« RH »

Maxime Borgeaud, SICA2a

CPNV, TPI 2024



*Chef de projet : Mr. Pascal Benzonana*

*Expert 1: Mr. Ernesto Montemayor*

*Expert 2: Mr. Antoine Mveng Evina*

Table des matières

[1 Introduction 3](#_Toc167703327)

[2 Analyse 4](#_Toc167703328)

[2.1 Méthode de projet 4](#_Toc167703329)

[2.2 Backup 4](#_Toc167703330)

[2.3 Fonctionnalités prévues 4](#_Toc167703331)

[2.4 Planification initiale 4](#_Toc167703332)

[2.5 Maquettes 7](#_Toc167703333)

[2.6 Modèle conceptuel de données 22](#_Toc167703334)

[3 Conception 23](#_Toc167703335)

[3.1 Architecture 23](#_Toc167703336)

[3.2 Outils 23](#_Toc167703337)

[3.3 Concepts 23](#_Toc167703338)

[3.4 Modèle Logique de Données 24](#_Toc167703339)

[3.5 Diagrammes de classes 25](#_Toc167703340)

[3.6 Gestion des erreurs 29](#_Toc167703341)

[4 Réalisation 29](#_Toc167703342)

[4.1 Base de données 29](#_Toc167703343)

[4.1.1 Structure 29](#_Toc167703344)

[4.1.2 Mise en ligne de la base de données 31](#_Toc167703345)

[4.1.3 Droits d’accès 31](#_Toc167703346)

[4.1.4 Connexion 31](#_Toc167703347)

[4.1.5 Données 32](#_Toc167703348)

[4.2 Initialisation du projet 32](#_Toc167703349)

[4.3 Génération des modèles 33](#_Toc167703350)

[4.4 Structure du projet 33](#_Toc167703351)

[4.4.1 Dossier « db » 33](#_Toc167703352)

[4.4.2 Dossier « models » 34](#_Toc167703353)

[4.4.3 Dossier « helpers » 34](#_Toc167703354)

[4.5 Dossier « routes » 34](#_Toc167703355)

[4.6 Dossier « SRC » et « dist » 35](#_Toc167703356)

[4.7 Logique des endpoints 35](#_Toc167703357)

[4.8 Stratégie de tests 37](#_Toc167703358)

[5 Conclusion 37](#_Toc167703359)

[5.1 Erreurs/imprécisions restantes 37](#_Toc167703360)

[5.2 Améliorations possibles 37](#_Toc167703361)

[5.3 Conclusion personnelle 37](#_Toc167703362)

[6 Archives 37](#_Toc167703363)

[6.1 Base de données 37](#_Toc167703364)

[6.1.1 MCD 38](#_Toc167703365)

[6.1.2 MLD 39](#_Toc167703366)

[7 Glossaire 40](#_Toc167703367)

[8 Annexes 40](#_Toc167703368)

[9 Sources 40](#_Toc167703369)

# Introduction

Le projet consiste en un petit système de gestion des employés d’une entreprise. L’objectif est de développer une application web qui permet de gérer les informations des employés ainsi que les offres d’emploi et candidatures. L’idée est d’optimiser les processus RH qui peuvent être long et répetitifs afin de permettre aux utilisateurs d’éviter de générer des erreurs dans le système.

L’application comprend la consultation de ses données personnelles et professionnelles ainsi que ses demandes de congés et ses évaluations personnelles. Les employés RH ont évidemment des droits particuliers. Ils peuvent valider ou refuser des demandes de congé, consulter et modifier les informations des autres employés ainsi que gérer les offres d’emplois, candidatures et entretiens d’embauches. Cette application est développée dans le cadre du CPNV de Ste-Coix comme projet de TPI.

# Analyse

## Méthode de projet

La méthode utilisée est « AGILE » afin d’avoir une gestion flexible du projet et de mettre en pratique des connaissances théoriques apprises durant la formation. Un Icescrum est disponible à l’adresse suivante : <https://icescrum.cpnv.ch/p/TPIRH/#/>

## Backup

J’effectue des sauvegardes tous les jours de l’avancée du projet sur Github, dont voici le lien : <https://github.com/maxDevelopement/TPI_RH>

## Fonctionnalités prévues

Voici une description des fonctionnalités souhaitées pour l’application

* **Gestion des employés**

Un employé RH doit pouvoir consulter la liste des employés actuels ou ex-employés avec toutes leurs informations personnelles et de leurs contrats. Il doit également pouvoir y ajouter un nouvel employé ainsi que modifier d’éventuelles informations

* **Gestion des demandes de congé**

Un employé peut consulter à tout moment ses demandes de congés sur un calendrier dynamique. Il peut y voir le statut des demandes (accepté, refusé ou en attente) mais également faire une nouvelle demande de congé. Un employé RH peut consulter à tout moment la liste des demandes de congés en attente et leurs attribuer un nouveau statut, (refusé ou accepté).

* **Gestion des performances**

Chaque employé reçoit une évaluation par année. Celle-ci contient une note (1 à 6), une note positive et une négative. Un employé RH peut Remettre une évaluation d’un employé pour l’année courante, s’il n’en a pas déjà reçu une.

* **Gestion du recrutement**

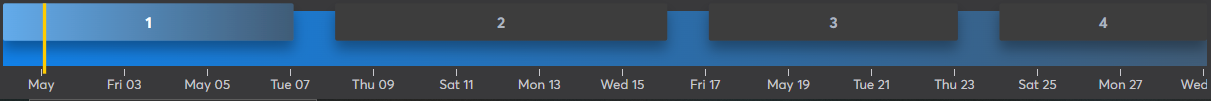
Les employés RH peut également gérer les offres d’emplois et leurs candidatures. Ils peuvent ajouter/modifier/supprimer une candidature pour un poste, programmer un entretien et attribuer un statut à la candidature (accepté, refusé ou en attente).

## Planification initiale

Vous trouverez les éléments de ma gestion de projet dans le Iscescrum (publique) suivant :

<https://icescrum.cpnv.ch/p/TPIRH/#/planning>

Le projet débute le 30 avril 2024 à 8h et se termine le 29 mai à 11h35. Il se déroule sur 90 heures et va se diviser en 4 parties distinctes : 20% d’analyse, 45% d’implémentation, 10% de tests, 25% de documentation. Il y a également une découpe temporelle de 4 sprints d’environ une semaine dont voici une représentation visuelle :



Sprint 1

* Date : 30.04 🡺 07.05
* Goal : *Analyse/conception du projet bien amorcée*

Sprint 2 :

* Date : 08.05 🡺 16.05
* Goal : *Backend bien amorcée + adaptation conception*

Sprint 3 :

* Date : 17.05 🡺 23.05
* Goal : *Backend terminée et frontend amorcé + adaptation conception*

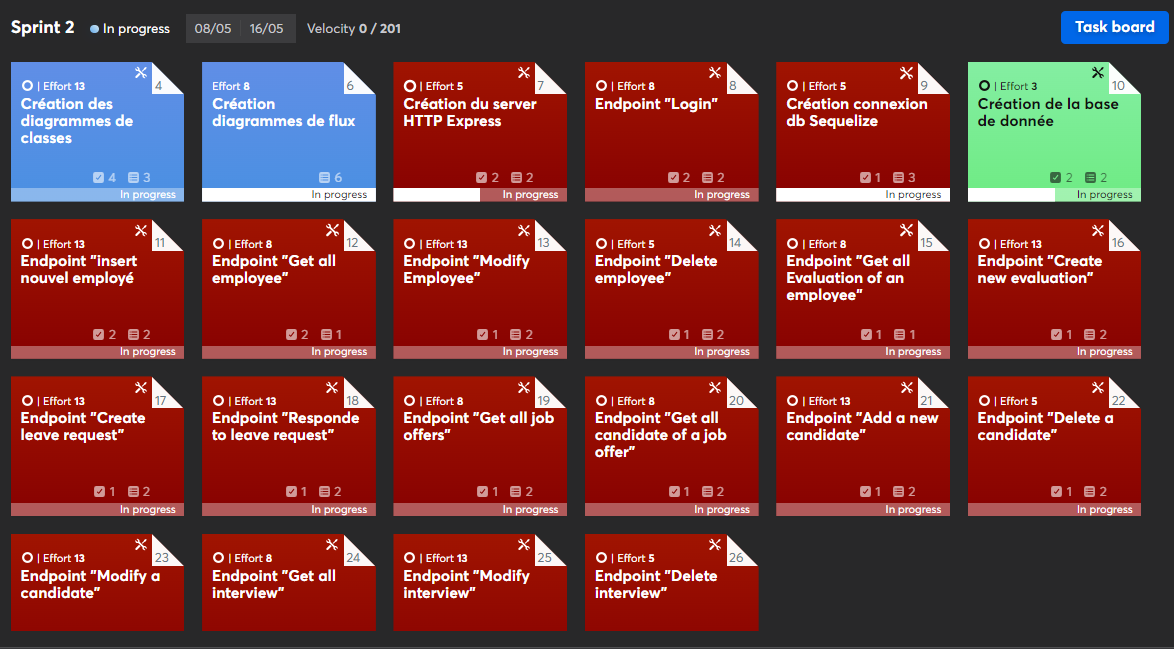
Sprint 4 :

* Date : 24.05 🡺 29.05
* Goal : *Frontend terminé + adaptation de la conception*

Voici la planification du sprint 1 :

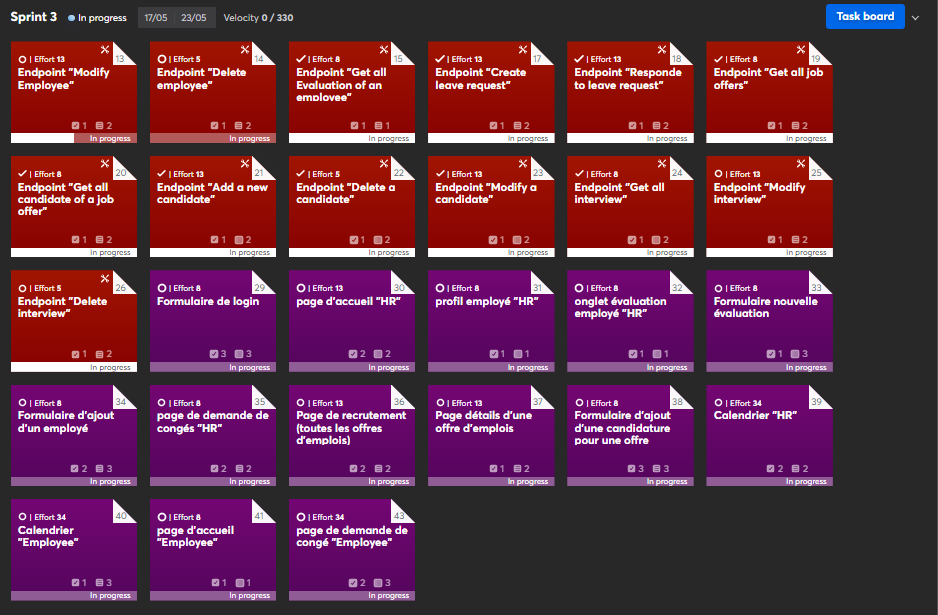


Voici la planification du sprint 2 :



Comme vous pouvez le voir les stories des diagrammes de classes et de flux n’ont pas été entièrement terminée lors du sprint précédent

Voici la planification du sprint 3 :



Comme vous pouvez le constater il y a eu un léger problème à ce stade. Après avoir écrit le code de tous les endpoints du 2ème sprint, le jour avant le sprint review j’ai décidé de légèrement modifié la structure de la base de données (ajout d’une nouvelle table ainsi que des changements dans les champs. Ceci a perturbél le comportement des endpoints, suffisamment pour faire échouer la plupart des tests. J’ai donc décidé de commencer le frontend et de modifier le backend au fur et à mesure afin de faire correspondre le backend avec le frontend.

## Maquettes

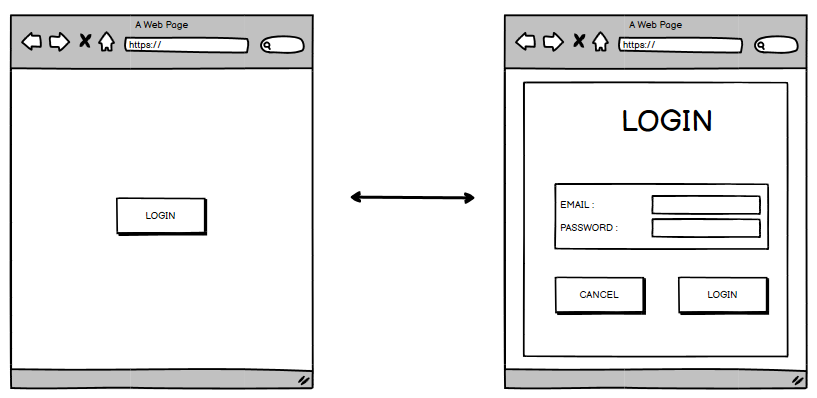


Figure 1 : page de Login

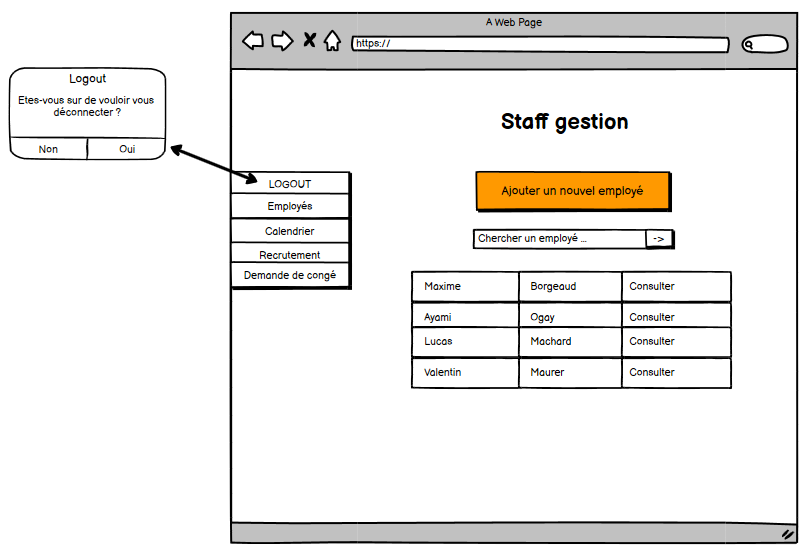


Figure 2 : RH - Page d’accueil

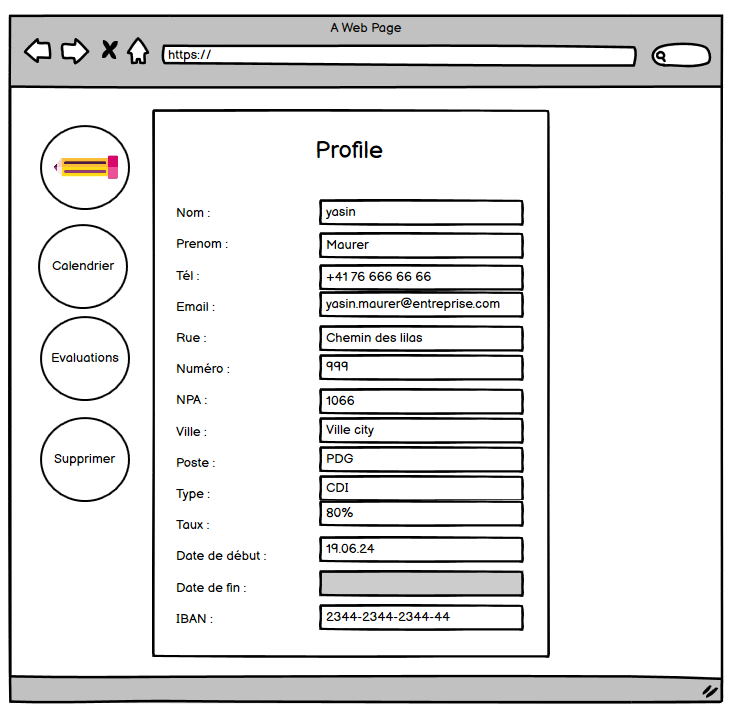


Figure 3 : RH – profil d’employé

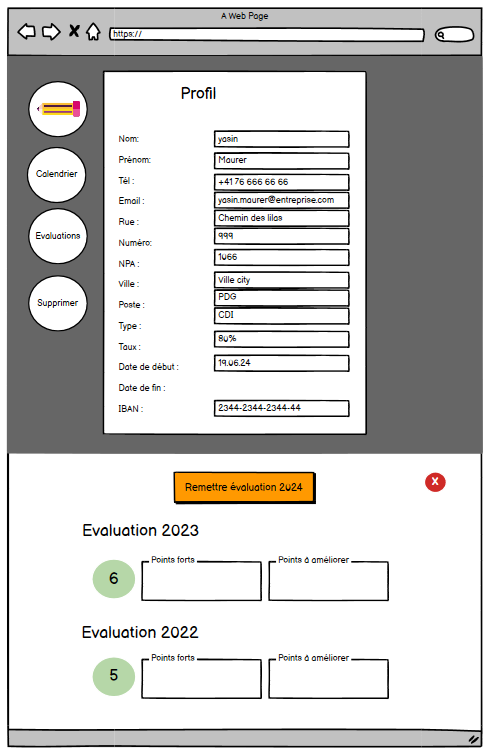


Figure 4 : RH – profil d’employé avec évaluations ouvertes

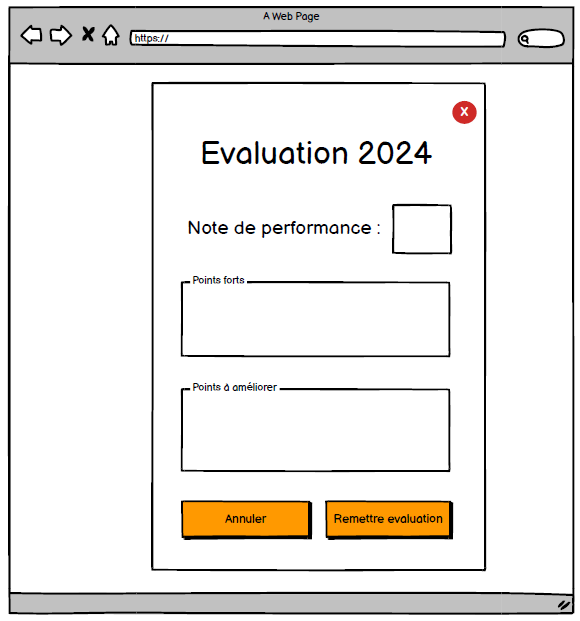


Figure 5 : RH – Ajouter une nouvelle évaluation

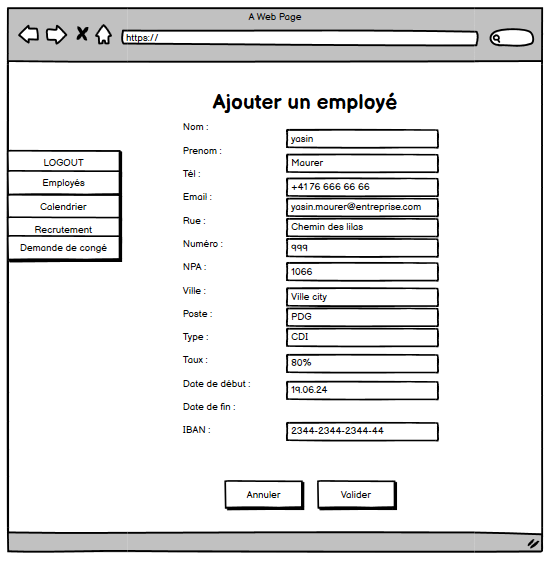


Figure 5 : RH - Ajouter un nouvel employé

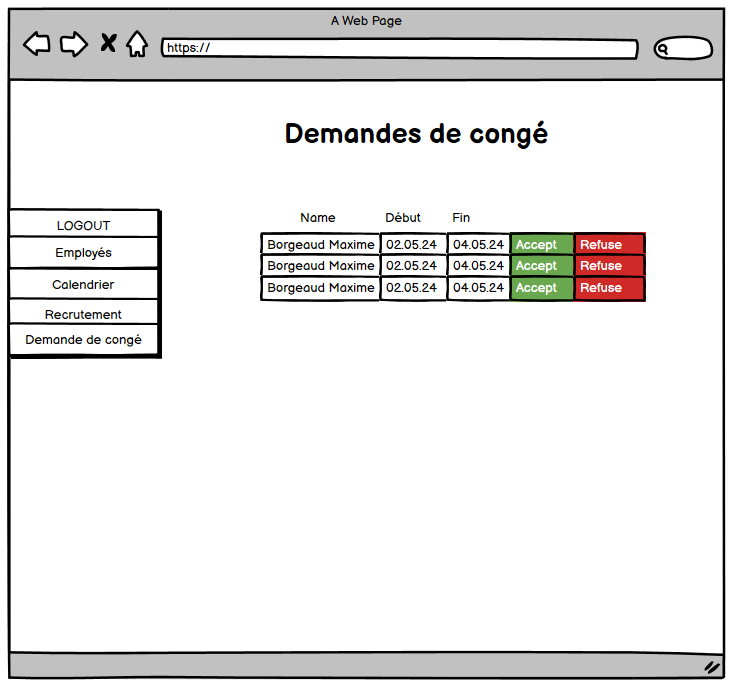


Figure 6 : RH - Gestion des demandes de congés

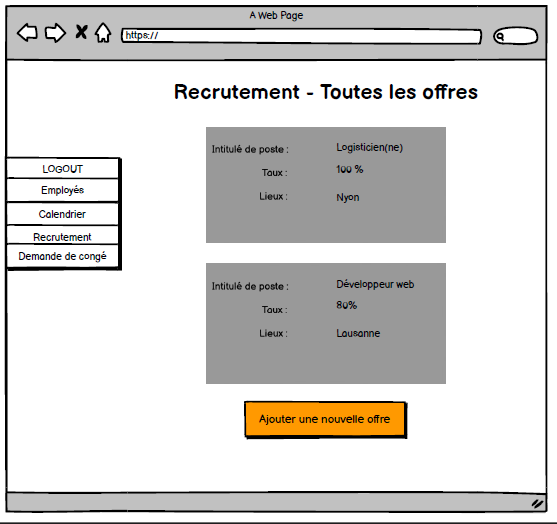


Figure 7 : Toutes les offres d’emplois

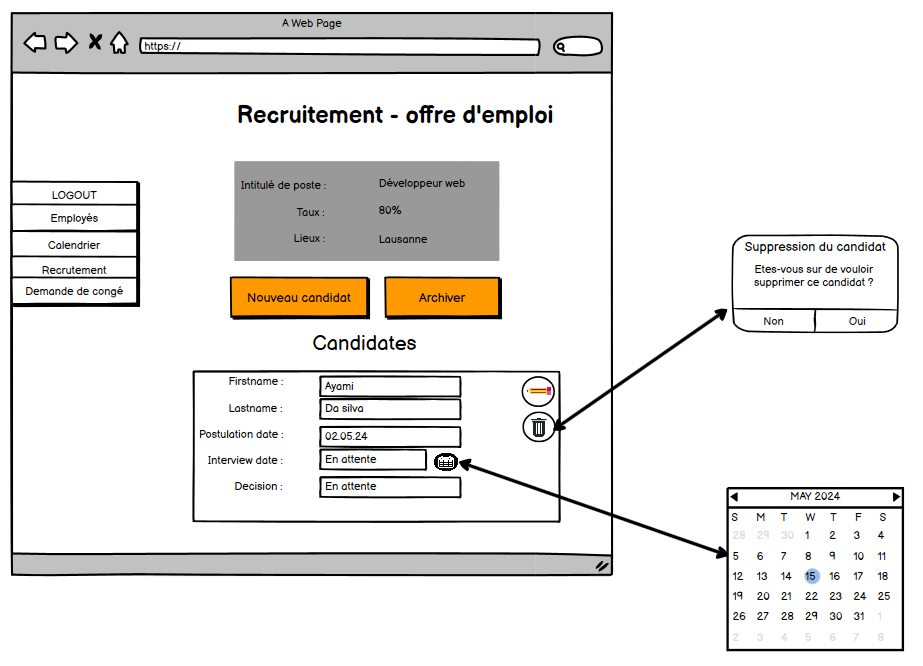


Figure 8 : RH - Détails d’une offre d’emploi

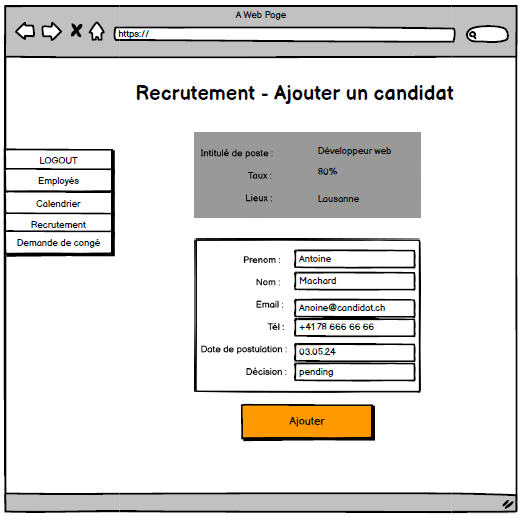


Figure 9 : RH - Ajouter un candidat

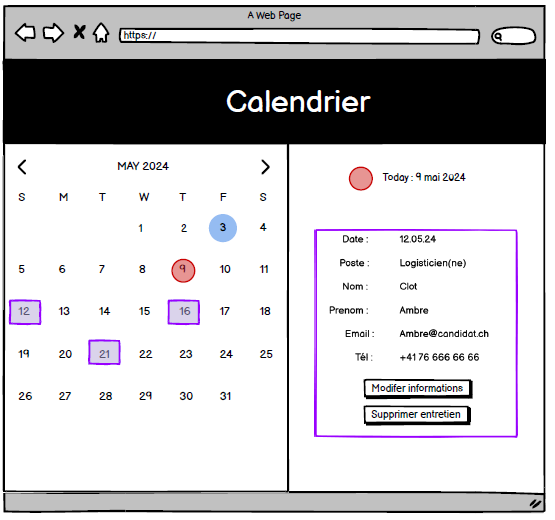


Figure 10 : RH – Calendrier

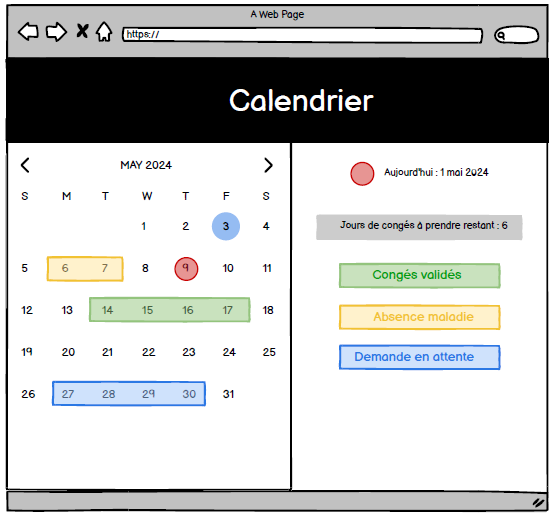


Figure 11 : RH - Calendrier d’un employé

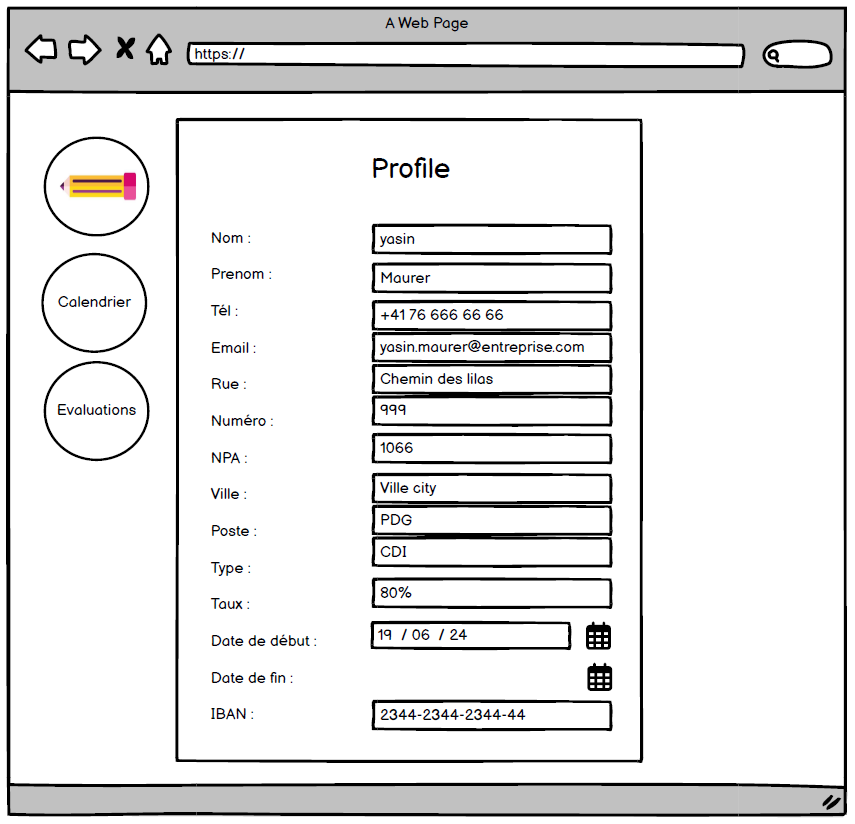


Figure 12 : employee – page d’accueil

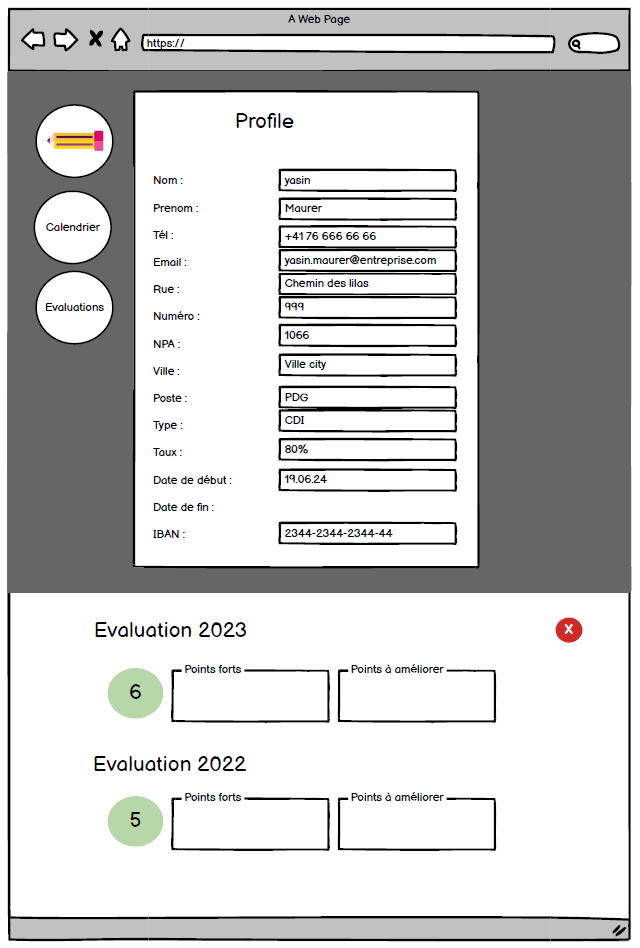


Figure 13 : Employé – page d’accueil avec évaluations ouvertes

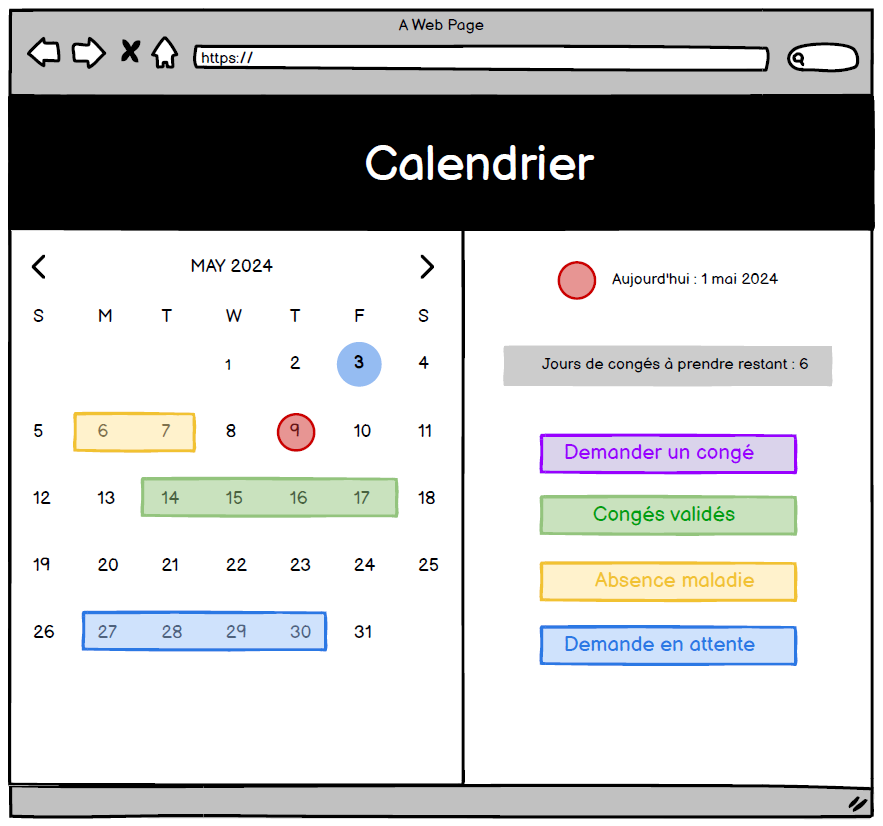


Figure 14 : Employé - calendrier

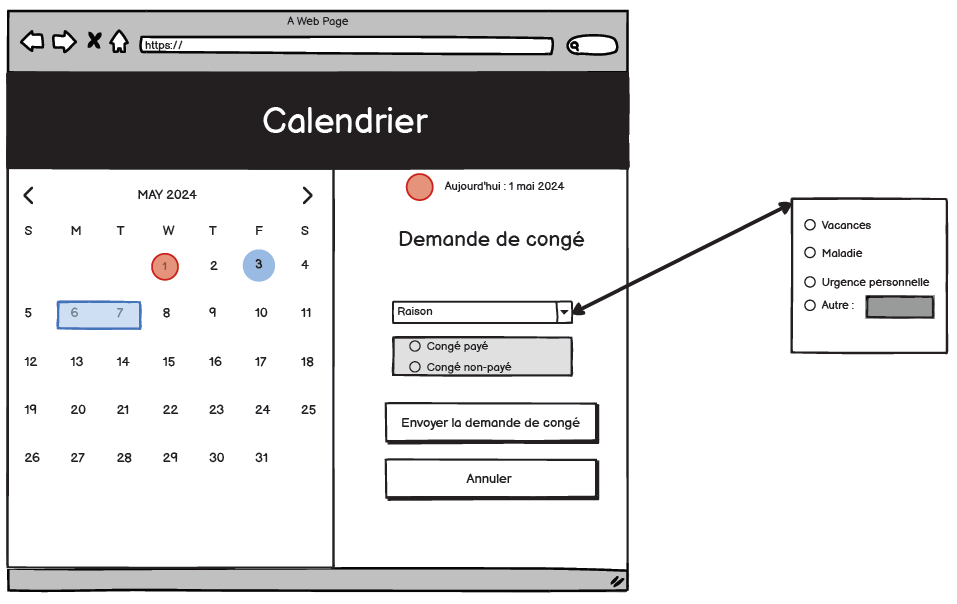
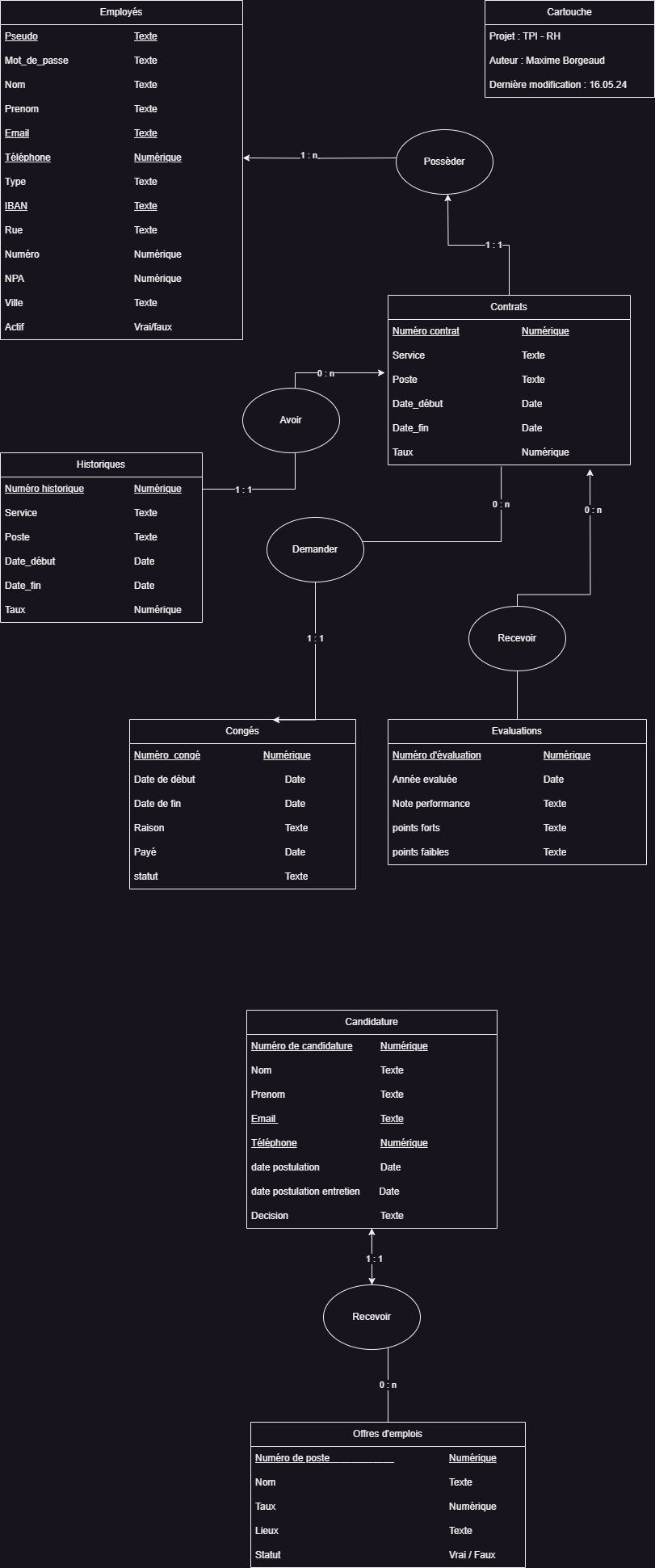


Figure 15 : Employé - Demande de congé

## Modèle conceptuel de données

Voici le MCD final



Concrètement la table « Employé » contient les données personnelles d’un membre du personnel. Sa table associée, « Contract » a les données complémentaires lié à son travail dans l’entreprise. Si des informations doivent changer pour cet employé, un backup des anciennes données est automatiquement créé afin de garder trace de tous changements. Les tables « Evaluation » et « congé » sont également liées à « Contract ».

# Conception

## Architecture

Cette application est conçue comme « client léger » dans le sens qu’il y a une architecture type client-serveur mais rien (ou presque) n’est installé sur le client. J’ai opté pour une architecture application web plutôt que « site web ». Le serveur, ne fournit qu’un seul fichier html qui évolue selon les actions de l’utilisateur, ce qui optimise les performances car le serveur renvoie beaucoup moins de fichiers. La création de bundle Webpack va également dans cette direction.

## Outils

Les outils utilisés pour la réalisation de l’application sont les suivants :

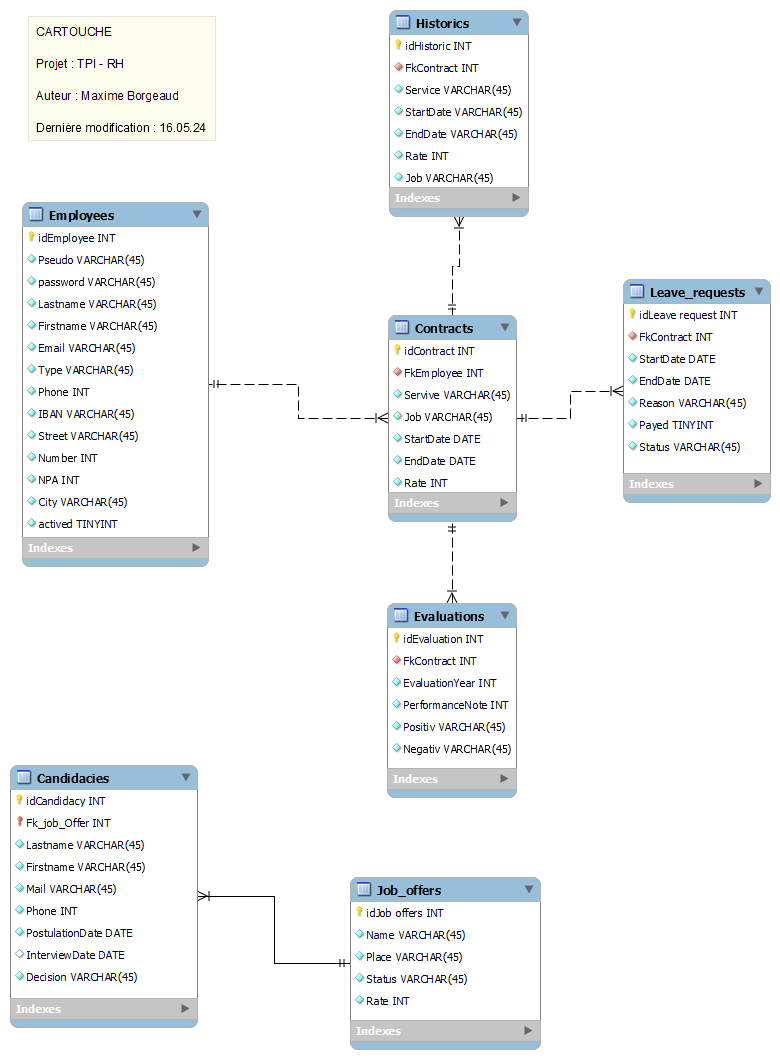
* Editeur de texte : Visual Studio Code
* Hébergement (db) : Apache (local), SwissCenter(online)
* Requêtes http : Insomnia
* Langages de programmation : Javascript, Html-css
* Frameworks/bibliothèques : NodeJS, VueJS, webPack

## Concepts

Voici les principaux concepts qui composent le projet :

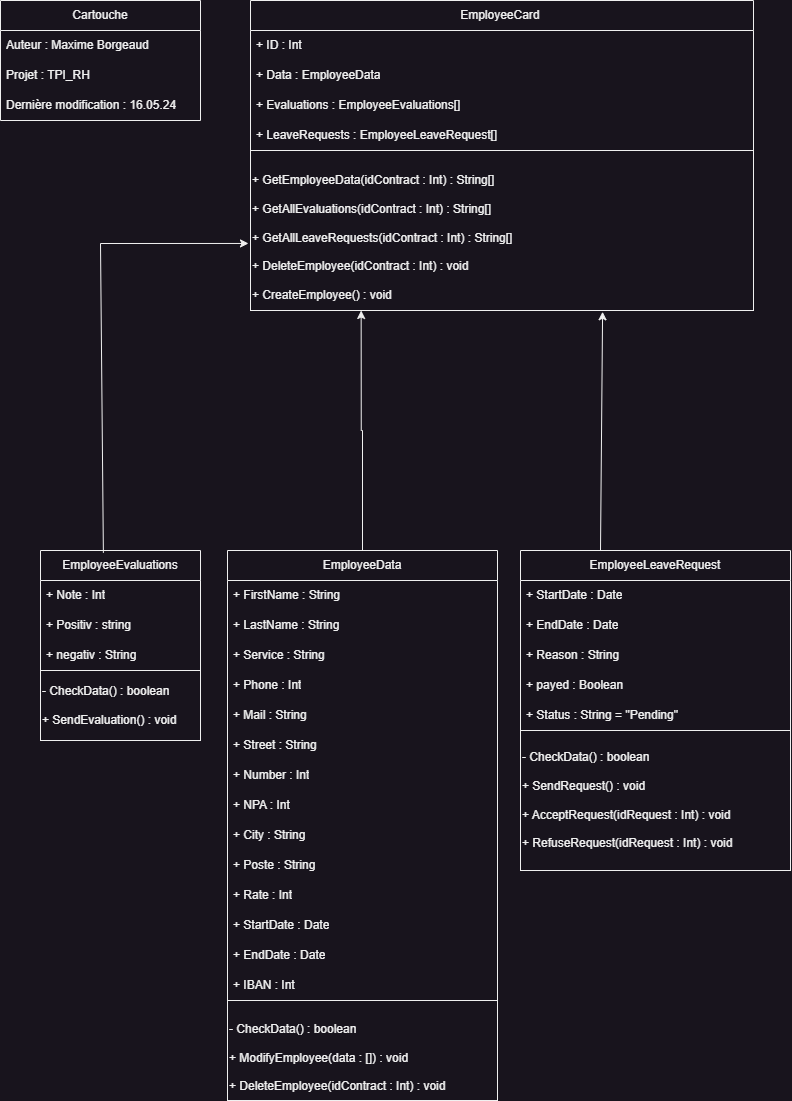
* Modèle – Vue - Contrôleur
* Programmation Orienté Objet
* Requêtes HTTP
* Un système d’authentification hashé
* Base de données MariaDb

## Modèle Logique de Données

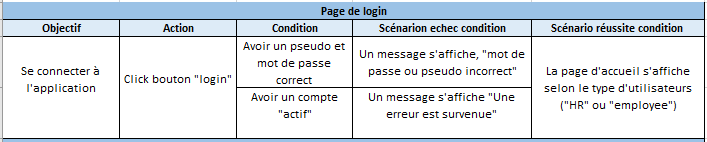


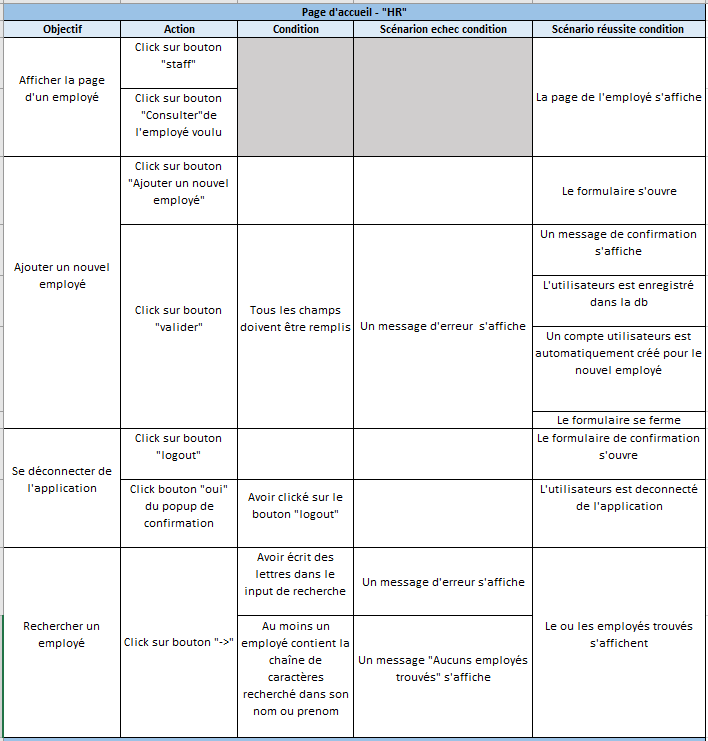
## Diagrammes de classes

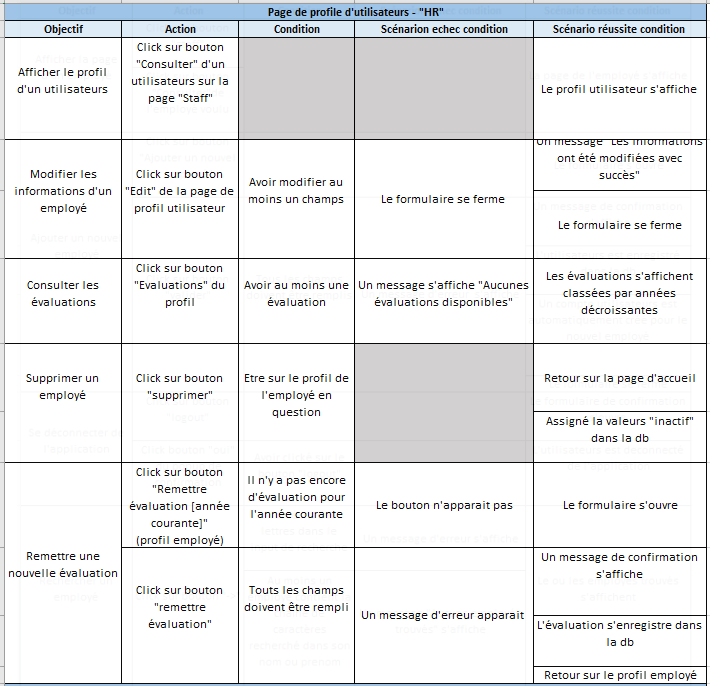
Voici la version finale du diagramme de classe

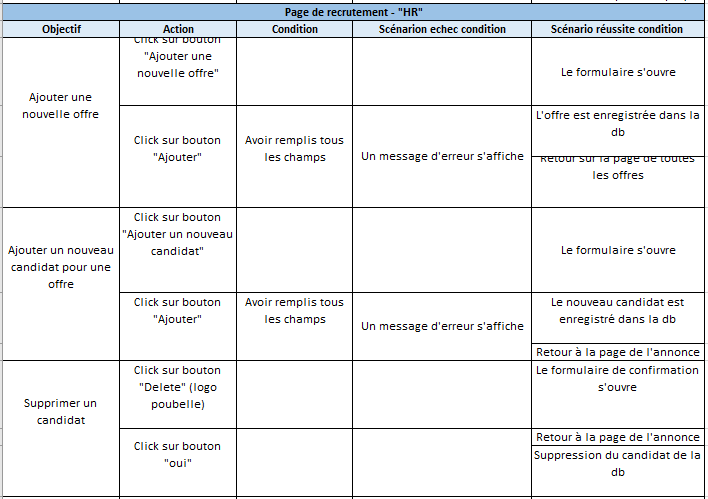


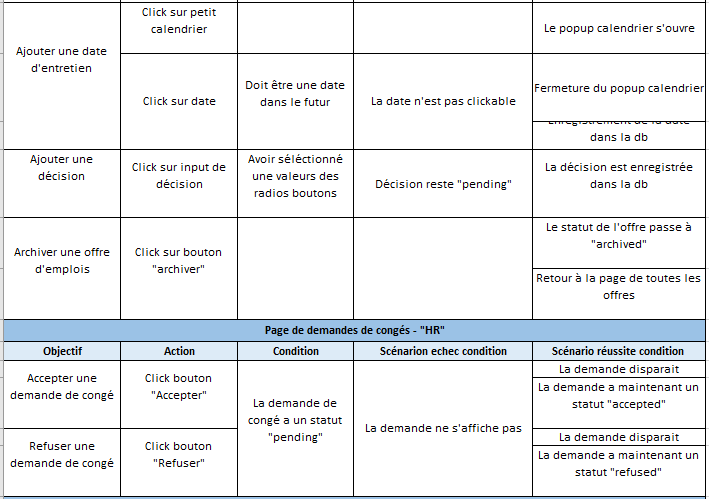
User cases

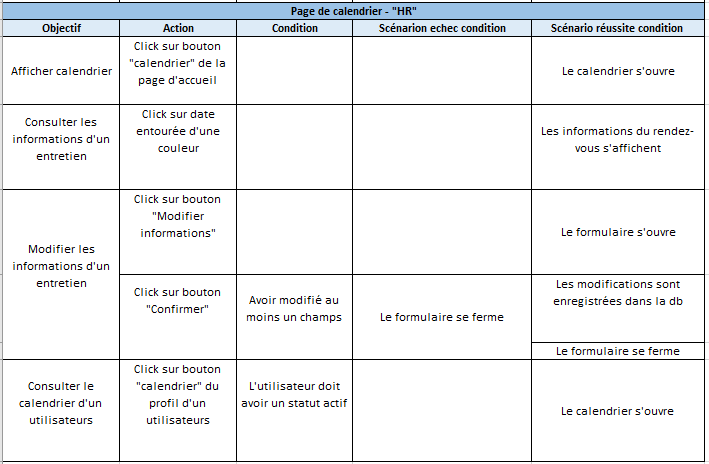












## Gestion des erreurs

Lorsque les utilisateurs envoient des requêtes au serveur, il peut se passer une multitude d’erreurs qui peuvent bloquer le serveur. Le but est que le serveur ne s’arrête jamais même lorsqu’un cas problématique se présente. Afin de gérer cela correctement j’attache un statut de requête à chaque réponse du serveur :

* Les erreurs causées par les utilisateurs, par exemple lorsqu’il envoie des données invalides etc. (status : 400)
* Les erreurs causées par le serveur, par exemple s’il y a des problèmes avec le réseau, avec la base de données ou un bug (status : 500)
* Une requête valide qui a réussi sa tâche (status : 200)

J’attache également toujours un message à la requête afin d’afficher un message selon le résultat. Concernant les erreur 400 il y en a 2 type « error\_unicity » si l’on tente d’insérer des données qui contredit une règle d’unicité et « error\_data » qui est retourné lorsque les données envoyées par le clients sont invalides pour différentes raisons.

# Réalisation

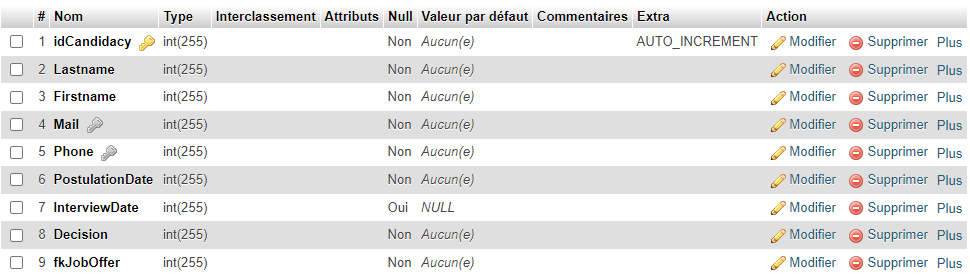
## Base de données

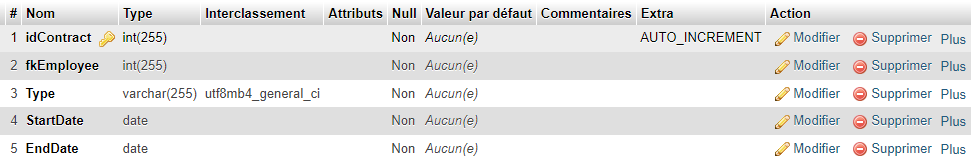
### Structure

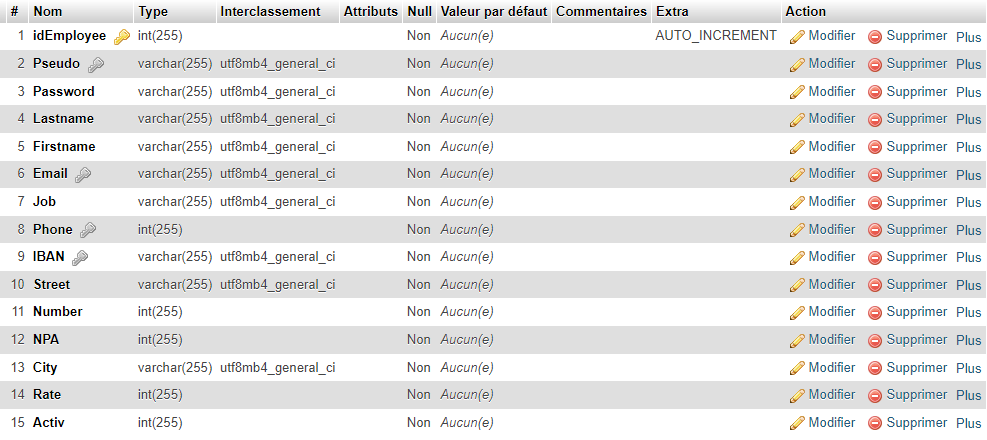
Il s’agit d’une base de données « Mariadb » que j’ai créé manuellement en local avec Xampp. La liaison avec NodeJS sera faite grâce à l’ORM\* « Sequelize ». Voici la structure :

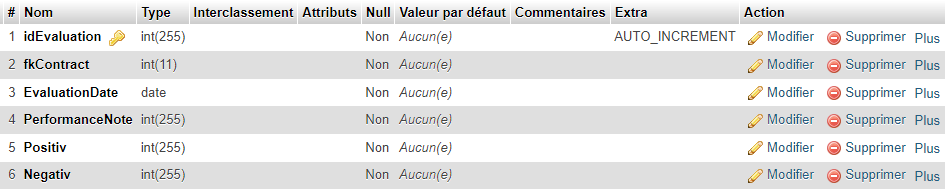
*Clefs jaune 🡺 Primary keys*

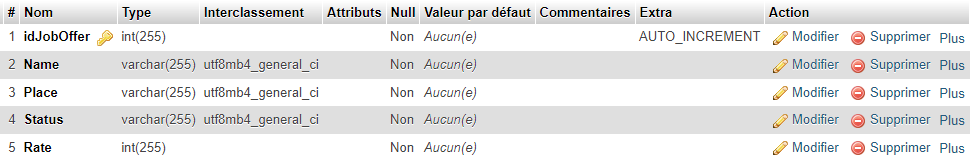
*Clefs grises 🡺 unicity constraint*

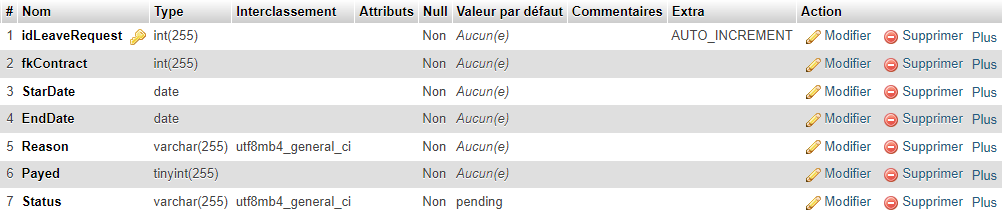




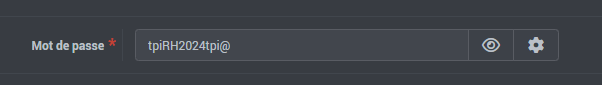






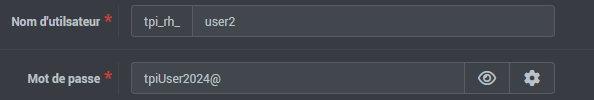


Mdp : admin



User :





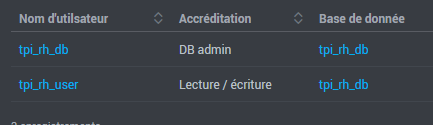
### Mise en ligne de la base de données

Pour que toutes les parties prenantes du projet puissent tester la base de données, j’ai décidé de la mettre sur swisscenter. Pour cela j’ai simplement importé le fichier SQL généré par Xampp.

J’ai également décidé de créer 2 bases de données, la première est pour le développeur (moi-même) afin de pouvoir tester sans limites ni craintes. La deuxième est celle que je remettrai lors du rendu du projet, elle contiendra un jet de données qui permettra de tester toutes les fonctionnalités de l’application.

### Droits d’accès

Pour des questions de sécurité, j’ai créé un second utilisateur qui n’a que les droits lecture/écriture. C’est donc celui-ci qui sera utilisé par défaut dans l’application afin de ne pas utiliser ‘root’ à chaque requête.

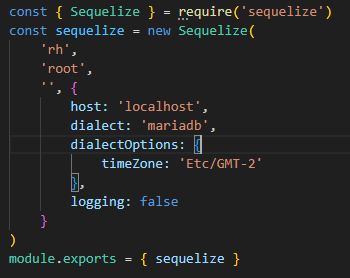


### Insertion des données

J’ai demandé à ChatGPT de générer les fichiers JSON avec les données qui respectent la logique de ce qu’attendent les endpoints. A l’heure où je rends ce dossier il y a 3 utilisateurs RH dans la base de données vous trouverez ces fichiers en annexes avec de nouvelles données à tester.

### Connexion

Voici la configuration initiale en local et la version finale, qui se trouve sur le serveur de Swisscenter, afin de se connecter à la base de données :



Version finale 2 (online)

Version initiale 1 (locale)

### Données

J’ai inséré manuellement les données, qui ont été générée avec l’aide de Chat GPT, dont je fichier JSON se trouve en annexe. Il n’y a que les données de la table « joboffers » que j’ai ajouté manuellement depuis PhpMyAdmin car pour le moment le projet ne comprend pas une gestion complète des offres d’emploi.

## Initialisation du projet

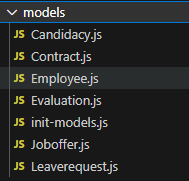
Une fois la base de données créée en local, j’ai initialisé NodeJS avec la commande « npm init ». Puis j’ai modifié le fichier « Package.json » comme ci-dessous afin d’insérer les dépendances minimums nécessaire au développement de l’application. Puis en faisant la commande « npm install », tous les fichiers sont téléchargés dans le dossier « node\_modules ».



## Génération des modèles

J’ai utilisé l’utilitaire « sequelize-auto » afin de générer automatiquement tous les modèles qui correspondent aux tables de la base de données. J’ai pu donc gagner un certain temps et éviter des erreurs. Voici la commande puis son résultat :



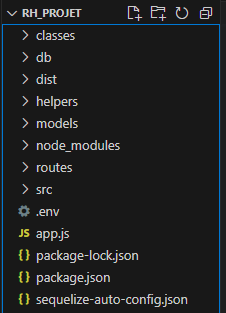


Le fichier « init-models.js » initie les modèles. J’y ai la déclaration des associations de tables (qui sont toutes reliées via une relation 1 à n).

Les autres fichiers sont donc les modèles utiles à Sequelize pour se synchroniser avec la base de données. Je les ai légèrement modifiés afin d’y avoir accès plus facilement.

## Structure du projet

Voici à quoi ressemble l’arborescence du projet :

« app.js » : Il s’agit du fichier racine du server Express. Il gère toutes les requêtes http du type « subscribe » « login » etc. Si aucune route n’est

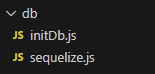
« package.json «  et « package-lock.json » : sont des fichiers qui montrent les dépendances du projet, ce dernier montre plus de détails.

« node\_modules » : contient toutes les librairies nécessaires au bon fonctionnement de l’application.

« .env » : fichier qui contient les variables d’environnement

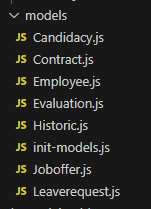
‘’ACCESS\_TOKEN\_SECRET’’ & ‘’REFRESH\_TOKEN\_SECRET’’, qui servent de clefs d’identifications des utilisateurs.

### Dossier « db »

« initDb » : S’authentifie auprès de la base de données grâces aux infos présentes dans « sequelize.js ». Cette partie de code s’exécute lors du démarrage du server.

« sequelize.js » : Présente les informations de connexion à la base de données comme le nom d’utilisateur, mot de passe etc.

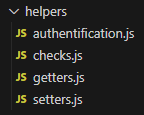
### Dossier « models »



« init-models » : synchronise tous les modèles avec la connexion de la base de données (« db/sequelize.js »). Il s’y trouve également la déclaration des relations entre les tables.

Les autres fichiers sont les modèles eux-mêmes.

### Dossier « helpers »

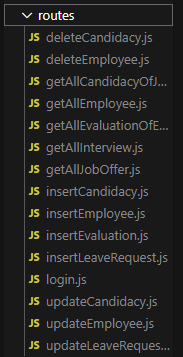
« authentifications » : Ce fichier fait la gestion des TokenJWT des requêtes, afin d’autoriser ou non l’accès aux endpoints.

« checks » : Ce fichier contient des fonctions qui checkent la validité des données avant de les insérer.

« getters » : Ce fichier contient quelques fonctions qui facilitent la récupération de données de la base de données.

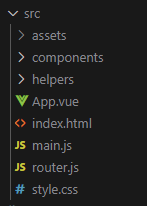
« setters » : Ce fichiers contient une fonction qui formate les tableaux de données à renvoyer aux clients.

## Dossier « routes »



Ce dossier contient les fichiers serveurs qui réceptionne les requêtes http, effectuent le traitement adéquat des données puis renvoie une réponse au client. Les fichiers sont nommés selon l’action du endpoint (insert – get – update – delete).

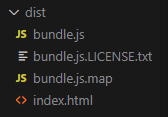
## Dossier « SRC » et « dist »



Il s’agit des fichiers de code frontend sur lesquels j’ai travaillé et qui ne servent qu’au développement. Ce dossier a été généré automatiquement par NPM .« assets » dans lequel devrait se trouver normalement les fichiers CSS et images. Le dossier « components » contient tous les composants VueJS du projet. Le dossier helpers ne contient que une seule fonction très utilisée qui vérifie que l’utilisateurs est connecté et renvoie toutes les données.

« App.vue » : Il s’agit du composant racine de Vue pour notre application. On y utilise notamment le routeur de Vue pour afficher le bon composant

« main.js » : C’est le fichier qui sert de point d’entrée de l’application. Pour créer son bundle, webpack part de ce fichier.



Il s’agit du dossier finalement renvoyé au client, c’est le bundle de webPack. Tous le code javascript est minifié et optimisé afin d’être plus léger.

## Logique des endpoints

Dans cette section je décris chaque endpoint et la logique qui lui est associée.

* **Login**

Ce endpoint va checker si les informations reçues correspondent bien à un pseudo et mot de passe valides. Si c’est le cas, le serveur renvoie toutes les données personnelles, contractuelles ainsi que les évaluations et demandes de congés de l’utilisateur.

* **Insert employee**

Ce endpoint insère les données d’un nouvel employé dans les tables « employees » et « contracts ». Il crée dynamiquement un pseudo pour l’utilisateur avec son prenom et son ID de la table « employees ». Si un autre employé a déjà la même email, téléphone ou IBAN le serveur n’insère rien et renvoie une erreure.

* **Insert leave request**

Ce endpoint récupère les informations d’une demande de congé, vérifie les informations avant des les insérer ou de renvoyer une erreur. Pour le moment le endpoint vérifie qu’il n’y aie pas d’autre demande de congé avec une date de début identique. **Insert Candidacies**

Ce endpoint vérifie l’existances de l’offre d’emplois pour laquelle la candidature se présente, puis check que la candidature n’existe pas déjà pour cette offre (par son email et numéro de téléphone).

* **Insert évaluation**

Ce endpoint vérifie s’il existe déjà une évaluation existente pour l’année de la requête et bloque l’insertion si c’est le cas. L’idée est de ne pas avoir deux évaluations pour une même année.

* **Get all job offers**

Ce endpoint va chercher toutes les offres d’emplois qui ont un statut « pending » (en attente). S’il n’y en a aucunes, il renvoie un tableau vide

* **Get all candidacies of a job offer**

Ce endpoint retourne toutes les candidatures associées à une offre d’emploi existante.

* **Get all employees**

Ce endpoint renvoie l’ID, le nom et prenom de tous les employés actifs. Je voulais utiliser ce endpoint pour la page « chercher un employé » des RH. Afin d’économiser les requêtes au serveur, au chargement de la page le programme charge ces données afin de pouvoir filtrerss les employés sans devoir faire une requête.

* **Get all evaluations of an employee**

Ce endpoint retourne toutes les données qui concernent les évaluations d’un employé existant.

* **Get all interviews dates**

Ce endpoint retourne toutes les candidatures qui ont une date d’interview fixée dans le futur.

* **Update candidacy**

Ce endpoint remplace les anciennes données par les nouvelles. La vérification que au moins un champ a été modifié est faite dans le frontend avant d’envoyer la requête.

* **Update leave request**

Ce endpoint sert aux RH afin de refuser ou d’accepter une demande de congé. Actuellement il check si la demande existe bien et si le statut est valide (« accepted », « refused » ou « pending »).

* **Update employee**

Ce endpoint enregistre les nouvelles informations d’un employé existant.

* **Update Contract**

Ce endpoint enregistre les nouvelles informations du contrat d’un employé. A chaque modification d’un record, un nouvel enregistrement est fait dans la table « historique » qui contient toutes les données de l’ancien contrat.

* **Delete candidacy**

Ce endpoint supprime une candidature existante d’une offre d’emploi existante.

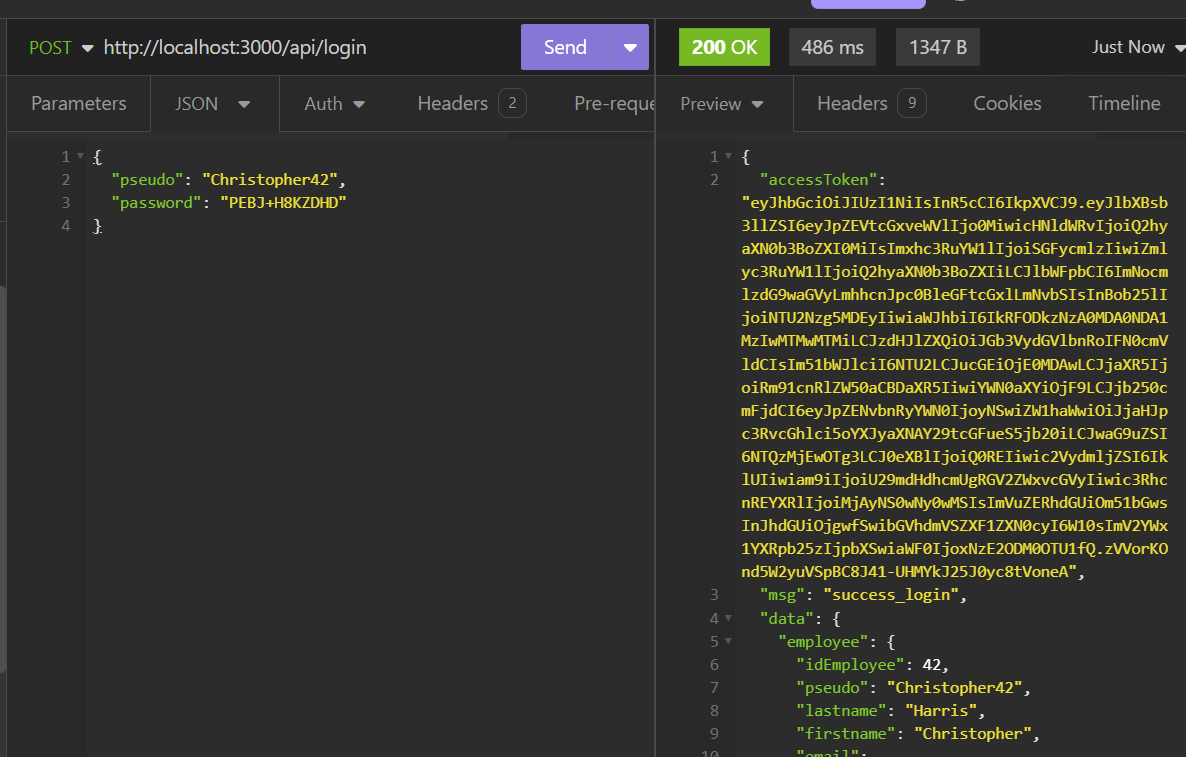
* **Delete Employee**

Ce endpoint passe le champs « activ » de la table « employee » à false puis supprime le record de la table « contracts » correspondant à l’employé tout en faisant un backup dans la table « historics ». Il va également supprimer toutes les demandes de congé et les évaluations liées à cet utilisateur.

## Stratégie de tests

J’ai utilisé l’outil gratuit « Insomnia » afin de tester les endpoints des requêtes. Vous trouverez la collection de requêtes dans les annexes ainsi qu’une série de données en json afin de pouvoir tester tous les endpoints.

### LOGIN



Avec des identifiant incorrects :

Une image contenant texte, Logiciel multimédia, logiciel, capture d’écran

Description générée automatiquement

### Insert employee

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

Si on tente de réinsérer une deuxième fois les mêmes données :

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

### Insert leave request

Une image contenant texte, Logiciel multimédia, logiciel, capture d’écran

Description générée automatiquement

Si on tente de réinsérer une deuxième fois la même « startDate » :

Une image contenant texte, Logiciel multimédia, logiciel, capture d’écran

Description générée automatiquement

### Update leave request

Une image contenant texte, Logiciel multimédia, logiciel, capture d’écran

Description générée automatiquement

Si on tente de modifier une demande de congé qui n’existe pas :

Une image contenant texte, Logiciel multimédia, logiciel, capture d’écran

Description générée automatiquement

Si on tente de modifier une demande de congé avec un autre statut que « accepted », « refused » ou « pending » :

Une image contenant texte, Logiciel multimédia, logiciel, capture d’écran

Description générée automatiquement

### Insert candidacy

Une image contenant texte, logiciel, Logiciel multimédia, capture d’écran

Description générée automatiquement

Si je tente d’insérer les mêmes données pour une offre d’emploi différente (fkJobOffer) :

Une image contenant texte, logiciel, Logiciel multimédia, capture d’écran

Description générée automatiquement

Si je tente d’insérer les mêmes données pour la même offre d’emploi que précédemment :

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquement

Maintenant lorsque je tente d’insérer des données pour une offre d’emploi qui est inexistante ou qui a un statut différent de « en attente » :

Une image contenant texte, logiciel, Logiciel multimédia, Logiciel de graphisme

Description générée automatiquement

### Delete candidacy

Une image contenant texte, Logiciel multimédia, logiciel, capture d’écran

Description générée automatiquement

Si je tente de supprimer une candidature qui n’existe pas :

Une image contenant texte, Logiciel multimédia, logiciel, capture d’écran

Description générée automatiquement

### Delete Employee

Une image contenant texte, Logiciel multimédia, logiciel, capture d’écran

Description générée automatiquement

Si on tente de supprimer un employé qui n’existe pas :

Une image contenant texte, Logiciel multimédia, logiciel, capture d’écran

Description générée automatiquement

# Conclusion

## Erreurs/imprécisions restantes

* **Fonctionnalités frontend**

La première chose à signaler ici est la faite que beaucoup de fonctionnalités ne sont pas présentes dans l’application frontend. Je me suis donc concentré sur la partie des demandes de congés avec le temps que j’avais à ma disposition.

* **Check du endpoint « insertLeaveRequest »**

Si il y a une plage de date, il Faudrait checker la validité pour chacune et tout insérer si tout est valide et ne rien insérer si une date est invalide.

* **Changement de mot de passe première connexion**

Actuellement, le système génère automatiquement un mot de passe aléatoire, que l’employé RH transmettrait à l’employé concerné. Il faut absolument mettre en place un système qui force l’utilisateurs à changer de mot de passe à la première connexion et éventuellement par la suite une fois par années pour la sécurité

* **Modification d’une évaluation existante**

Pour rajouter cette petite fonctionnalité, il faudrait rajouter un nouveau endpoint et de plus je rajouterais les champs « updatedAt » et « updatedBy » dans la table « evaluations » qui contiendraient la date de modification ainsi que l’id de l’employé RH. Il faudrait également faire que les employés RH ne puisse pas modifier leurs propres évaluations.

* **Mise à jour des données du calendrier**

Lorsqu’un utilisateur envoie une demande de congé valide qui s’inscrit dans la base de donnée, il faudrait ajouter la nouvelle demande de congé dans les données de l’utilisateurs

## Améliorations possibles

* Il serait intéressant d’ajouter la gestion des salaires, avec par exemple l’établissement de grilles salariales, le calcule des charges sociales, calculs de bonus de performances/d’ancienneté.
* Il serait également possible de rajouter une plateforme ou les offres d’emplois sont disponibles sans login et avec des formulaires de postulation. Dans ce cas il serait utile de renommer la table »candidacy » en « candidate » et créer une table intermédiaire entre elle et « joboffers ». Ainsi un même candidat pourra postuler pour plusieurs poste sanspour autant créer de redondance dans la base de données
* Centraliser tous les query Sequelize dans un seul fichier pourrait éventuellement rendre le code plus propre.

## Conclusion personnelle

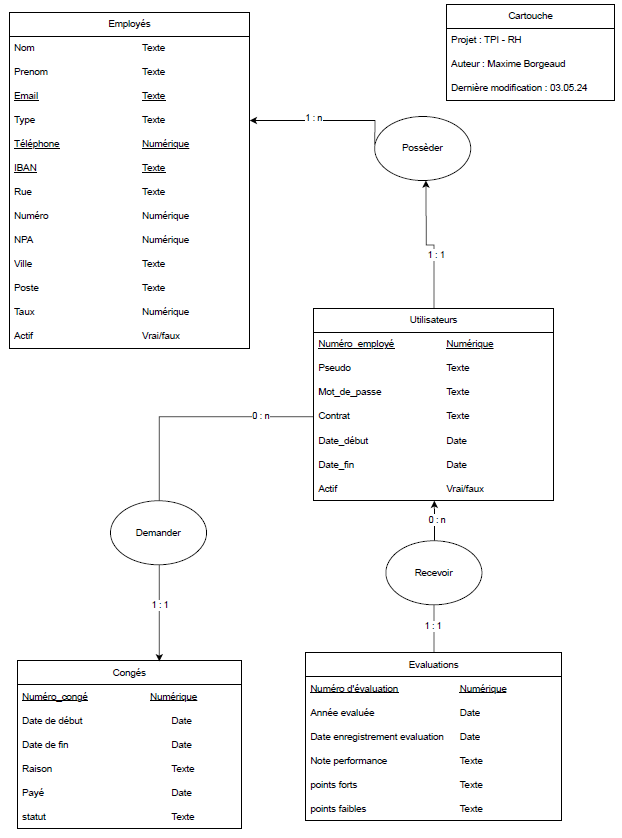
# Archives

## Base de données

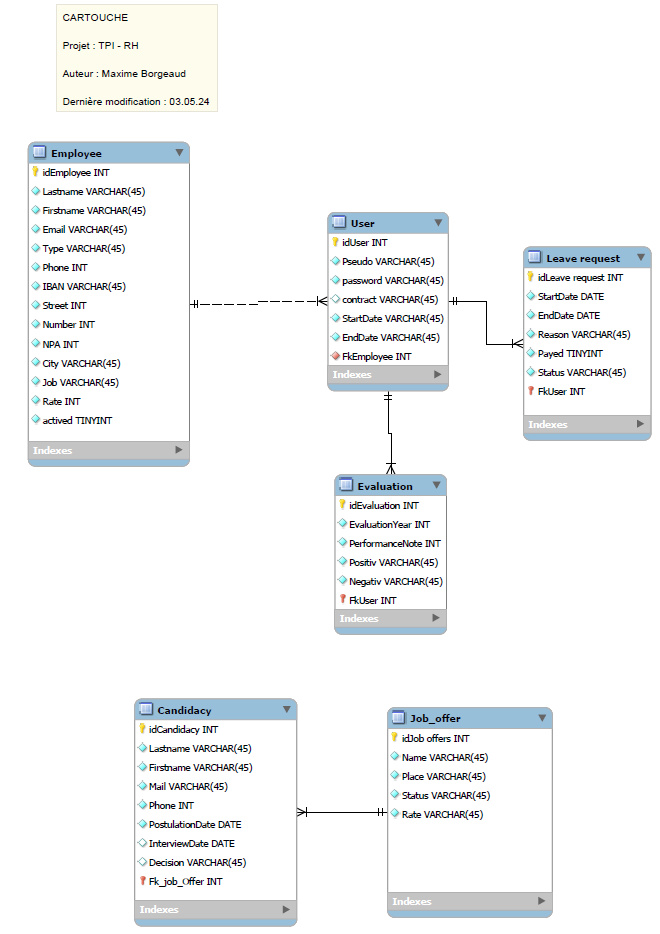
J’ai fait quelques modifications entre la dernière version de la base de données présentée plus haut et l’ancienne version présentée ici. J’ai déplacé les champs « Job » et « Rate » de la table « Employees » vers la table « Contracts ». J’y ai également ajouté la table « historics ». Lorsque’un utilisateurs modifiera une donnée de la table « contracts », il y aura une sauvagarde automatique des anciennes données dans la table « historics ».

PROBLEME SUPPRESSION CONTRACT

### MCD



### MLD



# Annexes

## Résumé du projet

Comme situation de départ, je dois créer une petite application qui doit permettre de gérer les points principaux qui composent les ressources humaines. Principalement la gestion des employés, demandes de congés, gestion des performances et des recrutements avec évidemment de la sécurité sur l’authentification des utilisateurs.

//Mise en œuvre

J’ai commencé la mise en œuvre par l’implémentation du backend. J’ai créé le serveur express, puis configuré les endpoints qui répondent aux requêtes http correspondantes. Ces requêtes font un CRUD complet sur les différents points évoqués au paragraphe précédent. On y insert, lit, modifie et supprime des éléments, ou non selon certains critères. Une fois que les endpoints ont été configuré, j’ai commencé le frontend avec VUE. C’est à cette étape que j’ai compris que finir toutes les interfaces serait impossible et donc nous avons convenu avec mon chef de projet que me concentrer sur la partie frontend des demandes de congés serait suffisant.

// Résultats

Comme résultat je présente une API NodeJS qui répond presque à toutes les demandes de l’applications (voir section « Erreurs/imprécisions restantes »). Les règles des checks furent une réelle difficulté car n’ayant aucunes compétences métiers je n’avais parfois pas trop idée quelles sont les bonnes pratiques.

## Liste documents fournis

## CDC

## Glossaire

Bibliothèque :

Bundle :

Champs :

Endpoint :

Framework :

http :

**ORM** **()** **:**

Record :

Requête :

**Sprint :**

**Table :**

Dans le contexte de la méthode Agile, un sprint est une période délimitée pendant laquelle une équipe spécifique travaille à compléter une ou plusieurs tâches

## Sources

Sequelize auto :

<https://ratneshpandey.hashnode.dev/auto-generate-models-for-sequelize-using-sequelize-auto>

Associations sequelize :

<https://sequelize.org/docs/v6/core-concepts/assocs/>

Recherches Javascripts diverses :

<https://www.syncfusion.com/blogs/post/js-commonjs-vs-es-modules>

Transactions Sequelize :

<https://sequelize.org/docs/v6/other-topics/transactions/>

ChatGpt : setRandomPassword