..

Dossier Projet

Titre professionnel Concepteur Développeur d'Application

Sommaire

1. Introduction

2. Liste des compétences couvertes

3. Résumé en anglais

4. Expression des besoins du projet

A. Contexte et description du cahier de charge du projet

EDAB SOLUTION est une jeune start-up accompagnant les entreprises ou des particuliers dans la réalisation et la concrétisation de leurs divers projets, de la digitalisation de celles existantes à la naissance de nouvelles.

L'entreprise propose une large gamme de services de gestion de projets, allant de la planification à l'exécution en passant par le suivi et la clôture.
Elle offre ses services dans différents secteurs d'activité tels que le développement logiciel, la construction, la recherche et développement, etc.

Elle souhaiterait donc disposer d'une solution pour pouvoir gérer de manière efficace et optimale l'ensemble de ces projets ainsi que ses projets internes afin de satisfaire au mieux les besoins de ses clients et d'être compétitive sur le marché.

Cela aidera EDAB SOLUTION de gérer efficacement ses projets, de collaborer avec ses équipes et d'interagir avec ses clients. La solution permettra également d'avoir un suivi en temps réel de l'avancement des projets autant au niveau de l'équipe projet qu'au niveau des clients. L'application sera disponible en version web et desktop.

Une première vue sera destinée aux administrateurs leur permettant d'avoir un contrôle total et un suivi sur les projets. Ensuite, une autre vue sera destinée aux clients (Users Dashboard), qui pourront suivre également de leurs côté l'avancement de chacun de leurs projets. Enfin, la dernière vue sera destinée à l'équipe projet pour le développement, la réalisation d'un projet spécifique.

Cette application vise à centraliser la gestion, l'organisation et le suivi en temps réel des projets au sein de l'entreprise EDAB SOLUTION. Elle sera un tout en un pour EDAB SOLUTION en répondant aux problématiques suivantes :

- Fournir une interface intuitive pour les administrateurs de l'entreprise afin de gérer les projets, les équipes et les clients et d'avoir un suivi en temps réel de l'ensemble des projets.

- Permettre aux équipes projets de collaborer en temps réel, de suivre l'avancement des tâches des projets et de communiquer efficacement.
- Fournir un tableau de bord aux clients afin de suivre en temps réel de l'avancement de chaque projet d'interagir si besoin
- Ressortir les statistiques relatives à un projet et/ou à l'ensemble des projets de l'entreprise

De nouvelles fonctionnalités pourront être développées afin de proposer une expérience enrichissante à l'utilisation de la solution.

La sécurité sera haute afin de garantir une bonne confidentialité des données des clients d'où le respect de la législation en vigueur concernant le respect de la vie privée des utilisateurs (RGPD).

Le logiciel est destiné principalement aux **équipes projets (Utilisateurs internes)** et aux **administrateurs** de EDAB SOLUTION mais il sera également accessible en mode suivi uniquement aux **clients (Utilisateurs externes)**.

Le périmètre des fonctionnalités qui seront disponibles sera plus explicité dans la suite de ce document. Une analyse détaillée des fonctionnalités sera proposée afin de permettre une meilleure compréhension du projet.

B. La solution attendue

Pour le développement en cours, il est attendu du développeur un premier livrable de la version web de la solution avec les fonctionnalités suivantes :

- la mise en place de la gestion de projets sur les vues administratives et équipes projets ;
- la mise en place de la gestion de suivi toujours pour les acteurs Administrateurs et équipe projet
-

C. Les acteurs impliqués

Les administrateurs

Encore désignés par les décisionnaires chez EDAB SOLUTION, ils ont une vue globale de l'ensemble des projets et ont accès à toutes les fonctionnalités de l'application. C'est eux qui constituent l'équipe projet à associer à chaque projet. Ils créent les différents membres de chaque équipes, les clients ainsi que les projets, en un mot c'est ceux qui ont tous les droits sur la solution.

L' équipe projet

L'équipe projet représente l'ensemble des employés de l'entreprise EDAB SOLUTION qui ont été désignés pour gérer un projet spécifique. Elle est constituée par un administrateur après la création du projet auquel elle veut associer. Les membres constitutifs de cette dernière sont choisis en fonction de leur domaine d'expertise.

Elle a accès à toutes les fonctionnalités de l'application liées à la gestion d'un projet, elle réalise un projet de bout en bout avec le plus de clarté possible pour permettre un suivi en temps réel.

Les utilisateurs externes

Plus précisément les entreprises clientes, ils ont un accès uniquement en lecture sur leur projet en cours de développement et ils pourront également y laisser des notes. Les utilisateurs externes, une fois leur projet créé pourront un suivi en temps sur l'avancement de leur projet et pourront également interagir à travers un système qui sera mis en place ultérieurement.

5. Gestion de projet

A. Planning et suivi

Le projet réalisé dans le cadre de mon titre Concepteur Développeur D'Application au cours de ma formation de Conceptrice de Projets SI au sein de l'Ecole des Technologies du Numérique Avancées (ETNA) est un projet fictif. L'idée de ce projet m'est venue lors de mon apprentissage au sein de la Mairie d'Argenteuil.

Il était attendu :

- Un cahier de charge au 13 juillet,
- Les spécifications détaillées au 03 août,
- Implémentation (Maquette, architecture et modèle de données de l'application) au 14 août
- Premier livrable attendu le 08 septembre

Afin de pourvoir me rapprocher des conditions réelles d'utilisation du logiciel au sein d'une vraie entreprise, je me suis rapprochée des chargés de projets du service projets où je suis apprentie. Ils ont donc été mes interlocuteurs pour l'établissement du cahier de charge lors de mes premières analyses. J'ai réalisé toute seule la rédaction détaill »e du cahier de charge, des spécifications fonctionnelles ainsi que technique, le déploiement du projet en passant par l'implémentation de la soultion elle-même.

Préciser les changement au fil du développement …..

Diagramme de Grant de l'organisation du projet final

Pour la réalisation du projet je me suis appuyée sur la méthodologie Agile. Cela permettra ainsi une flexibilité au cours de la réalisation de mon projet, d'être ponctuelle dans le respect des délais fixé et de pouvoir présenter le premier livrable fonctionnel de l'application. Ajouter l'évolution de la réalisation

Pour l'implémentation, mon approche a été d'implémenter une solution couvrant le maximum de fonctionnalités possibles. Toutes les fonctionnalités ne sont pas totalement implémentées, un aperçu est réalisé pour avoir une cohérence de l'application. pour accorder suffisamment de temps à la conception et l'implémentation et pouvoir ainsi présenter une première version fonctionnelle de l'application.
L'organisation du travail pour l'implémentation s'est faite grâce à la fonctionnalité GitLab de l'école et mon GitHub personnel.

Ajouter une capture de la répartition des tâches dans GitLab.

B. Environnement technique

Pour le développement de la solution, j'ai utilisé les outils et les technologies suivantes :

Visual Studio Code

Visual Studio Code est un des IDE (Environnement de Développement Intégré) les plus utilisés pour le développement d'application de logiciels et autres. Je l'ai donc utilisé comme environnement de développement et GitLab pour la gestion du code source et le suivi des modifications apportées au projet.

StarUML

StarUML est un outil de génie logiciel dédié à la modélisation UML et édité par la société coréenne MKLabs. Le logiciel StarUML a servi pour concevoir les diagrammes de cas d'utilisation des fonctionnalités afin de mieux illustrer les fonctionnalités des différentes vues ainsi que le model conceptuel de données.

Figma

Figma est un éditeur de graphique vectoriel et un outil de prototypage. L'ensemble des fonctionnalités de Figma est axé sur l'utilisation dans la conception de l'interface utilisateur et de l'expérience utilisateur. Ce dernier a servi pour concevoir la maquette de l'application, les wireframes, les diagrammes d'activité ainsi que la logique des différentes vues

MySQL Workbench

MySQL Workbench est un outil visuel unifié destiné aux architectes de bases de données, aux développeurs et aux administrateurs de base de données. Il fournit des outils de modélisation de données, de développement SQL et d'administration complet pour la configuration du serveur, l'administration des utilisateurs, la sauvegarde et bien plus encore. Ce logiciel a été principalement utilisé pour la conception de la base de données et la visualisation en temps réel. Avec ce dernier j'ai

conçu le Modèle Conceptuel de Données (MCD) du logiciel à l'aide des diagramme ERR, gérer les relations entre les entités qui ont permis de générer la base de données du logiciel.

GitLab

GitLab est un logiciel libre de forge basé sur git, permettant le versionnage du code (ainsi que sa mise en commun, lors de projets d'équipe). Il propose aussi la mise en place de pipelines CI/CD (Continuous Integration, Delivery, Deployment), qui ont permis de mettre en place une analyse de sécurité statique (SAST Static Application Security Testing), vérifiant le code source du projet à la recherche de vulnérabilités connues.

En backend :

- Le langage **Python**.

  Python est un langage de programmation populaire interprété, multi paradigme et multiplateforme. Il permet de créer des applications web innovantes et modernes. Il est doté d'un typage dynamique fort, d'une gestion automatique de la mémoire et d'un système de gestion d'exceptions.

- Le Framework **Flask**

  Flask est un micro Framework de développement web en python. Il est classé micro car il est très léger et flexible. Il serra utiliser pour définir les routes, gérer les requêtes http et les réponses, et gérer les interactions avec la base de données. Flask est basé sur le modèle d'architecture MVC (Modèles-Vues-Contrôleurs) et prend en charge différentes extensions pour une utilisation optimale comme **SQLAlchemy** (une extension qui facilite l'intégration de la base de données).

- **MySQL**

  Le développement a été réalisé à l'aide de **MySQL**, un système de gestion de base de données relationnelles (SGBDR) libre et open source. Il permet de stocker, manipuler, partager et gérer les informations d'une base de données.

Une base de données relationnelle est un ensemble de tables reliées entre elles.

Chaque table possède plusieurs attributs, dont une clef primaire, dont la valeur permet de distinguer de façon sûre une occurrence de toutes les autres. Chaque entrée d'une table dispose de ses propres valeurs concernant chaque attribut. Les tables sont reliées entre elles en incorporant la clef primaire d'une autre table (ou plusieurs) ; on parle alors de clef étrangère.

- **Pytest**

  Pytest est un Framework de test pour Python qui facilite l'écriture de tests unitaires pour un backend développé avec Flask. Il sera utilisé à cet effet

En Frontend :

- **VueJS**

  VueJS est un Framework et un écosystème qui couvre la plupart des fonctionnalités courantes nécessaires au développement frontend. C'est un Framework JavaScript progressif pour la construction d'interfaces utilisateur interactives. Il vous permet de créer des composants réutilisables, ce qui facilite la construction d'une interface utilisateur modulaire et maintenable.

- **Bootstrap**

  Bootstrap est une collection d'outils utiles à la création du design (graphisme, animation et interactions avec la page dans le navigateur, etc.) de sites et d'application web. Il a été utilisé en complément à VueJS pour plus de dynamisme dans le logiciel.

- **HTML**

  HyperText Markup Language est un langage standardisé de balisage permettant de structurer les pages web et de les relier entre elles, grâce notamment aux liens hypertexte. Il permet d'intégrer divers contenus, comme des images, vidéos ou encore des formulaires.

- **CSS**

  Cascading Style Sheets est un langage standardisé de présentation pour les documents HTML. Il permet de personnaliser l'apparence du contenu d'une page web (couleur,

décorations) mais aussi sa mise en page, notamment selon les dimensions de l'écran de l'utilisateur, permettant de mettre en place des interfaces responsives.

- **JavaScript**

  JavaScript est un langage de script haut niveau orienté objet à prototypes et standardisé sous le nom d'ECMAScript. Troisième langage principal du web avec HTML et CSS, il est interprété côté client par les navigateurs web pour rendre les pages web dynamiques.

- **Jest**

  Jest est un Framework de test populaire pour les applications Vue.js. Il sera utilisé pour écrire des tests unitaires et des tests d'intégration pour le frontend de la solution.

6. Spécifications fonctionnelles du projet

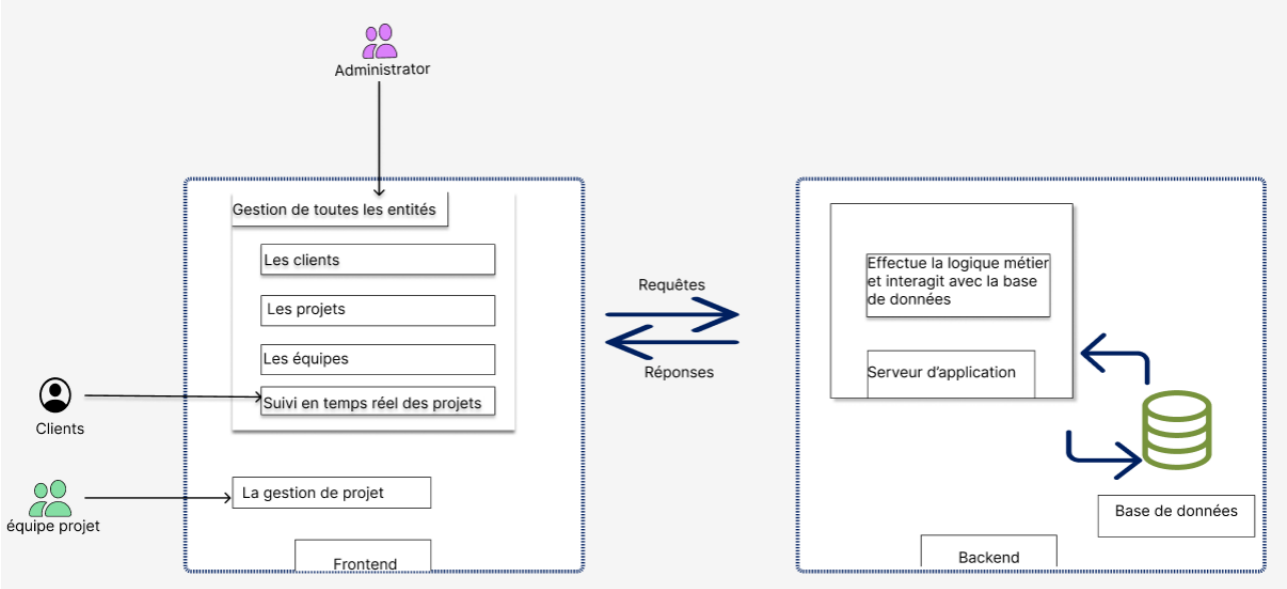

Figure 1 : Schéma fonctionnel

Nous pouvons remarquer que le schéma fonction du logiciel est composé de deux grands blocs qui regroupent les acteurs et entités principaux mis en jeu et qui communiquent à l'aide des requêtes http.

Les **entités** sont les principales composantes de l'application et comme entités présentes ici nous avons les clients, les projets, les équipes, la base de données et le serveur d'application.

Les **acteurs** sont les parties prenantes externes qui interagissent avec l'application, il s'agit des clients, des équipes projets, des administrateurs.

Ci-après les fonctionnalités principales détaillées du logiciel.

A. Gestion de projets

La gestion de projets est un des deux volets principaux de l'activité de EDAB SOLUTION. Il faut donc mettre en place les outils nécessaires à leur bon fonctionnement et suivi. L'expérience utilisateur tout comme le suivi et les mises à jour en interne doivent être fluides et intuitives.

Vue Administrateur

- Gestion de projets
    - Consulter les projets
    - Créer un nouveau projet
    - Modifier un projet
    - Archiver un projet
    - Supprimer un projet
    - Associer une équipe à un projet
    - Ajouter des commentaires sur un projet
    - Modifier des commentaires sur un projet
    - Supprimer des commentaires sur un projet
    - Sortir les statistiques relatives à un projet

- Gestion des équipes
    - Créer une nouvelle team
    - Ajouter un nouveau membre à l'équipe projet
    - Retirer un membre de l'équipe projet
    - Consulter les projets d'une team
- Modifier une tâche
- Consulter les tâches
- Supprimer une tâche

Vue Equipes Projets

- Gestion des tâches
    - Créer des tâches
    - Attribuer des tâches
    - Mette à jour des tâches
    - Retirer une tâche
    - Classification des tâches
        - A faire
        - En cours
        - Terminées
        - A revoir
    - Notifier les tâches à réaliser à venir
    - Consultation des tâches à réaliser
    - Gestion des échéances des taches
    -
-
- Gestion de l'équipe
    - Planifier des réunions en équipe
    - Planifier des réunions avec les intervenants

- Attribuer une tâche à un ou plusieurs membre de l'équipe
- Modifier le membre gérant une tâche donnée
- Consulter les membres de l'équipe

Vue Clients

- Interagir avec l'équipe projet

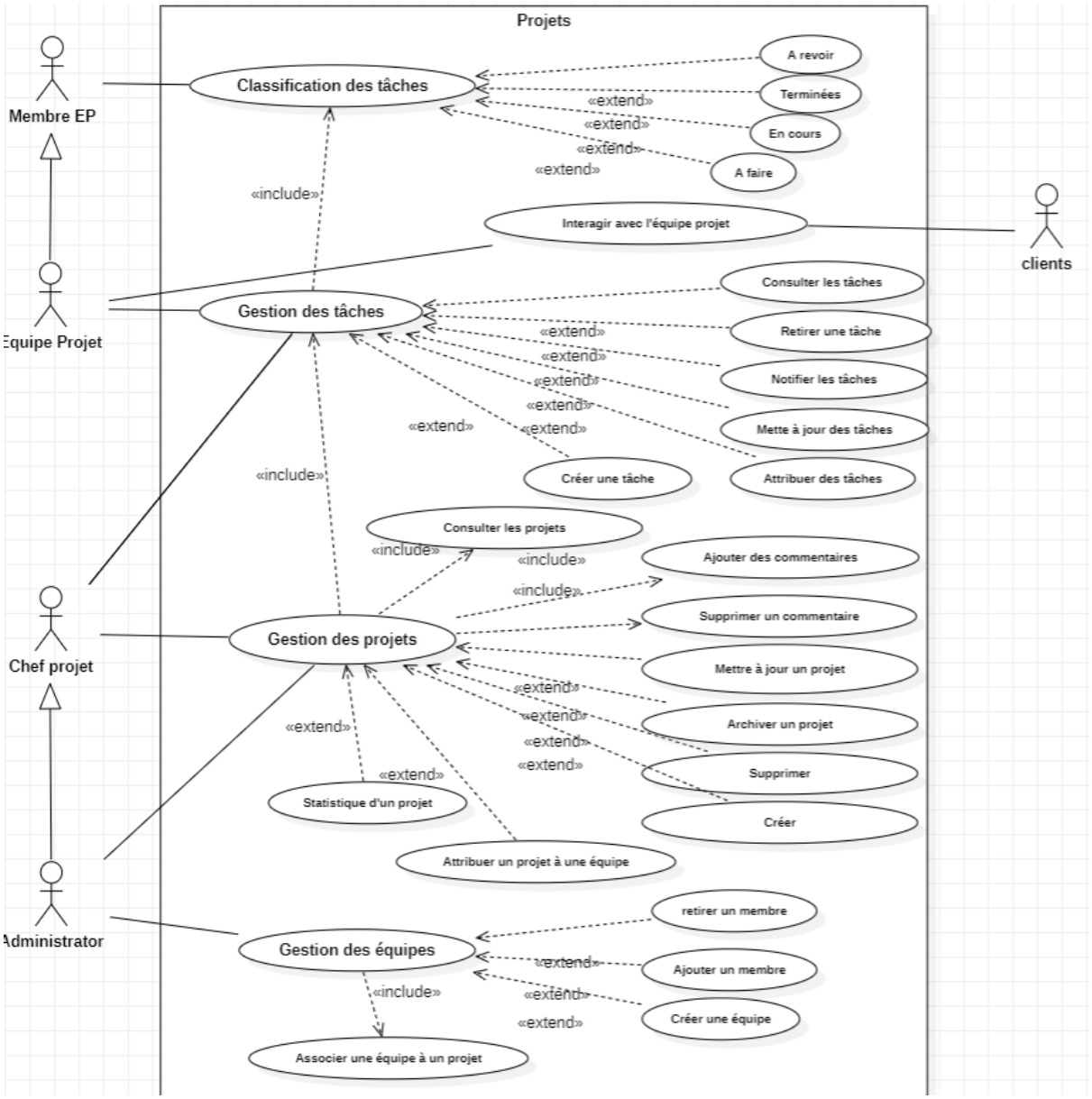

Figure 2 : Diagramme de cas d'utilisation pour la gestion de projets

B. Suivi de projets

Le suivi des projets est le deuxième volet principal de EDAB SOLUTION. Il faut donc mettre en place les outils nécessaires à leur bon fonctionnement et suivi. L'expérience utilisateur tout comme le suivi et les mises à jour en interne doivent être fluides et intuitifs.

Vue Administrateur

- Consulter les rapports des réunions
- Planifier une réunion avec l'équipe projet
- Planifier une réunion avec l'ensemble des intervenants du projet
- Consulter les projets
- Statistiques des tâches par équipe
- Statistiques d'un projet
- Statistiques de l'ensemble des projets
- Liste des réunions à venir

Vue Equipes projets

- Consulter les tâches
- Notifier chaque mise à jour de tâches
- Rapports hebdomadaires
- Suivre les échéances
- Consulter les projets
- Être notifié à chaque mise à jour du projet
- Liste des réunions à venir
- Sortir les statistiques d'un projet

Vue Clients

- Consulter l'état d'avancement de son projets
- Être notifié des prochaines réunions
- Être notifié à chaque mise à jour importante du projet
- Ajouter des commentaires sur un projet
- Sortir la statistique relative à un projet
- Liste des réunions à venir
- Contacter l'entreprise

Un diagramme de classe n'a pas été mis en place par

C. Gestion des rôles

La gestion est le dernier volet du logiciel, un volet assez critique qui sera développé minutieusement. Puisque tout partira d'un bonne gestion des rôles. Ci-après les fonctionnalités prévues à cet effet :

Vue Administrateur

- Créer un compte d'un membre d'une équipe et lui attribué les autorisations nécessaires
- Modifier un
- Créer un client et lui attribué les autorisations nécessaires
- Consulter les membres d'une équipe
- Modifier les autorisations d'un membre
- Supprimer un compte membre
- Archiver un compte client

Vue Equipes projets

- Se connecter avec les identifiants fournis
- Modifier les informations personnelles
- Avoir les droits nécessaire à la gestion de projet et au suivi de projet de sa vue

Vue Clients

- Se connecter avec les identifiants fournis
- Modifier les informations personnelles
- Avoir les droits nécessaires à la gestion de projet et au suivi de projet depuis son interface
- Se déconnecter

7. Spécifications techniques du projet

A. Maquette

Le maquettage marque la transition entre l'analyse et la conception technique. Il doit s'adapter aux contraintes techniques, alors que l'analyse ne se prononce pas sur le système utilisé, mais uniquement sur les fonctionnalités attendues.

Grâce au logiciel Figma, des maquettes ont été réalisées afin de guider la conception, en respectant la charte des couleurs fournie par le client.

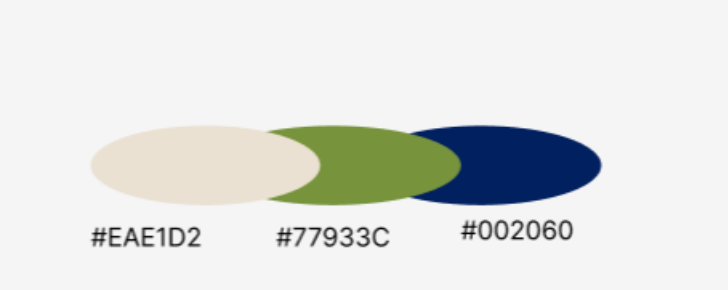

Figure 3 : Charte des couleurs de Allinone

Cette approche permet de prévoir l'utilisation des couleurs ainsi que l'organisation des informations, notamment selon la taille de l'écran.
Ci-après l'arche de la solution proposée :

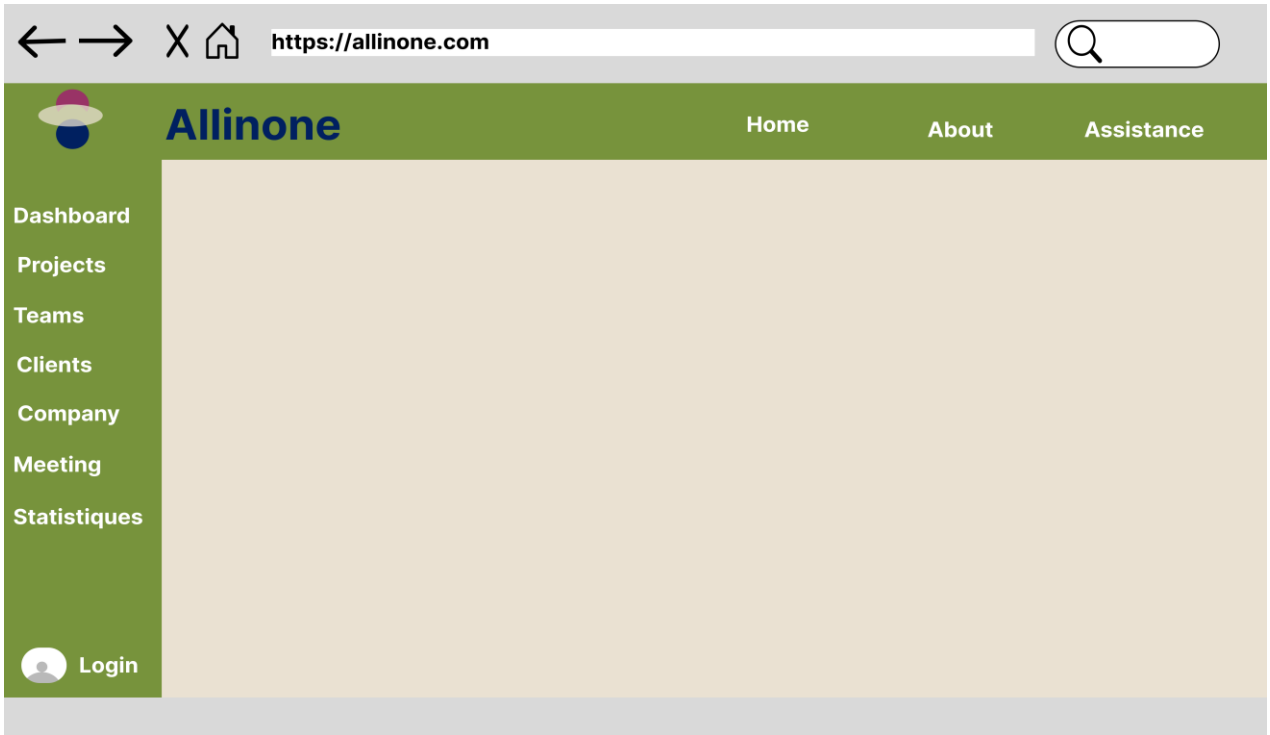

Figure 4 : Arche de la solution, Vues Admin & Equipe projet

L’arche permet de définir les éléments clés qui seront présents sur les autres pages de l’application.

Voici quelques une des maquettes conçues par Vue.

Vue admin

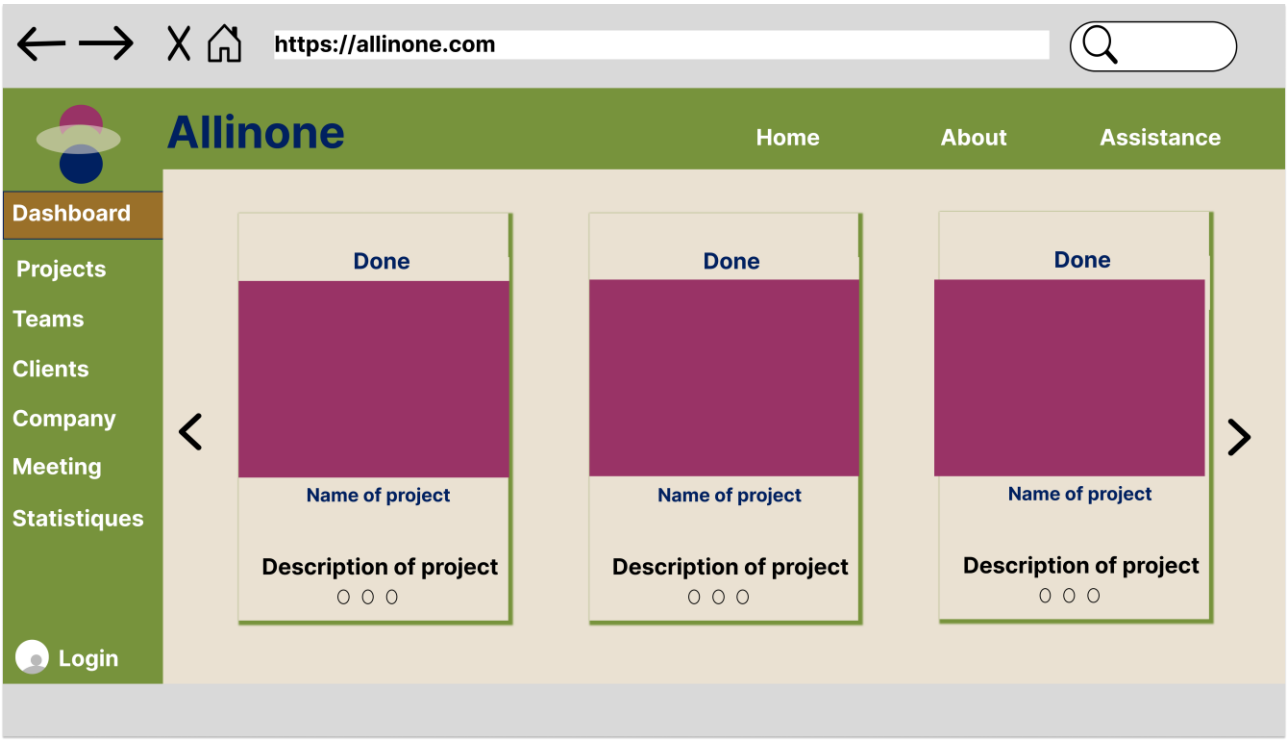

Figure 5 :Dashboard Page - Arche présente

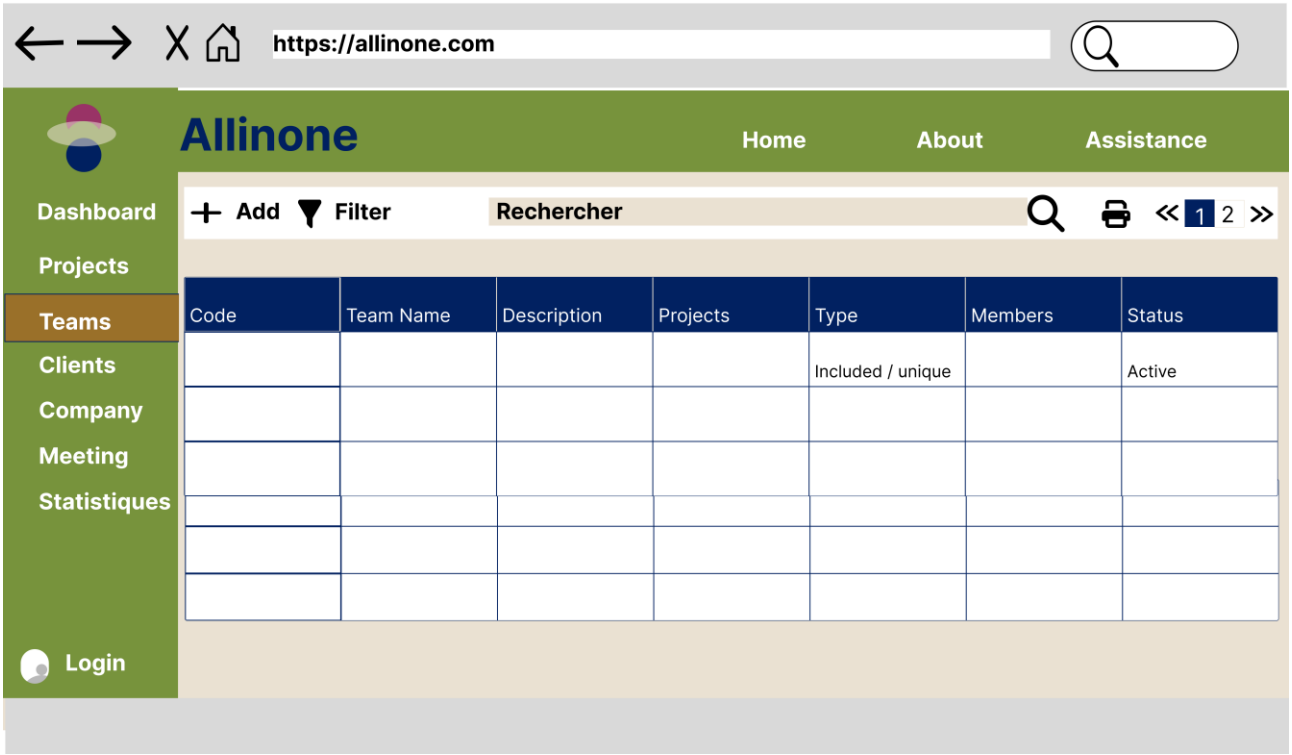

Figure 6 : Team Page : Arche présente

Vue Equipe

Dynamique des écrans

Enfin, la dynamique des écrans a été également prévue. Il s'agit de l'enchaînement d'un écran à un autre lors de la navigation sur le logiciel. Les pages auxquelles renvoie le menu principal de chaque vue ont été définies, ainsi que les renvois et redirections éventuels.

Ci-après les dynamiques d'écrans de la vue Admin et Equipe :

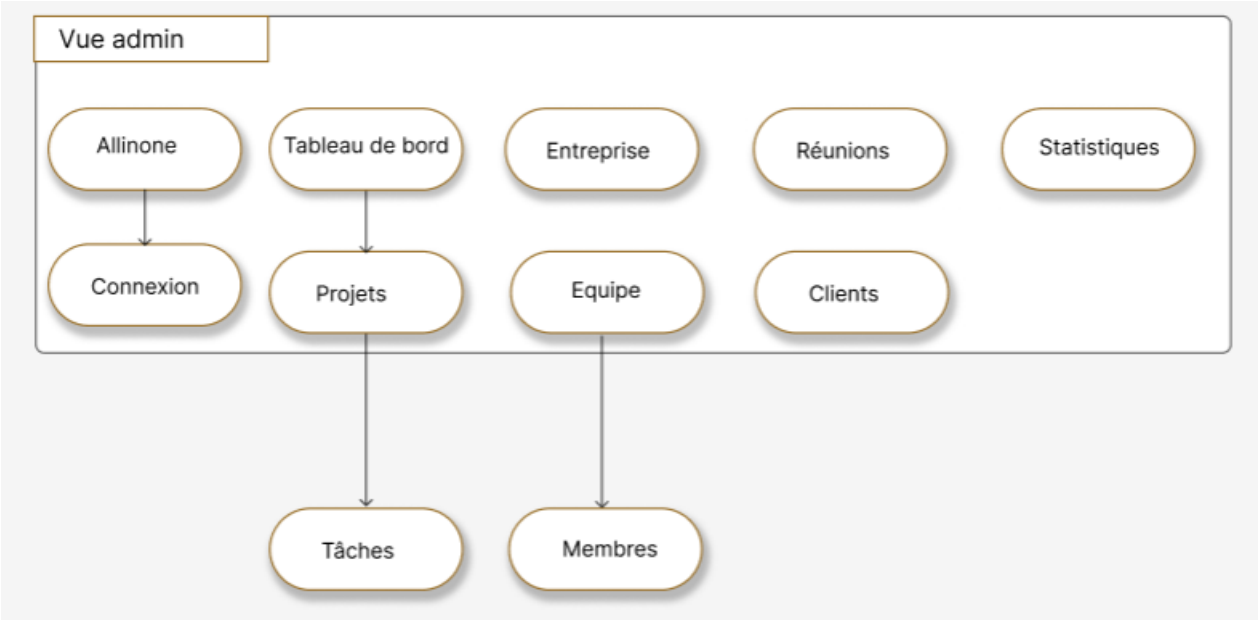

Figure 7 : Dynamique des écrans de la vue Admin

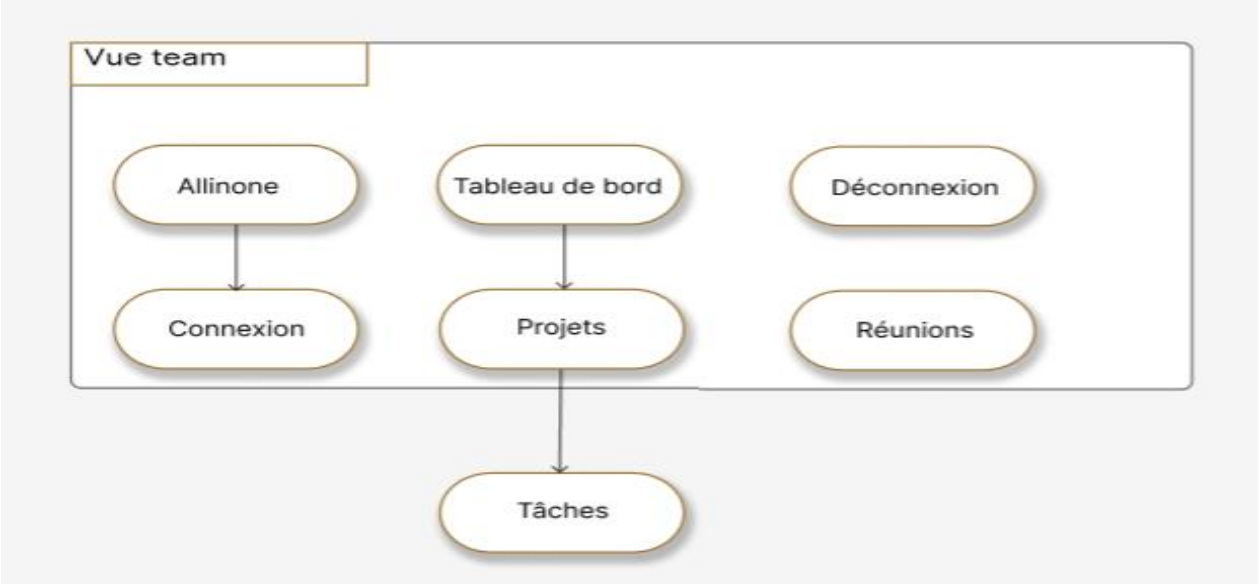

Figure 8 : Dynamique des écrans de la vue Equipe

B. Architecture du logiciel

Pour la conception de l'architecture du logiciel plusieurs éléments essentiels ont été pris en compte tels que :

- La séparation des responsabilités entre les différents modules du système;
- Les interactions entre les différents composants;
- La modularité et la réutilisabilité du code ;
- La maintenabilité et l'évolutivité du système;
- La performance et l'efficacité du système ;
- La sécurité et la fiabilité du système.

Ainsi, parmi les différents modèles d'architecture logicielle permettant de tenir comptes de ces éléments, j'ai adopté le modèle d'architecture MVC (Modèle-Vue-Contrôleur) pour organiser la structure de l'application Allinone.

MVC (Modèle-Vue-Contrôleur) est un modèle d'architecture logicielle qui permet de séparer les différentes fonctionnalités d'une application. C'est un concept très utilisé dans le développement web pour les applications web dynamiques. Elle est également très flexible et extensible, ce qui permet d'ajouter de nouvelles fonctionnalités facilement.

Modèle (Model) : Le modèle représente la logique métier et les données de l'application. Dans le cas de mon application, cette partie de l'architecture est gérée par le backend construit en Python avec Flask. Les modèles définissent la structure des données, leur validation et les interactions avec la base de données.

Vue (View) : La vue est responsable de l'interface utilisateur et de l'affichage des données. Dans le cas de mon application VueJS, les composants Vue (**Components**) représentent la vue. Ils sont responsables de la présentation des données et de la gestion des événements utilisateur. Elle est souvent représentée par des pages HTML et des formulaires.

Contrôleur (Controller) : Le contrôleur agit comme une liaison entre le modèle et la vue, c'est la partie centrale de l'architecture MVC. C'est là que les actions de l'utilisateur sont traitées, les données sont récupérées du modèle et la vue est mise à jour. Dans le cas de mon application, ils interviennent à la fois au niveau du Frontend et du Backend.

- Au niveau du frontend, les composants Vue agissent comme des contrôleurs légers, traitent les événements et effectuent des appels aux API du backend via des requêtes http de la bibliothèque Axios. Puis ils mettent à jour la vue en conséquence.
- Flask gère cette partie du backend en traitant les requêtes HTTP, en interagissant avec les modèles et en retournant des réponses appropriées à la aux contrôleurs vue.

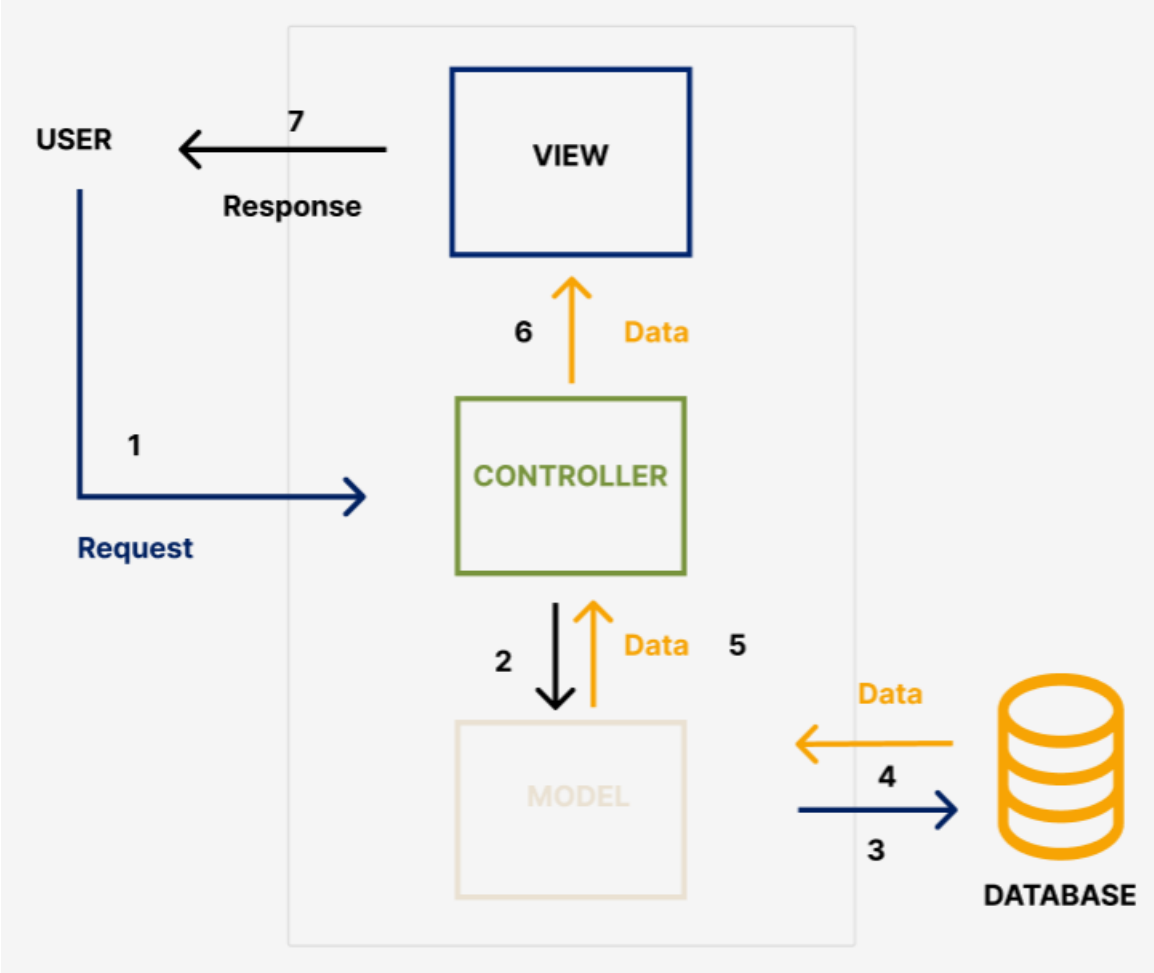

Figure 9 : Architecture du Logiciel mettant uniquement en relief le MVC

C. Modèle conceptuel de données

Le Logiciel StarUML a permis de créer le modèle conceptuel de données, à l'aide d'un diagramme de classe. Ce diagramme a permis de représenter les entités métier pertinentes et les attributs qu'elles contiennent et qui les définissent le mieux. L'accent a été également mis sur les différentes relations entre ces entités.

Une entité représente un objet ou un concept du monde réel qui peut être identifié de manière distincte.

Ces objets peuvent être concrets (un projet, une tâche, un client) ou abstraits (un rôle, le statut d'une tâche). Une entité a des attributs et peut être en relation avec une ou plusieurs autres entités.

Les attributs sont les caractéristiques ou les propriétés spécifiques d'une entité. Ils décrivent les détails ou les informations que nous souhaitons conserver sur une entité.

Les relations décrivent comment les entités sont liées les unes aux autres. Elles indiquent les connexions ou les interactions entre les entités. Les trois principaux relations sont : un à un (**One to One**), un à plusieurs (**One to many)** et plusieurs à plusieurs (**Many to many**).

- La relation entre les entités ***ClientUser*** et ***ClientAddress*** décrit bien une relation One to One, un enregistrement dans la table ***ClientUser*** est associé à un et un seul enregistrement dans la table ***ClientAddress***. Cela signifie que chaque client a une seule adresse.
- La relation entre les entités ***ClientUser*** et ***Meeting*** décrit une relation One to many. Un enregistrement dans la table ***ClientUser*** peut être associé à plusieurs enregistrements dans la table ***Meeting***, mais chaque enregistrement dans la table ***Meeting*** est associé à un seul enregistrement dans la table ***ClientUser***. Cela signifie qu'un client peut avoir plusieurs réunions, mais chaque réunion est associée à un seul client.
- La relation entre les entités ***Member*** et ***Role*** décrit bien une relation Many to Many, plusieurs ***Member*** peuvent être associées à plusieurs ***Role***, et vice versa. Par exemple, plusieurs ***Member*** peuvent avoir plusieurs ***Role*** différents, et un ***Role*** peut être associé à plusieurs ***Member***. Une relation Many to Many est généralement représentée par une table d'association qui contiendrait les clés étrangères reliées aux clés primaires des tables ***Member*** et ***Role.*** Seule la relation a été mise en exergue dans le diagramme de classe.

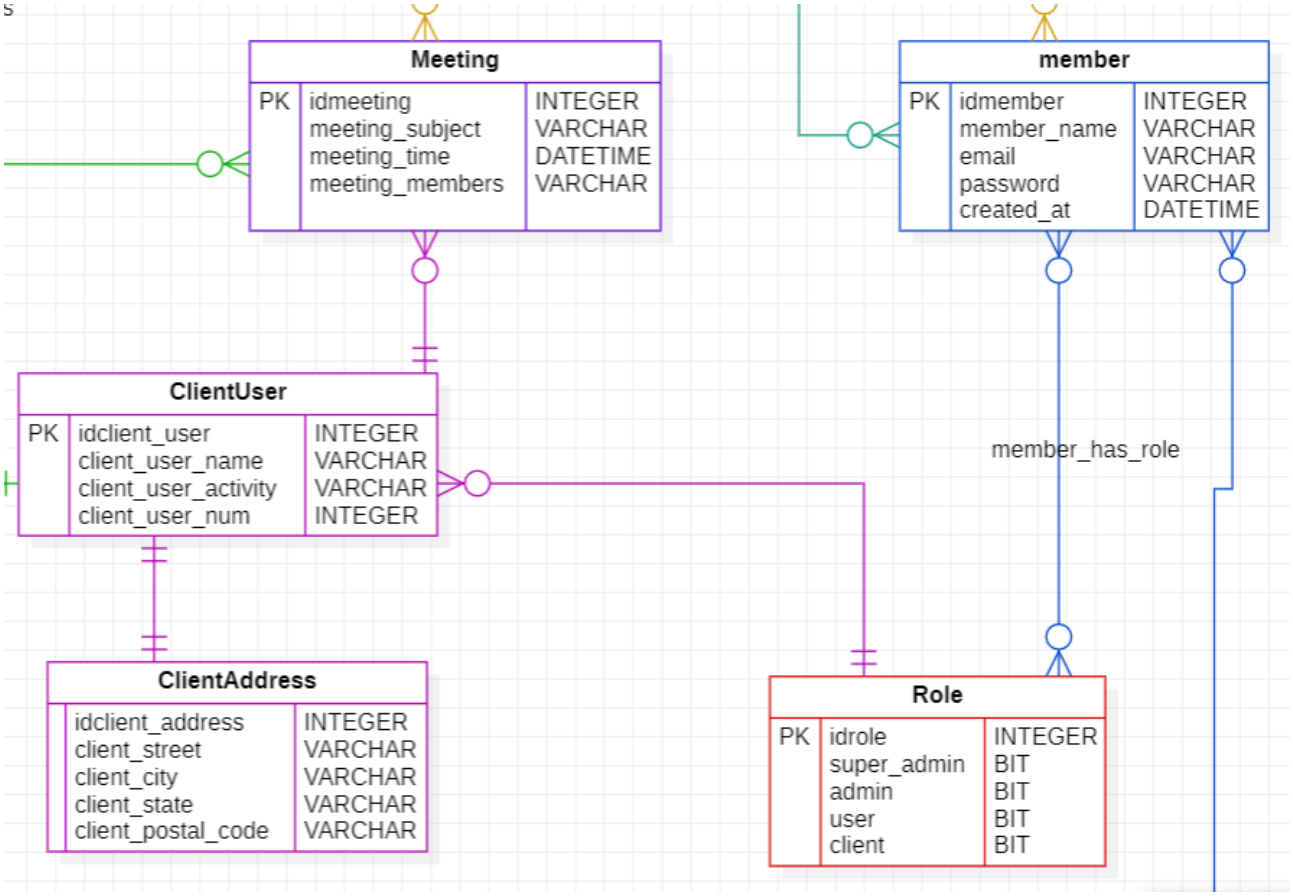

Figure 10 : extrait du diagramme de classe de Allinone, illustration des relations

Ainsi nous pouvons observer ces notions dans le digramme de classe complet représentant le modèle conceptuel de données ci-après :

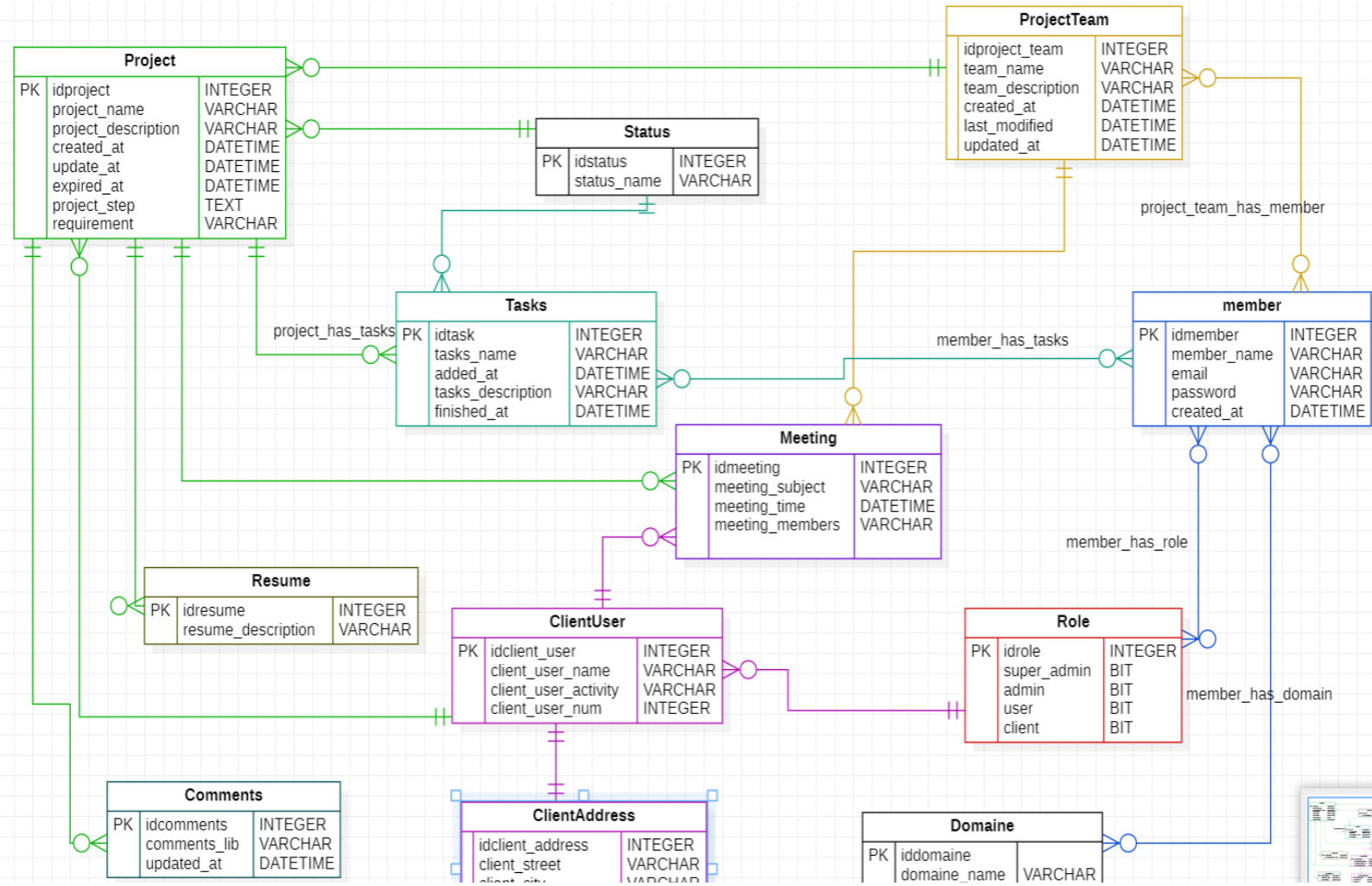

Figure 11 : diagramme de classe de la base de données de Allinone

8. Réalisation du candidat

Cette partie du dossier comprendra l'implémentation proprement dite du logiciel. Après avoir définies les spécifications technique du projet, je suis passée à son implémentation tout en respectant les contraintes techniques précédemment définies et en ayant pour repères les fonctionnalités du logiciel.

A. Structure du projet

Structure du Frontend

En rappel, le frontend du logiciel a été réalisé en VueJS. La structure d'un projet VueJS comprend plusieurs répertoires et fichiers organisés de manière à faciliter le développement, la maintenance et le déploiement. Grâce à la commande de création d'un projet Vue-cli, fournie dans la documentation de VueJS, j'ai initialisé le frontend de mon application dans le dossier front-allinone.

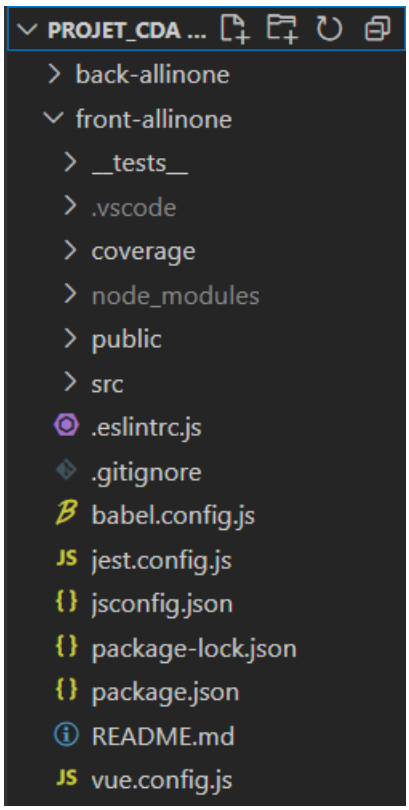

Figure 12 : capture de la structure du frontend dans VSCode

Les fichiers et dossiers générés sont tous nommés en anglais, j'ai ainsi cette habitude de nommage dans la suite du projet.

Sur la capture, nous pouvons observer plusieurs fichiers et dossiers dont la plupart sont générés lors de la création du projet et qui sont régulièrement mis à jour pendant le développement (.gitignore, package.json, vue.config.js, jsconfig.json, public, src) puis il y a ceux qui ont été ajoutés en cours de développement (README.md, jest.config.js, package-lock.json, __tests__, …).

Le dossier src est le dossier racine de l'application et à l'intérieur de ce dernier, nous avons les dossiers et fichiers représentant les grands axes de l'application :

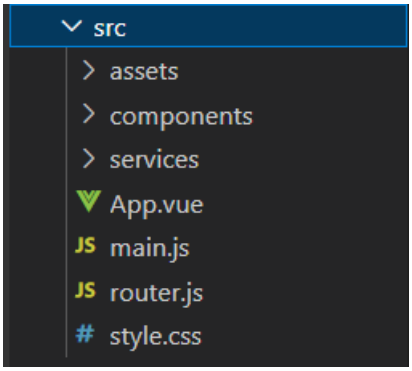

Figure 13 : Structure du projet_CDA/front-allinone/src

- Le dossier ***assets*** contient les fichiers statiques tels que les images, les polices, … de l'application. Son utilisation est spontanée.
- Le dossier ***components*** représente les vues principales et les composants réutilisables de l'application tels que :
    - les vues ***admin***, ***equipe*** et **clients** qui regroupent les components propres à chaque vue ;
    - et les vues _Arche et _Forms qui regroupent respectivement les components présents sur toutes les pages et les formulaires.

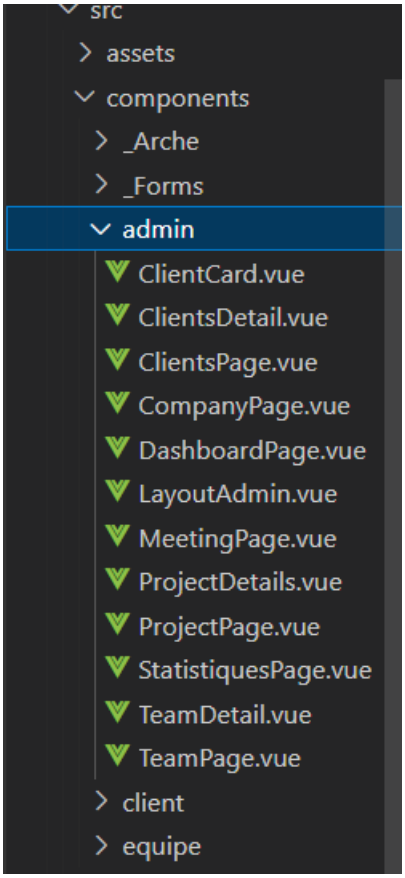

Figure 14 : Structure de la Vue admin de l'application

Nous pouvons remarquer qu'en plus de nommer les fichiers en anglais, les fichiers en **.vue** sont composés de deux mots et respectent la représentation CamelCase. Autrement dit chaque mot dans le nom du chier commence par une lettre majuscule (TeamBoard.vue, TasksList.vue, …).

- Le fichier **router.js** représente le fichier lié à la gestion du routage de l'application. Il est très important dans le fonctionnement de l'application, grâce à lui nous pouvons naviguer d'une page à une autre. Ainsi, à chaque component vue créé est associé une route dans le router.js.

  J'ai également utilisé de l'imbrication dans la définition de mes routes, cela a permis de regrouper les routes par vue autrement dit de créer une relation hiérarchique entre les vues. Ainsi, les routes parentes sont définies de la même manière que les routes normales avec une propriété supplémentaire appelée ***children***. Le ***children*** est un Array (type de données) qui regroupe toutes les routes enfants

```
27      {
28        path: '/admin',
29        component: () => import('./components/admin/LayoutAdmin.vue'),
30        children: [
31          {
32            path: 'dash',
33            name: 'DashboardPage',
34            component: () => import('./components/admin/DashboardPage.vue')
35          },
36          {
37            path: 'projects',
38            name: 'ProjectPage',
39            component: () => import('./components/admin/ProjectPage.vue'),
40          },
```

Figure 15 : Séquence de la hiérarchie entre les routes de la vue admin

L'accès à toutes les pages de vue admin doit s'effectuer par la route parent ***/admin*** du composant **LayoutAdmin.vue** en exemple :

```
<ul class="nav flex-column">
  <li class="nav-item">
    <a class="nav-link d-flex align-items-center gap-2 active" aria-current="page" href="/admin/dash">
      Tableau de bord
    </a>
  </li>
```

- Le dossier ***store*** contient les fichiers liés à la gestion de l'état global de l'application avec Vuex.
- Le fichier **style.css** contient les styles de tous les components de l'application. À ce jour, j'ai regroupé les styles de toutes les pages dans ce seul fichier qui est appelé dans chaque component afin d'appliquer le style correspondant.
- Le dossier ***services*** contient les modules liés à la récupération de données et à la communication avec l'API. Dans mon projet il est organisé de manière à avoir un module service pour chaque vue, cette partie sera plus développée plus tard dans les composants d'accès au backend.
- Le dossier ***public*** contient des fichiers statiques qui sont copiés directement dans le dossier de sortie lors de la construction de l'application.
- __tests__ : Contient les fichiers de tests de l'application …
- Le dossier ***coverage*** est lié à la couverture de code, souvent utilisée dans le contexte des tests automatisés. Dans ce dossier nous pouvons trouver les données de tests des fonctionnalités qui ont été testées, le proportions de codes testés et dans quelle mesure …

Structure du Backend

B. Composants d’accès aux données

Composants d’accès au backend

Dans la description de la structure du frontend j’ai abordé brièvement le rôle du dossier ***services*** dans la structure du dossier src.

Ce dossier contient les fichiers javascript qui servent d’interface avec le backend de l’application, ils sont pour faciliter la gestion des entités de l’application telles que les clients, les projets, les membres, les équipes.

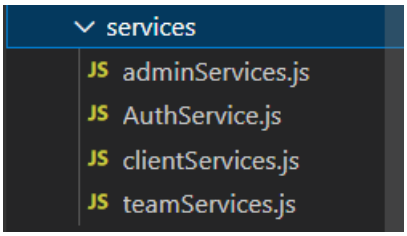

Figure 16 : Structure de front-allinone/src/services

Les services exploitent la bibliothèque Axios pour effectuer des requêtes http vers le backend.

Axios est une bibliothèque Javascript utilisée pour effectuer des requêtes HTTP depuis node.js ou des XMLHttpRequests depuis le navigateur qui prend également en charge l’API Promise ES6. Il simplifie le processus de communication en offrant une syntaxe élégante et des fonctionnalités telles que la gestion des intercepteurs de requêtes et de réponses, la transformation automatique pour les données JSON, le support côté client pour la protection contre XSRF.

Afin d’organiser au mieux le projet, j’ai créé un service par vue et le service adminService.js contient toutes les fonctions qui permettent d’accéder à toutes les routes accessibles par un administrateur.

```
import axios from 'axios';

const apiUrl = axios.create({
  baseURL: 'http://127.0.0.1:5000',
});

/* Client*/
export const getListClient = () => {
  return apiUrl.get('/admin/clients');
};
```

Figure 17 :Fonction pour récupérer la liste des clients du backend

En effet, ici, ***getListClient*** est une fonction qui utilise Axios (***apiUrl.get(...)***) pour effectuer une requête **GET** vers l'URL spécifiée (***'/admin/clients***'). Cette fonction renvoie une ***Promise*** (Type de réponses des requêtes Axios) qui sera résolue avec les données de la réponse de la requête. Cette fonction permet de récupérer la liste des clients depuis le backend.

Outre la communication avec le backend, les services jouent un rôle crucial dans le rendu des vues. Ils permettent aux components vue d'avoir les données dynamiques et régulièrement à jour. Voici une utilisation des services dans le component ***ClientsPage.vue*** qui fait appelle à la fonction ***getListClient*** :

Dans un premier temps la fonction est importée afin d'être accessible dans le component :

```
<script>
import {getListClient} from '@/services/adminServices';

export default {···
};
</script>
```

Ensuite, elle est utilisée dans ***methods*** d'une part pour récupérer la ***Response***, de la traiter pour ne récupérer que l'essentiel des données clients (traitement effectué dans la fonction ***fetchData()***) et ensuite de rendre disponibles et à jour à chaque appel du component (cela est gérer la fonction ***mounted()***) :

```
methods: {
  fetchData() {
    getListClient()
      .then((response) => {
        response.data.forEach((client) => {
        this.clients.push(client)
        });
      })
      .catch((error) => {
        console.error(error)
      });
  },
},
mounted() {
  this.fetchData();
},
```

Figure 18 : Utilisation de la fonction de récupération de la liste des Clients dans le component ClientsPage.vue

Le processus est donc le même pour les autres services et fonctions, des spécificités peuvent être néanmoins être ajoutées en fonction des URLs et/ou de l'utilisation des données récupérées.

Composants d'accès aux données

Les composants d'accès aux données sont été gérés par l'extension SQLAlchemy proposée par le Framework Flask. SQLAlchemy est un Object Relationnel Mapped (ORM) qui une suite complète de modèle de persistance de données et un accès efficace à une base de données SQL.

Les modèles SQLAlchemy définissent la structure des données de l'application. Chaque modèle représente une ***entité*** et donc une table dans la base de données. Et chaque propriété d'un modèle correspond à une colonne de la table de l'entité dans la base de données.
Les modèles sont définis avec le mot clé ***class*** et le paramètre ***db.model***. ***__tablename__*** est pour spécifié que ce modèle correspond telle table dans la base de données.

Chaque ***class*** correspondant à une entité métier doit disposer d'un attribut avec les paramètres ***db.Integer*** & ***primary_key=True***, qui correspondra à sa clef primaire et qui sera automatiquement générée grâce au paramètre ***autoincrement=True***. Cet attribut permet de représenter chaque ligne d'une table par un identifiant unique.

La déclaration d'un attribut avec le mot clé ***db.Column*** permet se spécifier que cet attribut correspond à une colonne dans la base de données.

```python
from app import db, bcrypt

You, 1 second ago | 1 author (You) | Comment Code
class Project(db.Model):
    __tablename__ = 'project'

    # Les columns de la table project
    idproject = db.Column(db.Integer, primary_key=True, autoincrement=True)
    project_name = db.Column(db.String(45), nullable=False)
    project_description = db.Column(db.String(255))
    created_at = db.Column(db.DateTime, default=db.func.current_timestamp())
    update_at = db.Column(db.DateTime, default=db.func.current_timestamp())
    expired_at = db.Column(db.DateTime)
    project_status = db.Column(db.Integer, db.ForeignKey('status.idstatus'), nullable=False, default=1)
    _idproject_team = db.Column(db.Integer, db.ForeignKey('project_team.idproject_team'))
    project_step = db.Column(db.Text)
    requirement = db.Column(db.String(255))
    project_team_idproject_team = db.Column(db.Integer)
    client_user_idclient_user = db.Column(db.Integer)
    _idclient_user = db.Column(db.Integer, db.ForeignKey('client_user.idclient_user'))

    # Les relations de la table project avec les autres tables
    project_team = db.relationship("ProjectTeam", back_populates="projects")
    client_user = db.relationship("ClientUser",  back_populates="project")
    status = db.relationship("Status", back_populates="projects")
    comments=db.relationship("Comments", back_populates="project")
    tasks = db.relationship("Tasks", back_populates="project")
    resumes=db.relationship("Resume", back_populates="project")
```

Figure 19 : Entité project, représentant un projet

En plus de déclarer le colonnes de la table dans le modèle, les relations de cette table avec les autres tables de la base de données doivent également être définies dans le modèle. Cela facilite la compréhension de la logique métier, la navigation entre les entités, la définition des contraintes d'intégrité référentielle entre les tables et une optimisation des requêtes.

Il existe types de relations, elles ont été décrites en détail dans le modèle conceptuel de données :

- La relation ***OneToOne***,
- La relation ***OneToMany***
- La relation ***ManyToOne***, inverse de *OneToMany*
- La relation ***ManyToMany***

Une relation est déclarée avec le mot clé ***db.relationship***, lors de sa déclaration l'entité avec laquelle la relation est établie est précisée. L'option ***back_populates*** indique la propriété correspondante à la relation inverse dans la deuxième entité.

Dans, la classe Project, l'attribut ***project_team*** définie la relation de cette table avec la table ***ProjectTeam*** :

```
 project_team = db.relationship("ProjectTeam", back_populates="projects") #
Relation avec la table ProjectTeam
```

Et dans la classe ***ProjectTeam*** , la relation inverse est également spécifiée :

```
projects = db.relationship("Project", back_populates="project_team") # Relation
inverse avec la table Project
```

Ainsi, lorsqu'on accède à l'attribut ***project_team*** d'une instance de la classe ***Project***, on peut obtenir un objet ou une liste d'objets représentant l'entité ***ProjectTeam*** associée à ce projet spécifique. De même, à travers la propriété ***projects*** d'une instance de la classe ***ProjectTeam*** , on peut accéder à la liste des projets associés à cette équipe projet. Cette relation correspond à une relation ***OneToMany/ManyToOne***

	idproject	project_name	project_description	project_status	_idproject_team
▶	1	Projet alpha	Description du projet	1	1
	2	Projet beta	Le projet beta!	1	NULL
*	NULL	NULL	NULL	NULL	NULL

Figure 20 : Extrait de la table project correspondant à l'entité Project

Les relations avec le mot clé ***secondary*** indique l'existence d'une table de jonction ce qui traduit une relation ***ManyToMany***. Dans la classe ***ProjectTeam***, l'attribut ***members*** définie la relation entre ***ProjectTeam*** et ***Member*** tout en précisant l'existence de table de jonction ***project_team_has_member*** :

```
members = db.relationship("Member", secondary='project_team_has_member',
back_populates="teams")
```

Les tables de jointures sont également représentées par des modèles et sont définies avec le mot clé class :

```
class ProjectTeamHasMember(db.Model):
    __tablename__ = 'project_team_has_member'

    project_team_idproject_team = db.Column(db.Integer, db.ForeignKey('project_team.idproject_team'), primary_key=True)
    member_idmember = db.Column(db.Integer, db.ForeignKey('member.idmember'), primary_key=True)
```

Figure 21 : Table de jointure entre les entités ProjectTeam et Member

Dans l'extrait suivant, on peut noter que la table de jointure n'associe que les clefs primaires des entités de bases :

	project_team_idproject_team	member_idmember
▶	1	3
*	NULL	NULL

Figure 22 : Extrait de la table de jointure des entités ProjectTeam et Member

Afin de pouvoir utiliser ces modèles SQLAlchemy pour interagir avec la base de données en effectuant des requêtes d'ajout, de modification, de mise à jour ou de suppression des données, SQLAlchemy a été configuré dans l'application Flask.

```python
app = Flask(__name__)
load_dotenv()

# Configurer la base de données SQLAlchemy
app.config['SQLALCHEMY_DATABASE_URI'] = os.getenv('SQLALCHEMY_DATABASE_URI')
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = os.getenv('SQLALCHEMY_TRACK_MODIFICATIONS')

db = SQLAlchemy()

# Configurer l'environnement de développement
app.config['FLASK_ENV'] = os.getenv('FLASK_ENV')
app.config['FLASK_DEBUG'] = os.getenv('FLASK_DEBUG')

from model.project import *
```

Figure 23 : Extrait de la configuration de SQLAlchemy dans l'application Flask

La configuration détaillée de SQLAlchemy est inscrite dans le fichier .env dans la structure du backend.

```python
@app.route('/admin/project', methods=['POST'])
def create_project():
    data = request.get_json()

    new_project = Project(project_name=data['project_name'],
                          project_description=data['project_description'],
                          expired_at=data['expired_at'])
    db.session.add(new_project)
    db.session.commit()

    return jsonify({'message': 'Project created successfully', 'idproject': new_project.idproject}), 201
```

Figure 24 : Route d'ajout d'un nouveau projet

On peut noter que l'interaction avec la base de données s'effectue par des sessions, la gestion des sessions reste très importante car elle permet de maintenir une cohérence des données lors des opérations complexes. Elle permet de clôturer chaque ouverture de sessions pour s'assurer que les ressources de la base de données sont correctement gérées.

La documentation des routes a été faite en docstring pour plus de clarté et une compréhension fluide du code.

A l'aide de l'outil MySQL Workbench, j'ai mis en place le diagramme ERD et on peut noté qu'il est identique au diagramme de classe du Modèle conceptuel de données.

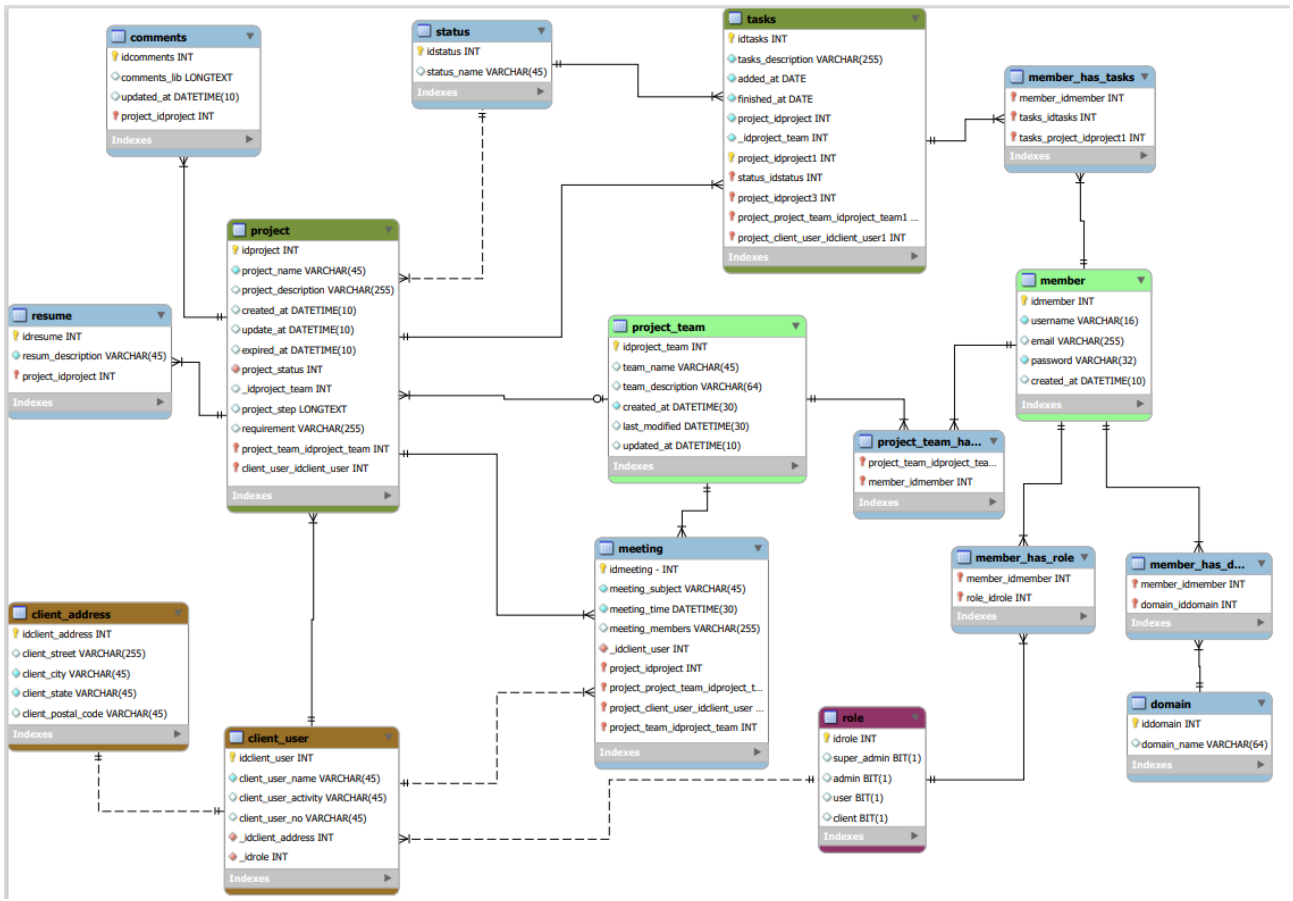

Figure 25 : Schéma de la base de données

C. Sécurité

Il est extrêmement important de respecter des normes de sécurité lors du développement d'un logiciel, et de faire en sorte de se prémunir autant que possible des attaques malveillantes ou des accidents. Même si une application n'est jamais sécurisée de façon absolue, des moyens de protection adaptés réduisent les risques et rendent les attaques malveillantes trop coûteuses pour être intéressantes.

La mise en place d'une **analyse de sécurité statique (SAST** - Static Application Security Testing), permet de vérifier que le code source du projet ne contient pas de vulnérabilité connue. Le procédé a été mis en place grâce à GitLab et permis de corriger plusieurs vulnérabilités, signalées plus bas. L'**OWASP** (Open Web Application Security Project), et plus particulièrement son **Top 10**, a été une source d'information utile sur les failles de sécurité à vérifier.
Les critères DICT définissent les quatre grands axes de sécurité qui ont été abordés pendant le développement :

- **D**isponibilité du système et de ses données
- **I**ntégrité du système et de ses données
- **C**onfidentialité des données
- **T**raçabilité des accès et altérations

La **disponibilité** est assurée en faisant appel à la persistance des données grâce à une base de données et un ORM (SQLAlchemy) pour faire le lien avec l'application.

L'**intégrité** consiste à s'assurer qu'une information n'est pas altérée et est unique. A nouveau, l'utilisation d'une base de données relationnelle permet d'y répondre en évitant les redondances, afin qu'une donnée ne soit stockée qu'une fois tout en restant utilisable aisément. L'utilisation de SQLAlchemy assure une protection contre les attaques par injections SQL, qui pourraient modifier ou supprimer des données. Enfin, ne pas stocker de valeurs calculées mais réaliser le calcul à chaque fois qu'on y fait appel permet d'éviter un décalage entre les valeurs de base et la valeur calculée.

La configuration du **CORS** contribue au maintien de l'intégrité des données coté serveur, en limitant les domaines qui sont autorisés à effectuer certaines actions. Il réduit le risque d'attaque CSRF (Cross-Site Request Forgery) où un site malveillant tente d'effectuer des actions au nom de l'utilisateur sur un autre site auquel l'utilisateur est authentifié.

```python
from flask_cors import CORS, cross_origin

app = Flask(__name__)

CORS(app, methods=['POST', 'GET', 'PUT', 'DELETE', 'OPTIONS'], resources={r"/*": {"origins": "http://localhost:8080", "supports_credentials": True}})
```

Figure 26 : Configuration du CORS côté serveur

Pour la **confidentialité**, plusieurs stratégies ont été mises en place.

- Sécuriser les connexions aux comptes des clients et des membres :

  Tout d'abord, le formulaire de login ne précise jamais si le courriel utilisé pour la tentative de login n'est pas enregistré ou si c'est le mot de passe qui est faux, afin de ne pas révéler d'information aux individus malveillants.

Le mot de passe n'est jamais stocké directement en base de données, il est toujours haché avant d'être enregistré dans la base de données avec la méthode ***set_password()*** et à chaque login une vérification est effectuée pour vérifier que le mot de passe entré correspond bien à ce qui est haché dans la base de données avec la méthodes ***check_password()*** :

```python
def set_password(self, passw):
    self.password = bcrypt.generate_password_hash(passw).decode('utf-8')

def check_password(self, passw):
    return bcrypt.check_password_hash(self.password, passw)
```

Figure 27 : Méthode de sécurisation des informations de connexion

Le hachage avec bcrypt est une méthode sécurisée de stockage des mots de passe. Bcrypt est un algorithme de hachage de mot de passe basé sur la fonction de dérivation de clé adaptative (Adaptive Key Derivation Function - KDF) bcrypt. Il est conçu pour être lent et résistant aux attaques par force brute, offrant ainsi une protection accrue contre les attaques par dictionnaire et les attaques par Rainbow tables.

Avant de hacher le mot de passe, un sel aléatoire est généré. Le sel est une valeur unique et aléatoire qui est stockée avec le mot de passe haché. Cela signifie que même si deux utilisateurs ont le même mot de passe, leur hachage sera différent en raison de l'utilisation de sels différents.

Le mot de passe de l'utilisateur est ensuite combiné avec le sel. Cette opération renforce la sécurité car elle garantit que même si deux utilisateurs ont le même mot de passe, le hachage sera différent en raison des sels uniques.

Le sel généré et le résultat du hachage (mot de passe + sel) sont ensuite stockés ensemble. Ce couple (sel, hachage) est stocké dans la base de données.

Lorsqu'un utilisateur tente de se connecter, le processus est inversé. Le sel stocké avec le mot de passe haché est extrait, combiné avec le mot de passe fourni par l'utilisateur, et la fonction de hachage

bcrypt est appliquée avec les mêmes itérations. Le résultat est comparé au mot de passe haché stocké dans la base de données.

L'utilisation des tokens JWT (Json Web Token) pour l'authentification permet de renvoyer un tokens après une authentification et donc de sécuriser les informations de connexion

Capture

- Reste celles du Front
- La configuration du CORS dans le front

  CORS est principalement conçu pour améliorer la confidentialité des données côté client. En empêchant les requêtes cross-origin non autorisées, CORS limite l'accès à certaines ressources à partir de domaines spécifiques. Cela empêche un site malveillant hébergé sur un domaine différent d'effectuer des requêtes et de récupérer des données qui sont normalement restreintes pour des raisons de confidentialité.

- Gestion des rôles

  La gestion des rôles fait parties des fonctionnalités développées pour l'application et cela permet gérer les autorisations. C'est une approche pour déterminer les privilèges et les 'accès d'accès aux comptes et aux fonctionnalités.

  Capture

La **traçabilité** à ajouter

9. Jeu d’essai élaboré par le candidat

- Les tests consistent généralement en 3 phases ; organiser, agir et vérifier.