Introduction

Ce rapport a pour but de présenter le travail de ce groupe composé de Manon CHENEVIERE, Hugo GILLES et Mathis LEROUX en se comportant comme une petite entreprise de conception web et d'expliquer de manière détaillée comment fonctionne l'application web conçu. En effet, le projet consiste à créer un outil support à la gestion des entreprises liées à l'université mêlant les modules Bases de données et Technologies Web. Nous aborderons, après avoir présenté le groupe, la partie Base de données de l'application, puis la partie PHP et enfin nous conclurons en imaginons ce qui serait possible d'ajouter dans cette application.

Notre entreprise

MHM est une entreprise créée en 2023 spécialisée dans le développement d'outils web de gestion.

Notre équipe est composée de trois membres : Hugo GILLES : développeur web, Mathis LEROUX : spécialiste base de données et Manon CHENEVIERE : développeuse web.

Pour réaliser ce projet, nous avons procédé comme suit : la base de données a été conçue par les trois membres de concert. Les vues SQL ont été réalisées par M LEROUX alors même que Mme CHENEVIERE réalisait la structure du code PHP.

Par la suite, les différentes vues à réaliser ont été partagées entre les trois membres à parts égales.

Afin de suivre l'avancement de chacun, nous avons utilisé l'outil suivant (l'image a été capturée à un stade antérieur au rendu du projet) :

Tâche	Liens utiles et description	État
Nom de l'application et de l'entreprise	MHM, ManageMe	Terminé
Création du dépôt Git		Terminé
Modèle UML de la BDD	Réalisation du modèle de la BDD, validation de l'encadrant à obtenir	Terminé
Rédaction des classes	Une par table de la BDD	En cours
Elaboration du squelette du code PHP	A réaliser suivant le modèle MVC (dossiers Repository, Entity, Controller)	En cours
Design		Pas comme…
Création de la base de données SQL	Selon le modèle réalisé précédemment	Terminé
Création des requêtes SQL	Selon les vues demander	Terminé
Création des vues SQL		Terminé
Rapport	Voir page précédente pour l'organisation du rapport	Pas comme…
Mise en place de la connexion à la BDD	Commit "mise en place bdd"	En cours

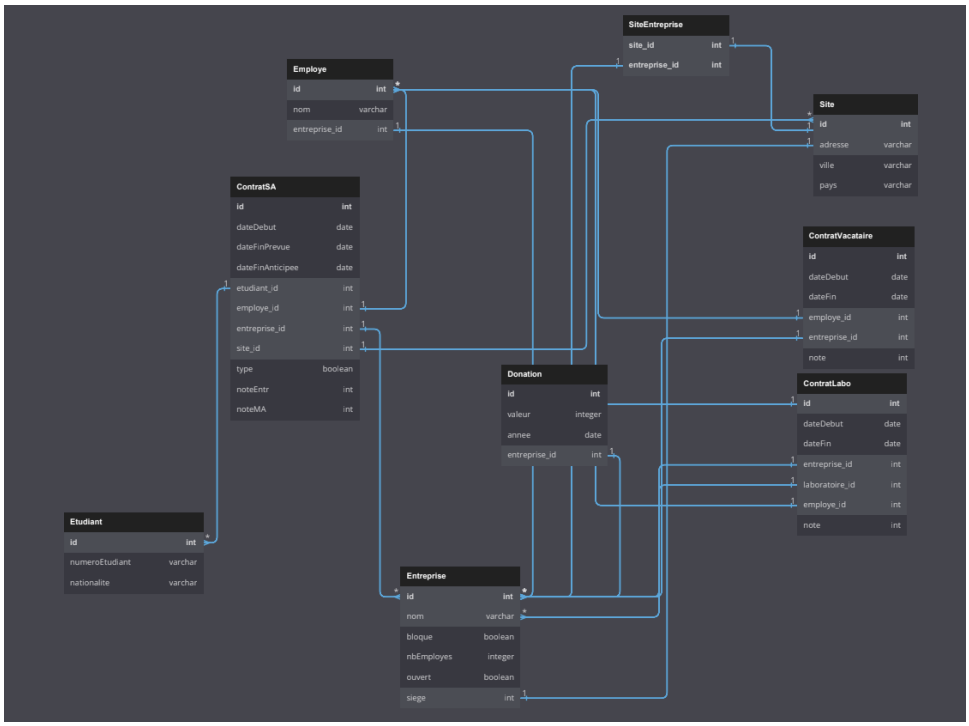
Un échange quotidien était également prévu afin de caractériser les éléments fonctionnels ou non et d'adresser les éventuels problèmes de chaque membre de l'équipe.

Partie base de données

Conception

Après étude de l'échange avec notre client, nous avons réalisé plusieurs propositions de base de données, que nous avons précisées en lien étroit avec M MAHEO.

En accord avec les demandes de notre client, nous avons conçu la base de données suivante :



Nous avons fait le choix de factoriser les vacataires, les tuteurs et les référents dans une même table Employé, de même qu'un laboratoire est considéré comme une entreprise et est donc enregistré dans la table Entreprise.

Chaque type de contrat possède une table, à l'exception des stages et des alternances qui se partagent la table ContratSA. Ces deux types de contrats sont différenciés par un booléen, 0 signifie que nous sommes face à un contrat d'alternance là où la valeur 1 représente un contrat de stage.

Toutes nos tables contiennent un attribut id, à l'exception de la table SiteEntreprise qui est identifiée de manière unique par un ensemble de deux attributs : l'id de l'entreprise et celui du site qui concernent chaque tuple. Cela permet de faciliter les accès aux données par les Repositories.

Afin de pouvoir manipuler notre base et tester nos requêtes, nous avons rempli notre base de données de valeurs respectant notre schéma. Celles-ci sont disponibles dans le dump.sql présent dans notre repository Git.

Modele logique

Notre base de données répond au schéma logique suivant :

Employe{id NOT NULL, nom, entreprise_id}

Avec Employe[entreprise_id]⊆ Entreprise[id]

Etudiant{id NOT NULL, numeroEtudiant, nationalite}

Entreprise{id NOT NULL, nom, bloque, nbEmployes, ouvert, siege}

Avec Entreprise[siege]⊆ Site[id]

ContratSA{id NOT NULL, dateDebut, dateFinPrevue, dateFinAnticipee, etudiant_id, employe_id, entreprise_id, site_id, type, noteEntr, noteMA}

Avec ContratSA[etudiant_id]⊆ Etudiant[id]

ContratSA[employe_id]⊆ Employe[id]

ContratSA[entreprise_id]⊆ Entreprise[id]

ContratSA[site_id]⊆ Site[id]

ContratLabo{id NOT NULL, dateDebut, dateFin, laboratoire_id, employe_id, entreprise_id,

note}

ContratLabo[laboratoire_id]⊆ Entreprise[id]

ContratLabo[employe_id]⊆ Employe[id]

ContratLabo[entreprise_id]⊆ Entreprise[id]

ContratVacataire{id NOT NULL, dateDebut, dateFin, employe_id, entreprise_id, note}

Avec ContratVacataire[employe_id]⊆ Employe[id]

ContratVacataire[entreprise_id]⊆ Entreprise[id]

Site{id NOT NULL, adresse, ville, pays}

SiteEntreprise{site_id NOT NULL, entreprise_id NOT NULL}

Avec SiteEntreprise[site_id]⊆ Site[id]

SiteEntreprise[entreprise_id]⊆ Entreprise[id]

Donation{id NOT NULL, valeur, annee, entreprise_id}

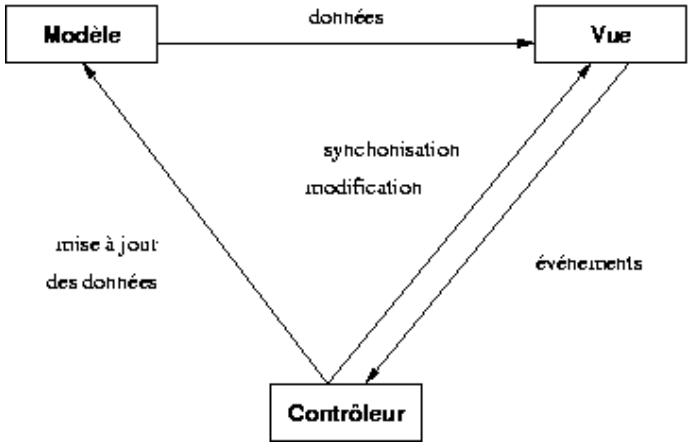
Avec Donation[entreprise_id]⊆ Entreprise[id]

A noter que la création de la table SiteEntreprise résulte de la présence d'une association de type 1-n.

Partie PHP

Le modèle MVC

Le MVC, ou Modèle-Vue-Contrôleur est une architecture logicielle utilisée pour faciliter la gestion d'interface graphique dans les applications web. Les 3 parties qui composent cette convention jouent un rôle précis et interagissent entre elles de la manière suivante :



Modèle :

Le modèle dans l'architecture MVC sert d'intermédiaire entre l'application et la base de données.

En l'occurrence dans notre application se trouvent deux types de fichiers représentant le modèle : les Entities et les Repositories.

Les Entities déclarent simplement les classes dérivées de notre modèle de données avec leurs attributs ainsi que les getters/setters y étant rattachés, là où les Repositories vont concentrer les méthodes permettant de faire le lien avec la base de données selon les vues imposées par notre client.

Vue :

La vue contient le contenu visible pour l'utilisateur. Elle concentre les informations récupérées par le modèle et les présente. Elle peut également contenir des formulaires de modification des informations. A noter que la vue ne fait aucun traitement sur les données, elle se contente de les envoyer aux couches sous-jacentes.

Controllers :

Le contrôleur peut être vu comme un chef d'orchestre servant d'intermédiaire entre les autres couches. Il permet d'ajouter une notion de sécurité en empêchant l'utilisateur d'avoir directement accès aux données. Il interagit avec le modèle en lui envoyant les données saisies par l'utilisateur et appelle les templates nécessaires à l'affichage des vues.

Connexion à la base de données

La connexion à la base de données se fait par l'intermédiaire d'une classe DBConnexion regroupant les informations nécessaires (mot de passe, login, nom de la base). On instancie ensuite des objets de cette classe dès qu'un besoin de connexion à la base de données se manifeste (en l'occurrence dès l'utilisation d'un Repository).

Templates

Les templates sont les fichiers permettant l'affichage des vues. Afin de limiter la redondance de code, nous avons fait le choix de créer un fichier nommé base.php qui contient toutes les informations nécessaires à la déclaration d'un fichier HTML. Les autres templates utilisent ainsi ce fichier pour fonctionner.

Les templates utilisent les valeurs déclarées dans les contrôleurs (souvent des tableaux) afin de les afficher.

Le fichier index.php

Le fichier index.php contient un routeur permettant de choisir la page à afficher selon un paramètre passé dans l'URL appelé "action".

Si action est présent dans l'URL, on rentre dans le routeur qui appelle le contrôleur adéquat, sans quoi la page d'accueil est appelée.

Projections

Rajout de crud

Notre application a atteint un statut de produit livrable, c'est-à-dire que l'intégralité des éléments qui sont demandés dans le projet figure dans notre application. Si nous avions plus de temps pour continuer l'avancement du projet, il serait intéressant d'ajouter une interface de commande pour l'utilisateur qui lui offre la possibilité de modifier la base de données en ayant la possibilité d'appliquer les 4 opérations qui répondent à la convention nommée crud. Un crud désigne les 4 opérations de gestion de base de données, Create, Read, Update et Delete. Une telle interface permettrait à un utilisateur ou simplement un super utilisateur d'administrer la base de données de l'application facilement.

Fonctionnalité responsive

Le "responsive" en web fait référence à la conception de sites web qui s'adaptent automatiquement aux différents appareils utilisés pour les consulter comme les ordinateurs de bureau, les smartphones et les tablettes. Il serait intéressant aussi d'adapter l'application web de telle sorte à ce qu'elle soit présentable et ergonomique pour un téléphone portable par exemple.

Conclusion

Le projet a atteint un statut de fonctionnement convenable, même si beaucoup de fonctionnalités utiles peuvent accompagner l'application actuelle, celle-ci répond au cahier des charges imposé par le client.