQUIPE listant les services ainsi que leur responsable.
- SERVICERH définissant le service RH en charge d'un groupe d'employé.

Entités faiblement évolutives :

- ROLE permettant de définir le domaine d'activité des utilisateurs.
- TYPE relative aux différents types d'absences.
- STATUT relative aux différents statuts des demandes.

A noter que l'entité EQUIPE doit posséder une cardinalité récursive qui permet, dans le cas d'un responsable, de déterminer l'équipe qu'il encadre, l'équipe dont il est membre et son propre responsable.

Les entités UTILISATEUR et ROLE, permettant de gérer les authentifications et les privilèges sont potentiellement à la portée de tous les internautes, elles doivent contenir le strict minimum nécessaire par mesure de sécurité contre les éventuels piratages de données. Les seules données concernées sont l'adresse mail, le rôle et le mot de passe qui est rigoureusement crypté.

REALISATIONS

a) L'étude

- **Vision fonctionnelle**

Il s'agit de réaliser une application Web de sorte que :

- Le Back End et le Front End soient en design-pattern MVC.
- L'application soit associée à une base de données.
- Les vues portées par le Front End respectent le Single Page Application et le mobile adaptive.

- **Equipe Projet**

En temps normal, ce type de projet doit faire intervenir plusieurs équipes mais dans le cadre du projet et de notre formation, notre équipe n'est constituée que de 3 apprenants qui endossent les casquettes de tous les acteurs censés intervenir de la demande du client jusqu'à la livraison de l'application. Cela a pour avantage de raccourcir le temps de réalisation qui est de ce fait tronqué des éventuelles échanges qu'aurait engendrées des réunions de calage en amont ainsi que les arbitrages qui pourraient découler des potentiels désaccords.

En résumé, pour notre équipe composée de 3 apprenants, il nous suffit d'être d'accords entre nous quitte à procéder à un vote où le minima de 2 pous permet d'entériner certaines décisions qui ne recueillent pas l'adhésion générale.

- **Délai de réalisation**

Le temps imparti pour réaliser l'application est estimé à 2 mois entrecoupés par nos différentes occupations dans la partie professionnelle de notre alternance.

- **Budget alloué**

Le contexte de notre formation et le fait de la situation multi-casquettes font que le budget alloué pour cette réalisation ne reposant que sur notre équipe projet est facilement estimé à 0€.

- **Vision technique**

La base de données est construite en MySQL en cohérence avec le diagramme de classes.

Le Back End est développé en Framework Spring-Boot embarquant les dépendances essentielles "Hibernate", "mysql-connector-java" et "Spring-boot-starter-web". Spring-Boot a pour avantage d'être simple, léger et modulaire, Il permet la prise en compte des annotations ce qui simplifie énormément le code (voir les annotations en annexe). Spring-Boot intègre facilement les bonnes pratiques liées à la persistance et à la sécurisation des données.

Le Front End est développé en ReactJS associé à JavaScript embarquant le Framework Bootstrap et la bibliothèque jQuery. Le choix de ReactJS est motivé par le rendu qu'il propose en Single Page Application (monopage). La particularité de ReactJS est qu'il gère les opérations dans un DOM virtuel avant de procéder à la restitution sur le DOM normal (ensemble des composants et comportements de l'interface Web). Le développement en ReactJS via des fichiers JS qui sont transpilés par NodeJS.

Ces choix ont été motivés dans la perspective de faciliter les échanges entre le Back End et la base de données grâce à JPA d'une part et entre le Back End et le Front End grâce à l'API REST de l'autre. L'avantage du couplage Front End/Back End en REST par requête HTTP, est qu'il donne facilement la possibilité de réutiliser l'association Back End et BDD avec une autre interface Web. Autre avantage, du fait du couplage en REST, il devient tout à fait possible de développer le Front End et le Back End dans n'importe quel ordre (voire en même temps). Dans notre cas, il nous a paru plus judicieux de commencer par développer notre Back End et puis d'enchaîner sur notre Front End.

b) La réalisation

- **Outils utilisés**

- **Suivi du projet**

- Gestion et répartition des tâches : Trello (voir en annexe).
- Outils de communication : Lync, Lync Attendee et Slack.

- **Stockage et développements collaboratif**

- Stockage et partage des sources : Github (voir en annexe).

- **Conception et maquettage**

- Diagrammes de classes : StarUml.
- Diagrammes de séquences : StarUml.
- Diagrammes d'utilisation : StarUml.
- Diagrammes d'activité : Balsamiq.
- Mockups : Balsamiq (voir en annexe).

- **Réalisation**

- Administration de Base de données : MySQL Workbench.
- Connexion à la base de données : JDBC et MySQL connector.
- EDI pour le Back End : Eclipse.
- EDI pour le Front End : Atom, Sublim text.

- **Développement**

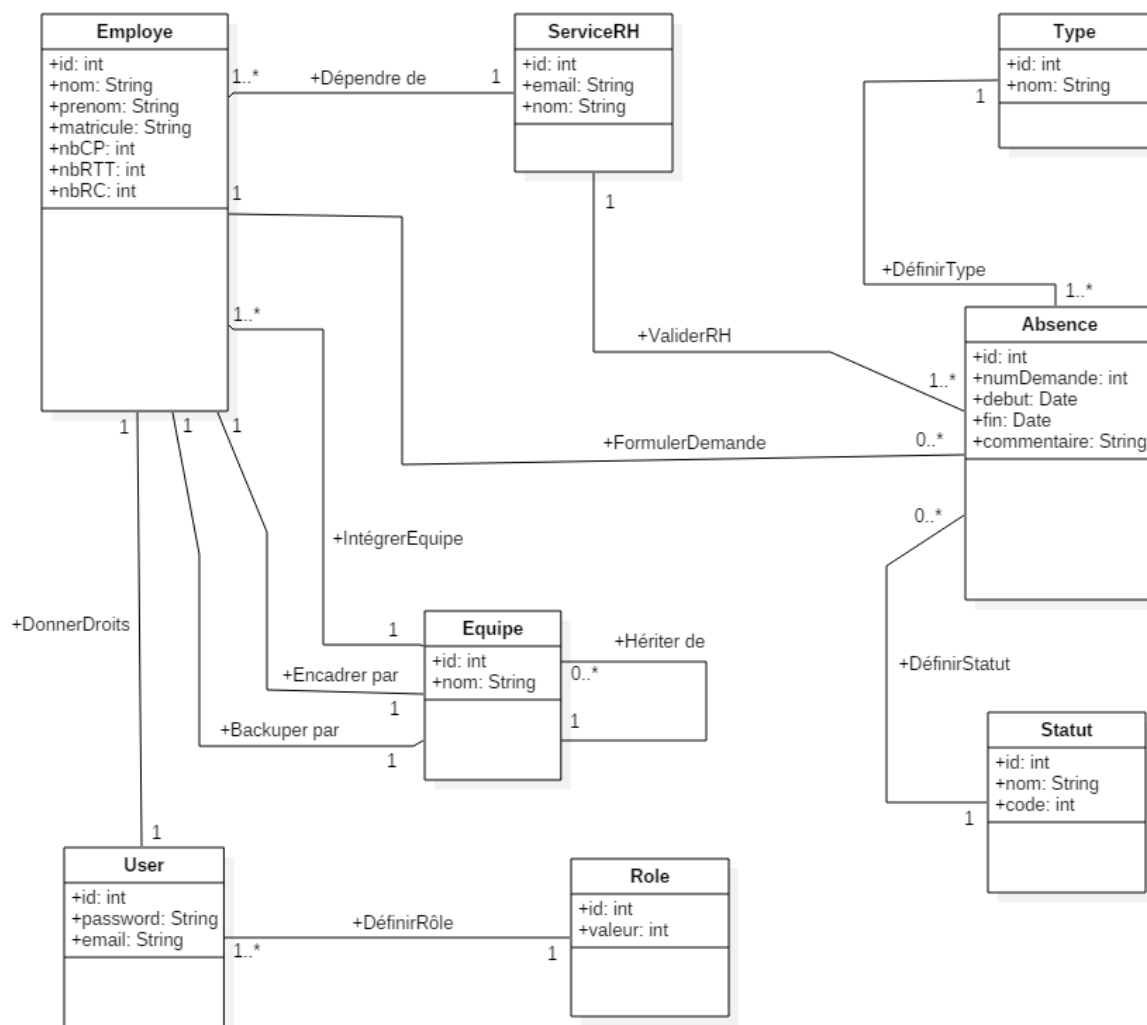
- Langages de programmation : Java, ReactJS, CSS, JavaScript.
- Framework et librairies additionnelles : Spring-Boot, Spring-Security, Maven, JUnit, Bootstrap, jQuery, Hibernate.
- Mise en place de la base de données : MySQL Workbench.

A noter : Nous avons aussi utilisé Maven dans le cadre de notre projet afin de réaliser la gestion des dépendances ainsi que le packaging de l'application.

1- Conception générale

Diagramme de classes

En tenant compte du cahier des charges, le diagramme des classes adapté aux besoins de l'application, tel que nous la concevons, s'établit comme suit :

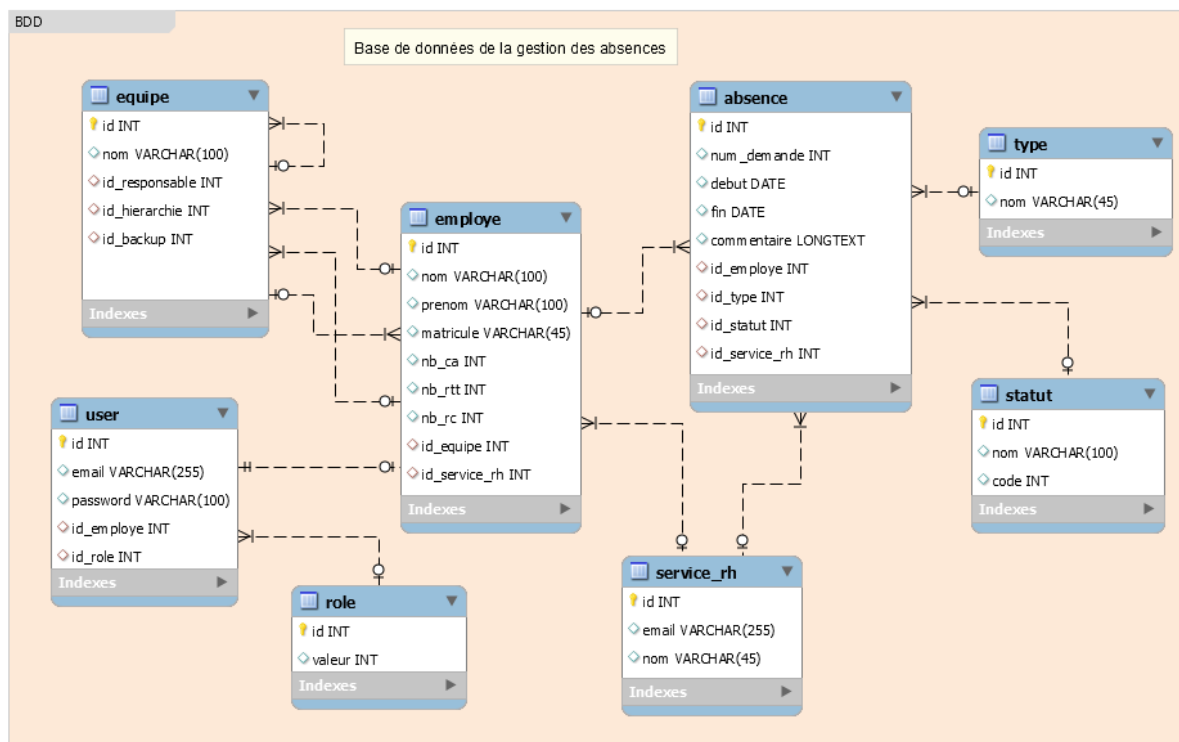


Détails :

- La liaison "Hériter de" permet, pour une équipe donnée, de déterminer l'équipe de son responsable hiérarchique.
- User et Role, utilisés en Back End par Spring-Security, servent à définir les authentifications et les privilèges des utilisateurs.

Modèle de données

Le diagramme de classes entraine le modèle de données suivant :



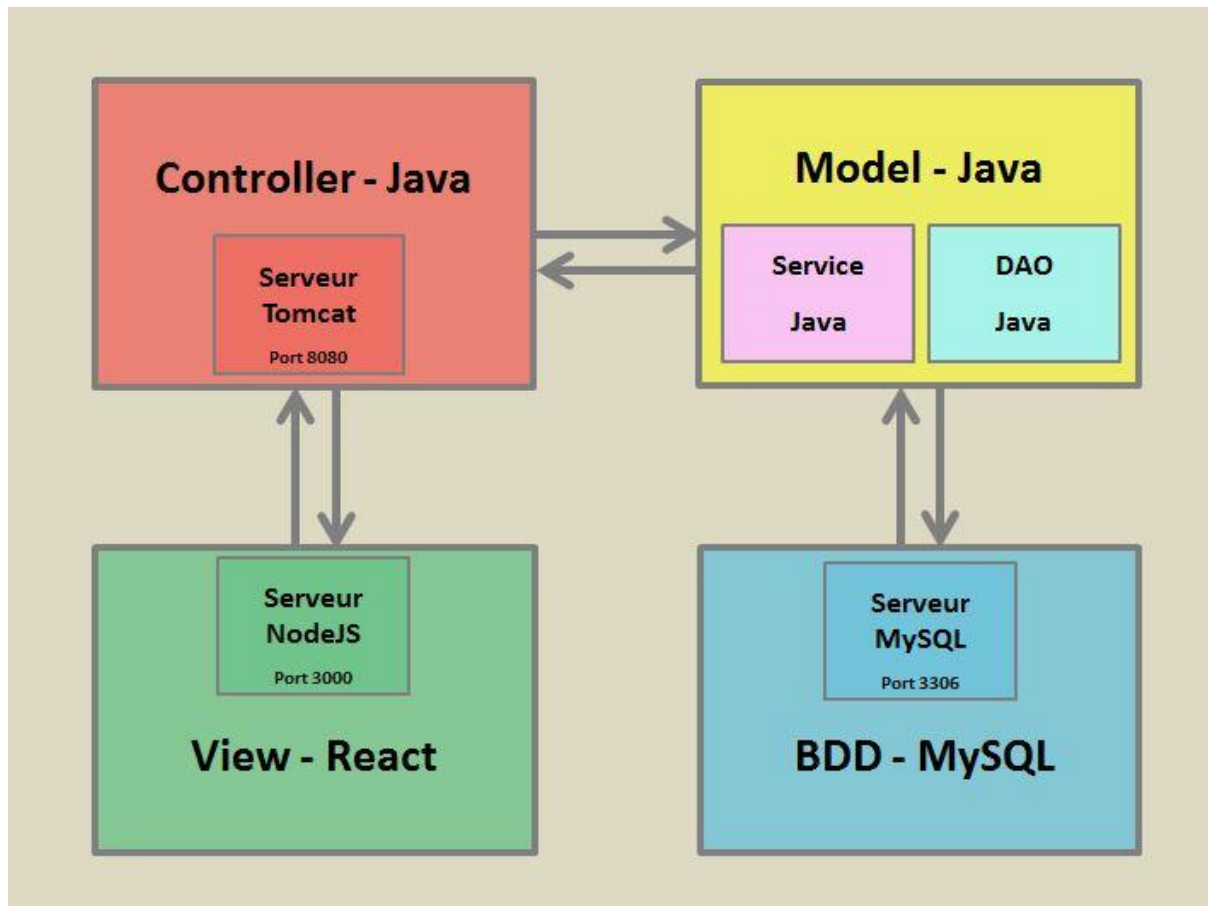
La modélisation a été élaborée grâce à MySQL Workbench.

Les liaisons entre les objets du diagramme de classes sont traduites en cardinalités, clés primaires et clés étrangères.

A noter : MySQL Workbench permet d'exporter le modèle vers la requête SQL de création de la base de données vierge conformément à la composition des tables et des champs présentés dans le modèle. Les spécificités liées aux clés primaires ou étrangères sont prises en compte dans l'élaboration de cette requête.

❖ Infrastructure détaillée

L'architecture en design-pattern MVC (Model-View-Controller) de l'application telle que nous avons décidé de la concevoir s'établit comme suit :



Détails :

- Le **Contrôleur** sert à distribuer les actions à effectuer.
- La **Vue** présente l'interface graphique.
- Le **Modèle** sert à modéliser les données à manipuler.
- Le Back End englobe le Contrôleur et le Modèle.
- Le Modèle embarque également les couches Service et DAO.
- Les échanges entre le Back End et la BDD sont facilités grâce à JPA Hibernate (Se reporter aux informations détaillées dans les parties Structure et packages et Annotations ci-après).
- Les communications entre Back End et Front End se font en REST, avec le Front End pilotant les demandes **CRUD** (**C**hange, **R**ead, **U**pdate et **D**eleter) par des requêtes HTTP.

2- Préparation

Nous avons, après étude, décidé de réaliser dans l'ordre suivant :

- Mise en place de la base de données.
- Réalisation du Back End.
- Réalisation du Front End.

Pour les besoins de l'application, nous avons implémenté les API suivantes :

- Côté Back End

- JavaMail afin de permettre les émissions de mails.
- Joda-Time pour opérer des traitements sur les dates.
- Google Calendar pour la création de tâches dans l'agenda des employés.
- Spring-Security pour la gestion des authentifications et le cryptage/décryptage des mots de passe.
- Mockito qui accompagne JUnit, pour les tests unitaires, propose des simulations d'objets, de données, etc...

- Côté Front End

- Javascript et jQuery pour un meilleur rendu de l'affichage des calendriers et aussi parce que ce module était déjà fonctionnel avant la prise en compte d'architecture technique du projet. Il a juste fallu l'implanter dans notre projet.
- Bootstrap pour la mise en forme des éléments de chacune des pages et pour faciliter le fonctionnement responsive.
- Axios (en React) et Ajax (en Javascript) pour gérer les échanges avec le Back End.

A noter : Dans le cadre de notre projet, nous avons choisi d'intégrer le template Bootstrap "SB Admin 2" plutôt que les natifs Semantic-UI ou React-Bootstrap parce qu'il nous propose des composants dont l'esthétisme nous convient.

3- Mise en place de la base de données

Toutes les opérations sont réalisées avec MySQL Workbench.

Dans le cadre de notre projet, nous avons créé un compte MySQL spécifiquement pour l'application.

Afin de créer la base de données, nous avons exporté le modèle vers la requête de création que nous avons ensuite exécuté. A ce stade, la base est vide, il convient de créer une autre requête afin d'inclure des données dans la base.

Dans notre cas, nous avons choisi de créer 3 requêtes chacun incluant la requête de création suivi de l'ajout de données selon les 3 volumétries suivantes :

- Base de données basiques : contenant quelques données.
- Base de données à mi charge : Sur la base de 100 employés.
- Base de données chargées : Sur la base de 400 employés.

Pour nous faciliter la tâche, nous avons généré des données à partir du site Generatedata accessible via <http://www.generatedata.com/?lang=fr#t1>.

4- Réalisation du Back End

Pour débiter notre Back End, nous avons créé dans Eclipse un nouveau projet de type "Spring Starter Project" auquel nous avons adjoint les dépendances Security (Core), Mail (I/O), JPA et MySQL (SQL), Web (Web).

Dépendances (emplacement racine du projet)

Le fichier de dépendances POM.XML situé à la racine du projet a été créé automatiquement avec ces dépendances. Par la suite, nous avons ajouté manuellement la dépendance Joda-Time permettant traiter les dates manipulées dans notre application et les dépendances nécessaires à la gestion des agendas. POM.XML sert à indiquer au processus Maven de récupérer les fichiers nécessaires au fonctionnement de ces dépendances.

A noter : JUnit est une dépendance par défaut, elle est embarquée de facto.

Paramétrage (emplacement src/main/resources)

Nous avons utilisé des fichiers Properties pour paramétrer notre application. Ainsi, nous y avons par exemple inscrit les paramètres de connexion à la base de données, les paramètres liés au compte émetteur des emails, etc...

Emplacement prévu le Front End (emplacement src/main/resources/static)

Le Front End une fois finalisé et "Buildé" sera implémenté dans cet emplacement ce qui nous permettra par la suite de packager notre application (ensemble Back End-Front End) afin de procéder à sa livraison.

Pour "builder" : En ligne de commandes à la racine du Front End (dossier contenant Package.json, exécuter **npm run build**, puis copier le résultat à l'emplacement.

Tests unitaires (emplacement src/test/java)

Nos tests unitaires développés en JUnit permettent de procéder aux tests et à la qualification de fragments de nos codes. Le but étant de s'assurer du bon fonctionnement en toutes circonstances.

Structure et packages (emplacement src/main/java)

Nous avons créé une structure Back End classique avec 4 packages Controller, Dao, Domain et Service, nous y avons inséré une classe (une interface pour le DAO) par table issue de notre base de données.

Pour une table donnée, l'ensemble de ses 3 classes et de son interface permettent les échanges entre Back End et BDD ainsi que la partie REST pour le Front End, le contrôleur se chargeant de les diriger.

Sachant que nous disposons de 8 tables, 24 classes et 8 interfaces sont donc créées pour couvrir l'ensemble des opérations d'échanges entre Back End et BDD.

Le code développé pour réaliser ces échanges s'appuie sur des annotations grâce à JPA, voir détails et explications sur les annotations en Annexes.

DAO : Peut être également nommé Repository, c'est la couche qui permet à l'application de dialoguer avec la base de données via **JPA Repository** : Pendant de CRUD Repository, ORM permettant de réaliser le CRUD, retourne les données sous forme de liste au lieu d'un "iterable". Dans le service, le résultat fourni est exploitable directement contrairement au CRUD repository où il faut parcourir l'iterable. Dans notre projet ce package est constitué uniquement d'interfaces.

Extrait de notre DAO pour Type d'absence :

```
...
@Repository
public interface TypeDao extends JpaRepository<TypeAbsence, Long> {

    /**
     * exemple de lecture personnalisée dans la base de données
     * @param name nom
     * @return liste de types
     */
    @Query("from TypeAbsence where nom like %?1%")
    public TypeAbsence findTypeByName(String nom);

}
...
```

@Repository : Prend en charge la classe en tant que Repository.

@Query : Permet de définir la requête SQL.

Controller : Sert à définir les différents mappages de l'appli, se charge de gérer et diriger les échanges de données.

Extrait de notre Controller pour Type d'absence :

```
...
/**
 * Controleur REST de la classe TypeAbsence
 * @author LeKhiCou
 */

@RestController
@RequestMapping("/type")
@CrossOrigin(origins="*")
public class TypeController {

    @Autowired
    TypeService typeService;

    /**
     * Liste des types
     * @param search : critère de recherche
     * @param searchnew : 2eme critere de recherche
     * @return liste des types
     */
    @GetMapping("/listeTypeAbsence")
    public ResponseEntity<?> findAll() {
        List<TypeAbsence> typeabsence;
        try {
            typeabsence =typeService.listeTypeAbsence();
        } catch (SQLException sqle) {
            return ResponseEntity.badRequest().body(sqle);
        }
        return ResponseEntity.ok(typeabsence);
    }
}
...
```

@RestController et **@RequestMapping("/type")** : Aident à la mise en place du contrôleur répondant aux requêtes liées à l'URI //localhost:8080/type.

@GetMapping("/listeType") : Permet de répondre aux requêtes GET liées à //localhost:8080/type/listeType.

@AutoWired : Injection de dépendances, alimente avec l'objet cité en dessous.

@CrossOrigin(origin="*") : Autorise la récupération des réponses aux requêtes.

Service : Peut être également nommé Métier, sert à opérer des traitements sur les données.

Extrait de notre Service pour Type d'absence :

```
...
/**
 * service gérant le type
 * C'est la couche métier.
 *
 * @author LeKhiCou
 *
 */
@Service
@Transactional
public class TypeService {

    @Autowired
    private TypeDao typeDao;

    /**
     * Liste des type d'absence
     *
     * @return une liste
     * @throws SQLException
     */
    /*
     * La methode [findAll] retourne une liste de la table
     * La methode [findAll] est override.
     */
    public List<TypeAbsence> listeTypeAbsence() throws SQLException {
        List<TypeAbsence> resultat;
        try {
            resultat = typeDao.findAll();
        } catch (ServiceException e) {
            throw new ServiceException("Hibernate Error !:
listeTypeAbsence" + e);
        }
        return resultat;
    }
}
```

@Service et **@Transactional** : Prend en charge la classe en tant que service interagissant avec la base de données et assurant la persistance des données.

Domain : Peut être également nommé Bean, POJO ou Model, chargé de modéliser et transformer les informations échangées entre base SQL et objets Java.

Extrait de notre Domain pour Type d'absence :

```
...
/**
 * entity Type
 * @author LeKhiCou
 */
@Entity
@Table(name = "type")
public class Type {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
    private Long id;

    @Column(name = "nom")
    @NotBlank(message = "Nom obligatoire")
    @Length(min = 2, message = "La chaîne doit avoir au moins 2 caractères")
    private String nom;

    @OneToMany(fetch = FetchType.LAZY, mappedBy = "type")
    @JsonBackReference
    private List<Absence> absence;

    ...
}
```

@Entity et **@Table** : Lient l'entité à la table Type de la base de données.

@Id et **@GeneratedValue** : Lient id de l'entité à l'id auto généré de la table.

@Column : Permet de lier une propriété de l'entité à un champ de la table.

@NotBlank et **@Length** : Imposent des contraintes à la propriété.

@OneToMany : Permet d'établir la jointure avec l'entité Absence.

@JsonBackReference : Associée à **@JsonManagedReference**, sert à prévenir contre les boucles infinies dans le JSON du fait de la récursivité.

Autres packages :

Dto : Data Transfert Object, limite et met en forme les données à échanger avec le Front End.

Extrait de notre Dto pour Employe :

```
...  
    private String nom;  
    private String prenom;  
    private String matricule;  
    private String email;  
    private int role;  
    private int nbCa;  
    private int nbRtt;  
    private int nbRc;  
    private String nomRh;  
    private String emailRh;  
    private String nomEquipe;  
    private String nomResponsable;  
    private String prenomResponsable;  
    private String emailReponsable;  
  
    public EmployeDto() {  
    }  
  
    public String getNom() {  
        return nom;  
    }  
  
    public void setNom(String nom) {  
        this.nom = nom;  
    }  
...
```

C'est une simple définition des propriétés de l'objet.
Dto est intégré dans le package Dto dépendant de Model.

Exception : Nous sert à stocker nos classes qui ne sont pas directement concernées par notre design-pattern MVC.

- Common: Gestion des exceptions et des erreurs (serviceException et emailException dans le cas de notre projet).

Services.utils :

- GestionAbsencesApplication (lanceur de l'application).
- Config.java permet de configurer l'application (par exemple Spring-Security).
- Différentes classes permettant des ajustements, calculs et traitements sur les couches métiers.

Tests unitaires :

Nous avons réalisé des tests unitaires à l'aide de JUnit couplé à Mockito qui nous sert à générer automatiquement des objets et des données factices à partir des classes existantes.

Extrait d'un exemple de test unitaire :

```
...
@RunWith(SpringRunner.class)
@SpringBootTest(classes = Config.class)
public class DemandeAbsencesApplicationTests {

    // @Mock prend une classe en argument et retourne un objet de cette classe.
    @Mock
    ServiceRhDao rhDao;

    @Mock
    ServiceRh rh;

    ...

    @InjectMocks
    ServiceRhService rhService;

    @Before
    public void serviceRhInit(){
    }

    @Test
    public void serviceRhBeanTest() {

        // Test avec simulation d'exception.
        @Test(expected = Exception.class)
        public void serviceRhTestException() throws SQLException{
            when(rhDao.findByName(anyString())).thenReturn(new Exception());
            rhService.getServiceRh(anyString());
        }

    }
    ...
}
```

Annotations apportées par JUnit et Mockito : **@Mock** : Simuler l'objet cité à la ligne suivante. Les autres annotations sont explicites d'après leur nom.

5- Réalisation du Front End

Notre Front End a été développé en ReactJS, idéal dans notre cas où nous avons souhaité réaliser une application REST en design-pattern MVC. Sachant que le Back End nous satisfait la partie Model Controller en REST, ReactJS propose toutes les conditions pour la partie View également orienté REST.

Liste des composants principaux

- **Apps** : Lanceur de ReactJS, charge et affiche le composant Site.
- **Site** : Point de lancement de notre site, intègre Entete, BarDeNav, Pages et PiedPage.
- **Entete** : Affiche l'identité ainsi que des détails sur l'utilisateur.
- **PiedPage** : Affiche des détails sur l'application.
- **BarDeNav** : Guide le choix de l'utilisateur (format menu déroulant).
- **Pages** : Permet de d'afficher le contenu de la page en fonction du contexte ou de la navigation.

Extrait de Site (contenant la transmission de données à Entete et Pages) :

```
...
render() {
  return (
    <div>

      // Appel d'une page fille accompagnée des données issues du State
      <Entete nom={this.state.nom}
        prenom={this.state.prenom}
        nbCa={this.state.nbCa}
        nbRtt={this.state.nbRtt}
        nbRc={this.state.nbRc} /> { /* Affiche le header de la page */}

      // Appel d'une page sans données du State
      <BarDeNav /> { /* Affiche la barre de navigation (latérale) */}
      <Pages employe={this.state} /> { /* Affiche le body selon navigation */}
      <PiedPage /> { /* Affiche le footer */}
    </div>
  )
  ...
}
```

A noter : Les composants Entete.js, BarDeNav.js et PiedPage.js sont en position absolute respectivement en haut, à gauche et en bas. Le scrolling des composants de Pages.js n'interfère pas sur cette disposition.

Le composant Pages intègre les différents formulaires, les tableaux et les calendriers. Ces derniers s'appuient sur Javascript et jQuery. Réact se charge de fournir une page contenant des boutons et des Divs identifiés par des Ids. Javascript et jQuery affichent et gèrent les données via le DOM des éléments de la page (une fois que ReactJS les aura rendus).

Chaque composant ReactJS est associé à un répertoire contenant le script ReactJS et sa feuille de style. A partir de Pages, nous avons choisi de constituer une arborescence en fonction des pages de navigation Inspirées des cas d'utilisation et des diagrammes de navigation.

Exemple de feuille de style en l'occurrence pages.css :

```
.page {  
  float: right;  
  margin-right: 30px;  
  width: 80%  
}  
  
.panel-title {  
  font-weight: bold;  
}  
  
.VOffsetPages {  
  height: 80px;  
}  
  
.VOffsetHautPages {  
  height: 7vh;  
}  
  
.VOffsetBasPages {  
  height: 7vh;  
}
```

Communications entre Front End et Back End :

Les informations concernant l'utilisateur sont chargées à partir du composant Site.js qui fait office de "mère" et sont à la disposition de toutes les pages dépendantes "filles". Les autres données, plus spécifiques, sont chargées directement depuis la page concernée. Cela a pour but de limiter la consommation de la bande passante réseau. Les données en State de la mère sont transmises aux filles en tant que Props (voir inventaire des communications en annexe).

Les données formatées en Json sont échangées avec le Back End via des requêtes HTTP s'appuyant sur les méthodes CRUD. Elles sont gérées à l'aide du composant Axios sauf pour les calendriers où les requêtes HTTP se font dans le fichier Javascript calendrier.js à l'aide de routines Ajax.

Extrait de Site qui fait office de mère (partie récupération de données et préparation du State)

```
...
class Site extends Component {
  constructor() {
    super();
    // Définition des propriétés du State
    // Les propriétés sont relatives aux renseignements concernant l'utilisateur
    // authentifié ainsi que la liste de ses absences.
    this.state = {
      nom: "",
      prenom: "",
      matricule: "",
    ...
    // Traitement asynchrone : Résultat est obtenu après récolte des données du back
    componentDidMount() {

    // Récupération des données du Back par requête HTTP
    axios.get('http://localhost:8080/user/getUser?email='+email)
    // Type GET paramétré avec l'email.
    .then(res => {
      this.setState({ // Incorpore les données dans le State
        nom: res.data[0].employeeDto.nom,
        prenom: res.data[0].employeeDto.prenom,
        matricule: res.data[0].employeeDto.matricule,
      ...
    ...
```

A noter : Le State vu de la mère devient Props côté fille.

Extrait de NouvelleDemande qui fait office de fille (partie récupération des données de la mère et constitution du state contenant des données spécifiques à cette page)

```
...
class NouvelleDemande extends Component {

  constructor(props) {
    super(props); // Récupère le Props du parent
    this.props=props;
    this.state={ // Définition des propriétés du State
      types: []
    }
  }

  // Traitement asynchrone : Résultat est obtenu après récolte des données du back
  componentDidMount() {
    axios.get('http://localhost:8080/type/listeTypeAbsence')
  }
  // Requête de type GET - Récupère la liste des absences
  .then(res => {
    this.setState({
      types: res.data
    });
  });
...

```

Extrait de Nouvelle demande (exploitation des données communes mère et spécifiques fille)

```
...
<div className="col-md-4 col-md-offset-1">  {/* Block formulaire gauche */}
  <div className="form-group">  {/* Champs Type d'absence */}
    <label>Type d'absence</label>
    <select className="form-control" onChange={this.handleTypeChange}>
      {this.state.types.map(
        (type, i) =>
        <option key={i}>{type.nom}</option>
      )}
    </select>
  </div>
...
<div className="col-md-4 col-md-offset-2">  {/* Block formulaire droit */}
  <div className="form-group">  {/* Champs Congés payés */}
    <label>Congés payés</label>
    <label className="form-control">{this.props.employe.nbCa}</label>
  </div>
...

```

L'inventaire des échanges de données est disponible en annexe.

Extrait de calendrier.js (En Javascript - Contenant une routine Ajax)

```
...
// Bloc exécuté au chargement
$(document).ready(function() {

    // Routine Ajax permettant d'exécuter une requête HTTP
    // afin de récupérer des infos du Back End en Json
    $.ajax({
        type: "GET",                // Type de requête
        url: "calendrier/perso",    // URL de la requête
        success: function(response) { // Exécute le bloc si la réponse est satisfaite
            listeAbsences = response; // Les données fournies sont stockées dans la
                                     // variable listeAbsences
        }
    });
    // JS poursuit ses traitements même si le traitement Ajax n'est pas terminé.

    // Le traitement Ajax étant asynchrone par rapport à JS, la routine ci-dessous
    // permet d'exécuter le bloc une fois la récupération des données est complète.
    $(document).ajaxStop(function () {
        afficherCalendrier();
        afficheLegend();
    });

    // Gestion du bouton gauche décalant d'un mois en arrière le calendrier.
    $('#leftcalendrier').click(function(){
        dateEnCours = new Date(annee, (mois - 1), 1); // Décalage d'un mois en arrière
                                                       // de la date de référence.


        afficherCalendrier();
    });
    ...
}
```

6- Conditionnement et packaging

En mode de Dev ou de Test, notre Front End et notre Back End sont dissociés mais interagissent entre eux du fait des échanges par requêtes HTTP. Dans le cadre de la livraison, le Front End une fois finalisé est "Buildé" puis déposé dans le dossier src/main/resources/static de notre Back End ce qui permet de consolider Back End et Front End afin de packager notre application. Nous avons choisi de packager en fichiers WAR embarquant le Front End et Back End consolidés ainsi que les librairies.

Exemples de copie d'écran issus de notre application (Voir aussi les mockups en annexe)

- Liste de mes demandes :



MERRILL Steven

Congés payés : 17 j
 RTT : 6 j
 Repos compensateurs : 22 h


[Accueil](#)
[Absence](#)
[Calendrier](#)
[Gestion](#)
[Aide](#)
[Session](#)

Liste de vos dernières demandes

N° demande	Nom	Prénom	Nom Resp	Prénom Resp	Type	Début	Fin	Statut	Action
DEM000003	MERRILL	Steven	DICKSON	Jacob	Congé payé	23/08/2017	30/08/2017	En attente de validation du Responsable	Décider
DEM000028	MERRILL	Steven	DICKSON	Jacob	Autres absences	12/06/2017	20/06/2017	Refusé par le Responsable	
DEM000061	MERRILL	Steven	DICKSON	Jacob	Repos compensateur	10/07/2017	12/07/2017	En attente de validation du Responsable	Décider

GestionAbsences©2017 - Par Fred, Jean et Mokhtar de La Poste.promo1@Simplon avec la participation de
 

- Formulaire de nouvelle demande d'absence :


MERRILL Steven

Congés payés : 17 j
 RTT : 6 j
 Repos compensateurs : 22 h

[Accueil](#)
[Absence](#)
[Calendrier](#)
[Gestion](#)
[Aide](#)
[Session](#)

Nouvelle demande

Type d'absence

Congés payés


Date de début

RTT


Date de fin

Repos compensateur

Valideur

GestionAbsences©2017 - Par Fred, Jean et Mokhtar de La Poste.promo1@Simplon avec la participation de
 

- Mon calendrier


MERRILL Steven

Congés payés : 17 j RTT : 6 j Repos compensateurs : 22 h


[Accueil](#)
[Absence](#)
[Calendrier](#)
[Gestion](#)
[Aide](#)
[Session](#)

Mon calendrier


Mai 2017							Juin 2017							Juillet 2017						
Lu	Ma	Me	Je	Ve	Sa	Di	Lu	Ma	Me	Je	Ve	Sa	Di	Lu	Ma	Me	Je	Ve	Sa	Di
1	2	3	4	5	6	7	29	30	31	1	2	3	4	26	27	28	29	30	1	2
8	9	10	11	12	13	14	5	6	7	8	9	10	11	3	4	5	6	7	8	9
15	16	17	18	19	20	21	12	13	14	15	16	17	18	10	11	12	13	14	15	16
22	23	24	25	26	27	28	19	20	21	22	23	24	25	17	18	19	20	21	22	23
29	30	31	1	2	3	4	26	27	28	29	30	1	2	24	25	26	27	28	29	30
5	6	7	8	9	10	11	3	4	5	6	7	8	9	31	1	2	3	4	5	6

—Légende—
 : Non travaillé
 : Congé payé validé
 : Congé payé en attente
 : RTT validé
 : RTT en attente
 : Repos validé
 : Repos en attente
 : Autre validé
 : Autre en attente

[<](#) [Aujourd'hui](#) [>](#)

 GestionAbsences©2017 - Par Fred, Jean et Mokhtar de La Poste.promo1@Simplon avec la participation de
 

- Le calendrier de mon équipe


MERRILL Steven


Congés payés : 17 j RTT : 6 j Repos compensateurs : 22 h

[Accueil](#)
[Absence](#)
[Calendrier](#)
[Gestion](#)
[Aide](#)
[Session](#)

Calendrier de mon équipe

Equipe	Mai 2017							Juin 2017														Juillet 2017						
	Je	Ve	Sa	Di	Lu	Ma	Me	Je	Ve	Sa	Di	Lu	Ma	Me	Je	Ve	Sa	Di	Lu	Ma	Me	Je	Ve	Sa	Di			
DICKSON Jacob																												
WINTER Bryan																												
MERRIL Steven																												
MARQUEZ Skyler																												
BURT Eagan																												

[<](#) [Aujourd'hui](#) [>](#)

 GestionAbsences©2017 - Par Fred, Jean et Mokhtar de La Poste.promo1@Simplon avec la participation de
 

CONCLUSIONS

Ce projet est pour moi l'aboutissement de notre formation où la majorité des compétences du cursus a pu être exprimée.

Nous nous sommes attelés à réaliser l'application la plus en adéquation possible avec le cahier des charges en ayant eu à surmonter des difficultés tels que la gestion particulière des calendriers, la maîtrise des API Spring-Security, Java-Mail et Google Calendar.

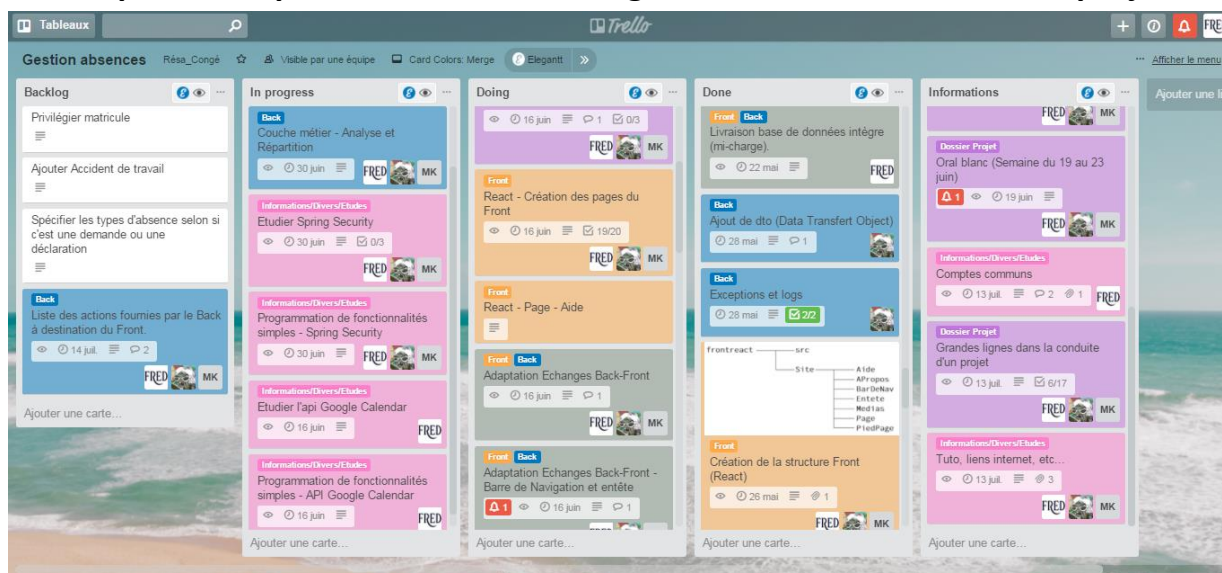
Notre réalisation nous a permis de livrer la version V1 de notre application, nous avons pensé apporter des évolutions à la V2 tels que la prise en compte de justificatifs d'absence, la restitution de l'organigramme de l'entreprise, le calcul des statistiques, la création d'une console d'administration (incluant le retour arrière du workflow et la correction en base) et nous n'excluons pas la construction de pages supplémentaires relatives à la configuration de l'entreprise (comprenant le choix du logo d'entreprise, la configuration des jours fériés, des jours de repos hebdomadaires, du quota de congés et des différentes dates de traitement).

Réaliser ce projet fut une expérience très enrichissante qui m'a permis de découvrir plusieurs étapes de la vie d'un projet, le travail collaboratif, le travail individuel, la bonne gestion du rush et l'optimisation du temps de travail.

Grâce à l'enseignement de Simplon qui pour moi sont symbolisé par les mots clés Savoir, Bonnes pratiques, Autonomie, Adaptabilité et Communication, je pense être en mesure de réaliser des applications du même type et pense pouvoir m'adapter à d'autres situations employant des technologies différentes.

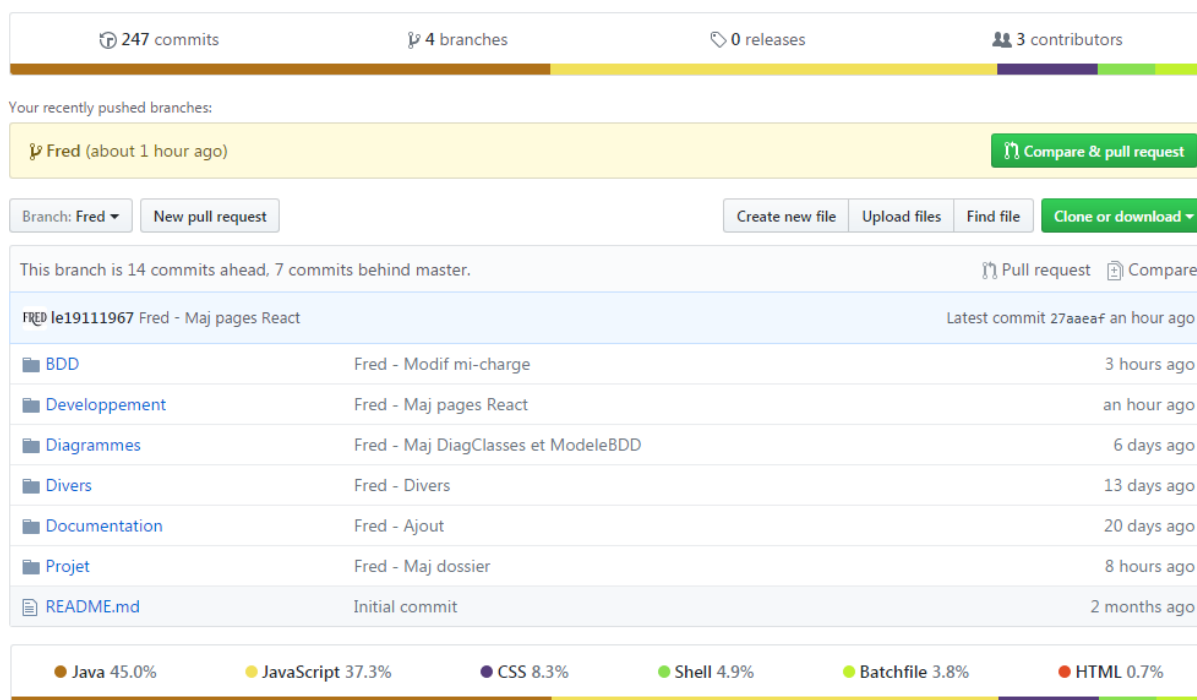
ANNEXES

Trello qui nous a permis de suivre et de gérer l'avancement de notre projet :



Trello est gratuit et est disponible à l'adresse <https://trello.com/>

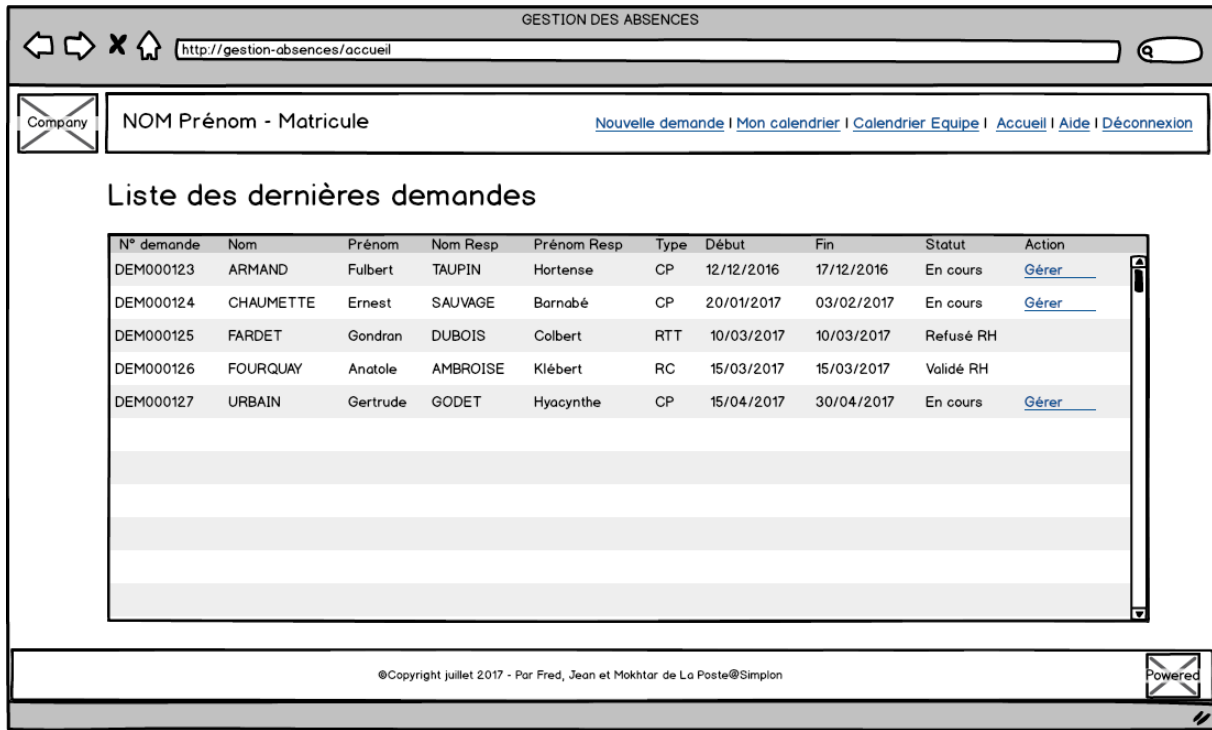
Notre ressource collaborative Github :



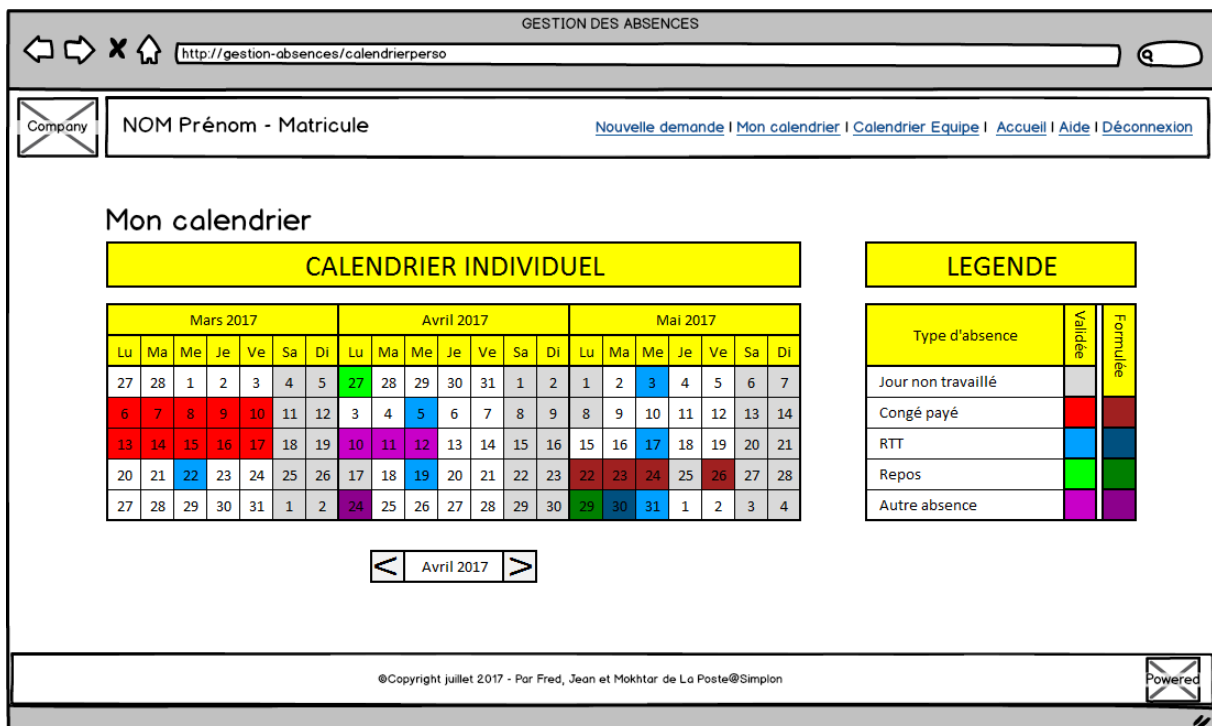
Github est gratuit et est disponible à l'adresse <https://github.com/git/git/>

2 exemples de mockup réalisés à l'aide de Balsamiq :

- Mockup de la liste de demandes



- Mockup du calendrier individuel



Récapitulatif des échanges par email réalisées par l'application

Dans le cas des demandes d'absences :

- **Demande formulée et soumise par le demandeur (notification de prise en compte)**
 - Destinataire : Le demandeur
- **Demande soumise à décision du responsable hiérarchique ou du backup (avec lien pour décision hiérarchique - voir l'exemple ci-dessus)**
 - Destinataire : Le responsable hiérarchique
 - En copie : Le backup
- **Demande validée par le responsable hiérarchique ou le backup (notification)**
 - Destinataire : Le demandeur
 - En copie : Le responsable hiérarchique
 - En copie : Le backup
- **Demande validée par le responsable hiérarchique ou le backup (avec lien pour décision RH)**
 - Destinataire : Le service RH
- **Demande refusée par le responsable hiérarchique**
 - Destinataire : Le demandeur
 - En copie : Le responsable hiérarchique
 - En copie : Le backup
- **Demande validée par le service RH**
 - Destinataire : Le demandeur
 - En copie : Le Responsable hiérarchique
 - En copie : Le backup
 - En copie : Le service RH
- **Demande refusée par le service RH**
 - Destinataire : Le demandeur
 - En copie : Le Responsable hiérarchique
 - En copie : Le backup
 - En copie : Le service RH

- **Demande annulée par le demandeur (cas d'une demande en cours)**
 - Destinataire : Le décisionnaire en cours
 - En copie : Le demandeur
 - En copie : Le responsable hiérarchique (si décisionnaire est le service RH)
 - En copie : Le backup
- **Demande annulée par le demandeur (cas d'une demande validée à date non échue)**
 - Destinataire : Le demandeur
 - En copie : Le responsable hiérarchique
 - En copie : Le service RH

Dans le cas des déclarations d'absences inopinées :

- **Déclaration formulée par le responsable hiérarchique (notification de prise en compte)**
 - Destinataire : Le responsable hiérarchique
- **Déclaration formulée par le responsable hiérarchique (avec lien pour décision RH)**
 - Destinataire : Le service RH
- **Demande validée par le service RH**
 - Destinataire : Le Responsable hiérarchique
 - En copie : L'employé
 - En copie : Le service RH
- **Demande refusée par le service RH**
 - Destinataire : Le Responsable hiérarchique
 - En copie : Le service RH

Dans le cas particulier de l'envoi de l'état mensuel des absences validées :

- **A date définie, l'application élabore puis envoie un récapitulatif des absences du mois au service de paye**
 - Destinataire : Le service de paye
 - Pièce jointe : Etat des absences pour la période du mois écoulé

Les annotations JPA

Très pratique, les annotations servent à se substituer à plusieurs lignes de codes tout en réalisant des opérations bien précises. Le système les interprète comme si ces lignes de codes étaient présentes.

JPA permet de relier la base SQL aux objets Java (par exemple transforme les données SQL en objet Java et vice-versa).

Ci-dessous, quelques annotations qui nous ont été utiles :

@GeneratedValue : Signifie que la colonne, généralement l'Id, est générée automatiquement sous réserve qu'il ait l'attribut Auto-Incrémental en base.

@Column : Spécifie le nom du champ de la table à traiter.

@DateTimeFormat : Spécifie le format de la date échangée avec la base (gestion bidirectionnel).

@Temporal : Spécifie que la donnée manipulée est de type Date.

@Future : Spécifie que la date manipulée doit être supérieure ou égale à la date du jour.

@ManyToOne ou **@OneToMany** ou **@OneToOne** : Servent à définir le type de cardinalité entre les tables.

@JoinColumn : Utilisé avec manyToOne, spécifie le champ de jointure de la cardinalité.

@JsonBackReference : Fournie les données de premier niveau de dépendance (souvent associé à JsonManagedReference).

@JsonIdentityInfo : Permet d'éviter les boucles Json de données à données.

@Cascade : Permet de prendre en compte les données de la table ainsi que ses dépendances liées aux cardinalités.

Inventaire des communications entre le Back End et le Front End (en fonction de nos pages)

La liste ci-dessous recense l'ensemble des communications qui ont lieu entre le Back End et le Front End à partir d'un protocole de langage élaboré par nous.

Explications avec par exemple **"Liste des demandes"** :

- **Liste de demandes** indique la page de Front End concernée.
- **Back**, c'est le Back qui fournit au Front (et vice-versa pour Front).
- **Employe(matricule)** indique le paramètre de la requête http et la racine de la réponse Json.
- **{nom, prenom}** indique les nom et prénom de employe(matricule).
- **=> employe(idResp)** sous-entend le responsable dont dépend l'employe(matricule).
- **=> [absence]** est la liste des absence dépendant de employe(matricule).
- **=> [type]** est l'ensemble des types dépendant des [absence]

Liste des demandes :

Requête : employe(matricule)

Back : employe.{nom, prenom} => employe(idResp).{nomResp, prenomResp}
=> [absence].{n°demande, debut, fin} ==> [type].{nom} ==> [statut].{nom}

Front : Néant

Nouvelle demande :

Requête : employe(matricule)

Back : employe.{nbCa, nbRtt, nbRC, nomResp, prenomResp}, [type].{nom}

Front : absence.{idEmploye, debut, fin} => type.{nom}

Déclaration d'absence :

Requête : employe(idEquipe)

Entée : [employe].{nom, prenom, nbCa, nbRtt, nbRC}, [type].{nom}

Front : absence.{idEmploye, debut, fin, commentaires} => type.{nom}

Reliquat de congés :

Requête : employe(idEquipe)

Entée : [employe].{nom, prenom, matricule, nomResp, prenomResp, nbCa, nbRtt, nbRC}

Front : néant

Calendriers :**- MonCalendrier**

Requête : employe(matricule)

Back : employe.{nom, prenom} => [absence].{debut, fin} ==> type.{nom} ==> statut.{nom}

- CalendrierEquipe

Requête : employe(idEquipe)

Back : [employe].{nom, prenom} => [absence].{debut, fin} ==> type.{nom} ==> statut.{nom}

- CalendrierEntreprise

Requête : employe

Back : [employe].{nom, prenom} => [absence].{debut, fin} ==> type.{nom} ==> statut.{nom}

Front : néant

Gestion du personnel :

Requête : employe

Back : [employe].{nom, prenom, matricule, email} => equipe.{nom, idResp} ==>

employe.{nomResp, prenomResp}

=> serviceRh.{nom, email}

=> role.{valeur},

[equipe].{nom, idResp} => employe.{nomResp, prenomResp},

[serviceRh].{nom, email},

[role].{valeur}

Front CRU : employe.{nom, prenom, matricule, email}, equipe.{idEquipe}, serviceRh.{idRh},
role.{valeur}

Front D : employe.{id}, user.{id}

Gestion d'équipe :

Requête : equipe

Back : equipe.{nom, idResp} => employe.{nomResp, prenomResp}

=> serviceRh.{nom, email},

[equipe].{nom, idResp} => employe.{nomResp, prenomResp},

[serviceRh].{nom, prenom},

Front CU : equipe.{nom, idResp}, serviceRh.{nom, email}

Front D : equipe.{id}

Avis hiérarchie :

Requête : absence

Back : absence.{idEmploye, debut, fin} => employe(matricule).{nbCa, nbRtt, nbRC}

=> type.{nom}

=> statut.{nom}

Front : absence.{idEmploye, debut, fin, commentaires}, employe(matricule).{nbCa, nbRtt, nbRC}, type.{nom}, statut.{nom}

Avis RH :

Requête : absence

Back : absence.{idEmploye, debut, fin} => employe(matricule).{nbCa, nbRtt, nbRC}

=> type.{nom}

=> statut.{nom}

Front : absence.{idEmploye, debut, fin, commentaires}, employe(matricule).{nbCa, nbRtt, nbRC}, type.{nom}, statut.{nom}

Authentication :

Requête : employe

Back : user.{idEmploye, email, motdepasse} => role(idUser).{valeur}

Front : user.{idEmploye} => role(idUser).{valeur}

Modification du mot de passe :

Requête : employe

Back : user.{idEmploye, email, motdepasse}

Front : user.{idEmploye, motdepasse}

Barre de navigation :

Requête : employe

Back : employe.{id} => role.{valeur}

Front : néant

Entête :

Requête : employe

Back : employe.{nom, prenom, matricule, nbCa, nbRtt, nbRC}

Front : néant