



IT214 : Projet au Fil de l'Année

---

# ERP Aquitaine Électronique Informatique

## Rapport Final

---

*Équipe :*

Nesrine ABID  
Mathieu DUPOUX  
Léo-Paul MAZIÈRE  
Lisa VEILLAT

Nadjime BARTEAU  
Maël PAUL  
Antoine RAOULT  
Marine VOVARD

*Encadrant :*

David RENAULT

9 mai 2023

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>4</b>
1.1	Aquitaine Électronique Informatique (AEI)	4
1.2	Objectifs du PFA	4
<b>2</b>	<b>Organisation</b>	<b>5</b>
2.1	Organisation du travail en équipe	5
2.1.1	Organisation du début de projet	5
2.1.2	Organisation au cours du projet	6
2.1.3	Organisation à la fin du projet et transmission du travail	7
2.2	Organisation du projet - code	8
2.2.1	Stratégie de gestion des branches et utilisation de Git	8
2.2.2	Mise en forme du projet	9
<b>3</b>	<b>Architecture</b>	<b>10</b>
3.1	Structure générale	10
3.2	Modèle de données	10
3.2.1	Analyse et simplification du modèle existant	10
3.2.2	Implémentation et contraintes d'intégrité	11
3.2.3	Version finale du modèle	12
3.3	Backend	13
3.3.1	SGBD et environnements	13
3.3.2	Migrations	13
3.3.3	Interactions Modèle-Contrôleur-Routeur	14
3.3.4	Endpoints	14
3.4	Frontend	15
3.4.1	Vue d'ensemble de l'interface	15
3.4.2	Technologies utilisées	15
<b>4</b>	<b>Fonctionnalités</b>	<b>17</b>

4.1	Authentification . . . . .	17
4.1.1	Backend . . . . .	17
4.1.2	Frontend . . . . .	17
4.2	Permissions . . . . .	18
4.3	Navigation sur l'ERP grâce à des menus . . . . .	18
4.4	Gestion des membres . . . . .	18
4.4.1	Tableau des membres . . . . .	18
4.4.2	Visualisation des informations d'un membre . . . . .	19
4.4.3	Ajout et mise à jour d'un membre . . . . .	19
4.5	Gestion des documents . . . . .	20
<b>5</b>	<b>Comparaison avec l'existant</b>	<b>21</b>
<b>6</b>	<b>Continuité du projet</b>	<b>22</b>
	<b>Lexique</b>	<b>23</b>
	<b>Annexe</b>	<b>1</b>
<b>A</b>	<b>Backend</b>	<b>1</b>
A.1	Endpoints . . . . .	1

# 1 | Introduction

---

## 1.1 Aquitaine Électronique Informatique (AEI)

AEI est la Junior Entreprise de l'ENSEIRB-MATMECA. Son objectif est de fournir des missions, que l'on appelle études, aux élèves de l'école afin qu'ils acquièrent des compétences dans les domaines enseignés en cours. Ces études sont fournies par des entreprises ou organismes extérieurs et les étudiants travaillant dessus sont rémunérés pour le travail qu'ils accomplissent. Dans le vocabulaire de la Junior Entreprise, on appelle ces étudiants des réalisateurs ou intervenants.

Pour gérer l'organisation de toutes ces études, une équipe d'une trentaine d'étudiants de l'école travaille pour faire le lien entre les entreprises et les réalisateurs. On les appelle administrateurs de la Junior Entreprise.

Cette équipe est séparée en plusieurs pôles :

- **Bureau** : responsables d'AEI, gère les événements internes et externes, les ressources humaines, les assemblées générales, les adhésions, ...
- **Trésorerie** : gère les comptes de la Junior Entreprise
- **Étude** : suit les études en cours
- **Développement Commercial** : se charge de démarcher les clients, faire de la prospection, rédiger les devis
- **Qualité** : relit les documents, établit les processus et procédures, gère les indicateurs et les plan d'actions (PDA) de chaque pôle
- **Communication** : s'occupe de la communication externe de la Junior Entreprise sur les réseaux sociaux et gère l'image d'AEI (site web, communication au sein de l'école et en dehors de l'école)
- **SI** : gère tous les systèmes informatiques d'AEI (gestion du serveur avec le site de l'ancien ERP, de WikiX (processus et procédures), du téléphone fixe et des backups des données)

## 1.2 Objectifs du PFA

Pour organiser le travail de cette large équipe, AEI dispose d'un ERP qui permet de gérer la relecture des documents, le suivi d'études, la gestion des adhérents, etc... Cependant, cet ERP a été développé par une autre Junior Entreprise, **N7 Consulting**, la Junior Entreprise de l'ENSEEIH.

Le code étant open-source, il a été repris il y a de nombreuses années par les administrateurs d'AEI. Cependant, il ne répond plus exactement aux besoins d'AEI et les années de bidouillages sur ce code l'ont rendu maintenant compliqué à maintenir.

Il a donc été décidé par le conseil d'administration d'AEI de redévelopper un nouvel ERP en repartant de zéro et nous avons profité des PFAs pour pouvoir travailler dessus en équipe.

## 2 | Organisation

### 2.1 Organisation du travail en équipe

#### 2.1.1 Organisation du début de projet

Au début du projet, nous avons pris la décision de regrouper toutes les informations au même endroit afin d'assurer une meilleure organisation. Pour cela, nous avons commencé à utiliser le tableau Kanban disponible sur GitHub. Ce tableau nous a permis de lister toutes les tâches à effectuer ainsi que les cas d'utilisation de notre projet. Parallèlement, nous avons entrepris de recueillir les besoins de tous les pôles de la Junior-Entreprise. Nous avons également réalisé une analyse de l'ERP existant afin de mieux comprendre les attentes pour ce projet.

En plus du tableau Kanban, nous avons mis en place un serveur Discord dédié à notre projet. Sur ce serveur, nous avons créé différents fils de discussion pour faciliter la communication. Nous avons un fil de discussion général pour les échanges courants, un fil de discussion dédié aux liens utiles liés au projet, ainsi qu'un fil de discussion spécifique pour suivre les commits effectués sur la branche de développement. Ce dernier fil de discussion s'est révélé particulièrement utile pour permettre à chacun de suivre l'avancement du projet et d'identifier d'éventuelles difficultés rencontrées lors du développement.

L'idée de la mise en place du serveur Discord était de favoriser une communication efficace et facile pour tous les membres de l'équipe. Étant donné que notre projet impliquait un groupe plus important que d'habitude, il était essentiel de veiller à ce que la communication reste claire au sein de l'équipe.

En ce qui concerne la répartition des tâches, nous avons établi les grandes lignes directrices en groupe, puis nous avons fait le point lors de nos réunions hebdomadaires sur ce qui avait été accompli, ce qui n'avait pas pu être terminé, ainsi que sur les éventuels problèmes ou points de discussion à aborder.

Nous avons également eu la chance de pouvoir utiliser le local dédié à la Junior-Entreprise lors de nos réunions. Cela nous a permis de nous approprier l'espace et en particulier d'utiliser l'un des deux tableaux présents dans la salle. Ces tableaux se sont révélés extrêmement efficaces pour faire des schémas, discuter ensemble des points à aborder et favoriser une planification visuelle plus tangible que sur un ordinateur. Les interactions directes et la visualisation sur tableau ont permis d'échanger efficacement.

Concernant le cahier des charges, nous avons établi un diagramme de Gantt prévisionnel visible Figure 2.1 pour planifier le déroulement du projet. Cependant, nous pensons qu'il peut être amélioré pour refléter de manière plus précise les différentes étapes, les dépendances entre les tâches et les échéances.

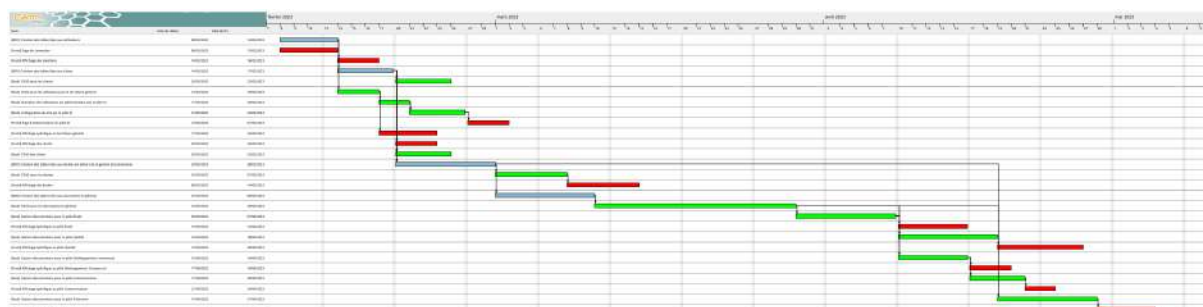


FIGURE 2.1 – Gantt au début du projet

### 2.1.2 Organisation au cours du projet

Nous avons décidé de revoir notre organisation lors de la rédaction du cahier des charges, car nous avons réalisé que l'outil sur GitHub n'était pas adapté à notre projet. Afin de remédier à ce problème, nous avons opté pour un outil simple et accessible à tous : Google Drive. Voici comment nous avons mis en place notre système : à la fin de chaque réunion hebdomadaire, nous faisons le point sur les sujets à aborder à l'aide d'un tableau, puis nous répartissons les tâches en fonction des affinités de chacun. Tout le monde pouvait ajouter les tâches importantes à effectuer pour la semaine suivante, puis nous établissions une liste de priorités. Une fois que nous avons défini un nombre suffisant de tâches pour les sept membres de l'équipe, nous les répartissons équitablement, en permettant à chacun de choisir l'aspect qui l'intéressait le plus en fonction de ses compétences et de son expérience.

[illegible]

FIGURE 2.2 – Exemple d'un tableau d'avancement en fin de projet

De plus, ayant généralement plusieurs projets en parallèle et du travail à effectuer à côté, il était parfois difficile de trouver du temps pour travailler sur l'ERP. Nous avons donc cherché à rendre le codage aussi simple, motivant et efficace que possible. Afin de maintenir la motivation de tous les membres de l'équipe, nous avons mis en place un système de cases à cocher qui permettait de mesurer l'avancement des tâches individuelles ainsi que des tâches de l'ensemble de l'équipe. Mais dans l'ensemble, il n'y a pas eu besoin de mettre une hiérarchie particulière au sein du projet.

Lorsque des problèmes ou des questions techniques sur le projet se posaient, nous avons rapidement réalisé qu'il était nécessaire d'avoir une personne de référence à qui s'adresser. Étant donné que certaines technologies étaient inconnues pour la plupart d'entre nous, il fallait prendre des décisions en avançant au mieux et parfois revenir en arrière lorsque nous constatons que cela ne fonctionnait pas. Par conséquent, nous avons unanimement désigné Antoine, ayant le plus d'expérience avec les technologies utilisées, comme leader technique. Dans un premier temps, il a mis en place un prototype du projet pendant que le reste du groupe travaillait sur la spécification des besoins.

Un autre avantage de tout consigner sur Google Drive est que cela permet de conserver une trace du projet, ce qui peut s'avérer utile à l'avenir pour nous, en cas de projets similaires ultérieurs, ainsi que pour les personnes qui travailleront sur ce projet à l'avenir, afin de pouvoir constater l'évolution du projet.

En conclusion, au cours du PFA, nous avons adapté notre méthode de travail afin de favoriser l'implication de chacun dans le projet et de consacrer le plus de temps possible à des tâches à valeur ajoutée, telles que la programmation. Notre objectif était d'avoir une approche flexible, simple et motivante, de sorte que tous les membres de l'équipe soient fiers de ce qui a été accompli à la fin du projet.

Cependant, cette méthode présente encore des axes d'amélioration. Les dernières personnes à choisir les tâches se retrouvent souvent avec des tâches moins motivantes. De plus, il n'existe pas d'indicateurs permettant de mesurer le temps passé par chacun sur les différentes tâches. En effet, l'un des points particulièrement compliqué à aborder lors de ce projet a été d'évaluer le temps et la complexité de chaque tâche. Cela est dû au fait que nous nous sommes lancés dans des technologies avec lesquelles nous n'étions pas familiers. Finalement, nous aurions aimé pouvoir réaliser une release et avoir un premier retour client sur les fonctionnalités implémentées lors de ce projet. Cela n'a pas pu

être mis en place par manque de temps mais également car un changement de mandat a été effectué au sein de la Junior-Entreprise en avril, date prévu de notre release. Cependant, les nouveaux entrants au sein du mandat ne sont pas encore familiers avec l'ancien ERP et les procédures à suivre. Nous avons donc estimé qu'il était préférable de leur accorder un certain laps de temps afin qu'ils puissent prendre conscience des problématiques de l'ancien ERP avant de leur présenter ce qui avait été réalisé au cours de ce PFA. Cependant, une démonstration de ce qui a été réalisé a mis en place à été faite pour les chargés du pôle DSI de la Junior-Entreprise.

En ce qui concerne les comptes rendu des réunions. Au début, chacun prenait des notes de son côté. Nous avons décidé par la suite dans notre répertoire commun sur le drive de mettre en place des documents partagés afin que chacun puissent contribuer au projet.

[METTRE UNE PHOTO DU GANTT "REEL"]

### 2.1.3 Organisation à la fin du projet et transmission du travail

Ce projet n'a pas pu être entièrement réalisé pendant ce PFA. Il est donc essentiel de pouvoir transmettre le travail effectué par la suite. En effet, l'un des problèmes majeurs de l'ancien ERP était la complexité croissante du code au fil du temps. Par exemple, dans la base de données, certaines tables étaient en français tandis que d'autres étaient en anglais, certaines commençaient par des majuscules et d'autres par des minuscules, ... Cet exemple simple illustre les problèmes auxquels nous devons faire face dans ce projet. De plus, la Junior-Entreprise repose sur un système de mandat annuel, ce qui implique que de nouvelles personnes travaillent sur l'ERP chaque année. Ainsi il est primordial que le travail puisse être facilement transmis et qu'une attention particulière soit portée à la maintenabilité du code à long terme.

De plus, l'ancien ERP était basé sur un projet open source développé par l'ENSHEEIT. Le but est donc de pouvoir sur le long terme partager ce code et le rendre publiques et accessibles à tous. Pour cela, nous avons pour objectif de publier notre code sous licence GNU Affero General Public Licence (GNU AGPL). Il s'agit d'une licence libre copyleft avec une clause supplémentaire par rapport à la licence GNU GPL obligeant les utilisateurs accédant au logiciel via un réseau (par exemple, un serveur web) à rendre disponible le code source du logiciel modifié. Le but étant de garantir que les utilisateurs du logiciel auront accès au code source, même si le logiciel est exécuté à distance.

Les autres caractéristiques de cette licence sont l'autorisation de la redistribution et de la modification du code source avec une garantie que les versions modifiées soient aussi sous licence AGPL (principe du copyleft). Mais aussi l'obligation de citer les auteurs d'origine du code source dans toutes les versions modifiées.

La licence ne permettant pas de base de restreindre la commercialisation du logiciel par des tiers, il faudra par contre qu'on ajoute une clause personnalisée afin d'empêcher toute commercialisation. Mais ce dernier reste en cours de discussion. Notamment car autoriser la commercialisation peut permettre à des entreprises de contribuer au développement du logiciel. Étant donné qu'elles resteraient dans l'obligation de publier le code modifié, cela pourrait tout de même bénéficier au logiciel d'origine tout en donnant envie à des entreprises qualifiées de participer au développement du projet.

Pour conclure sur ce point, nous estimons que la licence GNU AGPL serait la plus à même de bénéficier au bien d'AEI dans le but de reprendre le développement du projet mais aussi de bénéficier à l'ensemble des associations de types Junior Entreprise qui pourraient s'y investir également sans restrictions d'accessibilité du code source.

## 2.2 Organisation du projet - code

### 2.2.1 Stratégie de gestion des branches et utilisation de Git

Au début du projet, nous avons créé une association sur GitHub pour AEI. L'objectif est de permettre à tous les membres de l'association de contribuer, de coordonner leurs efforts et de travailler ensemble sur le projet. Ainsi, le code est donc relié à une entité (ici AEI) plutôt qu'à une personne. Cela est en accord avec notre démarche de vouloir rendre ce projet open source. En effet, les associations sur GitHub permettent l'utilisation d'outils de gestion de projet intégrés à GitHub, tels que les tableaux Kanban, les problèmes, les discussions et les permissions d'accès.

Au cours du projet, nous avons expérimenté différentes méthodes d'organisation des branches sur GitHub pour faciliter le développement collaboratif. Initialement, nous avons utilisé les branches suivantes :

- branche prototype : Cette branche a été créée pour le développement du prototype initial du projet. Elle était destinée à explorer les idées et les concepts sans perturber la branche principale ("main"). Cela nous a permis d'itérer rapidement sur les fonctionnalités et d'expérimenter différentes approches.
- branche main : C'est la branche principale du projet, où la version stable et fonctionnelle du logiciel était conservée. Les modifications apportées dans d'autres branches étaient fusionnées avec celle-ci une fois qu'elles étaient suffisamment matures.
- branche front et branche back : Ces branches ont été créées pour séparer le développement du front-end et du back-end du projet, respectivement. Cela permettait de travailler de manière indépendante et de minimiser les conflits lors de la fusion des modifications.

Au début, nous souhaitions éviter au maximum les conflits sachant que nous étions nombreux à travailler sur le projet. Ainsi, des branches reliées au front-end ont été ajoutées et des branches différentes pour le back-end également. Ce qui fait que nous nous sommes rapidement avec 5 branches différentes en même temps et nous avons rencontré des problèmes de synchronisation entre certaines branches.

Voyant le système se complexifier, nous avons donc décidé de modifier notre méthode d'organisation des branches. Nous avons ainsi adopté la structure suivante :

- branche main : Cette branche est devenue la branche principale du projet, où la version stable et fonctionnelle était maintenue. Toutes les modifications ont été fusionnées avec cette branche une fois qu'elles étaient prêtes pour une intégration complète.
- branche dev : Nous avons introduit une nouvelle branche dev pour le développement continu. Étant donné qu'il n'y avait pas de conflits majeurs entre le front-end et le back-end, nous n'avions plus besoin de les séparer. Cette branche a servi de terrain de jeu pour les développements en cours et les fonctionnalités en cours d'élaboration.
- Autres branches : Nous avons également créé des branches supplémentaires au besoin. Par exemple, lorsque nous ajoutons de nouvelles fonctionnalités telles que la gestion des documents ou l'authentification, nous créons des branches spécifiques pour ces tâches. Cela nous permettait de travailler sur ces fonctionnalités de manière isolée et de les fusionner avec la branche dev une fois qu'elles étaient suffisamment matures.

En résumé, la méthode d'organisation des branches a évolué au cours du projet. Nous sommes passés d'une approche avec des branches distinctes pour le prototype, le front-end et le back-end à une structure simplifiée avec les branches main et dev. Cette nouvelle structure nous a permis de travailler de manière plus fluide et de fusionner les fonctionnalités au moment approprié, tout en maintenant une branche principale stable.

Finalement, au cours du PFA, nous avons exploré une solution de vérification de la qualité du code en utilisant GitHub Actions et SonarQube. L'objectif était d'automatiser les processus d'analyse statique et de détection des problèmes de code. Cependant, après avoir évalué son utilisation et pris en compte les contraintes de notre environnement de développement, nous avons décidé de ne pas l'utiliser de



manière intensive. Ainsi, l'analyse ne se déclenche que lorsqu'il y a une modification sur la branche main.

Avec plus de temps, nous aurions aimé mettre en place une structure plus complète d'intégration continue à l'aide de GitHub Action en ajoutant le lancement des tests automatiques afin d'éviter une régression des tests (qui est actuellement fait à la main techniquement avant chaque commit). On aurait également pu envisager un déploiement automatique de l'application. Cela aurait grandement facilité l'échange avec le client.

## 2.2.2 Mise en forme du projet

L'adoption de conventions de codage claires et cohérentes joue un rôle essentiel dans le développement d'un projet logiciel de qualité. Dans notre projet, nous avons établi des conventions de codage spécifiques pour le back-end et la base de données afin de garantir une base solide pour notre code source.

Nous avons également ajouté au fur et à mesure du projet d'autres conventions. Voici des exemples de ce que nous avons mis en place :

- Commits significatifs avec gitmoji : Afin de rendre nos commits plus explicites et informatifs, nous avons adopté l'utilisation de gitmoji. Gitmoji est une convention qui utilise des emojis pour représenter différentes actions dans les messages de commit. Cela permet une meilleure compréhension du contenu des commits et facilite la recherche d'informations spécifiques dans l'historique du projet
- Utilisation de Prettier avec Husky : Pour maintenir une cohérence dans le style du code, nous avons intégré Prettier à notre flux de travail. Prettier est un outil de formatage automatique du code qui garantit une présentation uniforme du code source. En utilisant Husky, un gestionnaire de hooks Git, nous avons automatisé le processus de formatage du code lors des commits ou des pushes. Cela permet d'économiser du temps et d'éliminer les discussions inutiles sur le style du code au sein de l'équipe
- Documentation des recommandations de convention de codage : Afin de maintenir la cohérence et la lisibilité du code, nous avons créé un document décrivant les recommandations et les conventions de codage spécifiques au projet. Pour le back-end, nous avons précisé les modifications nécessaires pour ajouter des données dans un fichier README. Pour la base de données, nous avons choisi de conserver les noms en français, étant donné que certains termes sont spécifiques à la langue française et leur traduction aurait été étrange.

Le document de conventions de codage sert de référence pour l'équipe et garantit une meilleure compréhension du code tout au long du développement. En mettant en œuvre ces pratiques, nous avons constaté une amélioration significative dans la gestion de notre projet. Cela nous a permis de travailler de manière plus organisée, cohérente et collaborative, ce qui a contribué à l'efficacité et à la qualité de notre code.

Il est important de souligner que ces conventions ne sont pas figées, mais plutôt flexibles et évolutives. nous avons ajusté et affiné les conventions de codage pour répondre aux besoins spécifiques de notre équipe et du projet lui-même. Cette approche itérative nous a permis de tenir compte des retours et des nouvelles exigences qui ont émergé tout au long du développement.

## 3 | Architecture

### 3.1 Structure générale

Notre application utilise une base de données MySQL (en production) et est décomposée en trois parties principales : un système de gestion de base de données, une API REST (backend) qui interagit avec la base de données et une interface utilisateur (frontend) qui communique avec l'API pour permettre aux utilisateurs d'interagir avec l'application.

Le backend est écrit en TypeScript avec ExpressJS, utilise l'ORM Sequelize pour gérer les interactions avec la base de données et fournit une API REST. Le frontend, lui aussi en TypeScript, est construit avec VueJS et utilise PrimeVue pour faciliter la construction de l'interface utilisateur avec des composants VueJS.

Une illustration de l'architecture technique est fournie dans la Figure 3.1.

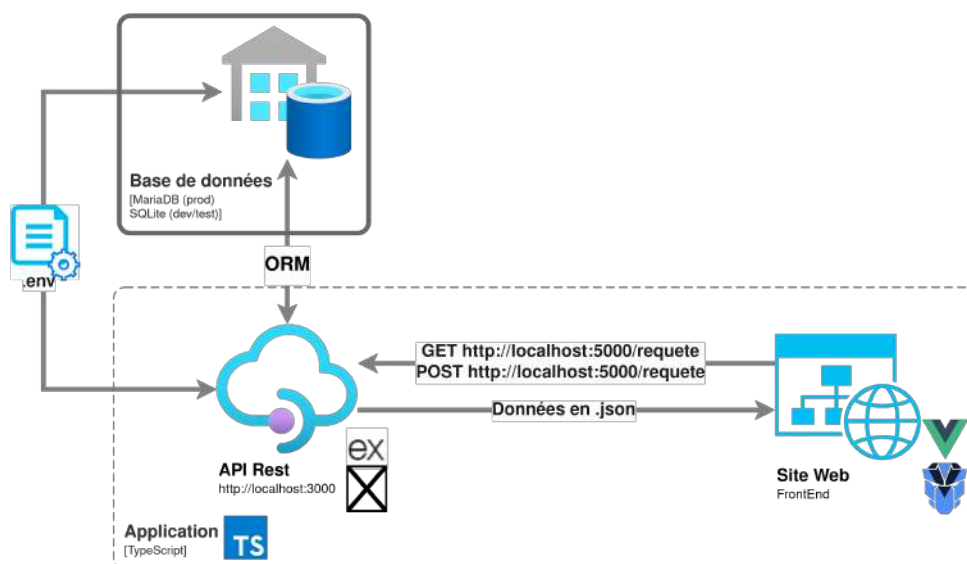


FIGURE 3.1 – Architecture technique proposée pour le nouvel ERP d'AEI

On remarque alors que l'intelligence de l'application (authentification, gestion des permissions, requêtes vers la base de donnée, vérifications d'intégrité...) est concentrée sur la partie backend. L'architecture permet alors d'implémenter de multiples versions de frontend pouvant être adaptées sur différents supports (application mobile, client lourd...)

### 3.2 Modèle de données

#### 3.2.1 Analyse et simplification du modèle existant

La première étape de notre démarche a consisté à analyser le modèle de données de l'ancienne base existante. Nous avons étudié sa structure et ses composants afin de comprendre son fonctionnement et d'identifier les éléments pertinents à conserver pour notre projet.

Cependant, nous avons rapidement réalisé que le modèle de données de l'ancienne base était complexe et comportait des éléments redondants ou peu utilisés. Dans le souci de simplifier notre nouveau modèle de données et d'améliorer sa lisibilité, nous avons entrepris une démarche de simplification de la base de données existantes qui s'est fait de manière itérative. En effet, lorsque nous avons extrait un schéma de la base de données existantes, nous avons obtenus 54 tables toutes reliées entre elles comme on peut le voir sur le schéma Figure 3.2. Certaines de ses tables ont été ajoutées au fur et à mesure de l'utilisation de l'ancien ERP. Le modèle de données s'est donc considérablement complexifié au cours du temps. De plus, un bon nombre de tables étaient reliées à des fonctionnalités qui n'étaient pas utilisées.

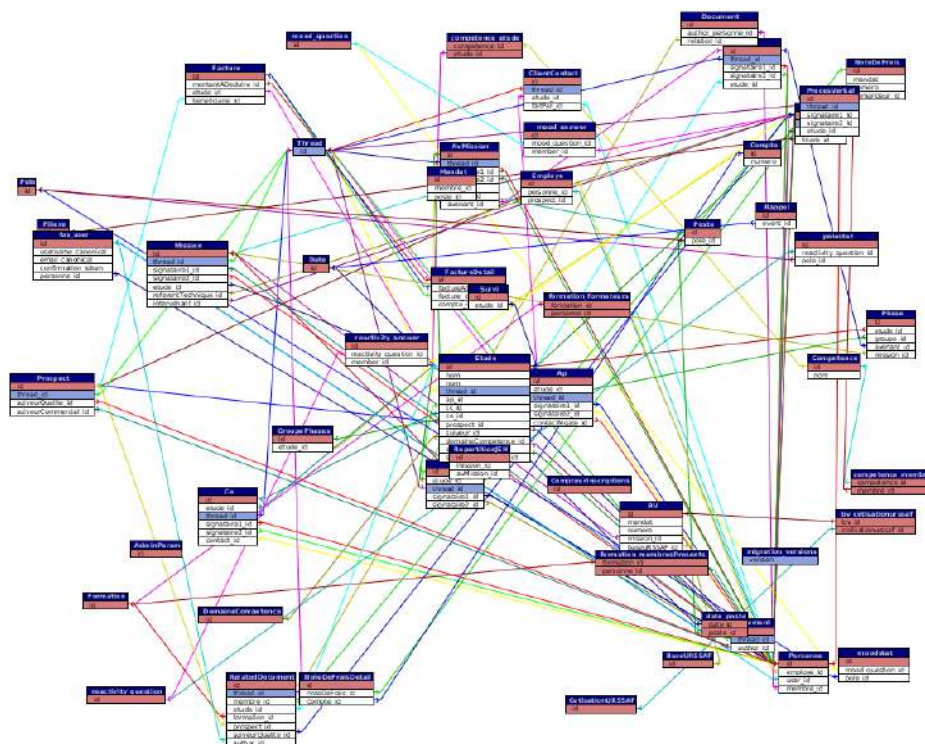


FIGURE 3.2 – Modèle de données de l'ancien ERP

Lors de l'élaboration de notre cahier des charges, nous avons réussi à considérablement réduire le nombre de tables de données, aboutissant finalement à 11 tables (sans compter les énumérations qui pourraient se transformer en tables si des modifications fréquentes étaient nécessaires). Dans ce modèle, seuls les types de données étaient indiqués. Il a donc fallu choisir en plus si certains attributs pouvaient être nuls, ainsi que les vérifications à apporter sur les données afin d'assurer leur cohérence dans le temps.

### 3.2.2 Implémentation et contraintes d'intégrité

Au cours de l'implémentation de la base de données, des modifications ont été apportées à ce modèle. Certains types ont été modifiés, comme par exemple le numéro de téléphone qui, initialement marqué comme un entier, a été transformé en chaîne de caractères afin de prendre en charge l'écriture des numéros internationaux.

Nous avons utilisé un ORM (Object-Relational Mapping) pour créer nos données, et par défaut, chaque table possède les attributs suivants : un identifiant unique en tant que clé primaire, une date de création et une date de modification. Afin de respecter les contraintes d'intégrités, nous avons utilisés deux mécanismes : la vérification sur la base et la vérification avec l'ORM en utilisant des validateurs. Ces deux mécanismes ont des avantages et inconvénients.

La vérification au niveau de la base de données offre certains avantages. Tout d'abord, elle garantit l'intégrité des données au niveau le plus bas, directement dans la base de données elle-même. Les contraintes d'intégrité définies au niveau de la base de données, telles que les clés primaires, les clés étrangères, les contraintes de valeur et les règles d'unicité, sont appliquées automatiquement lors des opérations de modification des données. Cela assure une cohérence et une fiabilité des données à un niveau fondamental.

En revanche, la vérification au niveau de la base de données peut présenter certains inconvénients. L'un des principaux inconvénients est que les contraintes d'intégrité définies dans la base de données sont souvent plus difficiles à modifier que les règles de validation au niveau de l'application. Modifier une contrainte au niveau de la base de données peut nécessiter une migration de schéma complexe et ainsi engendrer des erreurs lors de cette migration ou des problèmes conséquents sur l'application.

D'autre part, la vérification avec l'ORM en utilisant des validateurs offre une flexibilité plus grande. Les validateurs au niveau de l'ORM permettent de définir des règles de validation personnalisées pour les modèles de données. Nous avons en effet utilisé des validateurs déjà existants pour certaines vérifications (certains étant déjà présent dans l'ORM et d'autres venants d'une bibliothèque. Les validateurs sont plus faciles à modifier et à ajuster en fonction des besoins évolutifs de l'application. De plus, ils peuvent dans notre cas être réutilisé au niveau du front-end afin de s'assurer que les données ajoutées par l'utilisateur sont bien cohérentes avec la base.

Cependant, la vérification avec des validateurs au niveau de l'ORM peut présenter certains risques. Si les validateurs ne sont pas correctement configurés ou mis à jour, il est possible de contourner les règles de validation et d'introduire des données incohérentes ou incorrectes dans la base de données. De plus, nous avons constaté que lorsque nous importons nos données à partir des seeders (mécanisme qui permet d'ajouter des données initiales ou de remplir une base de données avec des données de test), les vérifications avec les validateurs n'étaient pas faites ce qui peut venir mettre en péril l'intégrité de la base.

```

1  @Column({
2      type: DataType.STRING,
3      allowNull: false,
4      validate: {
5          checkPhone(str: string) {
6              if (!validator.isMobilePhone(str)) {
7                  throw new Error("Invalid phone number");
8              }
9          },
10     },
11 })
12 telephoneMobile!: string;
```

FIGURE 3.3 – Validateurs pour numéro de téléphone dans la classe Adhérent

### 3.2.3 Version finale du modèle

On retrouve le modèle de données finale sur la figure Figure 3.4.

Lors de ce PFA, nous nous sommes concentrées sur les acteurs principaux de la Junior-Entreprise ainsi qu'un début d'implémentation pour les documents. Il sera nécessaire dans la suite du développement de créer des tables en lien avec les études.

Finalement, nous avons apporté un soin tout particulier pour corriger un bug présent dans l'ancien ERP qui consistait à ne pas reconnaître correctement l'historique des documents si on changeait le nom de celui-ci lors d'un import de ce document. Ainsi, à l'heure actuelle, un document est relié à des fichiers, ces fichiers étant des versions différentes du documents.

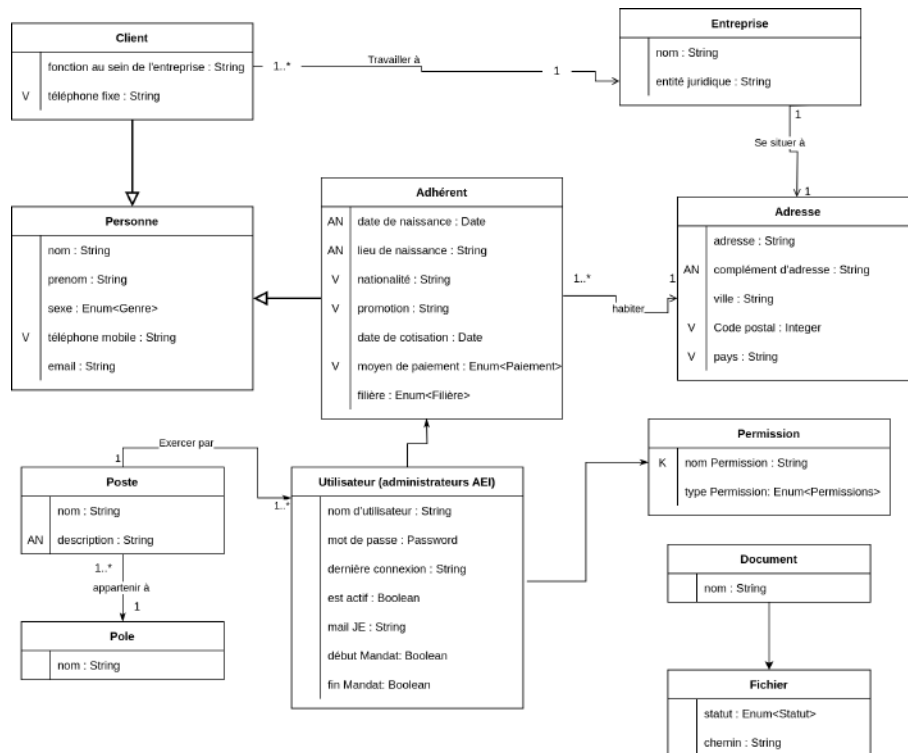


FIGURE 3.4 – Modèle de données final de la base implémentée

## 3.3 Backend

### 3.3.1 SGBD et environnements

Nous avons implémenté à l'aide de la variable d'environnement Node.JS `NODE_END` permettant de gérer différentes configurations en fonction de l'environnement dans lequel notre application est exécutée. Dans notre cas, nous avons créé trois environnements distincts : `dev` (développement), `test` (tests) et `production`. Chacun de ces environnements utilise une base de données différente.

L'environnement `dev` utilise d'une base de données SQLite stockée dans un fichier local. Cette approche nous permet de travailler efficacement et simple à mettre en place en fournissant une base de données locale pour tester et déboguer notre application sans dépendre d'un serveur de base de données externe. L'environnement `test`, utilise une base de données SQLite distincte de celle de développement. Cette séparation nous permet de garantir l'indépendance des tests et d'éviter toute interférence avec les données de développement. Cela nous a notamment permis de corriger des erreurs d'accès du fichier SQLite lors de l'exécution des tests

Enfin, l'environnement de production est paramétré à l'aide d'un fichier `.env` spécifiant le serveur à choisir. Il est prévu de choisir MySQL comme solution de gestion de base de donnée, car c'est une solution mature, robuste, performante et déjà utilisée par les différents services existant chez AEI.

### 3.3.2 Migrations

Nous avons mis en place un système de migration de notre base de donnée gérée par Sequelize. Pour être plus précis, nous utilisons la bibliothèque `umzug` qui facilite la gestion des migrations de base de données pour les applications Node.js. La fonctionnalité de migration permet de maintenir une base de données à jour au fil du temps en appliquant des modifications de schéma (ajout de tables, modification

de colonnes, etc.) de manière structurée et contrôlée.

Ces migrations sont définies sous forme de fichiers dans le répertoire migrations, externalisée des sources de l'application. Chaque fichier de migration, écrits en TypeScript, correspond à une modification de schéma particulière. Il existe deux types de migrations avec umzug : les *migrations* qui correspondent à une modification de la structure de la base de donnée et les *seeders* qui permettent de peupler la base de donnée avec des données particulières.

Umzug permet de facilement gérer l'exécution des migrations, en vérifiant quelles migrations ont déjà été appliquées et en appliquant uniquement les migrations manquantes dans l'ordre approprié. Il est également possible de faire des « rollback » des migrations appliquées en cas de besoin. Cette exécution peut être gérée de manière programmatique (fonction `up()` et `down()`) ou en ligne de commande l'aide des scripts `migrate.js` et `seed.js` exécutable avec les raccourcis décrits dans le listing 1 définis dans `package.json`.

```
npm run migrate -- up           #Up all migrations
npm run migrate -- up --to <n>  #Up to the n-th migration
npm run migrate -- down        #Down last migration
npm run migrate -- down --to <n> #Down all migration to n-th migration
npm run migrate -- create --name=<Name> #Generate new empty migration from template

npm run seed -- up             #Up all seeders
npm run seed -- up --to <n>    #Up to the n-th seeder
npm run seed -- down          #Down last seeder
npm run seed -- down --to <n>  #Down all seeders to n-th seeder
npm run seed -- create --name=<Name> #Generate new empty seeder from template
```

Listing 1 – Extrait du README.md décrivant l'utilisation des migrations

### 3.3.3 Interactions Modèle-Contrôleur-Routeur

Dans le schéma d'interaction décrit en Figure 3.5, nous remarquons que toute requête, une fois réceptionnée par un des routeurs passe par une partie « middleware », vérifiant notamment si la requête est authentifiée ou non. Si la requête est bien authentifiée, alors elle est renvoyée au contrôleur dédié qui se chargera de traiter la requête, en manipulant les différents modèles définis à l'aide de Sequelize. Ce sont ces modèles qui permettent d'interagir avec la base de donnée de l'application. Une fois la requête traitée, le contrôleur renvoie une réponse, si besoin en passant par un gestionnaire d'erreur.

### 3.3.4 Endpoints

Cette partie backend est implémentée comme une API REST servie par ExpressJS et les routeurs présentés précédemment. Notre API possède ainsi nombre de requêtes API utilisable par notre frontend. Celles-ci sont listés dans l'annexe A.0, mais on a globalement des requêtes suivant le schéma CRUD (Create, Read, Update, Delete) suivant :

1. Ajout d'un nouvel élément
2. Consultation d'un élément existant
3. Liste de tous les éléments existants
4. Mise à jour d'un élément existant
5. Suppression d'un élément existant



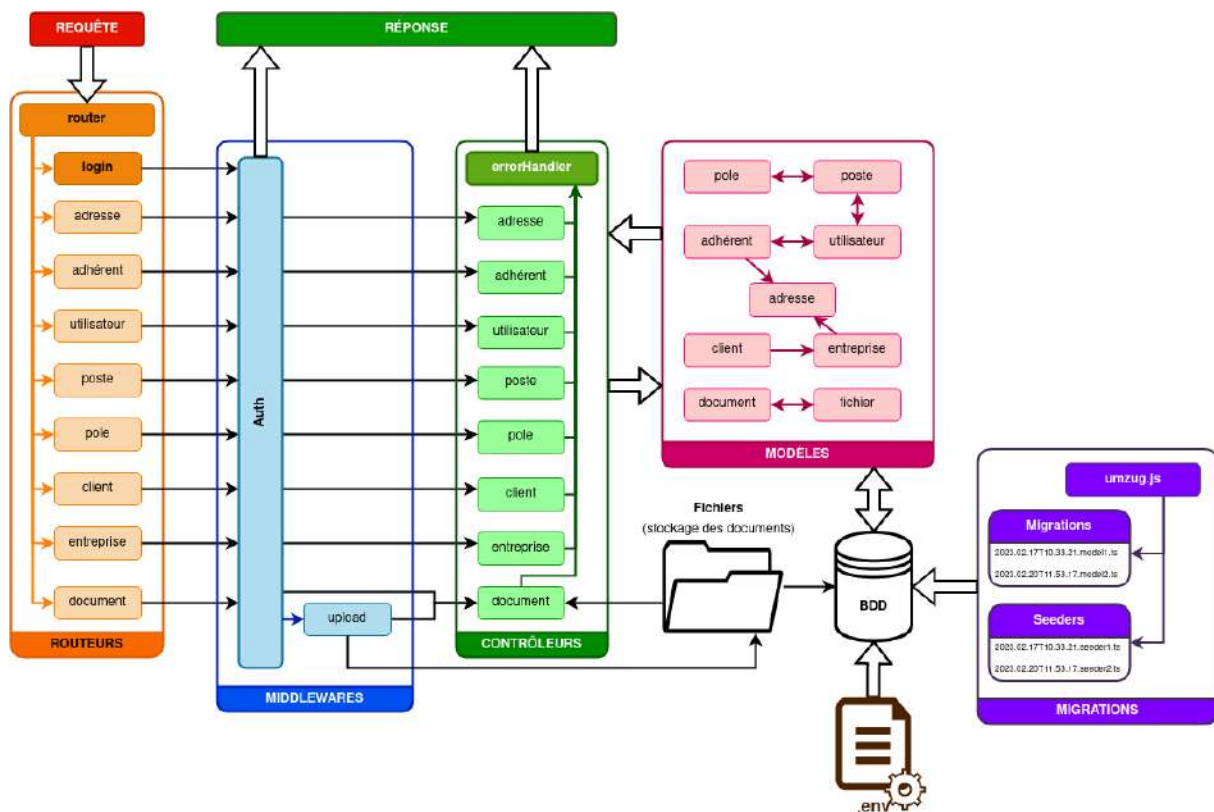


FIGURE 3.5 – Schéma de principe du fonctionnement du backend

## 3.4 Frontend

Dans cette partie, nous allons traiter de l'interface qui a été développée pour l'ERP.

### 3.4.1 Vue d'ensemble de l'interface

D'un point de vue général, l'interface actuelle permet en grande partie de faire de la gestion des adhérents de la Junior-Entreprise et de leurs prospects. De plus, une partie sur la gestion des documents a aussi commencé à voir le jour.

De manière plus précise, l'interface est composée d'une page d'authentification (qui sera développée dans une partie plus tard) qui permet d'accéder au reste de l'ERP. Une fois connecté, il est possible à l'utilisateur de visualiser la liste des utilisateurs, les informations sur l'utilisateur choisi, de modifier ces informations et même d'ajouter un utilisateur. La majorité des fonctionnalités énoncées au-dessus sont également possibles pour le tableau de prospects (qui fonctionne de la même manière). Par la suite, nous traiterons uniquement la partie sur les membres en raison de sa similitude avec la partie sur les prospects. Un utilisateur peut aussi accéder à la liste des postes et aussi à une partie de gestion des documents. Enfin, un utilisateur peut également se déconnecter et aussi supprimer son compte.

### 3.4.2 Technologies utilisées

Tout le développement s'est fait avec du JavaScript et PrimeVue. C'est une sur-couche de Vue qui permet d'utiliser des Components assez facilement pour créer les différents éléments présents sur un site web, comme des tableaux ou des pop-ups.

Notre choix concernant la librairie à utiliser pour avoir facilement accès à des composants de `Vue` n'était pas arrêté. Aucun d'entre nous n'avait d'expérience particulière avec ce type de langage. Après une longue hésitation entre `Vuetify` et `PrimeVue`, nous nous sommes finalement dirigés vers `PrimeVue` dont la dernière version est sortie très récemment. Le point positif étant que cette version serait maintenue plus longtemps mais avec comme inconvénient que la documentation ne serait pas très étoffée.

De plus, utiliser `VueJS`, nous a permis de créer une application single-page. Les avantages sont de ne pas devoir recharger la page à chaque interaction entre l'utilisateur et l'interface et ainsi avoir des interactions plus rapides. Cela permet de respecter la spécification non fonctionnelle concernant la vitesse de l'application.



## 4 | Fonctionnalités

### 4.1 Authentification

#### 4.1.1 Backend

Pour authentifier l'utilisateur, nous utilisons les JWT (JSON Web Token). Les JWT sont du texte hashé contenant des informations, et qui sont signés, ce qui permet de vérifier après coup l'authenticité de ceux-ci.

Dans notre cas, on a un endpoint `/login` qui prend dans le body un champ `nomUtilisateur` et un champ `motDePasse`. On va ensuite vérifier dans la base de données qu'il y a une correspondance entre le nom d'utilisateur et le mot de passe fourni. Si la combinaison est correcte, l'API va retourner dans sa réponse un JWT contenant le nom d'utilisateur. Si la combinaison est incorrecte, un message s'affiche et avertit l'utilisateur que les informations données sont incorrectes.

L'utilisateur va ensuite fournir ce JWT dans chacune de ses requêtes, ce qui permettra à l'API de vérifier grâce à un middleware que la requête provient bien d'un utilisateur authentifié.

#### 4.1.2 Frontend

La page d'authentification représente la page d'accueil de l'ERP. Il est impossible pour un utilisateur d'aller plus loin sur le site tant qu'il n'est pas authentifié. En effet, la Junior Entreprise doit, selon ses missions, gérer des documents confidentiels. Ainsi, il était évident de mettre une page d'authentification au départ pour gérer les accès et réguler les droits en fonction de son identité et donc de son poste.

De plus, des fonctions telles que se souvenir du mot de passe et redemander un mot de passe en cas d'oubli sont visibles sur la page mais ne sont pas encore implémentées. Néanmoins, sur cette page d'accueil, le menu principal est encore visible mais tendra à disparaître dans les versions futures.

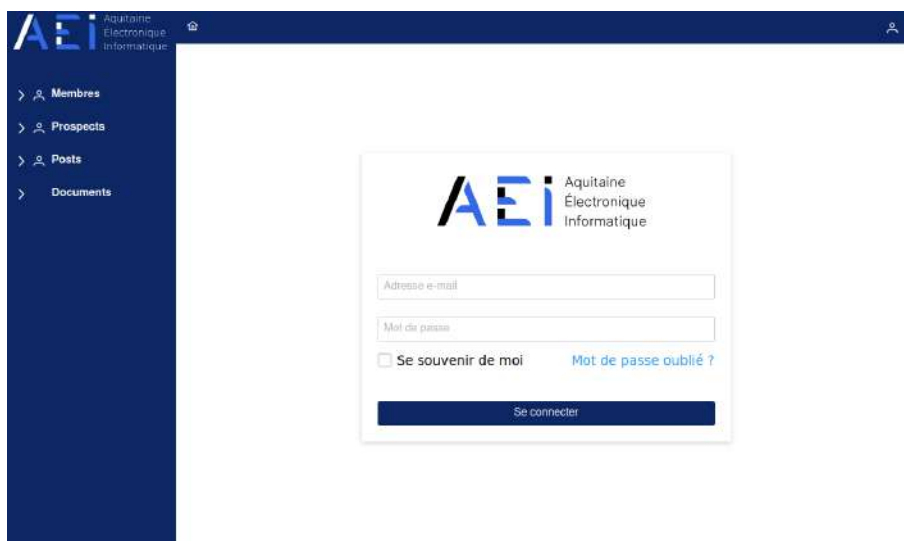


FIGURE 4.1 – Page d'authentification

## 4.2 Permissions

Être authentifié est une bonne chose, mais cela ne signifie pas que l'on doit pouvoir avoir accès à tous les endpoints librement. Pour gérer les droits des utilisateurs, on a donc implémenté un système de permissions.

Ce système fonctionne grâce à une table qui contient 2 colonnes : `nomPermission` et `idUtilisateur`. Le nom de la permission correspond à une action, par exemple `ajouterUtilisateur`, et l'id utilisateur correspond simplement à l'identifiant d'un utilisateur dans la base de données.

Pour vérifier qu'un utilisateur a bien les droits d'effectuer une action, on ajoute un middleware sur la route correspondante. Par exemple, sur la route `POST /adherent` qui permet d'ajouter un adhérent, on ajoute le middleware `verifyPermission("ajouterUtilisateur")`. La fonction `verifyPermission` étant une fonction qui retourne le middleware, paramétré avec le nom de la permission donné en paramètres.

## 4.3 Navigation sur l'ERP grâce à des menus

Deux menus sont présents sur toutes les pages et permettent à l'utilisateur de naviguer sur l'ERP. Grâce au menu à gauche, l'utilisateur peut accéder à tous les onglets traitant d'une tâche centrale pour la Junior-Entreprise, c'est-à-dire : consulter la liste des membres d'AEI, des prospects, des postes et accéder à la partie concernant les documents. Il est également possible pour un utilisateur de savoir où il se trouve dans l'arborescence à partir de ce menu grâce au chemin écrit dans l'entête de la page web.

Le menu à gauche concerne plus un utilisateur que la Junior Entreprise en général. Grâce à ce menu, visible uniquement lorsque l'utilisateur clique sur l'icône d'une personne, un utilisateur peut se déconnecter ou supprimer son compte. Lorsqu'un utilisateur souhaite supprimer son compte, une pop-up apparaît pour demander à l'utilisateur de confirmer son choix. Ainsi, chaque utilisateur est libre de supprimer son compte sans devoir demander directement au ou à la Secrétaire Général.e de supprimer son compte. Pour l'instant, uniquement la pop-up s'affiche et aucune action après la confirmation ne s'effectue.

## 4.4 Gestion des membres

### 4.4.1 Tableau des membres

Dans cette partie et les deux suivantes, nous allons parler uniquement de l'onglet concernant les membres. Néanmoins, elle est grandement similaire à la partie concernant les prospects. Cette partie est également semblable pour le tableau des postes.

Pour le Bureau, le la Secrétaire Général.e doit pouvoir gérer les adhérents d'AEI. C'est à lui de pouvoir ajouter des membres et les modifier. On a tout d'abord créé un tableau pour pouvoir visualiser tous les membres. Ce tableau affiche les informations principales pour chaque membre, c'est-à-dire : son nom, son prénom, son numéro de téléphone, son adresse email, son poste et sa promotion. La dernière colonne permet d'accéder directement à des informations plus détaillées sur un membre et les modifier. Il a également été décidé de rajouter des filtres sur certaines colonnes pour pouvoir retrouver des personnes plus rapidement. Ainsi, uniquement les colonnes nom, prénom et promotion possèdent un filtre. Ce tableau est basé sur le `DataTable` et est représenté par la figure 4.2.

Nom	Prénom	Téléphone	Email	Poste	Promotion
Vovard	Marine	+33634567034	marine.vovard@bordeaux-inp.com	2024	
Je	Manque	+33234567034	d'inspiration@bordeaux-inp.com	2024	
Update	Design	+33684300098	nouvel-essai@gmail.com	2028	

FIGURE 4.2 – Tableau des membres avec une base de données de développement

#### 4.4.2 Visualisation des informations d'un membre

A partir du tableau précédent, il est possible de visualiser toutes les informations concernant un membre en cliquant sur une des lignes. On obtient ensuite des blocs de textes contenant : les informations personnelles, l'adresse, les informations concernant l'école et des informations sur leur naissance concernant un membre. Toutes ces informations sont récupérées dans la base à partir de l'identifiant du membre pour lequel on veut avoir accès aux informations. Il sera également possible pour la personne consultant ces informations, donc pour le/la Secrétaire Général.e de pouvoir supprimer le membre, directement à partir de cette page. De plus, la propriété visant à créer un site responsive est respectée ici, tous les blocs se mettant sur la même colonne lorsque la fenêtre est réduite.

**Informations personnelles**

Nom Vovard

Prénom Marine

Sexe F

Telephone Mobile +33634567034

Email marine.vovard@bordeaux-inp.com

**Adresse**

Adresse 999 rue des cerberes

Complément d'adresse appt. 6

Ville Enfer

Code Postal 66666

pays FRA

FIGURE 4.3 – Information sur un adhérent

#### 4.4.3 Ajout et mise à jour d'un membre

Dans cette partie, on s'intéresse aux deux notions restantes de CRUD, après la lecture et la suppression. Pour l'ajout d'un membre, on peut le faire à partir du tableau. L'utilisateur a juste remplir les champs qui seront plus tard affichés dans la partie précédente. On remarque bien sur les figures précédentes 4.2 et 4.4 que la structure de fond est la même. Mais au lieu d'afficher de l'information, on utilise des balises du type `InputText`, par exemple. La subtilité pour l'ajout des membres et des prospects était de prendre en compte le fait que les adresses, les personnes (et les entreprises pour l'ajout de prospect) ne se trouvent pas tous dans la même table. Ainsi lorsqu'un ajout pour un membre échouait et que ce

FIGURE 4.4 – Ajout d'un adhérent

n'était pas l'adresse qui posait problème, on la supprime pour éviter de la surcharger.

Enfin, certains champs pour l'ajout et la mise à jour possède un masque. (C'est le cas pour les numéros de téléphones, notamment) Cela permet de limiter les erreurs lors de la saisie en forçant un utilisateur à écrire un seul type de données. Tous les champs qui ont pu être précisés, comme les codes postaux, numéros de téléphones et dates ont été fait avec des composants particuliers, pour éviter les erreurs.

On notera également que la mise à jour pour la partie concernant les prospects n'a pas été faite mais fonctionnera de la même manière que ce qui est fait pour la partie membres.

Point important pour l'ajout des prospects, celui-ci est affilié à une entreprise or plusieurs prospects pouvant appartenir à une même entreprise, l'ajout d'un prospect se fait par une page intermédiaire qui ressemble à ce qui est illustré en Figure 4.5.

FIGURE 4.5 – Ajout ou recherche d'une entreprise.

Cela permet d'associer un prospect à une entreprise facilement.

## 4.5 Gestion des documents

Enfin, la dernière partie visible sur l'ERP est la partie de gestion de document.

## 5 | Comparaison avec l'existant

---

De nombreuses fonctionnalités présentes sur la version existante de l'ERP utilisé par la Junior Entreprise AEI ne sont pas implémentées sur notre version de l'ERP, cela résulte, pour la majorité d'entre elles, du fait que ces fonctionnalités ne sont jamais utilisées et sont donc inutiles.

En ce qui concerne la page de connexion, ainsi que le système d'authentification, notre ERP ne présente pas de différences de fonctionnalités avec le précédent.

Concernant la gestion du compte, sur l'ancien ERP il est possible de modifier son mot de passe, ce qui n'est pas possible sur le notre. Cependant, il n'est pas possible de demander la suppression du compte, chose que l'on peut faire sur notre ERP.

La gestion associative s'effectue de la même manière pour les membres de la Junior Entreprise et les informations récoltées sont les mêmes que sur l'ancien ERP. Cependant, il faudrait créer un poste "intervenant" pour pouvoir dissocier les administrateurs des autres membres, ce qui est fait sur le précédent ERP.

Sur l'ancien ERP, il est également possible de lier des documents à un membre, tel que le certificat de scolarité, et on peut aussi ajouter les études auxquelles il a participé, ce n'est pas possible dans notre cas et il serait intéressant d'implémenter ces éléments pour avoir un ERP plus complet.

La gestion des prospects s'effectue elle aussi de la même manière que sur l'ancienne solution et les informations collectées sont les mêmes. Cependant, tout comme dans le cas des membres, dans notre solution, il n'est pas possible de lier des documents et des études à un prospect, ce qu'il est possible de faire avec la solution existante.

La gestion des postes est un élément spécifique à notre version de l'ERP, cela permet d'ajouter et de définir des nouveaux postes au sein de la Junior Entreprise, ainsi que de gérer les postes existants. C'est intéressant car des postes existants peuvent être supprimés dans le futur pendant que de nouveaux postes seront créés.

La gestion des documents est la même entre notre version de l'ERP et celle existante. On peut en effet, uploader/télécharger/supprimer un document avec un tag permettant d'identifier si celui doit être relu ou modifié, ou s'il peut être envoyé à un client. Cependant, ceux-ci ne peuvent pas être liés à un membre/prospect ou à une étude ce qui constitue le plus gros point négatif de notre implémentation.

La principale différence entre la version existante de l'ERP et notre ERP est la gestion des études. En effet, sur l'ERP existant, on peut créer/modifier/supprimer des études. Chaque étude possède une page principale résumant les principales informations de celle-ci (description, chefs de projet, intervenants, prix), les différentes phases sont visibles dans un diagramme de Gantt lorsqu'elles ont été renseignées et une page intitulée "Documents", liste tous les documents relatifs à l'étude et permet de les gérer (ajout/modification/suppression).

Une dernière différence est que les liens vers les différents outils, tel que le Wiki, de la Junior Entreprise sont présents sur la version existante, mais pas sur notre version.

## 6 | Continuité du projet

---

Si une nouvelle équipe souhaite reprendre notre projet, il lui suffit d'installer les différents outils utilisés pour ce projet (Vue.js, Sequelize, sqlite, ...). Puis, en lisant les fichiers README.md du client et du serveur et en s'inspirant des fonctionnalités déjà réalisées, il est assez aisé de comprendre comment prendre en main les différents outils et ainsi pouvoir implémenter de nouvelles fonctionnalités, ou modifier celles existantes. Il est également nécessaire d'ajouter certaines fonctionnalités indispensables pour avoir un ERP complet qui convienne au besoin de la Junior Entreprise.

Liste des fonctionnalités à ajouter :

- Implémenter les études
- Changement de mot de passe sur la page de gestion du compte
- Mise à jour de ses informations personnelles sur la page de gestion du compte
- Lier un document à un membre/prospect
- Lier une étude à un membre/prospect
- Créer un poste "intervenant"
- Ajouter les liens vers les outils de la Junior Entreprise tel que WikiX

L'implémentation des études est indispensable pour avoir un ERP fonctionnel, le reste des fonctionnalités ci-dessus permettent d'améliorer la gestion de la Junior Entreprise. D'autres fonctionnalités pertinentes peuvent également être ajoutées si besoin.

Il serait également intéressant d'afficher des statistiques sur les résultats de la Junior Entreprise en page principale.

# Lexique

**AEI** Aquitaine Électronique Informatique. 10

**ERP** Un ERP est un système informatique intégré utilisé pour gérer les opérations d'une entreprise, telles que la gestion de la production, des finances, des ressources humaines et de la chaîne d'approvisionnement. Il vise à améliorer l'efficacité opérationnelle en offrant une vue d'ensemble de toutes les activités de l'entreprise et en permettant une meilleure coordination entre les différents départements. 10





# **Annexe**

# A | Backend

---

## A.1 Endpoints

### Auth Gestion de l'authentification des utilisateurs

**GET** /login Authentifier l'utilisateur sur le système

### Utilisateur Requêtes liés aux utilisateurs de l'ERP

**GET** /utilisateur Obtenir la liste de tous les utilisateurs

**POST** /utilisateur Créer un nouvel utilisateur

**PUT** /utilisateur Mettre à jour un utilisateur existant

**GET** /utilisateur/:id Obtenir un utilisateur existant par son identifiant

**DELETE** /utilisateur/:id Supprimer un utilisateur par son identifiant

### Adhérent Requêtes liés aux adhérents d'AEI

**GET** /adherent Obtenir la liste de tous les adhérents

**POST** /adherent Créer un nouvel adhérent

**PUT** /adherent Mettre à jour un adhérent existant

**GET** /adherent/:id Obtenir un adhérent existant par son identifiant

**DELETE** /adherent/:id Supprimer un adhérent par son identifiant

### Adresse Requêtes liés aux adresses

**GET** /adresse Obtenir la liste de toutes les adresses

**POST** /adresse Créer une nouvelle adresse

**PUT** /adresse Mettre à jour une adresse existante

**GET** /adresse/:id Obtenir une adresse existante par son identifiant

**DELETE** /adresse/:id Supprimer une adresse par son identifiant

## Client Requêtes liés aux clients d'AEI



GET	/client	Obtenir la liste de tous les clients	▼
POST	/client	Créer un nouveau client	▼
PUT	/client	Mette à jour un client existant	▼
GET	/client/:id	Obtenir un client existant par son identifiant	▼
DELETE	/client/:id	Supprimer un client par son identifiant	▼

## Entreprise Requêtes liés aux entreprises clientes d'AEI



GET	/entreprise	Obtenir la liste de toutes les entreprises	▼
POST	/entreprise	Créer une nouvelle entreprise	▼
PUT	/entreprise	Mette à jour une entreprise existante	▼
GET	/entreprise/:id	Obtenir une entreprise existante par son identifiant	▼
DELETE	/entreprise/:id	Supprimer une entreprise par son identifiant	▼

## Pôle Requêtes liés aux pôles de la Junior Entreprise



GET	/pole	Obtenir tous les pôles existants	▼
POST	/pole	Créer un nouveau pôle	▼
GET	/pole/:nom	Obtenir un pole par son nom	▼
DELETE	/pole/:nom	Supprimer un pôle par son nom	▼

## Poste Requêtes liés aux postes en place dans la JE



GET	/poste	Obtenir la liste de tous les postes	▼
POST	/poste	Créer un nouveau poste	▼
PUT	/poste	Mette à jour un poste existant	▼
GET	/poste/:id	Obtenir un poste existant par son identifiant	▼
DELETE	/poste/:id	Supprimer un poste par son identifiant	▼

Document <span>Requêtes liés aux documents gérés par AEI</span> <span>^</span>		
GET	/document	Obtenir la liste de tous les documents <span>∨</span>
POST	/document	Créer un nouveau document <span>∨</span>
GET	/document/file/:id	Télécharger une version donnée (par son identifiant) d'un document <span>∨</span>
POST	/document/:id/:statut	Uploader une nouvelle version d'un document (:id) avec un nouveau statut (:statut) <span>∨</span>

FIGURE A.0 – Requêtes REST proposés par le backend de l'ERP AEI développé