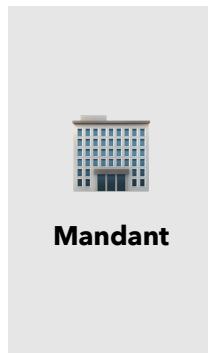


Travail de Bachelor

Étude et mise en place d'une plateforme web
facilitant la gestion de projets entre mandants et
équipes de développeurs indépendants

Non confidentiel



- ➡ Liste de fonctionnalités
- ➡ Délai
- ⬅ Cahier des charges
- ➡ Coût
- ✓ Produit / Rémunération



Étudiant :

Thibaud Alt

Travail proposé par :

Guillaume Wägli
Boulevard de Pérrolles 20
1700 Fribourg

Enseignant responsable :

Patrick Lachaize

Année académique :

2020-2021

Travail de Bachelor 2020-2021

Étude et mise en place d'une plateforme web facilitant la gestion de projets entre mandants et équipes de développeurs indépendants

Résumé publiable

Aujourd'hui, la gestion de projets informatiques respecte la plupart du temps un même processus long et complexe. Les problèmes rencontrés lors de ces projets sont nombreux, les plus problématiques sont les suivants :

- Pour le mandant : Dépassement des coûts, non-respect des délais, diminution des fonctionnalités...
- Pour les développeurs : Cahier des charges, problèmes technologiques, manque de temps...

Au final, le jour du délai fixé aucun livrable n'est prêt, les coûts planifiés sont dépassés et la documentation est faible voire inexistante.

La solution développée se base sur la communication directe entre le mandant et l'équipe de développeurs. Le processus se simplifie alors comme suit :

1. Un mandant met en concours une liste de fonctionnalités et un délai pour la production d'un produit
2. Différentes équipes de développeurs décrivent un cahier des charges comprenant une liste de fonctionnalités réalisables, un choix technologique, une qualité et un coût dans le délai donné
3. Le mandant accepte et signe un des cahiers des charges proposés
4. L'équipe de développeurs implémente le projet pour lequel elle s'est engagée et livre un produit final que le mandant paie

Pour mettre en relation les mandants et les développeurs, faciliter la communication, proposer des devis, gérer les délais et les flux financiers la solution s'apparente à créer et développer une plateforme web.

Au travers de cette plateforme, les équipes de développeurs sont en "compétition sociale" entre elles. Le fait de choisir ses coéquipiers, de définir les technologies, les prix, les choix laissés aux équipes et la dimension sociale motive et permet de lisser la plupart des problèmes. En cas de livraison d'un produit final exemplaire ou au contraire d'un échec (non-respect d'un délai d'un projet, diminution des fonctionnalités, abandon, etc.) les équipes sont notées publiquement. Ainsi la réputation d'une équipe permet à celle-ci de grandir, de décrocher plus de projets et assure la qualité et la réussite des projets.

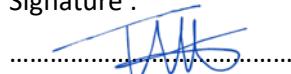
Étudiant :

Alt Thibaud

Date et lieu :

Fribourg, le 06.10.2021

Signature :



Enseignant responsable :

Lachaize Patrick

Date et lieu :

.....

Signature :

.....

Nom de l'entreprise/institution :

Guillaume Wägli

Date et lieu :

.....

Signature :

.....

Préambule

Ce travail de Bachelor (ci-après TB) est réalisé en fin de cursus d'études, en vue de l'obtention du titre de Bachelor of Science HES-SO en Ingénierie.

En tant que travail académique, son contenu, sans préjuger de sa valeur, n'engage ni la responsabilité de l'auteur, ni celles du jury du travail de Bachelor et de l'École.

Toute utilisation, même partielle, de ce TB doit être faite dans le respect du droit d'auteur.

HEIG-VD

Le Chef du Département

Yverdon-les-Bains, le 7 octobre 2021

Authentification

Le soussigné, Thibaud Alt, atteste par la présente avoir réalisé seul ce travail et n'avoir utilisé aucune autre source que celles expressément mentionnées.

Fribourg, le 7 octobre 2021

Thibaud Alt



Remerciements

Je tiens à témoigner toute ma reconnaissance aux personnes suivantes, pour leur précieuse aide dans la réalisation de ce travail de Bachelor :

Monsieur Patrick Lachaize, pour son temps, ses critiques toujours constructives et ses précieux conseils qui ont permis de guider mes réflexions tout au long de la réalisation de ce travail de Bachelor.

Monsieur Fabien Alt, pour avoir pris de son temps pour relire et corriger ce travail de Bachelor.

Ma compagne et ma famille, pour leur soutien moral constant et leurs encouragements.

Mes camarades de classe, pour leur aide et leur accompagnement.

Table des matières

1	<i>Introduction</i>	8
1.1	But du document	8
1.2	Problématique	8
1.3	Objectifs	8
1.4	Valeur ajoutée	9
2	<i>Cahier des charges</i>	11
2.1	Éléments généraux	11
2.2	Éléments d'études	12
2.3	Besoins fonctionnels	13
2.4	Besoins non fonctionnels	14
2.5	Extensions	14
3	<i>Analyse</i>	16
3.1	Étude de marché	16
3.2	User stories	18
3.3	Graphisme et ergonomie	22
3.4	Réception des livrables, validation et paiement	29
3.5	Livraison et intégration continue	30
4	<i>JavaScript</i>	31
4.1	Applications web « <i>State-of-the-Art</i> »	31
4.2	Le JavaScript c'est quoi ?	31
4.3	Environnements d'exécutions	31
4.4	Frameworks <i>front-end</i>	32
4.5	Frameworks <i>back-end</i>	38
4.6	Tests technologiques	42
5	<i>Système de gestion de base de données</i>	55
5.1	Base de données relationnelle	55
5.2	Base de données orientée documents	57
5.3	Base de données orientée graphe	58
6	<i>Planification</i>	62
6.1	Sprint N°1 : Développement de l'API	62
6.2	Sprint N°2 : Développement des interfaces utilisateur	62
6.3	Sprint N°3 : Intégration des interfaces et de l'API	62
6.4	Sprint N°4 : Mise en place des relations et recommandations	63

6.5	Sprint N°5 : Finalisation du projet	63
7	Réalisation.....	64
7.1	API avec <i>Express.js</i>	64
7.2	Front-end avec <i>Vue.js</i>	82
7.3	Recommandations sur <i>Neo4j</i> à l'aide de <i>Cypher</i>	93
7.4	Tests de bout en bout	97
8	Améliorations	98
8.1	Back-end.....	98
8.2	Front-end	98
9	Conclusion	99
10	Annexes.....	107
10.1	Repository <i>Github</i>	107
10.2	Ressources externes.....	107
10.3	Réalisation de requêtes avec le langage <i>Cypher</i>	108
10.4	Historique du développement web	110
10.5	Architectures logicielles	112
10.6	Frameworks	116
10.7	Solutions « <i>stack</i> »	118

1 INTRODUCTION

1.1 But du document

Ce document représente le travail de Bachelor réalisé durant le dernier semestre de formation de la filière Informatique avec orientation « *Systèmes de gestion* » à la Haute École d'Ingénierie et de Gestion du canton de Vaud.

1.2 Problématique

Aujourd'hui, la gestion de projets informatiques respecte la plupart du temps le même processus :

1. Un mandant soumet une liste de fonctionnalités et un délai pour la production d'un produit
2. Un vendeur (et/ou un directeur, un chef de projet) décrit un cahier des charges et propose un devis au mandant
3. Le vendeur négocie avec le mandant la forme du projet
4. Le mandant accepte et signe un devis comprenant un coût, une qualité et un délai
5. Le vendeur impose un cahier des charges et des délais de réalisation à une équipe de développeurs
6. L'équipe de développeurs implémente le projet et crée un livrable
7. Le vendeur livre le produit fini
8. Le mandant paie le projet (et les dépassements de coûts, la baisse de qualité, le non-respect des délais !)



Les problèmes rencontrés lors de ces projets sont nombreux, les principaux et les plus problématiques sont les suivants :

- Pour le **mandant** : Dépassement des coûts, non-respect des délais, diminution des fonctionnalités...
→ *Finalement le mandant se retrouve le jour du délai demandé avec aucun livrable ou un livrable ne contenant qu'une partie des fonctionnalités et avec un coût planifié atteint voire dépassé.*
- Pour les **développeurs** : Cahier des charges, problèmes technologiques, manque de temps...
→ *Finalement les développeurs doivent fournir un livrable bâclé présentant des bugs dus au manque de temps, avec une documentation faible, voire inexistante.*

1.3 Objectifs

La solution envisagée se base sur la communication directe entre le mandant et l'équipe de développeurs. Le processus se simplifie alors comme suit :

1. Un mandant met en concours une liste de fonctionnalités et un délai pour la production d'un produit
2. Différentes équipes de développeurs décrivent un cahier des charges comprenant une liste de fonctionnalités réalisables, un choix technologique, une qualité et un coût dans le délai donné
3. Le mandant accepte et signe un des cahiers des charges proposés
4. L'équipe de développeurs implémente le projet pour lequel elle s'est engagée et livre un produit final que le mandant paie

Pour mettre en relation les mandants et les développeurs, faciliter la communication, proposer des devis, gérer les délais et les flux financiers la solution s'apparente à créer et développer **une plateforme web**.



Au travers de cette plateforme, les équipes de développeurs sont en "compétition sociale" entre elles. Le fait de choisir ses coéquipiers, de définir les technologies, les prix, les choix laissés aux équipes et la dimension sociale motivent et permettent de lisser la plupart des problèmes.

En cas de livraison d'un produit final exemplaire ou au contraire d'un échec (non-respect d'un délai d'un projet, diminution des fonctionnalités, abandon, etc.) les équipes sont notées publiquement. Ainsi la réputation d'une équipe permet à celle-ci de grandir, de décrocher plus de projets et assure la qualité et la réussite des projets.

1.4 Valeur ajoutée

Le but premier de cette plateforme web est d'éliminer un ou plusieurs intermédiaires ceux-ci étant une cause de multiplication des problèmes. Prenons l'exemple du célèbre jeu du *téléphone arabe* ; plus il y a d'acteurs, moins le message initial a de chance d'arriver correctement et non déformé au destinataire final. De plus, chaque être humain à une façon personnelle d'émettre et d'interpréter les informations qu'il reçoit d'autres êtres humains. Dans un projet, ces éléments s'appliquent également ; plus il y a d'acteurs, plus le contenu du projet diverge, plus le temps de celui-ci est allongé et plus les coûts sont élevés.

L'optique de cette plateforme web est d'éliminer au maximum les acteurs intermédiaires et de mettre en valeur les acteurs apportant une réelle plus-value à un projet. Ceci dans le but d'arriver à créer un produit en réduisant les coûts tout en respectant les délais et en visant une qualité requise. Ces acteurs sont, à l'identique du *téléphone arabe* l'*émetteur et le destinataire du message*, le ou les mandants et le ou les développeurs. Pour que ces deux mondes se comprennent et arrivent à collaborer dans le but de créer un produit répondant au besoin, cela demande des efforts de compréhension des deux côtés ainsi qu'une implication forte de chacun dans le projet.

Pour aider la communication entre les différents acteurs tout au long du projet, diriger la gestion de celui-ci et ainsi remplacer et améliorer les rôles des intermédiaires, la plateforme web disposera de différentes valeurs ajoutées. Premièrement, la gestion des projets sera faite en respectant un cadre de travail SCRUM qui permet d'encadrer, de standardiser et ainsi de faciliter la conduite des projets. De plus, la plateforme mettra à disposition de ses clients différents services notamment :

- Un espace de travail incluant un forum permettant un échange direct d'informations, de fichiers, d'images, etc. qui sera accessible et à disposition de tous les acteurs du projet.
 - En plus de la version originale du message et si nécessaire, cet espace inclura un système de traduction des messages dans la langue préférée de l'utilisateur. Permettant ainsi à des acteurs de toutes régions du monde de travailler ensemble facilement.
 - Des livrables avec une gestion des versions seront mis à disposition du mandant tout au long du projet, celui-ci pourra alors vérifier et corriger en cours de réalisation le bon déroulement du projet.
 - Une intelligence artificielle pourra être interpellée en cas d'incompréhension de vocabulaire entre mandant et développeur, celle-ci pourra reformuler certaines phrases, vulgariser certains termes et apporter des éléments supplémentaires nécessaires à une bonne compréhension.
- La mise à disposition sporadique de professionnels externes pour traiter un point spécifique. Par exemple le recours à un médiateur en cas de conflit entre certains acteurs du projet, l'appel à un consultant ou à un expert pour prendre une décision stratégique, le suivi par un coach en cas de baisse de motivation, etc.
- Un système de sous-traitance de tâches simples et répétitives permettant de répondre rapidement à un besoin ponctuel. Par exemple la saisie de centaines d'articles dans un magasin de vente en ligne, le nettoyage d'une base de données, la recherche d'images d'illustrations, etc.

- Un *espace de connaissances* mettant à disposition des ressources, des cours, des tutoriels, des outils, etc. apportant des notions théoriques ainsi que différents savoirs nécessaires liés à la gestion de projets.
- Une équipe d'assistance disponible en tout temps pour répondre aux différentes questions liées à la plateforme et ainsi aider ses clients à tout moment.

Dans ce travail de Bachelor, nous nous focalisons sur des produits informatiques pour développer l'idée, mais ce concept pourrait très bien s'adapter à tout type de projets et d'industries. Nous pourrions par exemple imaginer une entreprise de construction avec comme projet l'élaboration d'un bâtiment même si dans ce cas, il y aurait certainement plus que deux types d'acteurs.

2 CAHIER DES CHARGES

2.1 Éléments généraux

2.1.1 Objectifs du travail de diplôme

Les objectifs de ce projet pour ce travail de Bachelor sont les suivants :

- Réaliser une étude de marché sommaire
 - Travail de recherche de "ce qui se fait" actuellement et des éventuels produits existants concurrents
- Analyser via un "*State of the art*" les différentes techniques permettant le développement d'applications web en 2021 dans le but d'une sélection pour la réalisation
- Définir la structure et la technologie de la ou des base(s) de données à utiliser
- Développer une première version de l'application web client-serveur
- Proposer des améliorations et/ou d'autres fonctionnalités à développer dans des versions postérieures de l'application web

2.1.2 Périmètre

Dans sa première version, il est attendu au minimum de l'application web les fonctionnalités et point de conceptions suivants :

- La plateforme permettra à un utilisateur de se créer un compte, de gérer son profil et de rejoindre une ou plusieurs équipes de développeurs
- Un mandant pourra se créer un compte, gérer son profil, soumettre un ou plusieurs projets, choisir une équipe de développement et attribuer une évaluation à une équipe de développeurs lorsqu'un projet sera finalisé
- Une équipe de développeur (via un *team leader*) pourra gérer son profil, soumettre sa candidature pour des projets proposés et déposer des livrables pour ses projets en cours
- L'application web sera monolingue et sera proposée en anglais
- L'application web disposera d'une interface fonctionnelle sur les navigateurs web récents et sur une taille d'écran d'ordinateurs classiques

2.1.3 Planning

Pour ce projet, deux possibilités de rendus sont possibles :

1. La première nécessitant un taux de travail à 100% consiste à un rendu intermédiaire à la mi-juillet et un rendu final à la fin août.
2. La première nécessite un taux de travail d'uniquement 60% et consiste à un rendu intermédiaire à la fin juillet et un rendu final à la fin septembre.

Ayant des obligations professionnelles et ne pouvant pas réduire mon taux de travail pour les mois de juillet à septembre, j'ai opté pour la seconde option proposée. De ce fait, le planning suivant en découle.

Date	Échéance
Vendredi 21 mai 2021	<i>Rendu du cahier des charges</i>
Mardi 13 juillet 2021	<i>Rendu intermédiaire incluant le rapport intermédiaire</i>
Jeudi 7 octobre 2021	<i>Rendu final incluant le rapport final et l'application fonctionnelle</i>
Du 25 octobre au 5 novembre	<i>Soutenance du travail de Bachelor</i>

Le planning détaillé des tâches ainsi que le suivi de celles-ci est réalisé dans le document « *TB_Planning_Alt-Thibaud.xlsx* » disponible sous forme d'annexe.

2.2 Éléments d'études

2.2.1 Étude de marché sommaire

Un travail de recherche de "ce qui se fait" actuellement sera réalisé et une étude de marché sommaire présentera les éventuels produits existants concurrents. Cette étude pourra être composée d'une matrice d'affaires (*Business Model Canvas*), des différentes cibles visées par la plateforme web, du marché potentiel et du profil des clients, du secteur d'activité ou encore des éventuels risques et menaces.

2.2.2 Technologies

Le but est de réaliser une application web client-serveur entièrement en JavaScript à l'aide de Node.js et de frameworks comme Express.js, React.js, Vue.js ou équivalent. Pour réaliser cette plateforme, l'utilisation des technologies web récentes et actuelles semble cohérente ; ces choix techniques devront être vérifiés et validés dans une phase d'analyse.

2.2.2.1 State of the art

Un état de l'art des techniques permettant le développement d'applications web en 2021 sera réalisé. Celui-ci s'intéressera plus particulièrement aux technologies JavaScript choisies pour réaliser ce projet. Cet état de l'art étudiera les deux axes de développement nécessaires, à savoir le *front-end* avec des frameworks JavaScript et le *back-end* avec les environnements d'exécution et les frameworks.

2.2.2.2 Persistance des données

Le choix de la technologie de la ou des bases de données à utiliser devra être étudié. Pour stocker les informations des utilisateurs, les informations spécifiques aux projets, les évaluations, etc. une unique base de données SQL semble adéquate. Cependant ce choix devra être confirmé et validé durant la phase d'analyse. Une application monolithique semble adaptée et plus facile à mettre en place dans un premier temps, cependant l'appel et l'utilisation d'éventuels microservices pourra être envisagé.

2.2.2.3 Gestion de dépendances

La gestion de dépendances sera réalisée avec *npm* qui est le gestionnaire de paquets officiels de Node.js si ce dernier est choisi lors de la phase d'analyse pour y développer la plateforme. Ce gestionnaire de paquets est très pratique, car il fonctionne avec un simple terminal, gère les dépendances par application et permet d'installer très facilement des paquets Node.js disponibles sur le dépôt npm. En outre, toutes les informations nécessaires au développement et au déploiement sont écrites en clair dans un fichier JSON ce qui permet de gérer les dépendances de librairies tierces et d'automatiser leur téléchargement.

2.2.2.4 Livraison et intégration continue (CI / CD)

L'environnement d'intégration continue et de déploiement continu utilisé sera git à l'aide de la plateforme web *github.com* et du logiciel *Github Desktop*. Cet environnement et ses outils associés mettent à disposition un système de gestion des versions complet, ainsi qu'un puissant système de tests et de déploiement.

2.2.2.5 Livrables

Désirant réaliser le développement de l'application web avec la méthodologie SCRUM, celle-ci prévoit de générer autant de livrables que de *sprints* agendés. Une fois les différents *sprints* définis, chaque livrable sera clairement identifié sur le système de gestion des versions. Celui-ci pourra alors éventuellement être déployé, hébergé et soumis au client.

2.2.2.6 Hébergement

Les différents livrables pourront être déployés au fur et à mesure de son développement sur une plateforme cloud tel que AWS (*Amazon Web Services*), Heroku ou encore Netlify. Les principaux avantages de ces plateformes cloud sont qu'elles permettent un déploiement extrêmement rapide qui peut être automatisé avec plusieurs outils de livraison continue ; qu'elles ne nécessitent pas de configurations complexes et qu'elles sont gratuites dans une certaine mesure.

2.3 Besoins fonctionnels

2.3.1 User stories

Dans les user stories suivantes, nous prendrons cinq points de vue différents à savoir :

- Un utilisateur (il s'agit ici d'un développeur)
- Une équipe de développeurs (il s'agit ici de plusieurs développeurs)
- Un mandant
- Un modérateur
- Un intervenant externe (il peut s'agir d'un expert, d'un médiateur, d'un coach, d'un assistant, etc.)

Epic 1	Création d'un compte et authentification En tant que développeur ou en tant que mandant, je veux pouvoir me créer facilement un compte utilisateur puis l'utiliser par la suite pour m'authentifier.
Epic 2	Gestion de profil En tant que développeur ou en tant que mandant, je veux pouvoir gérer mon profil. En tant que développeur je peux rejoindre ou quitter une ou plusieurs équipes de développeurs.
Epic 3	Soumission de projets En tant que mandant, je veux pouvoir soumettre des projets et choisir une équipe de développement pour les réaliser. Pour ce faire, je peux consulter le profil de l'équipe de développeurs ainsi que les profils des différents membres de celle-ci.
Epic 4	Évaluation d'une équipe de développeurs En tant que mandant, je veux pouvoir évaluer une équipe de développeurs une fois un projet finalisé et livré ou abandonné. Mon évaluation est alors visible et consultable par tous les autres mandataires.
Epic 5	Soumission de candidature En tant qu'équipe de développeurs, je veux pouvoir soumettre ma candidature, mon cahier des charges et mes coûts pour un projet proposé.
Epic 6	Dépôt des livrables En tant qu'équipe de développeurs, je veux pouvoir téléverser des documents, des livrables et des informations tout au long du projet. Ceux-ci sont alors visibles pour tous les membres de l'équipe ainsi que pour le mandant.
Epic 7	Classement des équipes et des développeurs En tant que visiteur, je peux consulter le classement mensuel, annuel et « <i>de tous les temps</i> » des équipes de développeurs. En tant que visiteur, je peux également consulter le classement d'un développeur.
Epic 8	Espace de travail En tant qu'utilisateur de la plateforme, je peux accéder à mon espace de travail incluant un forum par projet en cours. Ces forums me permettent d'accéder à toutes les informations nécessaires au bon déroulement des projets.
Epic 9	Discussions instantanées En tant qu'utilisateur de la plateforme, je peux accéder via mon espace de travail à un système de discussions instantanées.
Epic 10	Gestion des projets et des utilisateurs En tant que modérateur, j'ai des droits de gestion sur des projets et des équipes qui me sont associées. Je peux valider la publication ou non d'un projet, bannir un utilisateur ou une équipe, etc.
Epic 11	Gestion de l'espace de connaissances En tant que modérateur, je peux analyser les ressources publiques et créer ou mettre à disposition de la communauté du matériel théorique comme : des cours, des tutoriels, des outils, des articles, etc.
Epic 12	Intervention externe En tant qu'intervenant externe je dois, lorsqu'on me sollicite, pouvoir intervenir et apporter mon expertise sur un projet ou sur une situation.

2.4 Besoins non fonctionnels

2.4.1 Contraintes dues à l'environnement

Ce projet étant nouveau et non lié à un environnement précis, il ne dispose pas de contraintes techniques définies. Il devra cependant pouvoir s'inscrire dans un portefeuille de projets web existants et de ce fait devra suivre les *bonnes pratiques* de développement actuelles.

Plus tard, il se pourrait que d'autres développeurs soient amenés à faire évoluer ce projet, c'est pourquoi celui-ci devra être correctement documenté et devra être développé avec des frameworks connus et maîtrisé par un grand nombre de développeurs.

2.4.2 Besoins de performance, d'ergonomie et de fiabilité

2.4.2.1 Interface et expérience utilisateur

L'interface utilisateur devra respecter une charte graphique et des maquettes définies. L'application web devra être intuitive ; l'expérience utilisateur devra être fluide et l'ergonomie agréable.

2.4.2.2 Charges et ressources

L'application ne devra pas, du moins dans sa première version, supporter un taux de charge excessif. Toutefois, elle devra être pensée et développée de telle sorte à pouvoir l'adapter à ces points dans des versions postérieures.

2.5 Extensions

2.5.1 « *Si temps le permet* »

2.5.1.1 Inscription et connexion via des services tiers

Aujourd'hui nous possédons tous de nombreux comptes sur internet et il ne nous est pas toujours facile de nous souvenir quelles combinaisons nom d'utilisateur/mot de passe nous avons définis. De plus, les inscriptions à un service web sont souvent des étapes lentes et contraignantes qui vont à l'encontre de l'expérience utilisateur. Partant de ce constat, il serait judicieux d'ajouter des services tiers (Google, Apple...) comme moyen de connexion à la plateforme web.

2.5.1.2 Multilinguisme

L'application étant dans sa première version uniquement disponible en anglais, il serait judicieux de la traduire dans d'autres langues permettant ainsi d'attaquer différents marchés. Dans un premier temps, et pour le marché suisse, l'application pourra être traduite en Allemand et en Français.

2.5.1.3 Adaptation « *responsive* »

L'application web étant fonctionnelle sur un ordinateur de bureau classique, il serait fort agréable pour l'utilisateur d'également disposer d'une interface sur ses appareils mobiles. Cette interface, éventuellement réduite, devra donc être agréablement utilisable et fonctionnelle sur des téléphones mobiles récents et sur des tablettes récentes.

2.5.2 Dans des versions futures

2.5.2.1 Solutions de paiement

Dans une version future, il serait intéressant d'étudier puis d'implémenter différentes solutions de paiement à la plateforme web.

2.5.2.2 Intégration de professionnels externes

Une des valeurs ajoutées de la plateforme est la possibilité de faire appel à un professionnel externe pour traiter un point spécifique du projet. Par exemple le recours à un médiateur en cas de conflit entre certains acteurs du projet, l'appel à un consultant ou à un expert pour prendre une décision stratégique, le suivi par un coach en cas de baisse de motivation, etc. Ces acteurs externes seraient des partenaires indépendants, validés par un système de sélection et leurs services seraient proposés aux mandants et/ou aux développeurs au travers de la plateforme.

2.5.2.3 Système de sous-traitance

Un système de sous-traitance de tâches simples et répétitives permettant de répondre rapidement à un besoin ponctuel pourra être intégré directement à la plateforme. Celui-ci pourra être utilisé par exemple pour la saisie d'articles dans un magasin de vente en ligne, le nettoyage d'une base de données, la recherche d'images d'illustrations, etc. L'idée ici serait de trouver différents partenaires effectuant ce type de services et de les intégrer entièrement à la plateforme. De ce fait l'utilisation de ce système par les développeurs serait très simple et ils n'auraient pas à quitter la plateforme.

2.5.2.4 Espace de connaissances

Un espace de connaissances mettant à disposition des ressources, des cours, des tutoriels, des outils, etc. et apportant des notions théoriques ainsi que différents savoirs nécessaires liés à la gestion de projets pourra être intégré. Cet espace se construirait au fur et à mesure des projets via les connaissances acquises par les différents acteurs et sur le principe de « *l'open source* ». De plus, il pourrait être envisageable de mandater un expert en gestion de projets qui réaliserait une série de tutoriels vidéo vulgarisées, accessibles à tous et intégrant les différentes spécificités de la plateforme.

2.5.2.5 Vulgarisation et reformulation

Dans une version future, une intelligence artificielle pourra être interpellée en cas d'incompréhension de vocabulaire entre mandants et développeur lors de discussions textuelles instantanées. Cette intelligence artificielle pourra reformuler certaines phrases, vulgariser certains termes et apporter des éléments supplémentaires nécessaires à la bonne compréhension du message.

3 ANALYSE

3.1 Étude de marché

Le but de ce chapitre n'est pas de faire une étude de marché complète, mais plutôt de définir le marché visé, d'identifier la demande potentielle ainsi que de faire un tour d'horizon des produits similaires existants. Il sera alors possible de créer un résumé à l'aide d'une matrice d'affaires représentant les partenaires, les ressources, les activités, les coûts, les valeurs et tout ce qui s'apparente à la relation client.

3.1.1 Définition de la zone géographique et de la taille du marché

La zone géographique visée est centrée, dans un premier temps, au niveau des pays européens et plus particulièrement de la Suisse. Plus tard, il serait intéressant de réaliser une étude concernant d'autres pays avec des fonctionnements managériaux différents, par exemple le marché Asiatique.

En 2020, l'association *Swico*¹ estimait un chiffre d'affaires de 33,8 milliards de francs pour le marché suisse des technologies de l'information et de la communication. Cependant, il est difficile d'estimer la taille du marché Suisse visé, tant celui-ci peut avoir un spectre très large. En effet, cette application mettant en relation un mandant et des développeurs, permet de cibler de multiples domaines. De fait, tout domaine ayant besoin d'utiliser des ressources informatiques, de quelque manière que ce soit, est susceptible d'être intéressé par ce type de plateforme.

3.1.2 La demande

Ce projet s'adresse principalement à deux types de profils distincts puisqu'il a pour but de les mettre en relation :

1. Les PME et grandes entreprises

Les clients pouvant proposer des projets sur la plateforme peuvent être des PME ou de grandes entreprises voulant développer un projet spécifique dont la liste de fonctionnalités et les délais sont définis précisément. Au lieu d'engager spécifiquement du personnel pour ce projet ou de mandater une entreprise externe, ces entreprises peuvent proposer leur projet sur la plateforme.

2. Les développeurs informatiques

De l'autre côté, le projet a besoin de développeurs informatiques, indépendants ou employés, désirant développer un ou plusieurs projets en équipe. Ces développeurs doivent maîtriser une ou plusieurs technologies de développements, être autonomes et savoir travailler en équipe.

3.1.3 L'offre

3.1.3.1 Concurrence directe

De nombreuses plateformes de mise en relation entre entreprise et travailleurs indépendants existent et leur principe de fonctionnements est souvent identique : ils permettent la mise en relation entre des entreprises qui cherchent des indépendants, et des indépendants qui cherchent des missions.

Deux principaux modes de mise en relation se distinguent, soit l'indépendant répond à des descriptifs de missions proposées, soit l'entreprise pioche parmi des profils et/ou des listes de services d'indépendants.

D'un point de vue des coûts, les 3 modes de tarifications principaux que j'ai pu identifier sont les suivants :

- *Gratuit* : l'indépendant ne paie rien. Il peut s'inscrire et rechercher des missions gratuitement.
- *Un coût mensuel* : le développeur indépendant paie un forfait par mois pour pouvoir chercher des missions.
- *Un pourcentage du chiffre d'affaires* : l'indépendant paie un pourcentage des factures des missions qu'il a obtenu via la plateforme

Pour illustrer l'offre disponible, j'ai choisi d'analyser plus en détail deux acteurs du marché, *codeur.com* et *Freelance.com*.

¹ <https://www.swico.ch/fr/>

Codeur.com

« *Codeur.com est la première place de marché des développeurs indépendants en France. Avec plus de 60'000 développeurs inscrits, vous êtes certains de trouver le développeur freelance idéal pour votre projet.* »

Ce service propose de trouver un développeur free-lance en trois étapes qu'ils qualifient de simples, rapides et efficaces. Celles-ci consistent à décrire un projet en quelques mots et le publier sur la plateforme, à recevoir une dizaine de devis en quelques minutes puis à choisir sans obligation le free-lance idéal. Il met en avant la création de sites internet, le développement d'applications mobiles, la refonte de sites e-commerce ou encore l'optimisation de référencement (SEO). D'après leurs chiffres de septembre 2021, Codeur.com dit avoir 251'080 développeurs free-lance inscrits et posséder un portefeuille de 189'613 clients. De plus, ils prétendent pouvoir proposer un devis dans les 5 minutes suivant la publication d'un projet.

Site web : www.codeur.com

Freelance.com

« *Avec notre communauté de 370'000 consultants et experts indépendants, et notre réseau de 1'000 PME et start-up, nous connectons les entreprises avec les meilleurs experts.* »

Ce site offre la possibilité aux free-lance de trouver des missions grâce à une plateforme, mais également de leur proposer une assurance professionnelle et au besoin une avance de trésorerie. En parallèle, elle propose aux PME et aux grandes entreprises de trouver un expert indépendant via une sélection de leurs équipes, mais offre également la possibilité de tester un collaborateur dans le but de le salariser. De plus, ces entreprises peuvent avoir accès à des services d'externalisation, ainsi qu'à des forums et des ressources utiles d'ordre administratives, d'assurances, de gestion du personnel, de management des risques, etc.

Site web : www.freelance.com

3.1.3.2 Concurrence indirecte

Nous retrouvons comme type de concurrences indirectes de multiples entreprises spécialisées dans la réalisation de projets informatiques et notamment des agences web offrant des services de développement similaires à la plateforme. Il s'agirait ici de l'offre de conduites de projets que l'on pourrait qualifier de classique et qui reste encore la plus répandue aujourd'hui.

3.1.4 Business Model Canvas

Après avoir défini les différents points décrivant la proposition de valeur, l'infrastructure, les clients et les finances, etc., je les ai intégrés dans un « *Business Model Canvas* ». Ce modèle de gestion stratégique, proposé en 2005 par *Alexander Osterwalder*, est souvent utilisé pour développer de nouveaux modèles commerciaux.

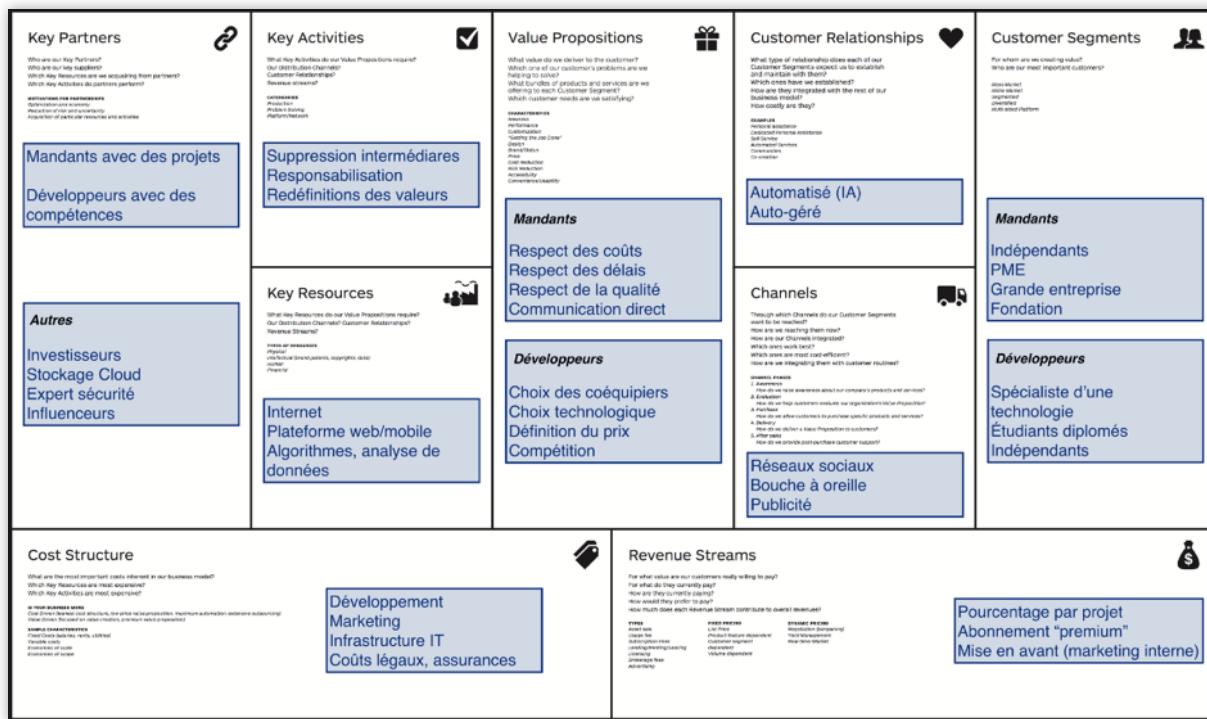


Figure 1 : Idée de « Business Model Canvas » pouvant servir de base pour la plateforme web à réaliser

3.2 User stories

Une « user story » est une explication non formelle, générale d'une fonctionnalité logicielle écrite du point de vue de l'utilisateur final. Son but est d'expliquer comment une fonctionnalité logicielle apportera de la valeur au client.

Dans les user stories suivantes, nous prendrons cinq points de vue différents à savoir :

- Un utilisateur (il s'agit ici d'un développeur)
- Une équipe de développeurs (il s'agit ici de plusieurs développeurs)
- Un mandant
- Un modérateur
- Un intervenant externe (il peut s'agir d'un expert, d'un médiateur, d'un coach, d'un assistant, etc.)

Les degrés de priorisation vont de 1 à 3 ; 1 étant la priorité la plus importante et 3 la priorité la moins importante. Les niveaux de complexités s'étendent de 1 à 5 ; 1 étant une tâche facile et 5 une tâche complexe.

3.2.1 Crédit d'un compte et authentification

Identifiant	Epic 1
Titre	Création d'un compte et authentification
Description	En tant que développeur ou en tant que mandant, je veux pouvoir me créer facilement un compte utilisateur puis l'utiliser par la suite pour m'authentifier.
Tâches associées	<ol style="list-style-type: none"> 1. L'utilisateur doit pouvoir se créer un compte avec au minimum un nom, une adresse email et un mot de passe. 2. L'utilisateur doit pouvoir se connecter et accéder à son compte. 3. L'utilisateur doit pouvoir récupérer son mot de passe en cas d'oubli de celui-ci.
Priorisation	3 / 3
Complexité	3 / 5
Auteur	Thibaud Alt

3.2.2 Gestion de profil

<i>Identifiant</i>	Epic 2
<i>Titre</i>	Gestion de profil
<i>Description</i>	En tant que développeur ou en tant que mandant, je veux pouvoir gérer mon profil. En tant que développeur, je peux rejoindre ou quitter une ou plusieurs équipes de développeurs.
<i>Tâches associées</i>	<ol style="list-style-type: none"> 1. L'utilisateur doit pouvoir ajouter/modifier/supprimer ses données de contacts (nom, prénom, adresse, etc.). 2. L'utilisateur doit pouvoir modifier ses informations de connexions (email et mot de passe). 3. L'utilisateur doit pouvoir ajouter/modifier/supprimer ses informations professionnelles (CV, langages connus, etc.). 4. L'utilisateur doit pouvoir rejoindre et quitter une équipe de développeurs. 5. L'utilisateur doit pouvoir créer une équipe de développeurs. 6. L'utilisateur doit pouvoir gérer les informations de contacts d'une équipe de développeurs (adresse mail, informations bancaires, etc.). 7. L'utilisateur doit pouvoir gérer les membres de son équipe de développeurs. 8. L'utilisateur doit pouvoir transmettre la gestion et la responsabilité d'une équipe de développeurs à un utilisateur tiers.
<i>Priorisation</i>	3 / 3
<i>Complexité</i>	2 / 5
<i>Auteur</i>	Thibaud Alt

3.2.3 Soumission de projets

<i>Identifiant</i>	Epic 3
<i>Titre</i>	Soumission de projets
<i>Description</i>	En tant que mandant, je veux pouvoir soumettre des projets et choisir une équipe de développement pour les réaliser. Pour ce faire, je peux consulter le profil de l'équipe de développeurs ainsi que les profils des différents membres de celle-ci.
<i>Tâches associées</i>	<ol style="list-style-type: none"> 1. Le mandant doit pouvoir publier un projet à réaliser. 2. Le mandant doit pouvoir modifier les informations liées à un nouveau projet (listes de fonctionnalités, délais). 3. Le mandant doit pouvoir choisir une équipe de développeurs pour réaliser le projet. 4. Le mandant doit pouvoir consulter les profils des équipes de développeurs. 5. Le mandant doit pouvoir consulter les profils des développeurs.
<i>Priorisation</i>	2 / 3
<i>Complexité</i>	2 / 5
<i>Auteur</i>	Thibaud Alt

3.2.4 Évaluation d'une équipe de développeurs

<i>Identifiant</i>	Epic 4
<i>Titre</i>	Évaluation d'une équipe de développeurs
<i>Description</i>	En tant que mandant, je veux pouvoir évaluer une équipe de développeurs, une fois un projet finalisé et livré ou abandonné. Mon évaluation est alors visible et consultable par tous les autres mandataires.
<i>Tâches associées</i>	<ol style="list-style-type: none"> 1. Le mandant doit pouvoir attribuer une note à une équipe de développeurs. 2. Le mandant doit pouvoir écrire un rapport (feed-back) sur le déroulement d'un projet. 3. Les notes et les rapports obtenus par une équipe de développeur doivent s'afficher sur le profil de celle-ci.

Priorisation	2 / 3
Complexité	3 / 5
Auteur	Thibaud Alt

3.2.5 Soumission de candidature

Identifiant	Epic 5
Titre	Soumission de candidature
Description	En tant qu'équipe de développeurs, je veux pouvoir soumettre ma candidature, mon cahier des charges et mes coûts pour un projet proposé.
Tâches associées	<ol style="list-style-type: none"> 1. Tous les développeurs doivent pouvoir consulter la liste des projets ouverts à la réalisation. 2. Une équipe de développeurs (via son responsable) doit pouvoir soumettre sa candidature pour un projet proposé. 3. Une équipe de développeurs doit être notifiée en cas de sélection de leur soumission pour un projet.
Priorisation	2 / 3
Complexité	2 / 5
Auteur	Thibaud Alt

3.2.6 Dépôt des livrables

Identifiant	Epic 6
Titre	Dépôt des livrables
Description	En tant qu'équipe de développeurs, je veux pouvoir téléverser des documents, des livrables et des informations tout au long du projet. Ceux-ci sont alors visibles pour tous les membres de l'équipe ainsi que pour le mandant.
Tâches associées	<ol style="list-style-type: none"> 1. Une équipe de développeurs (via son responsable) doit pouvoir téléverser des documents électroniques de différents formats (« .pdf », « .docx », « .xlsx », etc.). 2. Une équipe de développeurs (via son responsable) doit pouvoir téléverser des archives « .zip » de livrables du projet. 3. Le mandant doit être notifié lorsqu'un téléchargement est effectué. 4. Le mandant doit pouvoir consulter tous les documents et livrables liés au projet.
Priorisation	3 / 3
Complexité	4 / 5
Auteur	Thibaud Alt

3.2.7 Classement des équipes et des développeurs

Identifiant	Epic 7
Titre	Classement des équipes et des développeurs
Description	En tant que visiteur, je peux consulter le classement mensuel, annuel et « <i>de tous les temps</i> » des équipes de développeurs. En tant que visiteur, je peux également consulter le classement d'un développeur.
Tâches associées	<ol style="list-style-type: none"> 1. Un visiteur doit pouvoir consulter une page de classement présentant les équipes de développeurs et comportant des options de tris et de filtres. 2. Un visiteur doit pouvoir consulter le classement d'un développeur via sa page de profil.
Priorisation	1 / 3
Complexité	4 / 5

Auteur

Thibaud Alt

3.2.8 Espace de travail

<i>Identifiant</i>	Epic 8
<i>Titre</i>	Espace de travail
<i>Description</i>	En tant qu'utilisateur de la plateforme, je peux accéder à mon espace de travail incluant un forum par projet en cours. Ces forums me permettent d'accéder à toutes les informations nécessaires au bon déroulement des projets.
<i>Tâches associées</i>	<ol style="list-style-type: none"> 1. Un acteur du projet doit pouvoir ajouter différents types de fichiers (texte, images, etc.) 2. Un acteur du projet doit pouvoir consulter toutes les ressources liées à un projet. 3. Un acteur du projet doit pouvoir transmettre une information à tous les participants d'un projet. 4. Le responsable d'une équipe de développeurs doit pouvoir mettre à jour régulièrement un livrable du code du projet, ce livrable inclut une chronologie des versions. 5. Le mandant doit pouvoir télécharger en tout temps les différents livrables.
<i>Priorisation</i>	2 / 3
<i>Complexité</i>	3 / 5
<i>Auteur</i>	Thibaud Alt

3.2.9 Discussions instantanées

<i>Identifiant</i>	Epic 9
<i>Titre</i>	Discussions instantanées
<i>Description</i>	En tant qu'utilisateur de la plateforme, je peux accéder via mon espace de travail à un système de discussions instantanées.
<i>Tâches associées</i>	<ol style="list-style-type: none"> 1. Un acteur du projet doit pouvoir discuter, sous forme textuelle, avec un membre du projet. 2. Un acteur du projet doit pouvoir discuter, sous forme textuelle, avec un groupe de membres du projet. 3. Si nécessaire, un utilisateur peut activer un système de traduction instantané des messages dans sa langue préférée.
<i>Priorisation</i>	1 / 3
<i>Complexité</i>	4 / 5
<i>Auteur</i>	Thibaud Alt

3.2.10 Gestion des projets et des utilisateurs

<i>Identifiant</i>	Epic 10
<i>Titre</i>	Gestion des projets et des utilisateurs
<i>Description</i>	En tant que modérateur, j'ai des droits de gestion sur des projets et des équipes qui me sont associées. Je peux valider la publication ou non d'un projet, bannir un utilisateur, etc.
<i>Tâches associées</i>	<ol style="list-style-type: none"> 1. Un modérateur doit pouvoir être notifié lorsqu'un nouveau projet est mis en ligne par un mandant. 2. Un modérateur doit pouvoir analyser les données d'un nouveau projet mis en ligne 3. Un modérateur doit pouvoir valider ou refuser la publication d'un nouveau projet. En cas de refus, il doit pouvoir le justifier via un rapport. 4. Un modérateur doit pouvoir bannir un utilisateur et le justifier 5. Un modérateur doit pouvoir bannir une équipe et le justifier
<i>Priorisation</i>	1 / 3

Complexité	3 / 5
Auteur	Thibaud Alt

3.2.11 Gestion de l'espace de connaissances

Identifiant	Epic 11
Titre	Gestion de l'espace de connaissances
Description	En tant que modérateur, je peux analyser les ressources publiques et créer ou mettre à disposition de la communauté du matériel théorique comme : des cours, des tutoriels, des outils, des articles, etc.
Tâches associées	<ol style="list-style-type: none"> 1. En tant que modérateur de l'espace de connaissances, j'ai accès aux ressources publiques de tous les projets. Je peux les filtrer, les trier et les catégoriser. 2. En tant que modérateur de l'espace de connaissances, je peux ajouter une nouvelle ressource et la publier. 3. En tant que modérateur de l'espace de connaissances, je peux modifier ou supprimer une ressource que j'ai publiée.
Priorisation	1 / 3
Complexité	2 / 5
Auteur	Thibaud Alt

3.2.12 Intervention externe

Identifiant	Epic 12
Titre	Intervention externe
Description	En tant qu'intervenant externe je dois, lorsqu'on me sollicite, pouvoir intervenir et apporter mon expertise sur un projet ou sur une situation.
Tâches associées	<ol style="list-style-type: none"> 1. Tout utilisateur doit pouvoir solliciter l'aider d'un intervenant externe 2. Un intervenant externe doit pouvoir discuter, sous forme textuelle, avec un membre du projet. 3. Un intervenant externe doit pouvoir discuter, sous forme textuelle, avec un groupe de membres du projet. 4. Un intervenant externe doit pouvoir accéder aux ressources du projet 5. Un intervenant externe doit pouvoir ajouter des ressources au projet (rapport d'expertise, PV de séance, etc.) 6. Un intervenant externe doit pouvoir notifier un modérateur en cas de conflit qui sort de son domaine de compétences
Priorisation	1 / 3
Complexité	2 / 5
Auteur	Thibaud Alt

3.3 Graphisme et ergonomie

3.3.1 Charte graphique

L'application web à réaliser se doit d'être attrayante, moderne, intuitive, fluide et agréable à utiliser. Elle devra s'inspirer des caractéristiques des types de design « *flat patterns* » et « minimalistes ».

- Dispositions en grille
- Palette de couleurs limitée ou monochrome
- Fonctionnalités d'un élément graphique restreint à une seule fonction

- Utilisation de grandes images ou de vidéos d'arrière-plan
- Aplats de couleurs et iconographie en images vectorielles simple
- Utilisation et maximisation de « *l'espace négatif* »

Le « *flat patterns* » ou « *flat design* » est un style graphique et un sous-genre du courant minimaliste. Il se caractérise par des formes simples, sans textures ni effets de volumes. Des couleurs vives sont souvent utilisées en aplats. Des icônes sont dessinées sous forme synthétique. Ainsi que des polices de caractères qui sont souvent originales et de taille importante dans les titrages.

Les sources d'inspirations pourraient être notamment le système d'exploitation *macOS*² à partir de sa version 10.10 et son dérivé *iOS*³ depuis la version 7. L'interface graphique *d'Android* à partir de la version 5 et son « *Material Design*⁴ ».

3.3.1.1 Couleurs

Deux couleurs principales ont été définies, il s'agit d'une teinte de bleu profond pour la couleur principale et d'une teinte de turquoise pour la couleur secondaire. Pour ce faire, j'ai utilisé les teintes *Pantone*⁵ principalement utilisées dans l'imprimerie qui sont normalisées et référencées.



Figure 2 : Couleur primaire (PANTONE 534 CP) et couleur secondaire (PANTONE 2398 CP),
<https://www.pantone.com/eu/fr/color-finder/534-CP>, <https://www.pantone.com/eu/fr/color-finder/2398-CP>

À ces deux couleurs, différentes nuances de gris peuvent venir s'ajouter pour réaliser les arrière-plans et les éléments de support.

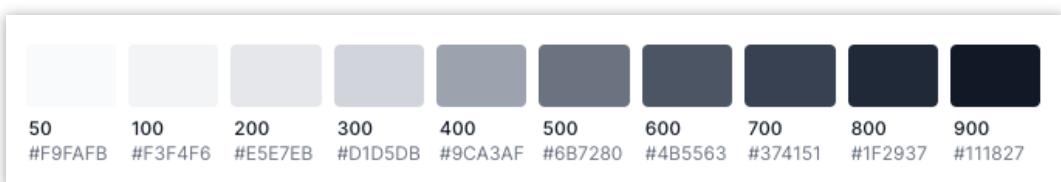


Figure 3 : Palette de nuances de gris choisie pour la réalisation de l'application,
<https://tailwindcss.com/docs/customizing-colors>

L'application se devra donc de respecter et d'utiliser au maximum ces couleurs définies. Toutefois, au besoin, des couleurs complémentaires peuvent venir s'ajouter afin de dynamiser un contenu ou de mettre en avant une donnée spécifique.

² <https://developer.apple.com/design/human-interface-guidelines/macos/overview/themes/>

³ <https://developer.apple.com/design/human-interface-guidelines/ios/overview/themes/>

⁴ <https://material.io/design/guidelines-overview>

⁵ <https://www.pantone.com/eu/fr/>

3.3.1.2 Images et image de marque

L'image de marque est primordiale lors de la création d'une application de ce genre, elle véhiculera son identité et ses visions. Les images utilisées et qui permettront de définir visuellement cette image de marque sont donc très importantes. De ce fait, il faut choisir des images qui vont venir soutenir la stratégie de communication et, qui en représentant l'application, lui seront alors associées.

Pour compléter les couleurs choisies, l'océan et ses mouvements semblent être un bon point de départ pour supporter cette application qui a pour vision de rallier différents types de navires (les développeurs et les mandants) dans l'immensité de l'océan (les domaines de l'informatique en générale).

La banque d'images librement utilisables unsplash.com regorge d'images de tout type et pourra être utilisée pour trouver des images pertinentes à intégrer.

3.3.1.3 Nom et logotype

Pour rester dans le thème de la mer, j'ai choisi d'utiliser deux poissons comme premier logo pour l'application. Ceux-ci représentent les deux types de clients qui pourront se retrouver dans la plateforme web ainsi que leur mise en relation via un projet.



Figure 4 : Logotype et nom de la plateforme dans les couleurs définies et sa nuance en noir et blanc

Pour le nom de la plateforme, j'ai choisi d'utiliser le terme « *MoonFish* ». Celui-ci étant la traduction littérale française du « poisson lune » mais n'existant pas en anglais, j'ai trouvé intéressant de l'utiliser tel quel. Un poisson lune a la particularité de changer d'apparence selon l'angle de vue depuis lequel on l'observe, tout comme les projets. En effet, de face le poisson lune est tout fin et ovale alors que de profil il est plutôt rond et imposant. Tout comme les projets qui sont souvent perçus d'une façon différente par tous ses acteurs ayant chacun un point de vue différent.

3.3.2 Maquettage

Pour réaliser des maquettes et des « *wireframe* », il existe de nombreux outils allant du simple dessin à la main, au service en ligne tel que draw.io⁶, en passant même par des applications dédiées telles que *Sketch*⁷ ou *Adobe Photoshop*⁸. Pour ce projet et dans le but de gagner du temps, j'ai choisi de réaliser des maquettes à la main sur mon iPad puis de les transposer directement dans un framework CSS. De ce fait, ces maquettes seront utilisables telles quelles, et ce sans nécessiter de redéveloppement dans l'application web finale. Le choix du framework permettant de réaliser ces maquettes est étudié et justifié dans le point suivant.

3.3.3 Choix du framework

Pour faciliter la création d'un design d'une page web, différents frameworks sont à disposition des développeurs. Le plus connu et le plus utilisé depuis plusieurs années est *Bootstrap*⁹ développé par Twitter depuis 2011. La solution de facilité aurait été de directement utiliser ce framework puisque je l'ai déjà utilisé et éprouvé dans différents

⁶ <https://app.diagrams.net>

⁷ <https://www.sketch.com>

⁸ https://www.adobe.com/ch_fr/products/photoshop.html

⁹ <https://getbootstrap.com>

projets. Cependant, j'ai envie de découvrir s'il existe un autre framework aussi bon, voir meilleur. C'est pourquoi j'ai choisi de comparer *Bootstrap* à un tout nouveau venu : *Tailwind CSS*¹⁰.

3.3.3.1 Bootstrap

Le framework *Bootstrap* est développé par *Twitter* depuis 2011. Il a été conçu par Mark Otto et Jacob Thornton et son premier déploiement eut lieu lors de la première « *hackweek* » organisée par Twitter. Il se présente sous la forme d'une collection d'outils spécifiques permettant la création de designs et facilitant la mise en page, le graphisme, les animations et les interactions d'une page web.



Figure 5 : Logo de Bootstrap depuis sa version 5,
https://commons.wikimedia.org/wiki/File:Bootstrap_logo.svg

Techniquement, ce framework se présente sous la forme d'un ensemble de fichiers HTML et CSS ainsi qu'en option des extensions JavaScript. Il est composé de formulaires, de boutons, d'outils de navigation, de diaporama et de divers éléments interactifs via l'ajout des options JS. Depuis sa seconde version, *Bootstrap* supporte et améliore la conception de sites web adaptatifs, permettant ainsi à tous les projets l'utilisant de s'adapter dynamiquement aux différentes tailles d'écrans sur lesquels ils sont visionnés (PC, tablette, smartphone). De plus, *Bootstrap* est compatible avec les dernières versions de tous les navigateurs principaux du marché. Ce qui fait la force de ce framework, c'est notamment son système de positionnement des éléments sur une page web, via son système de « *grid* ». Ce système utilise les *flexbox* (*CSS Flexible Box Layout Module*) pour créer des mises en page diverses et variées s'adaptant à toutes les tailles grâce à un système à douze colonnes. Ces mises en pages sont très facilement implémentables grâce à un système de conteneurs, de lignes, de colonnes et de classes CSS prédéfinies.

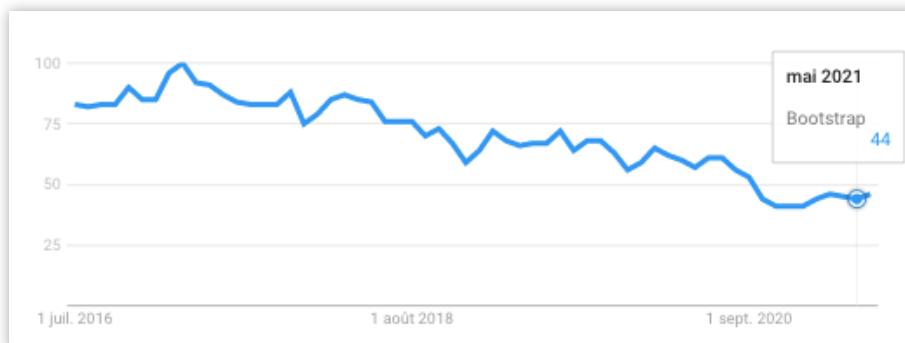


Figure 6 : Évolution du terme de recherche "Bootstrap" sur les 5 dernières années,
<https://trends.google.fr/trends/explore?date=today%205-y&q=Bootstrap>

De nombreux sites connus et reconnus utilisent *Bootstrap*, c'est notamment le cas de *gitlab.com*, du portail web de WhatsApp ou encore de *paypal.com*. Le site de Twitter, quant à lui, intègre des morceaux de *Bootstrap* dans ses menus déroulants, ses formulaires et certains de ses boutons. D'après le classement établi par le site *Wappalyzer*¹¹ sur les « *Part de marché des frameworks d'interface utilisateur* », *Bootstrap* sort en tête avec 72% de part de marché en juin 2021. *Google Trends* nous montre l'évolution du terme « *Bootstrap* » dans son moteur de recherche sur les 5 dernières années. Bien qu'assez stable, on peut tout de même observer une courbe descendante depuis début 2019. Celle-ci peut s'expliquer par l'arrivée sur le marché de nombreux autres concurrents fiables remettant la suprématie de *Bootstrap* en question.

Bootstrap est constamment maintenu et mis à jour par son équipe de développeur « *Bootstrap Core Team* ». Le 5 mai 2021, la version 5 est officiellement publiée. Les principaux changements de cette 5^e version sont : les nouveaux

¹⁰ <https://tailwindcss.com>

¹¹ <https://www.wappalyzer.com/>

composants pour les menus des téléphones mobiles, la suppression de jQuery au profit de l'utilisation de « *vanilla JavaScript* » et un système de grille amélioré.

3.3.3.2 Tailwind CSS

Tailwind CSS est un nouveau venu dans l'univers des frameworks CSS. Il a été imaginé et créé par *Adam Wathan* en 2017. Il se présente comme étant un utilitaire permettant de créer rapidement des interfaces utilisateur hautement personnalisables. Grâce aux différents blocs de construction qu'il met à disposition, il autorise des conceptions sur mesure et promet une création rapide sans avoir besoin de recourir à l'écriture de styles CSS. La force de *Tailwind CSS* est qu'il n'impose pas de spécification de conception ou de lignes directrices de ce à quoi devrait ressembler un site. Il se compose d'innombrables petits composants spécifiques permettant, en les rassemblant, de construire une interface utilisateur unique.



Figure 7 : Logo de Tailwind CSS,
<https://tailwindcss.com/brand/>

Techniquement, *Tailwind CSS* se compose de différentes classes utilitaires permettant un choix cohérent entre les couleurs, l'espacement, la typographie, les ombres, etc. Pour la partie responsive, *Tailwind CSS* permet de précéder n'importe quelle classe utilitaire d'une taille d'écran spécifique. Cette classe utilitaire sera alors appliquée uniquement à la taille spécifiée. Cela évite d'avoir à gérer de nombreuses « *media queries* » complexes. *Tailwind CSS* supprime automatiquement tous les styles inutilisés lors de la création de la version de production, ce qui signifie que le fichier CSS final est le plus petit possible.

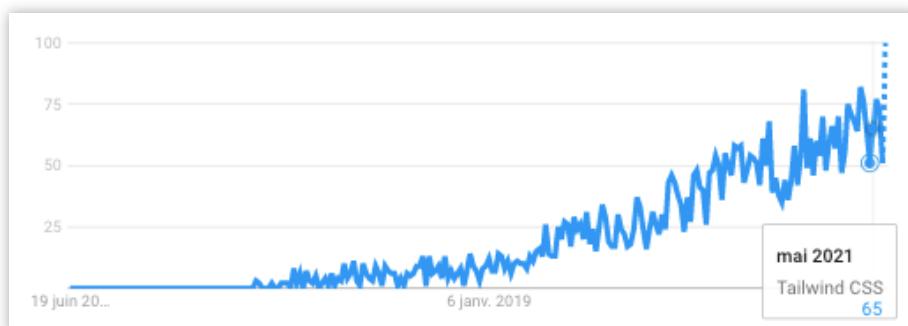


Figure 8 : Évolution du terme de recherche " Tailwind CSS" sur les 5 dernières années,
<https://trends.google.fr/trends/explore?date=today%205-y&q=Tailwind%20CSS>

Tailwind CSS, via son arrivée récente, tire parti de toutes les fonctionnalités CSS moderne, et ce notamment par la prise en charge de la grille CSS, des dégradés composables, des sélecteurs d'état modernes, etc. De par un engouement récent depuis début 2020, comme en témoigne l'évolution *Google Trends* du terme dans son moteur de recherche, ce framework n'est pas encore utilisé par énormément d'applications web. Cependant on peut tout de même citer que le site de covoiturage *BlaBlaCar* l'utilise actuellement sur son site web.

Plusieurs développeurs suivent de près l'évolution de *Tailwind CSS* comme en témoigne la page GitHub lui étant dédiée¹² et le développement de celui-ci est prometteur.

3.3.3.3 Framework choisi

Ayant eu la chance d'expérimenter *Tailwind CSS* lors d'un cours dispensé à la HEIG-VD, j'ai pu me faire un premier avis sur ce framework et cette première expérience était dans l'ensemble réjouissante. C'est pourquoi j'ai maintenant eu l'envie de le comparer à *Bootstrap*, le framework que j'utilise depuis des années. Nous l'avons vu, *Bootstrap* est principalement utilisé aujourd'hui grâce à sa facilité d'utilisation, sa documentation et de nombreux

¹² <https://github.com/tailwindlabs/tailwindcss>

« composants CSS prédéfinis ». Mais souvent *Bootstrap* est mal utilisé, charge beaucoup d’éléments et consomme de nombreuses ressources, ce qui dans un projet conséquent comme cette plateforme web peut poser problème.

Tailwind CSS quant à lui, à l'avantage de ne pas fournir de thème par défaut ce qui rend peu probable le fait de réaliser deux applications web similaires. Cependant il est tout de même structuré et offre une conception agréable. En partant de ce constat, cela permet déjà pour cette plateforme web d'avoir une identité propre.

La principale différence entre ces deux frameworks réside dans le fait que pour l'un (*Bootstrap*) il va falloir écrire relativement beaucoup de CSS alors que pour l'autre (*Tailwind CSS*) il va falloir ajouter plus de classes aux éléments HTML. Prenons un exemple concret qui nous servira de comparaison entre les deux frameworks. Nous allons réaliser un simple bouton bleu avec du texte blanc qui au survol de la souris doit devenir orange. Le but étant que ce bouton est le même aspect en utilisant *Bootstrap* puis *Tailwind CSS*.



Avec les styles de *Bootstrap* et la combinaison d'HTML et de CSS suivant, nous obtenons un tel bouton.

```
<style>
  .hover-orange:hover {
    border-color: orange;
    background-color: orange;
  }
</style>
<button type="button" class="btn btn-primary hover-orange">Mon bouton</button>
```

Pour obtenir le même résultat avec *Tailwind CSS*, nous n'avons pas besoin d'utiliser de CSS supplémentaire, mais devons ajouter plus de classes, comme montré ci-dessous.

```
<button type="button" class="py-2 px-4 bg-blue-600 hover:bg-yellow-500 text-white rounded">Mon bouton</button>
```

Les observations que l'on peut faire sont qu'avec *Tailwind CSS*, il semble possible de gérer tout le style de l'application en ajoutant simplement des classes HTML dites utilitaires. Avec *Bootstrap*, s'il l'on n'utilise pas un style par défaut, il est essentiel de devoir créer de nouvelles classes et d'y appliquer les différents styles désirés.

Finalement, il s'agit de deux approches différentes qui présentent toutes deux des avantages et des inconvénients, mais qui relèvent surtout, au final, des préférences personnelles des développeurs. Pour ma plateforme web, je vais donc choisir d'utiliser le framework *Tailwind CSS*. Les classes utilitaires à disposition semblent nombreuses et efficaces ; cela va me permettre de réaliser rapidement, je l'espère, une interface agréable, sympathique et unique pour ma plateforme web.

3.3.3.4 Installation de *Tailwind CSS*

L'installation de *Tailwind CSS* est très simple, celle-ci peut directement être réalisée via *npm* avec la commande suivante :

```
$ npm install -D tailwindcss@latest postcss@latest autoprefixer@latest
```

Puis, avec la commande suivante, on peut générer les fichiers de configuration. Cette commande créer le fichier '*tailwind.config.js*' à la racine du projet et le fichier '*postcss.config.js*' permettant d'activer des modules par défauts.

```
$ npx tailwindcss init -p
```

Dans le fichier de configuration principal, il faut spécifier les chemins des fichiers que le framework va nettoyer avant la compilation. Les différentes classes de *Tailwind CSS* sont alors prêtes à être utilisées.

3.3.4 Maquettes

3.3.4.1 Squelette

La toute première maquette à réaliser est le squelette général de l'application. Cette mise en page respecte la charte graphique définie ci-avant et sera identique sur toutes les pages du site. Elle se divise en 3 parties distinctes présentées ci-dessous.

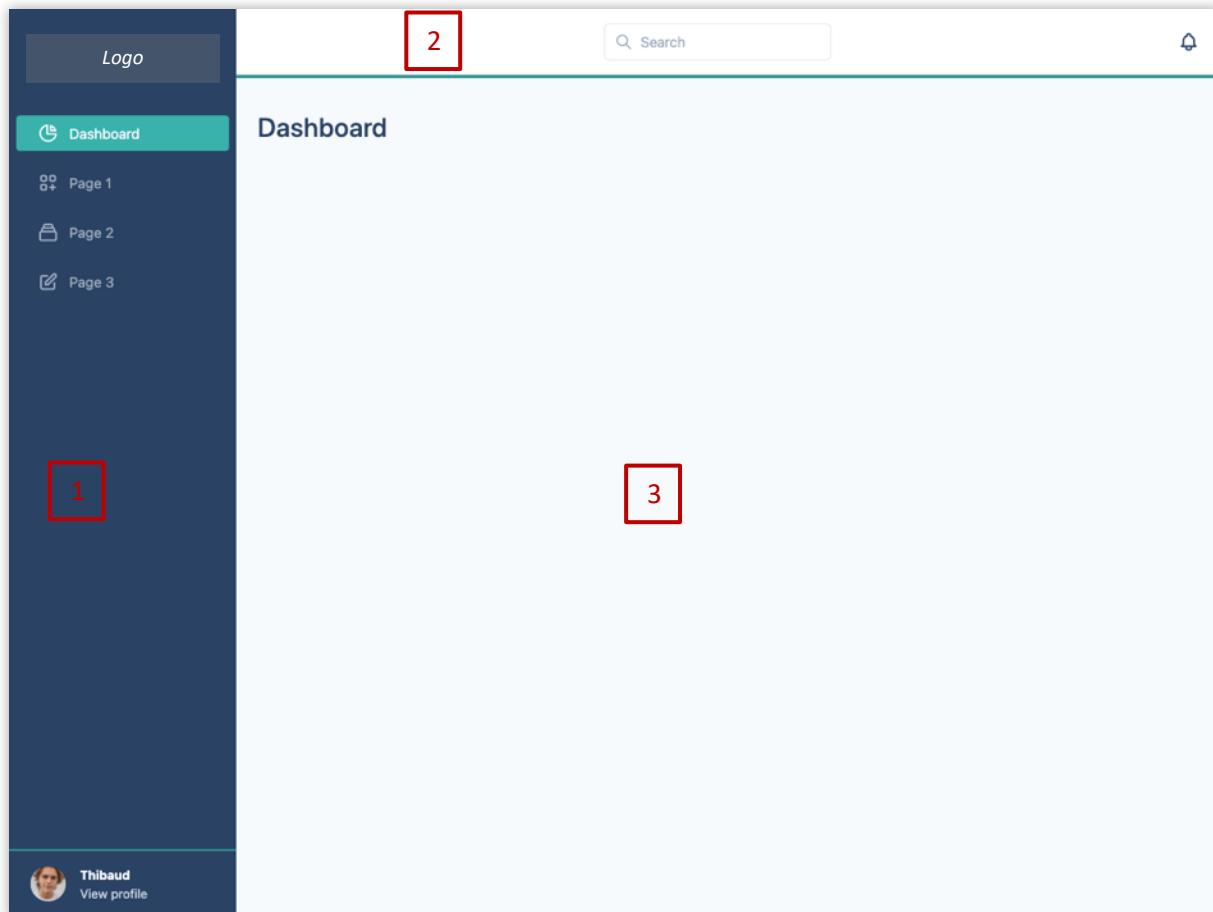


Figure 9 : Maquette représentant le squelette de l'application

1. Volet latéral

Le contenu de ce volet sera identique sur toutes les pages du site, il présente le logo de l'application, le menu de navigation entre les sections principales et le profil de l'utilisateur actuellement connecté.

2. Haut de page

Dans cette partie, l'application présentera un bouton permettant d'afficher le volet latéral contenant la navigation (uniquement au format mobile). Mais également une barre de recherche au centre et sur la droite un résumé des notifications actuelles sera disponible.

3. Contenu de la page

C'est dans cette partie que le contenu dynamique de l'application sera affiché. Dans cette partie, il y a la possibilité de faire défiler le contenu de haut en bas si celui-ci est plus grand que la hauteur de l'écran.

3.3.4.2 Page d'identification

Avant de pouvoir utiliser l'application, n'importe quel utilisateur devra s'y connecter à l'aide d'un identifiant et d'un mot de passe. Pour ce faire, j'ai réalisé la maquette suivante qui me servira de base lors de mes choix technologiques notamment des langages *front-end* et *back-end* qui seront utilisés.

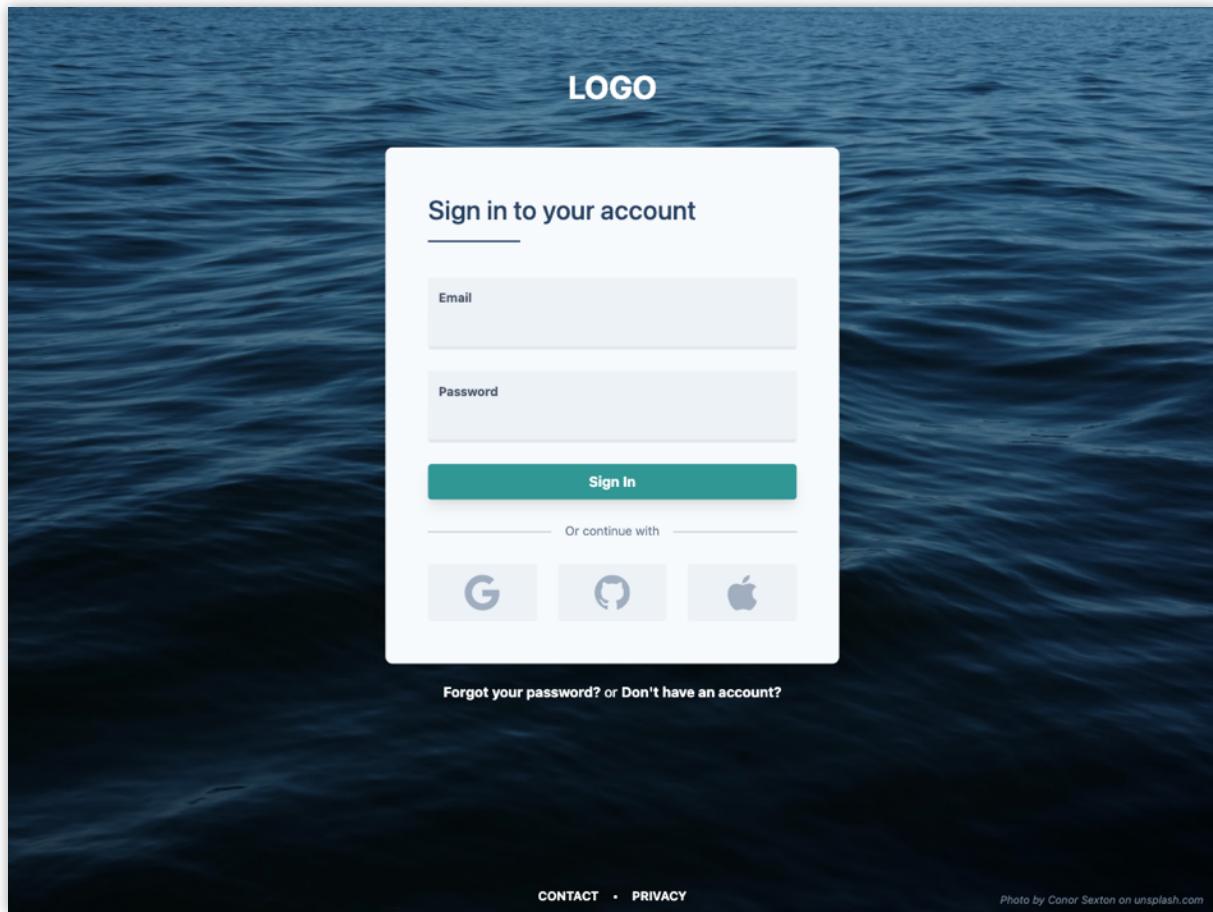


Figure 10 : Maquette représentant la page d'entrée et d'identification de l'application

Celle-ci respecte la charte graphique définie. Elle présente deux possibilités de s'identifier différentes : soit via un couple email / mot de passe, soit en utilisant un des services en ligne proposés. Elle permet également de s'inscrire ou de demander une réinitialisation de mot de passe en cas d'oubli.

3.4 Réception des livrables, validation et paiement

Le déroulement d'un projet dans la plateforme web, sa validation puis sa rémunération suivront les étapes suivantes.

1. Le mandant signe un devis et paie un acompte à la plateforme.
Développement du projet...
2. L'équipe de développeurs a terminé le projet, elle le notifie sur la plateforme.
3. Le mandant doit alors régler le montant total du projet à la plateforme.
4. L'équipe de développeurs livre et éventuellement installe le projet chez le mandant, puis en notifie la plateforme. La réception proprement dite des livrables est gérée directement entre le mandant et l'équipe de développeurs.
5. Le mandant complète un rapport sur le déroulement du projet et sur l'équipe de développeurs, puis valide la livraison et éventuellement l'installation du projet.
6. La plateforme rémunère l'équipe de développeurs et clôture le projet.

Si le mandant ne valide pas la livraison du projet, un expert de la plateforme intervient pour régler le conflit et trouver une solution entre les deux parties. Si le projet est abandonné en cours de route par le mandant, l'équipe de développeurs récupère l'acompte payé par celui-ci.

3.4.1 Changement d'équipe de développement

Si, en cours de projet, l'équipe de développeurs ou le mandant souhaitent rompre le contrat, les deux parties doivent remplir un rapport expliquant les raisons de cette rupture. Ensuite, le projet est proposé une seconde fois sur la plateforme dans son état actuel. Différentes équipes de développeurs peuvent alors proposer leurs services et le mandant doit à nouveau choisir une équipe pour terminer le projet. Le passage du code source, des processus de déploiement, de la documentation, etc. se fait via la plateforme par l'équipe de développeurs « sortante ». Celle-ci se doit de mettre à disposition de la nouvelle équipe de développeurs tous les éléments nécessaires à la bonne continuation du projet.

Un abandon du projet par une équipe de développeurs entraîne, de fait, un signalement sur le profil de cette équipe. Ce signalement contient le rapport des raisons de l'abandon et est noté par rapport à différents critères notamment sur le travail réalisé, la documentation, le passage des éléments nécessaires, la réactivité, etc.

3.5 Livraison et intégration continue

3.5.1 Déploiement continu

Pour le déploiement de mon application, tout au long de son développement, j'ai choisi d'utiliser un service en ligne nommé *Netlify*¹³. *Netlify* est une entreprise fondée en 2014 qui propose des services d'hébergement pour les sites web statiques et pour les applications Node.js. Ce service s'intégrant parfaitement et très facilement avec *GitHub.com*, il va me permettre de déployer des versions de mon application à chaque « *commit* » en toute transparence.

3.5.1.1 Configuration

La configuration de *Netlify* est très simple, puisqu'il suffit de se connecter avec son compte *GitHub* sur le site de *Netlify* et de donner l'accès au « *repository* » contenant le projet à déployer.

Par défaut, *Netlify* nous attribue une URL aléatoire. Celle-ci peut être modifiée dans les réglages généraux. Pour mon projet, j'ai décidé d'utiliser l'URL suivant : <https://heig-tb-moonfish.netlify.app/>. Dorénavant, mon projet et ses évolutions seront donc visibles via cette URL.

¹³ <https://www.netlify.com/>

4 JAVASCRIPT

4.1 Applications web « *State-of-the-Art* »

Aujourd’hui, il existe de multiples manières de réaliser des applications web, et ce au travers de nombreux langages et concepts de programmation. Pour ce projet, j’ai pris le parti d’utiliser uniquement le langage JavaScript pour réaliser mon application web. L’avantage du langage JavaScript est qu’il va non seulement me permettre de réaliser la partie *front-end* et les interactions destinées aux utilisateurs mais aussi la partie *back-end* et son API associée.

Au travers de ce chapitre, je vais présenter une partie des différents frameworks *front-end* et *back-end* existants sur le marché, puis réaliser des tests technologiques avec certains d’entre eux, dans l’objectif de trouver ceux avec lesquels je développerai mon application.

4.2 Le JavaScript c’est quoi ?



[https://commons.wikimedia.org/
wiki/File:Unofficial_JavaScript_logo_2.svg](https://commons.wikimedia.org/wiki/File:Unofficial_JavaScript_logo_2.svg)

Le JavaScript, souvent abrégé *JS*, est un langage de programmation de scripts majoritairement employé dans les pages web. Il a été créé par l’américain Brendan Eich, informaticien chez *Netscape Communications*, dans les années 1990. Sa toute première apparition date du 4 décembre 1995, ce qui en fait aujourd’hui un langage éprouvé. Il est maintenu régulièrement à jour et continue d’être développé par *Netscape* ainsi que par la *Mozilla Foundation*.

Avec HTML et CSS, JavaScript est l’une des technologies de base du World Wide Web. Les principaux navigateurs web du marché incorporent un moteur JavaScript dédié, permettant à l’appareil de l’utilisateur d’exécuter le code JS. JavaScript est conforme à la spécification *ECMAScript*¹⁴, est considéré comme un langage de haut niveau, est souvent compilé « *juste-à-temps* » et est multiparadigme (JavaScript prend en charge les styles de programmation événementiels, fonctionnels et impératifs). Bien qu’il existe des similitudes entre JavaScript et Java notamment au niveau de leur nom, de leur syntaxe ou de leurs bibliothèques standard, les deux langages sont distincts et considérablement différents dans leur conception.

4.2.1 TypeScript

TypeScript est également un langage de programmation de scripts. Il a été développé par *Anders Hejlsberg* en 2012 et est aujourd’hui maintenu par Microsoft. *TypeScript* n’est pas un langage en soi, mais plutôt un surensemble syntaxique strict pour JavaScript qui permet d’ajouter un typage statique. Son but premier est d’améliorer et de sécuriser la production de code JS. *TypeScript* peut être utilisé pour développer des applications JavaScript tant du côté client que du côté serveur. Avant d’être interprété, *TypeScript* doit être compilé, soit avec son propre vérificateur soit à l’aide de *Babel*¹⁵.



[https://commons.wikimedia.org/
wiki/File:TypeScript_logo_2020.svg](https://commons.wikimedia.org/wiki/File:TypeScript_logo_2020.svg)

Concrètement, *TypeScript* apporte un typage statique optionnel des variables et des fonctions ainsi que la création de classes et d’interfaces et l’import de modules. Tout ceci en conservant l’approche non contraignante de JavaScript et en supportant la spécification *ECMAScript 6*.

4.3 Environnements d’exécutions

Un environnement d’exécution parfois abrégé « *runtime* » est un logiciel responsable de l’exécution de programmes écrits dans un langage de programmation donné, par exemple en JavaScript. Son rôle principal est de fournir aux applications tout ce dont elles ont besoin afin d’être correctement exécutées sur une plateforme. Cette plateforme dispose de toutes les ressources nécessaires pour permettre au programme de fonctionner, et ce indépendamment du système d’exploitation. Un environnement d’exécution peut être vu comme une machine virtuelle, car tout comme elle, il offre les services d’exécution natifs de la machine hôte à du code objet. Ces services peuvent être des entrées-sorties, l’arrêt des processus, le traitement des erreurs de calcul, la génération d’événements, etc.

¹⁴ https://developer.mozilla.org/fr/docs/Web/JavaScript/Language_Resources

¹⁵ <https://babeljs.io/>

4.3.1 Node.js

Dans le monde du JavaScript *back-end*, l'environnement d'exécution principal est *Node.js*. *Node.js* est open source, multiplateforme et permet d'exécuter du code JavaScript en dehors d'un navigateur Web. De ce fait, il permet d'exécuter des scripts JS du côté du serveur, permettant ainsi de produire du contenu dynamique avant d'envoyer une page web au navigateur de l'utilisateur.



https://commons.wikimedia.org/wiki/File:Node.js_logo.svg

Node.js a été créé par *Ryan Dahl* un ingénieur software. Sa première version date de mai 2009. Il est aujourd'hui l'environnement d'exécution JavaScript le plus utilisé et est régulièrement maintenu par la *OpenJS Foundation*. *Node.js* est utilisé comme plateforme de serveur Web par de nombreux acteurs comme *LinkedIn*, *Microsoft*, *Netflix* ou encore *PayPal*. D'après le site *W³Techs*¹⁶, *Node.js* est utilisé sur 1,3% du web, ce qui signifie qu'il fait tourner au moins 20 millions de sites web.

4.3.1.1 npm

Node.js ne serait rien sans *npm*, son fidèle gestionnaire de paquets. *npm* se compose d'une interface en ligne de commande et d'une immense base de données en ligne mettant à disposition de nombreux *packages* publics ou certaines fois privés. Il permet une gestion des dépendances simplifiée en s'occupant de les télécharger, de les installer, de les mettre à jour et de les désinstaller. Et tout ceci en maintenant à jour une liste de l'état de ces dépendances permettant à n'importe quel développeur de recréer un environnement de développement à l'aide d'une seule commande. En outre, *npm* fait partie de l'environnement *Node.js* et est donc automatiquement installé à l'installation de ce dernier.

4.3.2 Deno

Deno est également un environnement d'exécution. Il pourrait être considéré comme le petit frère de *Node.js*, car il a également été développé par *Ryan Dahl*. Il a été annoncé en mai 2018 lors d'une conférence donnée par *Ryan Dahl*: « *10 choses que je regrette à propos de Node.js* ». *Deno* se concentre sur la productivité et endosse non seulement le rôle d'environnement d'exécution, mais également celui de gestionnaire de paquets.



<https://commons.wikimedia.org/wiki/File:Deno.svg>

4.3.3 Choix d'environnement

Bien qu'intéressant, l'environnement d'exécution *Deno* manque de maturité et de documentation. Je ne vais donc pas me lancer dans son utilisation pour réaliser mon application. Cependant il reste un concurrent sérieux au couple éprouvé *Node.js - npm* et est donc à surveiller pour le développement JavaScript de ces prochaines années.

4.4 Frameworks front-end

Pour les frameworks *front-end*, le langage JavaScript est l'unique langage côté client utilisé actuellement par les sites web. Selon les données de *W³Techs*, il est présent sur 97.4% des sites web en juin 2021. Le *Flash* est encore présent sur 2% des sites web. Il s'agit certainement d'anciens sites qui n'ont jamais été mis à jour depuis son abandon pour des raisons sécuritaires.

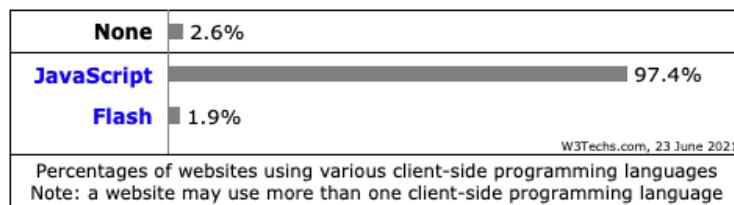


Figure 11 : Pourcentages de répartition des langages de programmation côté client utilisés par les sites web en juin 2021,
https://w3techs.com/technologies/overview/client_side_language

¹⁶ <https://w3techs.com/technologies/details/ws-nodejs>

Du fait que JavaScript soit l'unique langage de programmation *front-end* utilisé aujourd'hui, il existe de nombreux moyens de l'implémenter et de l'utiliser. C'est pourquoi, il subsiste aujourd'hui sur le marché des centaines de frameworks JavaScript, chacun étant plus ou moins adapté et conçu pour tel ou tel type d'applications et fournissant plus ou moins de fonctionnalités. Parmi eux, certains prennent le dessus et sont alors majoritairement utilisés par les développeurs. Dans cette section, je vais tenter de comprendre pourquoi certains framework sortent du lot et en détailler les avantages et les inconvénients.

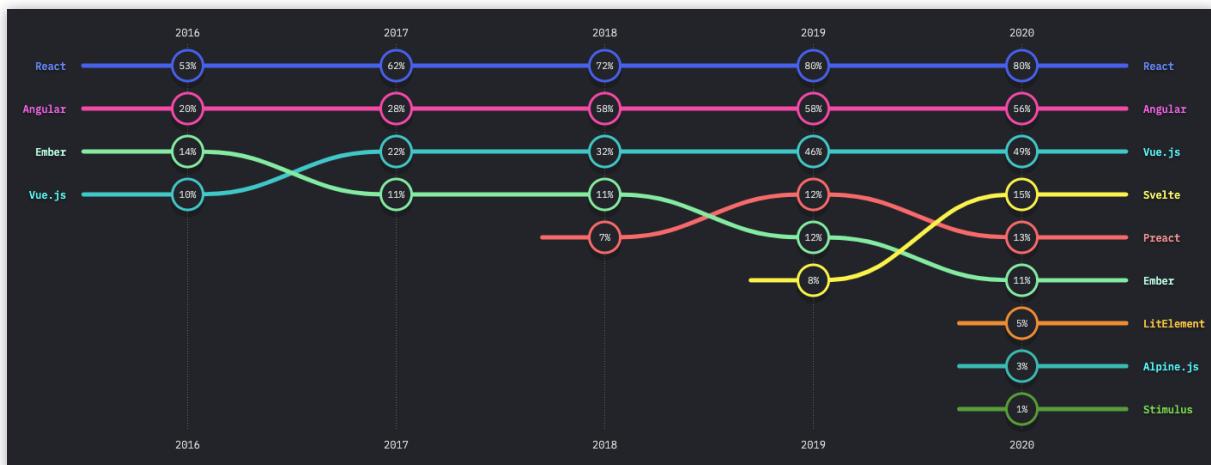


Figure 12 : Évolution d'utilisation des frameworks JavaScript front-end de 2016 à 2020,
<https://2020.stateofjs.com/en-US/technologies/front-end-frameworks/>

4.4.1 jQuery

jQuery est une bibliothèque JavaScript conçue en 2006 par John Resig pour simplifier la manipulation du DOM HTML, la gestion des événements, les animations CSS et les requêtes Ajax. Son utilisation est gratuite et open source, sa distribution utilise la licence MIT. *jQuery* est considéré comme le tout premier framework JS, même s'il n'en est pas réellement un, qui a permis au langage JavaScript d'être découvert puis d'être utilisé en masse par les développeurs web. Encore aujourd'hui, il s'agit et de loin de la bibliothèque JS la plus largement déployée sur le web, avec entre 1/3 et 2/3 (en incluant *jQuery UI* et *jQuery Migrate*) de part de marché.



jQuery
write less, do more.
<https://fr.wikipedia.org/wiki/Fichier:jQuery-logo.png>

- 2006 (15 ans)
- The *jQuery Team*
- JavaScript
- Licence MIT
- jquery.com

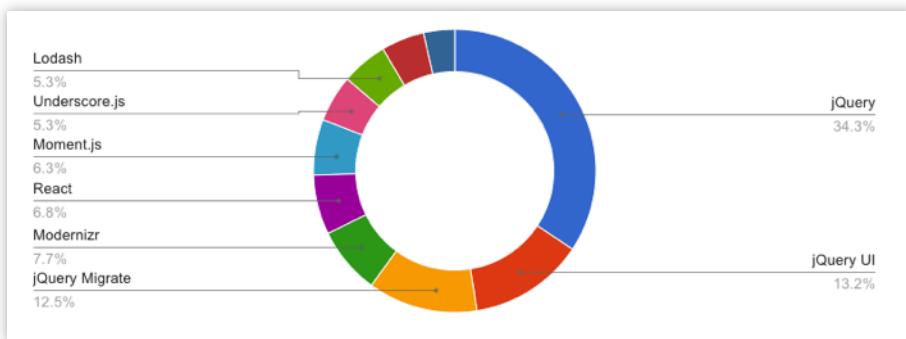


Figure 13 : Technologies de bibliothèques JavaScript les plus utilisées en se basant sur la part de marché en 2021,
<https://www.wappalyzer.com/technologies/javascript-libraries/>

À l'époque de sa sortie, *jQuery* est très intéressant, car il offre de nouvelles possibilités aux programmeurs notamment en permettant de rendre des pages web jusqu'alors statiques, dynamiques et ce, plutôt facilement. *jQuery* a également l'avantage de résoudre des problématiques de compatibilité du JS avec les nombreux navigateurs. En effet un seul code *jQuery* fonctionne à l'identique sur tous les navigateurs. Cependant *jQuery* inclut certains inconvénients, et pas des moindres ! Son code est très difficilement structurable et son implémentation

n'est pas standardisée. Il montre certaines limitations au niveau de la testabilité, de la performance et de la scalabilité. La scalabilité étant la capacité de supporter une montée en charge d'utilisation d'un logiciel.

4.4.1.1 Avantages / Inconvénients

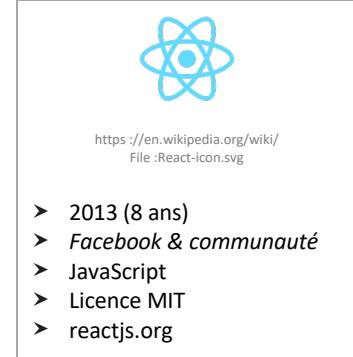
Avantages	Inconvénients
<ul style="list-style-type: none"> - Courbe d'apprentissage rapide - Bonne documentation - Facilite la mise en place des requêtes AJAX - Il existe de nombreux plug-ins 	<ul style="list-style-type: none"> - Peut être lent - Non structuré, code « <i>spaghetti</i> » - Fonctionnalités limitées - Non rétrocompatible - Conflit de plug-ins multiples

4.4.1.2 Hello, World !

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <script src="https://code.jquery.com/jquery-3.6.0.min.js"
           crossorigin="anonymous" integrity="sha256-/xUj+3OJU5yExlq6GSYGHk7tPXikynS7ogEvDej/m4=></script>
    <script>
        $(document).ready(function () {
            alert("Hello, World!");
        });
    </script>
</head>
<body>
</body>
</html>
```

4.4.2 React

React ou *React.js* est une bibliothèque JavaScript libre et open source (sous licence MIT) développée depuis 2013 par *Facebook*. Son principal but est de faciliter la création d'application web monopage, et ce via la création de différents composants dépendants d'un état. À chaque changement d'état, le composant associé se met à jour. *React* ne s'occupant que de la gestion des états et de leur rendu dans le DOM, la création d'applications avec *React* nécessite l'utilisation de bibliothèques supplémentaires pour le routage par exemple.



4.4.2.1 Avantages / Inconvénients

Avantages	Inconvénients
<ul style="list-style-type: none"> - Vitesse de traitement élevée grâce au DOM virtuel - Syntaxe JSX simple pour les modèles - Large communauté - Fonctionnalités de liaison de données (<i>data binding</i>) efficace - Code réutilisable et facile à tester - Migration simple - Possibilité de faire des composants mobiles natifs 	<ul style="list-style-type: none"> - Communauté divisée sur certains aspects gestion des feuilles de style, utilisation de <i>JSX</i>) - Style de programmation non « <i>orientée objet</i> » - JSX peut rapidement devenir compliqué si les modèles sont nombreux et interconnectés - Documentation parfois désuète - Mises à jour constantes, difficiles à intégrer

4.4.2.2 Hello, World !

```
import React, { Component } from 'react';

class App extends Component {
  render() {
    return (
      <div className="App">
        <h1>Hello, World!</h1>
      </div>
    );
  }
}

export default App;
```

4.4.3 Angular

Angular, à ne pas confondre avec *AngularJS* est également un framework open source sous licence MIT. Il est basé sur *TypeScript* et est maintenu par *Google* depuis ses débuts en 2016.



[https://en.wikipedia.org/wiki/
File:Angular_full_color_logo.svg](https://en.wikipedia.org/wiki/File:Angular_full_color_logo.svg)

- 2016 (5 ans)
- Google
- TypeScript
- Licence MIT
- angular.io

4.4.3.1 Avantages / Inconvénients

Avantages	Inconvénients
<ul style="list-style-type: none"> - Prise en charge de <i>TypeScript</i> - Fort potentiel d'évolutivité (<i>scalability</i>) - Algorithmes de liaison de données (<i>data binding</i>) sans faille - Prise en charge de l'injection de dépendances basée sur les modules - Nombreuses bibliothèques et fonctionnalités avancées - Architecture MVVM (<i>Modèle-Vue-VueModèle</i>) et structure claire - Documentation efficace et détaillée - Framework JS le plus populaire 	<ul style="list-style-type: none"> - Le code peut être difficile à lire - Beaucoup de composants structurels - Performances plus lentes (par rapport à la concurrence) - Processus de rendu lent - Pas de système de routage intégré - Options de référencement limitées - Courbe d'apprentissage lente

4.4.3.2 Hello, World !

```
import { Component } from '@angular/core';

@Component ({
  selector: 'my-app',
  template: `<h1>{{title}}</h1>`,
})

export class AppComponent {
  title = 'Hello, World !';
}
```

4.4.4 Vue.js

Vue.js, parfois abrégé *Vue*, est framework open source respectant l'architecture MVVM. Il a été créé par *Evan You* en 2014 et est régulièrement maintenu par lui-même ainsi que par son équipe de développement initial. Contrairement à d'autres grands frameworks, aucune grande entreprise n'a d'emprise sur *Vue.js*. Il est principalement utilisé pour la création d'interfaces utilisateur et d'applications monopage. *Vue.js*, tout comme *Angular*, est basé sur *TypeScript*.



[https://en.wikipedia.org/wiki/
File:Vue.js_Logo_2.svg](https://en.wikipedia.org/wiki/File:Vue.js_Logo_2.svg)

- 2014 (7 ans)
- *Evan You & core team*
- *TypeScript*
- Licence MIT
- vuejs.org

4.4.4.1 Avantages / Inconvénients

Avantages	Inconvénients
<ul style="list-style-type: none"> - La courbe d'apprentissage très rapide - Petite taille - Très bonnes performances - Intégration flexible - Forte évolutivité possible - Communauté importante et amicale - Composants réutilisables 	<ul style="list-style-type: none"> - Petite communauté - Faible part de marché - Moins de plug-ins que ces concurrents - Problèmes de performances d'anciens navigateurs web

4.4.4.2 Hello, World !

```
<div id="app">{{ message }}</div>
<script>
    var app = new Vue({
        el: '#app',
        data: {
            message: 'Hello, World!'
        }
    })
</script>
```

4.4.5 Svelte

Svelte, créé par *Rich Harris* en 2016 et distribué sous licence MIT est un compilateur gratuit et open source. Les applications *Svelte* génèrent du code pour manipuler le DOM, ce qui permet de réduire la taille des fichiers transférés et ainsi améliorer les performances d'exécution. *Svelte* est écrit en *TypeScript* et possède son propre compilateur lui permettant de convertir du code applicatif en JavaScript exécutable du côté client.



[https://en.wikipedia.org/wiki/
File:Svelte_Logo.svg](https://en.wikipedia.org/wiki/File:Svelte_Logo.svg)

- 2016 (5 ans)
- *Rich Harris*
- *TypeScript*
- Licence MIT
- svelte.dev

4.4.5.1 Avantages / Inconvénients

Avantages

- Moins de code, donc moins de bugs potentiels
- Pas de DOM virtuel
- Très réactif
- Implémentation mobile possible
- Courbe d'apprentissage minimale
- Rapidité d'exécution
- Peut être utilisé pour créer l'intégralité de l'application ou utilisé de manière incrémentale
- Communauté amicale

Inconvénients

- Faible part de marché
- Pas de vérification de type
- Confusion dans les noms des variables et de la syntaxe
- Difficultés liées à un framework de type compilateur
- Pas de soutien majeur
- Petite communauté

4.4.5.2 Hello, World !

```
<h1>{title}</h1>
<script>
  import App from './App.svelte';

  const app = new App({
    target: document.body,
    props: {
      title: 'Hello, World!'
    }
  );

  export default app;
  export let title;
</script>
```

4.4.6 Preact

Preact se décrit comme une alternative rapide et extrêmement légère (3 Ko) à *React.js*. Il a été développé dans le but de créer un framework de petite taille tout en offrant la même API et les mêmes fonctionnalités que *React.js*.

Cependant pour réussir à faire en sorte que *Preact* soit si petit, il fait des compromis et retire certaines fonctionnalités notamment au niveau de la gestion des événements, de la gestion de validation ou encore au niveau des moteurs de rendu « *renderer* ».



Preact semble de prime abord intéressant, cependant de par son manque de fonctionnalités, je ne vais pas l'utiliser par peur d'être bloqué à un certain moment du projet. Toutefois, si mon choix se porte sur *React.js* pour réaliser mon application et que cela en vaut la peine, il me sera possible de switcher sur *Preact* assez facilement.

4.4.7 Ember.js

Ember.js, créé initialement par *Yehuda Katz* en 2011 et distribué sous licence MIT est un framework incorporant des idiomes communs permettant de créer des applications web évolutives d'une seule page. Sa particularité est qu'il est également possible de créer des applications de bureau et des applications mobiles avec *Ember.js*.

Ember.js, est un framework avec une courbe d'apprentissage lente et difficile par rapport à ces concurrents, de plus il peut s'avérer excessivement lourd et contraignant pour des applications simples. Comme le montre la courbe de « *state*



of JS » ci-dessus, il n'a pas réussi à attirer beaucoup de développeurs et en a même perdu depuis 2016 dans la compétition des frameworks *front-end*.

➤ emberjs.com

De ce fait, je ne veux pas me risquer à me lancer dans l'apprentissage et dans l'utilisation de ce framework pour ce projet. Le temps imparti me semble trop court pour me risquer à la courbe d'apprentissage laborieuse d'*Ember.js*.

4.5 Frameworks *back-end*

Bien que le JavaScript soit majoritairement utilisé pour réaliser la partie *front-end* des applications, il peut également être utilisé du côté du serveur pour réaliser la partie *back-end*. Aujourd'hui, l'utilisation du JS pour les parties *back-end* n'est pas encore très répandue (1.4% en juin 2021). C'est pourquoi il est intéressant de comprendre son positionnement *back-end* et ses possibilités au travers de différents frameworks basés sur cette technologie JS.

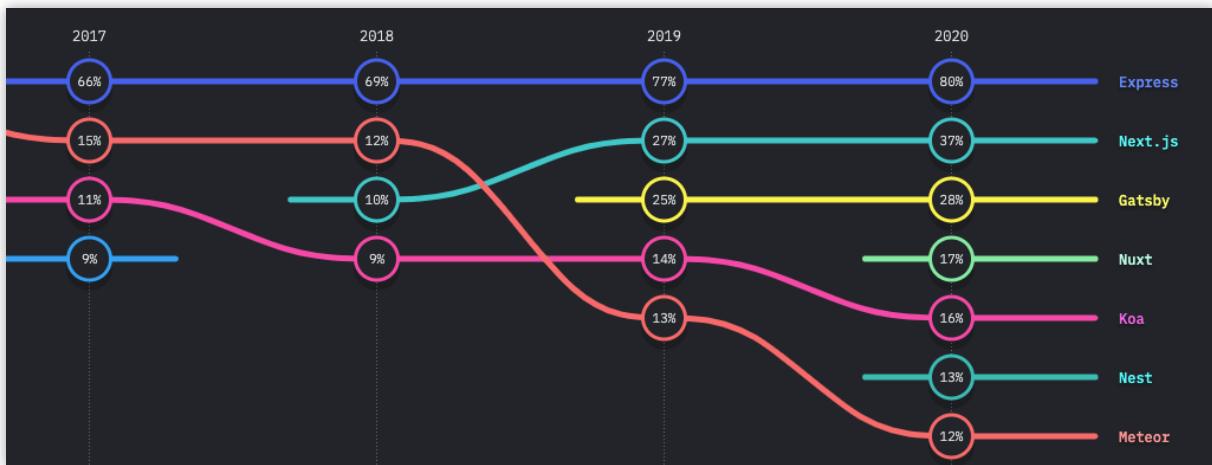


Figure 14 : Évolution d'utilisation "(l'utiliserait à nouveau + ne l'utiliserait plus) / total" des frameworks JavaScript *back-end* de 2017 à 2020, <https://2020.stateofjs.com/en-US/technologies/back-end-frameworks/>

4.5.1 Express.js

Express.js, ou plus simplement *Express*, est un framework open source permettant d'utiliser *Node.js*. Il a été pensé et développé dans le but d'être utilisé pour créer les API des applications web. *Express.js* a été créé en 2010 par *TJ Holowaychuk*, celui-ci s'est ensuite fait acquérir par IBM en 2015. Son auteur le décrit comme « *minimaliste tout en ayant la possibilité d'étendre ses fonctionnalités via des plug-ins* ». Actuellement, il est le standard pour le développement de serveur en *Node.js* et le framework *back-end* JS le plus utilisé.

Le principal avantage de l'utilisation d'*Express* est de faciliter et d'accélérer le codage des tâches compliquées du côté du serveur. De plus, *Express.js* fournit un puissant mécanisme de routage qui, contrairement à celui fourni par *Node.js*, prend en charge les URL dynamiques.

4.5.1.1 Avantages / Inconvénients

Avantages

- *La documentation impressionnante*. *Express* est complété d'une documentation riche ainsi que de nombreuses ressources d'aides et de tutoriels.
- *Le gain de temps*. *Express* accélère grandement et rationalise le développement

Inconvénients

- *La sécurité*. Il est de l'entièvre responsabilité du développeur d'assurer la sécurité de l'application. *Express* n'offrant aucune solution de sécurité.
- *Les problèmes de « callback »*. Certaines fois, les *callbacks* sont tellement imbriqués dans d'autres *callbacks* et ce sur plusieurs niveaux, ce qui rend

express

[https://en.wikipedia.org/wiki/
File:Expressjs.png](https://en.wikipedia.org/wiki/File:Expressjs.png)

- 2010 (11 ans)
- *TJ Holowaychuk & StrongLoop*
- JavaScript
- Licence MIT
- expressjs.com

d'applications par rapport à l'utilisation brute de *Node.js*.

- *Le routage.* Un mécanisme de routage robuste est fourni pour définir les routes en fonction des méthodes HTTP et des URL.
- *Le support communautaire.* Express étant un framework mature, il dispose d'une énorme banque communautaire.
- *La courbe d'apprentissage.* Au profit de sa structure et de sa syntaxe simple, *Express* est assez facile à apprêhender pour les développeurs.

potentiellement difficile la compréhension et la maintenance du code.

4.5.1.2 Hello, World !

Le code *Express* ci-dessous démarre un serveur Web à l'écoute sur le port 3000.

```
Const express = require('express');
const app = express();

app.get('/', (req, res) => res.send('Hello, World!'))

app.listen(3000, () => {
  console.log('Server listening on port 3000')
});
```

4.5.2 Next.js

Basé sur *React.js*, *Next.js* a été créé par société néerlandaise *Vercel* en 2017. Les points forts de *Next* sont le rechargeement de code « à chaud », le fractionnement de code automatisé, le routage automatisé et la gestion intégrée du référencement.

Même si *Next.js* se veut être un moteur de rendu côté serveur, il propose également des générateurs de pages statiques pour les applications basées sur *React.js*. Les applications *React.js* restituant principalement leur contenu dans le navigateur côté client, trouvent avec *Next.js* de nouvelles fonctionnalités supplémentaires permettant d'être exécutées du côté serveur.

4.5.2.1 Avantages / Inconvénients



[https://en.wikipedia.org/wiki/
File:Nextjs-logo.svg](https://en.wikipedia.org/wiki/File:Nextjs-logo.svg)

- 2016 (5 ans)
- *Vercel & communauté*
- JavaScript & TypeScript
- Licence MIT
- nextjs.org

Avantages

Inconvénients

- *Le rendu côté serveur.* Next inclut un moteur de rendu SSR qui fournit des performances beaucoup plus rapides. En effet, il n'est pas nécessaire d'attendre que le navigateur du client affiche le contenu pour que le moteur SSR commence le rendu HTML. Par ce biais, il est déjà possible d'obtenir un rendu initial de l'application pendant que le chargement du code continue en arrière-plan.

- *L'optimisation pour le référencement.* En général, les applications cliente des frameworks classiques n'ont pas de bonnes performances de référencement, car il est difficile pour les moteurs de recherche de les indexer. Au contraire, *Next* offre des performances de référencement élevées grâce à sa

- *Les performances.* Les performances de référence (benchmarks) de *Nuxt.js* et de *Gatsby* sont nettement meilleures que celles de *Next*.

- *Le moteur de rendu.* Pour de petites applications, les générateurs statiques sont considérés comme meilleurs pour fournir un rendu que la compilation côté serveur de *Next*.

capacité de rendu du côté serveur avec laquelle il est possible de créer des « *meta tags* » optimisant le référencement.

- *Le rechargement du code « à chaud ».* Le système de rechargement automatique des pages offert par *Next* dès qu'il y a une modification est très efficace.

4.5.2.2 Hello, World !

Next.js utilise une structure de pages déclarative, basée sur la structure du système de fichiers. Après avoir créé un projet *Next* avec *npm*, le code *Next.js* ci-dessous permet d'afficher un message en visitant la racine d'un site web.

```
Function HomePage() {  
  return <div>Hello, World!</div>  
}  
  
export default HomePage
```

4.5.3 Gatsby

En 2015, *Sam Bhagwat* et *Kyle Mathews* fondent *Gatsby.js* qui se veut être un générateur de sites statiques contemporain et flexible basé sur *GraphQL* et *React.js*. *Gatsby*, à l'opposé de *Next.js*, n'est pas un moteur de rendu côté serveur SSR, mais un générateur de pages statiques. Il offre des performances rapides, de très bons résultats de référencement ainsi qu'une sécurité accrue. Il est également possible d'ajouter des plug-ins à *Gatsby* pour étendre ces fonctionnalités de base.



<https://www.gatsbyjs.com/guidelines/logo>

- 2017 (4 ans)
- *Gatsby Inc.*
- JavaScript & TypeScript
- Licence MIT
- gatsbyjs.com

4.5.3.1 Avantages / Inconvénients

Avantages

Inconvénients

- *Les performances rapides.* Les sites créés avec l'aide de *Gatsby* sont généralement plus rapides que les sites « normaux » construits à l'aide d'autres frameworks.
- *L'optimisation pour le référencement.* Les robots des moteurs de recherche peuvent facilement lire et indexer le contenu statique généré par *Gatsby*.
- *La sécurité accrue.* *Gatsby* offre, par définition, une sécurité de premier choix, car il n'a besoin d'aucune base de données ou de serveur pour fonctionner.
- *La prise en charge de différentes sources de données.* Les informations peuvent être collectées par *Gatsby* sur de nombreuses sources d'informations distantes telles que *WordPress*, *Drupal*, *Trello*, etc.

- *Pas idéal pour les sites à grande échelle.* Les sites riches en contenus, comme les commerces en ligne, ne devraient pas être réalisés avec *Gatsby*. En effet, le temps de construction augmente significativement par rapport à la taille des données et du contenu.
- *Les prérequis.* Bien qu'il ne soit pas très compliqué en soi d'apprendre et d'utiliser *Gatsby*, une compréhension préalable de *GraphQL* et de *React.js* est nécessaire.

4.5.3.2 Hello, World !

Le code *Gatsby* ci-dessous permet d'afficher un message sur une page web. Celui-ci sera compilé par *Gatsby* qui générera un fichier statique. Comme on peut le remarquer, le code est identique à du code React.

```
Import React from "react"

export default function Home() {
  return <div>Hello, World!</div>
}
```

4.5.4 Nuxt.js

À l'instar de *Next.js* pour *React.js*, on peut considérer *Nuxt.js* comme un amplificateur pour Vue. Cependant, *Nuxt.js* ne peut pas fonctionner seul et par conséquent on ne peut pas le considérer comme un framework *back-end* complet. En effet, *Nuxt.js* est plutôt une combinaison de composants et de bibliothèques *Vue* officiels. *Nuxt.js* se veut être un « *métaframework pour créer des applications universelles* ». Cela signifiant que le code de l'application est initialement exécuté par le serveur puis dans le navigateur client. *Nuxt.js* permet également la génération de pages web statiques pouvant être servies par n'importe quel serveur web.

L'utilisation de ce framework a de nombreux avantages comme l'amélioration des processus de l'optimisation pour les moteurs de recherches, du fait du rendu côté serveur des pages web avant leur envoi vers le client, ce qui n'est pas fait de manière générale dans les applications web d'une seule page. Son utilisation, comme pour *Next.js* permet d'améliorer et d'optimiser l'indexation des pages web par les moteurs de recherches.

4.5.4.1 Avantages / Inconvénients



- 2016 (5 ans)
- Communauté
- JavaScript
- Licence MIT
- nuxtjs.org

Avantages

Inconvénients

- *La structure du projet.* Par défaut, le code est organisé par *Nuxt* dans une structure évolutive, logique et simple à comprendre.
- *La communauté.* La communauté autour de *Nuxt.js* est grande et met à disposition une collection de plusieurs API, kits de démarrage, modules, bibliothèques, etc.
- *Le fractionnement de code.* Une version statique de chaque page de l'application est générée par *Nuxt*. Par conséquent, le code JavaScript peut être divisé en plusieurs fichiers plus petits, améliorant ainsi la vitesse et les performances.
- *L'installation et le développement rapides.* La majorité de la configuration initiale et de l'installation est prise en charge par *Nuxt.js*, ce qui permet de commencer à coder immédiatement.

- *L'intégration de bibliothèques.* L'intégration de bibliothèques et surtout de bibliothèques personnalisées avec *Nuxt* peut être longue et difficile.
- *Les problèmes de débogages.* *Nuxt* génère de nombreux problèmes de débogage, ce qui peut s'avérer être assez frustrant.

4.5.4.2 Hello, World !

Dans cet exemple, le code *Nuxt.js* ci-dessous montre comment afficher un message sur une page web.

```
<template>
  <h1>Hello, World!</h1>
</template>
<script>
export default {
  asyncData() {
    return {
      rendering: process.server ? 'server' : 'client'
    }
  }
}
</script>
```

4.6 Tests technologiques

De par la diversité des frameworks *front-end* et *back-end* présents sur le marché aujourd’hui, il m'est impossible de réaliser des tests technologiques avec chacun d’entre eux. J’ai donc dû faire des choix et éliminer certains frameworks en me basant sur les recherches que j’ai pu réaliser jusqu’ici.

Pour les frameworks *front-end*, j’ai choisi d’en présélectionner trois d’entre eux d’après leurs fonctionnalités, leur documentation, leur communauté, leur courbe d’apprentissage et leurs avantages/inconvénients cités précédemment. Mon choix s’est porté sur : **React.js**, **Vue.js** et **Angular**. J’ai éliminé de ma liste de concurrents **jQuery**, car celui-ci n’a pas été pensé pour réaliser des applications complètes et il serait donc trop compliqué de l’utiliser pour ce projet. J’ai longuement hésité à embarquer **Svelte** dans ma liste de comparaison, néanmoins après de nombreuses recherches, celui-ci souffre encore de son manque de maturité et d’une liste de plug-ins. Finalement, le fait qu’il ne respecte pas exactement les exigences définis par le cahier des charges en étant un compilateur et non pas un framework à part entière, a entériné ma décision et a définitivement exclu **Svelte** de la liste des concurrents présélectionnés.

Pour les frameworks *back-end* et mise à part **Express.js**, chacun d’entre eux se réfère à un framework *front-end*. De ce fait, le choix de celui-ci sera forcément lié au choix du framework *front-end* et inversement. Le tableau suivant résume les différents liens entre ces frameworks.

Framework <i>back-end</i>	Associé au framework	Prévu pour
Express.js	-	Développer un serveur Node.js
Next.js	React.js	Rendu côté serveur et génération de pages statiques
Gatsby	React.js	Génération de pages statiques
Nuxt.js	Vue.js	Rendu côté serveur et génération de pages statiques

À la suite de mes recherches, je peux éliminer **Gatsby** qui ne permet pas de réaliser une API, mais de faire de la génération de pages statiques ce qui, dans un premier temps, n'est pas le but recherché. **Next.js** et **Nuxt.js** étant tous les deux des frameworks encore jeunes par rapport à **Express.js** et surtout fortement liés à leurs framework *front-end* respectifs, j’ai décidé de ne pas les utiliser. J’ai donc choisi d’utiliser **Express.js** pour le développement *back-end* de mon application par son usage éprouvé, sa polyvalence, sa documentation et son support communautaire. Le fait qu'il ne soit pas lié à un framework *front-end* est un avantage supplémentaire, si d'aventure le framework *front-end* choisi devait être remplacé, toute la partie *back-end* pourrait être conservée.

4.6.1 Stratégie de test

Comparer rapidement et efficacement l’ensemble des frameworks *front-end* et *back-end* est malheureusement impossible. C'est pourquoi j'ai présélectionné trois frameworks avec lesquels je vais réaliser une page de login d'application. À l'issue de ce test, j'aurais pu mieux découvrir ces différents frameworks et il me sera alors plus facile de définir lesquels j'utiliserai pour développer mon application.

Le cas d'utilisation d'une page de login est intéressant, car il permet de tester l'intégration d'une maquette visuelle tout en faisant appel à un formulaire. Il mettra également en avant la communication avec une base de données

4.6.2 Express.js

4.6.2.1 Prérequis

- *Node.js* version 10.0 ou supérieur (<https://nodejs.org/en/download/>)
- *npm* version 5.2 ou supérieur (<https://www.npmjs.com/package/download>)

4.6.2.2 Création d'une application

Pour ce test, nous utiliseront la même API pour les différents frameworks *front-end* testés. Pour ce faire, nous créerons cette API dans un dossier externe aux frameworks.

Pour créer facilement une API *Express*, il est possible d'utiliser « *express-generator* », celui-ci permet de générer un squelette d'application de base et d'y appliquer des configurations par défaut. Pour ce faire, il faut commencer par créer un dossier où l'application sera installée. Puis en exécutant la commande suivante dans ce dossier, la structure est créée.

```
$ npx express-generator && npm install
```

Par défaut Express utilise le port 3000, cependant cela risque de nous poser un problème avec *React.js* qui l'utilise lui aussi. De ce fait, nous allons changer le port utilisé par Express par le port 3001 via la commande suivante qui crée un fichier de configuration en renseignant le nouveau port à utiliser.

```
$ echo "PORT=3001" > .env
```

Il faut ensuite installer la dépendance « *dotenv* ».

```
$ npm install dotenv
```

Et finalement ajouter la ligne suivante au tout début du fichier « *app.js* » pour qu'*Express* prenne en compte ce fichier de configuration.

```
require('dotenv').config();
```

4.6.2.3 Lancement du serveur

Une fois les configurations faites, nous pouvons utiliser la commande suivante pour lancer le serveur *Express* puis ouvrir l'adresse « <http://localhost:3001/> » dans un navigateur pour vérifier le bon fonctionnement de celui-ci.

```
$ npm start
```

4.6.2.4 Configuration de « *Sequelize ORM* »

Sequelize est un *ORM* (interface entre un applicatif et une base de données relationnelle) *Node.js* fonctionnant avec *MariaDB*. Il est basé sur les « *promise* ¹⁷ » et offre une prise en charge des transactions solide, la gestion des relations, un chargement rapide, etc.

Pour l'installer, il faut l'ajouter comme dépendance avec la commande suivante.

```
$ npm install sequelize
```

Puis ajouter le connecteur du type de base de données désiré, ici *MariaDB*.

```
$ npm install mariadb
```

Nous pouvons dès lors créer un fichier « *db.config.js* » qui contiendra les paramètres de connexions à la base de données (hôte, nom d'utilisateur, mot de passe, temps d'attente, etc.)

¹⁷ https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/Promise

4.6.2.5 Création du *endpoint* « /users »

Ce premier *endpoint* pourra lister l'ensemble des utilisateurs présents dans la base de données. La création de celui-ci me permettra de me familiariser avec la connexion entre *Express* et *MariaDB*.

Pour commencer, il faut créer un fichier « *users.js* » dans le dossier « *routes* », c'est dans celui-ci que nous pourrons spécifier les différents chemins et les fonctions exécutés à l'appel de ceux-ci. Voici par exemple la route spécifiée pour récupérer tous les utilisateurs.

```
// File : routes/users.js
const express = require('express');
const router = express.Router();
const users = require("../controllers/users");

// Retrieve all users
router.get('/', users.findAll);

module.exports = router;
```

Ce fichier de routes fera un appel à la fonction « *findAll* » définie dans un contrôleur lors de l'accès au *endpoint* « /users ». Dans ce fichier contrôleur, la définition de cette fonction est assez simple.

```
// File : controllers/users.js
const db = require('../models');
const User = db.user;
const Op = db.Sequelize.Op;

// Retrieve all Users from the database.
exports.findAll = (req, res) => {
    const email = req.query.email;
    const condition = email ? {title: {[Op.like]: `%"${email}"%`}} : null;

    User.findAll({where: condition})
        .then(data => { res.send(data); })
        .catch(err => {
            res.status(500).send({ message: err.message || "An error occurred while retrieving users." });
        });
};
```

On peut remarquer que cette fonction « *findAll* » fait intervenir un modèle « *User* ». Ce modèle est le dernier fichier essentiel au bon fonctionnement. Il nous permet de représenter la table présente en base de données sous la forme d'un objet.

```
// File : models/user.js
module.exports = (sequelize, Sequelize) => {
    const User = sequelize.define('user', {
        email: { type: Sequelize.STRING(64) },
        password: { type: Sequelize.STRING(128) }
    });

    return User;
};
```

4.6.2.6 Interrogation avec *Postman*

Pour réaliser tous les appels à l'API facilement, j'utilise l'application cliente de *Postman*¹⁸. Celle-ci permet d'envoyer des requêtes, d'inspecter la réponse et de déboguer facilement. Par exemple, une fois la base de données initialisée et un utilisateur ajouté, nous pouvons obtenir la liste de tous les utilisateurs avec la requête suivante.

¹⁸ <https://www.postman.com/product/api-client/>

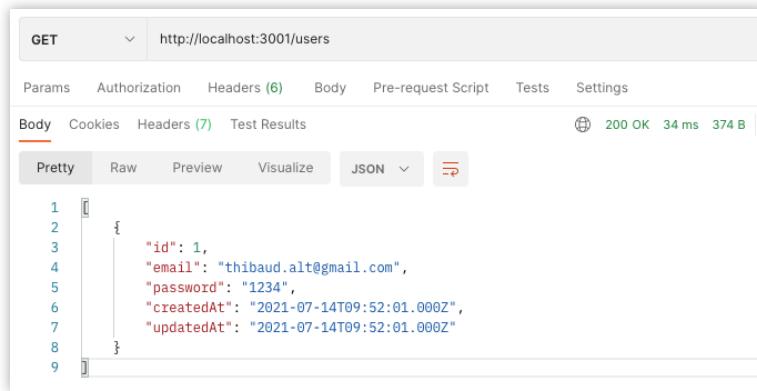


Figure 15 : Interrogation avec Postman du endpoint « /users »

4.6.3 React.js

4.6.3.1 Prérequis

- Node.js version 10.0 ou supérieur (<https://nodejs.org/en/download/>)
- npm version 5.2 ou supérieur (<https://www.npmjs.com/package/download>)

4.6.3.2 Création d'une application

La création d'une application React.js passe par un outil nommé « *Create React App* », cet outil permet de coder une application très rapidement. Cela grâce à des configurations par défaut de *webpack* et de *Babel* notamment.

Une fois les prérequis installés, il suffit de se rendre avec un terminal dans le dossier où l'on désire créer l'application et d'y d'exécuter la commande suivante.

```
$ npx create-react-app my-app
```

L'exécution de cette commande créera un répertoire appelé « *my-app* », y générera la structure initiale du projet et y installera les dépendances nécessaires.

4.6.3.3 Lancement de l'application

Une fois l'application *React.js* créée et toujours avec le terminal, il faut se rendre dans le dossier « *my-app* » et y exécuter la commande suivante qui aura pour effet de lancer le serveur.

```
$ npm start
```

À ce moment-là, une nouvelle fenêtre du navigateur s'ouvre sur l'adresse « <http://localhost:3000/> » et la nouvelle application *React.js* est prête à être codée. Cette page se rechargera automatiquement lors de la modification du code source.



Figure 16 : Point de départ d'une application React.js

4.6.3.4 Ajout des dépendances

React.js ne gère pas l'accès à l'API et le routage par défaut, c'est pourquoi il est essentiel de lui ajouter des dépendances pour gérer ce module de login. Il faut tout d'abord ajouter le module *npm* « *axios* » qui permet d'envoyer des requêtes API.

```
$ npm install axios
```

Puis, il faut ajouter le module *npm* « *react-router-dom* » permettant de gérer les chemins d'accès et le routage. C'est grâce à ce module que nous pourrons accéder par exemple à la page de connexion en ajoutant « */login* » dans l'URL.

```
$ npm install react-router-dom
```

4.6.3.5 Développement de l'application de test

Le développement de cette page de login avec *React.js* fut plutôt long et complexe. En effet, il a tout d'abord fallu créer la structure du projet, ce qui signifie qu'en *React.js* celle-ci est libre. Puis il a fallu ajouter aux *inputs* la gestion des interactions permettant ainsi de récupérer le texte entré par l'utilisateur. Cette gestion se fait à l'aide de variables d'état dans *React.js* qui sont des variables dont les valeurs peuvent être mises à jour dynamiquement et que l'on peut utiliser pour mettre à jour divers éléments de l'interface utilisateur.

```
// useState permet de déclarer et de mettre à jour Les variables d'état dans divers composants fonctionnels
// useState renvoie deux paramètres : Les variables d'état et une fonction pour mettre celles-ci à jour
const [state, setState] = useState({
  email: '',
  password: '',
})

// Cette fonction a La responsabilité de mettre à jour les variables d'état
const handleChange = (e) => {
  const {id, value} = e.target
  setState(prevState => ({
    ...prevState,
    [id]: value
  }))
}

// Lors du changement d'un input, la fonction handleChange est appelé
// Les variables d'état sont directement passées aux inputs, de ce fait tous changements de leur valeur seront
// répliqués sur l'interface
return (
  ...
  <input id="email" type="email" placeholder="Enter email" value={state.email} onChange={handleChange} />
  <input id="password" type="password" placeholder="Password" value={state.password} onChange={handleChange} />
  ...
)
```

Une fois les informations entrées par l'utilisateur récupérées, il a fallu les transmettre au *back-end* via l'API *Express* mise en place. Pour ce faire, il faut ajouter un gestionnaire d'événements déclenché lors du clic du bouton de connexion.

```
// Grâce au module axios, nous pouvons faire une requête au serveur et recevoir le résultat
// Cette fonction doit encore être améliorée pour la gestion des erreurs

const handleSubmitClick = (e) => {
  e.preventDefault();
  if(state.email.length && state.password.length) {
    const payload = {
      "email": state.email,
      "password": state.password,
    }
    axios.post(API_BASE_URL + '/auth', payload)
      .then(function(response) {
        if(response.status === 200) {
          localStorage.setItem('ACCESS_TOKEN', response.data);
          redirectToHome();
        }
      })
  }
}
```

```

        .catch(function(error) {
          props.showError(error.message);
        });
      }

// Lors du clic sur le bouton, la fonction handleSubmitClick est appelé
return (
  ...
  <button type="submit" onClick={handleSubmitClick}>Sign In</button>
  ...
)

```

Finalement, il faut encore configurer les routes qui permettront d'afficher les différentes pages de l'application d'après les URLs saisis.

```

function App() {
  const [title, updateTitle] = useState(null);
  const [errorMessage, updateErrorMessage] = useState(null);
  return (
    <Router>
      <div className="App">
        <Header title={title} />
        <Switch>
          <PrivateRoute path="/" exact={true}>
            <Home/>
          </PrivateRoute>
          <Route path="/login">
            <LoginForm updateTitle={updateTitle} errorMessage={errorMessage} updateErrorMessage={updateErrorMessage}/>
          </Route>
        </Switch>
      </div>
    </Router>
  );
}

```

Ici, nous avons déclaré deux routes, la première est « `/login` » redirigeant vers la page de connexion, la seconde est la racine « `/` » permettant d'accéder à une page d'accueil. Cette dernière est protégée par le composant `PrivateRoute` qui effectue un contrôle d'identification avant d'afficher ses composants enfants.

Le résultat ainsi que le code complet de cette page de login en `React.js` est disponible dans le *repository* du projet : https://github.com/weevoood/HEIG-VD_Travail-de-Bachelor/tree/main/4.Tests-Technos/React.js.

4.6.4 Vue.js

4.6.4.1 Prérequis

- `Node.js` version 8.9 ou supérieur (<https://nodejs.org/en/download/>)
- `npm` version 5.2 ou supérieur (<https://www.npmjs.com/package/download>)

4.6.4.2 Création d'une application

Pour créer une application `Vue.js` facilement, nous pouvons utiliser `Vue CLI`. `Vue CLI` est un système en ligne de commande complet qui aide au développement rapide d'applications `Vue.js`. Pour installer `Vue CLI`, il faut exécuter la commande suivante dans un terminal.

```
$ npm install -g @vue/cli
```

Après l'installation, le binaire « `vue` » sera disponible directement dans le terminal. Nous allons utiliser ce nouveau binaire pour créer une application `Vue.js`. Pour ce faire, il faut se rendre dans le dossier où l'on veut créer l'application et y exécuter la commande suivante.

```
$ vue create my-app
```

Il est alors possible de choisir entre `Vue 2` ou `Vue 3`, pour ce test nous utiliserons `Vue 3` qui est la dernière version de `Vue.js` contenant de nouvelles fonctionnalités.

4.6.4.3 Lancement de l'application

Une fois l'application *Vue.js* créée et toujours avec le terminal, il faut se rendre dans le nouveau dossier créé « *my-app* » et y exécuter la commande suivante qui aura pour effet de lancer le serveur.

```
$ npm run serve
```

Il faut alors se rendre, via un navigateur, sur l'adresse « <http://localhost:8080/> » pour découvrir la nouvelle application *Vue.js* et commencer à coder.

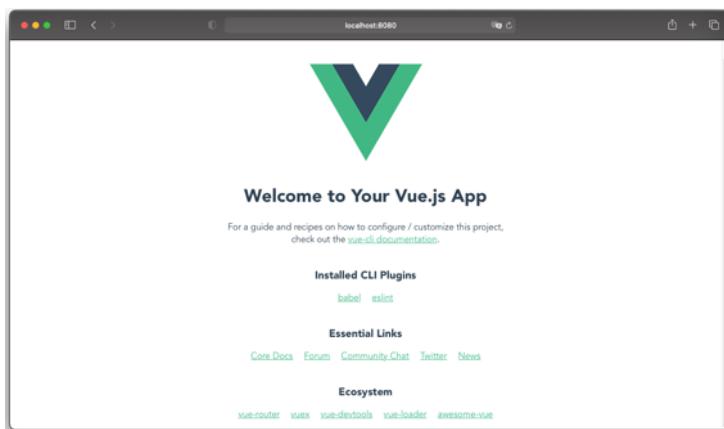


Figure 17 : Point de départ d'une application *Vue.js* fraîchement créée

4.6.4.4 Ajout des dépendances

Pour ce test avec *Vue.js*, nous aurons besoin d'ajouter certaines dépendances notamment pour gérer le routage, la validation des formulaires ou encore les appels API.

```
$ npm install vue-router@4 # The official router for Vue.js
$ npm install vuex@next # A state management pattern and library
$ npm install vee-validate@next # A form validation
$ npm install yup # A schema builder for value parsing and validation
$ npm install axios # Promise based HTTP client for the browser and node.js
```

4.6.4.5 Développement de l'application de test

La première étape de création de ce formulaire de login avec *Vue.js* est de créer un service d'authentification. Ce service, servant d'interface, fournira les méthodes de connexion et de déconnexion nécessaire à l'application en faisant appel au module *axios* et en utilisant donc les requêtes vers l'API.

```
// File : services/auth.service.js
import axios from 'axios';

const API_URL = 'http://localhost:3001/';
class AuthService {
  ...
  login(user) {
    return axios
      .post(API_URL + 'auth', {
        email: user.email,
        password: user.password
      })
      .then(response => {
        if(response.data.token) { return this.me(response.data.token) }
      });
  }
  ...
}
export default new AuthService();
```

La deuxième étape consiste à ajouter VueX, le gestionnaire d'état de *Vue.js*. Celui-ci sert de zone de stockage de données centralisée pour tous les composants d'une application. Pour le service d'authentification, nous allons donc créer un fichier permettant de gérer les différents états intervenant dans celui-ci.

```
// File: store/auth.module.js

import AuthService from '../services/auth.service';

const user = JSON.parse(localStorage.getItem('user'));
const initialState = user ? {status: {loggedIn: true}, user} : {status: {loggedIn: false}, user: null};

export const auth = {
  namespaced: true,
  state: initialState,
  actions: {
    login({commit}, user) {
      return AuthService.login(user).then(
        user => {
          commit('loginSuccess', user);
          return Promise.resolve(user);
        },
        error => {
          commit('loginFailure');
          return Promise.reject(error);
        }
      );
    },
    logout({commit}) {
      AuthService.logout();
      commit('logout');
    }
  },
  mutations: {
    loginSuccess(state, user) {
      state.status.loggedIn = true;
      state.user = user;
    },
    loginFailure(state) {
      state.status.loggedIn = false;
      state.user = null;
    },
    logout(state) {
      state.status.loggedIn = false;
      state.user = null;
    }
  }
};
```

Nous pouvons alors créer les composants et le formulaire d'authentification. Le module *yup* permet aux champs d'entrées, ici l'email et le mot de passe, d'être directement validés grâce à un schéma déclaré dans la fonction *data*. La fonction *handleLogin* est appelée lors de la validation du formulaire, celle-ci utilise le module *VueX* déclaré précédemment pour rediriger l'utilisateur vers sa page de profil une fois connecté ou le cas échéant afficher un message d'erreur.

```
// File : components/Login.vue

<template>
<Form class="flex flex-col" @submit="handleLogin" :validation-schema="schema">
  <div class="mb-6 pt-3 rounded bg-gray-200">
    <label for="email">Email</label>
    <Field id="email" name="email" type="text"/>
    <ErrorMessage name="email" class="error-feedback"/>
  </div>
  <div class="mb-6 pt-3 rounded bg-gray-200">
    <label for="password">Password</label>
    <Field id="password" name="password" type="password"/>
    <ErrorMessage name="password" class="error-feedback"/>
  </div>
  <button :disabled="loading">Sign In</button>
</Form>
</template>
```

```
<script>
import {Form, Field, ErrorMessage} from 'vee-validate';
import * as yup from 'yup';

export default {
  name: 'Login',
  components: {Form, Field, ErrorMessage},
  data() {
    const schema = yup.object().shape({
      email: yup.string().required("Email is required!"),
      password: yup.string().required("Password is required!"),
    });
    return {
      schema,
      loading: false,
      message: '',
    };
  },
  methods: {
    handleLogin(user) {
      this.loading = true;
      this.$store.dispatch('auth/login', user).then(
        () => { this.$router.push('/profile'); },
        (error) => { this.loading = false; this.message = error.message }
      );
    }
  }
};
</script>
```

Comme pour *React.js*, il reste à déclarer les routes gérées ici par le module *vue-router*. Ce module permet d'exécuter une fonction avant de rediriger l'utilisateur via sa fonction *beforeEach*. Ici nous l'utilisons pour vérifier que l'utilisateur est connecté avant qu'il accède à sa page de profil.

```
// File : router.js
import {createWebHistory, createRouter} from 'vue-router';
import Login from './components/Login.vue';
const Profile = () => import('./components/Profile.vue')
const routes = [
  {
    path: '/',
    name: 'login',
    component: Login,
  },
  {
    path: '/profile',
    name: 'profile',
    component: Profile,
  },
  ...
];
const router = createRouter({ history: createWebHistory(), routes });

router.beforeEach((to, from, next) => {
  const publicPages = ['/login', '/register'];
  const authRequired = !publicPages.includes(to.path);
  const loggedIn = localStorage.getItem('user');

  // trying to access a restricted page + not Logged in = redirect to Login page
  if (authRequired && !loggedIn) { next('/login') ; }
  else { next() ; }
});

export default router ;
```

Le résultat ainsi que le code complet de cette page de login en *Vue.js* est disponible dans le *repository* du projet : https://github.com/weevood/HEIG-VD_Travail-de-Bachelor/tree/main/4.Tests-Technos/Vue.js.

4.6.5 Angular

4.6.5.1 Prérequis

- *Node.js* version 12.14 ou supérieur (<https://nodejs.org/en/download/>)
- *npm* version 6.12 ou supérieur (<https://www.npmjs.com/package/download>)

4.6.5.2 Création d'une application

Pour créer une application *Angular* facilement, nous pouvons utiliser *Angular CLI*. *Angular CLI* est un système en ligne de commande qui permet de créer des projets *Angular*, de générer du code, d'intégrer des bibliothèques et d'effectuer diverses tâches de développement (les tests, le déploiement, etc.) Pour installer *Angular CLI*, il faut exécuter la commande suivante dans un terminal.

```
$ npm install -g @angular/cli
```

Après l'installation, le binaire « *ng* » sera disponible directement dans le terminal. Nous allons utiliser ce nouveau binaire pour créer une application *Angular*. Pour ce faire, il faut se rendre dans le dossier où l'on veut créer l'application et y exécuter la commande suivante.

```
$ ng new my-app
```

Cette commande nous invite à fournir des informations sur les fonctionnalités à inclure dans l'application initiale. Pour ce test, nous utiliserons les valeurs configurées par défaut en appuyant directement sur la touche « *Enter* ».

4.6.5.3 Lancement de l'application

Angular CLI comprend un serveur permettant d'exécuter une application *Angular* localement. Pour le lancer, il faut accéder au dossier créé précédemment « *my-app* » et y exécuter la commande suivante.

```
$ ng serve -open
```

Cette commande lance le serveur et reconstruira l'application lorsque des modifications seront apportées au code source. De plus, l'option « *--open* » ouvre automatiquement une nouvelle page dans un navigateur sur l'adresse « <http://localhost:4200/> » contenant l'application *Angular*.

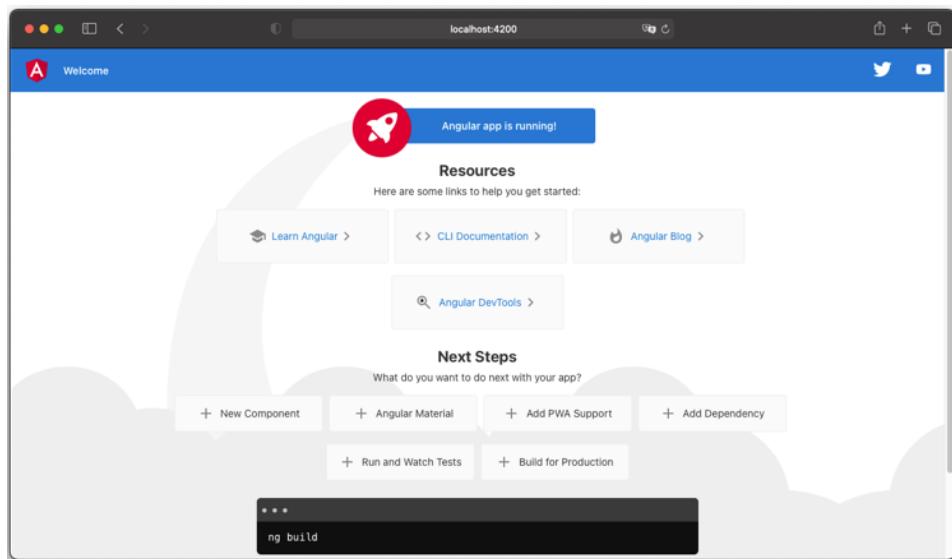


Figure 18 : Page d'accueil d'un projet Angular lors de sa création

4.6.5.4 Développement de l'application de test

Avec *Angular* il n'y a pas besoins d'ajouter de dépendances. En effet celui-ci embarque déjà toutes les fonctionnalités requises. Par sa nature orientée-objet et sa surcouche *TypeScript*, le code d'*Angular* est beaucoup plus verbeux et les fichiers plus nombreux. Dans *Angular*, à la différence de *React.js* et de *Vue.js*, le HTML et le JS ne sont pas écrits

dans le même fichier. De ce fait pour créer notre page de login, nous aurons besoin de créer trois fichiers en commençant par le fichier HTML.

```
// File : app/account/Login.component.html

<form class= »flex flex-col » [formGroup]= »form » (ngSubmit)= »onSubmit() »>
...
<input type= »email » formControlName= »email » [ngClass]= »{ 'is-invalid' : submitted && f.email.errors } »/>
<div *ngIf= »submitted && f.email.errors » class= »invalid-feedback »>
  <div *ngIf= »f.email.errors.required »>email is required</div>
</div>
...
<input type="password" formControlName="password" [ngClass] = "{'is-invalid': submitted && f.password.errors }"/>
<div *ngIf="submitted && f.password.errors" class="invalid-feedback">
  <div *ngIf="f.password.errors.required">Password is required</div>
</div>
...
<button type="submit" [disabled]= "loading">
  <span *ngIf="loading" class="spinner-border spinner-border-sm mr-1"></span>Sign In
</button>
</form>
```

Puis par le fichier *TypeScript* correspondant et répondant aux fonctions appelées dans le *HTML*.

```
// File : app/account/Login.component.ts

@Component({templateUrl: 'login.component.html'})
export class LoginComponent implements OnInit {
  form!: FormGroup;
  loading = false;
  submitted = false;

  ...
  ngOnInit() {
    this.form = this.formBuilder.group({
      email: ['', Validators.required],
      password: ['', Validators.required]
    });
  }
  ...
  onSubmit() {
    this.submitted = true;
    this.alertService.clear(); // reset alerts on submit
    if (this.form.invalid) { return; } // stop here if form is invalid
    this.loading = true;
    this.accountService.login(this.f.email.value, this.f.password.value)
      .pipe(first())
      .subscribe({
        next: () => { // get return url from query parameters or default to home page
          const returnUrl = this.route.snapshot.queryParams['returnUrl'] || '/';
          this.router.navigateByUrl(returnUrl);
        },
        error: (error: any) => { this.loading = false; this.alertService.error(error); }
      });
  }
}
```

La définition des routes est très simple dans *Angular* puisque nativement supportée. Celle-ci se fait dans chaque composant, ce qui permet une lecture simplifiée.

```
// File: app/account/account-routing.module.ts

const routes: Routes = [
  path: '', component: LayoutComponent,
  children: [
    {path: 'login', component: LoginComponent},
    {path: 'register', component: RegisterComponent}
  ]
];
```

Finalement, il nous reste à définir le service permettant de communiquer avec l'API. Ici aussi, nul besoin d'ajouter un module supplémentaire puisque *Angular* met à disposition son service de requête nommé « *http* ».

```
// File : app/_services/account.service.ts

export class AccountService {
    private userSubject: BehaviorSubject<User>;
    public user: Observable<User>;
    constructor(private router: Router, private http: HttpClient) {
        this.userSubject = new BehaviorSubject<User>(JSON.parse(localStorage.getItem('user')));
        this.user = this.userSubject.asObservable();
    }
    ...
    login(email: any, password: any) {
        return this.http.post<User>(`${environment.apiUrl}/auth`, {email, password})
            .pipe(switchMap(data => {
                return this.http.post<User>(`${environment.apiUrl}/users/me`, {token: data.token})
                    .pipe(map(user => {
                        localStorage.setItem('user', JSON.stringify(user));
                        this.userSubject.next(user);
                        return user;
                    }));
            }));
    }
    logout() {
        localStorage.removeItem('user');
        this.userSubject.next(null);
        this.router.navigate(['/account/login']);
    }
}
```

Le résultat ainsi que le code complet de cette page de login avec *Angular* est disponible dans le *repository* du projet : https://github.com/weevoood/HEIG-VD_Travail-de-Bachelor/tree/main/4.Tests-Technos/Angular.

4.6.6 Choix des frameworks

On trouve d'innombrables comparatifs de frameworks JavaScript sur internet, au niveau de la performance, de la courbe d'apprentissage, de la popularité, des visions futures, etc. J'aurais ainsi pu choisir l'un de ces frameworks simplement sur ces points techniques, mais grâce à ces trois pages de connexion réalisées à l'aide de *React.js*, de *Vue.js* puis *d'Angular*, j'ai pu davantage me familiariser avec la syntaxe de base de ces frameworks et ainsi découvrir les fonctionnements et les mécanismes implémentés par chacun d'entre eux. Cela m'a permis d'y voir plus clair et de mieux comprendre quel framework choisir pour quelle situation.

Tout d'abord au niveau des modules et des fonctionnalités *React.js* et *Vue.js* jouent la carte de la légèreté en n'incluant pas par défaut de nombreux éléments (routage, requêtes, etc.). Au contraire *d'Angular* qui se veut un framework « *tout-en-un* » incluant donc par défaut ces éléments utilisés dans la plupart des projets. Ces deux visions se confirment quand on observe et compare la taille des projets réalisés.

 Vue.js Modified: Today, 15:33	117.7 MB	 Angular Modified: Today, 15:26	321.7 MB	 React.js Modified: Today, 15:33	162.2 MB
---	----------	--	----------	---	----------

Figure 19 : Comparaison du poids de chacun des projets réalisés

Le choix du bon framework JavaScript est évidemment crucial pour ce projet, car celui-ci ne pourra pas (ou très difficilement) être changé par la suite. Comme il s'agit d'un choix difficile, j'ai décidé d'établir une liste des besoins de l'application. De cette liste, je pourrais identifier les besoins couverts ou non par les trois frameworks et ainsi faire un choix définitif. Ces trois frameworks couvrant déjà les besoins initiaux d'une application web (compatibilité, accessibilité, performances, sécurité) ceux-ci sont écartés de la liste. La liste des besoins essentiels de mon application est donc la suivante :

- Un pattern de gestionnaire d'état (« *state management pattern* »)
- Une communication avec des APIs
- Un système de routage efficace
- Un système de gestion de *template*
- Une gestion des erreurs performante
- Une architecture solide

Au vu de cette liste, *React.js* ne répond pas aux besoins du moins dans sa version initiale sans l'ajout de plusieurs dépendances externes, c'est pourquoi il est écarté des choix. *Vue.js*, dans sa troisième version avec *Vuex*, couvre la plupart des points. Quant à *Angular*, il embarque l'ensemble de ces besoins et semble donc être le choix judicieux. On comprend alors mieux pourquoi la courbe d'apprentissage d'*Angular* est considérée comme plus compliquée, du fait des fonctionnalités initiales mises à dispositions. Cependant si l'on ajoute à *React.js* et *Vue.js* toutes les dépendances nécessaires pour répondre à ces mêmes besoins, ceux-ci deviennent également plus denses et plus difficiles.

Ces trois frameworks, avec plus ou moins d'extensions, permettent donc tous de répondre aux mêmes besoins. Toutefois, chacun d'entre eux à sa « *spécialité* » qui peut être établit comme suit :

- *React.js* : idéal pour concevoir des composants individuels spécifiques à ajouter à un site web existant
- *Vue.js* : idéal pour créer des applications web rapidement ou pour transformer progressivement un site web existant
- *Angular* : idéal pour créer une nouvelle application complexe avec des bases solides

Angular, dans sa version *TypeScript*, paraît être un framework robuste et bien structuré. Cependant il est difficile à appréhender et partant de zéro, il me semble trop complexe à mettre en place dans les délais que j'ai à disposition.

Finalement, c'est donc assez naturellement que je choisis *Vue.js* en complément d'*Express* sur *Node.js* pour réaliser mon application web. Plébiscité pour sa facilité de prise en main, sa productivité et ses performances, *Vue.js* adopte les meilleurs compromis entre puissance, simplicité d'apprentissage et plaisir d'utilisation. Celui-ci devrait me permettre d'obtenir des résultats assez rapidement tout en mettant en place une architecture solide, durable et évolutive.



Figure 20 : Choix final des frameworks JavaScript back-end et front-end

5 SYSTÈME DE GESTION DE BASE DE DONNÉES

Un système de gestion de base de données, souvent abrégé « *SGBD* » est un logiciel système permettant de stocker, de manipuler et de gérer des données dans une base de données. La principale fonction d'un *SGBD* est de réaliser ces manipulations et opérations en diminuant leur complexité et en garantissant la qualité, la pérennité et la confidentialité des informations.

La plupart des *SGBD* permettent de manipuler des bases de données relationnelles qui sont encore aujourd'hui les plus courantes, mais il en existe également pour les autres types de modèles. Parmi les différents types de modèles existants (hiérarchiques, réseau, relationnel, orienté objet, documents, graphe et XML), j'en ai identifié trois qui peuvent être intéressants pour la réalisation de mon application.

5.1 Base de données relationnelle

Le modèle de bases de données relationnelles a été introduit par l'informaticien *Edgar F. Codd* en 1970. Il s'agit du modèle de base de données le plus courant et le plus répandu aujourd'hui. Dans les bases de données relationnelles, l'information est organisée dans des tableaux à deux dimensions appelés « *tables* » qui contiennent un ensemble d'enregistrements (les lignes). Les tables peuvent être assemblées entre-elles par des « *relations* » et connectées par les valeurs qu'elles contiennent (clé primaire - clé étrangère) ou par des tables de relations intermédiaires.

5.1.1 MariaDB



MariaDB est un « *fork* » communautaire de *MySQL* destiné à rester un logiciel libre, né suite à l'acquisition de *MySQL* par Oracle Corporation en 2009. Son développement, sous la licence publique GNU, est assuré par certains des développeurs originaux de *MySQL* ainsi que par une grande communauté de développeurs. *MariaDB* maintient une compatibilité élevée avec *MySQL*, ce qui lui confère de nombreux points communs ainsi qu'une capacité de remplacement de *MySQL* très simple.

5.1.1.1 Installation

Pour simplifier l'installation et le déploiement ainsi que pour pouvoir recréer facilement un environnement de développement, j'ai choisi d'utiliser un conteneur *Docker*¹⁹ pour déployer ma base de données *MariaDB*. J'ai créé un fichier « *docker-compose* » pour définir les différents services à démarrer en appui de *MariaDB* ainsi que les scripts initiaux à exécuter. De ce fait, pour installer la base de données en local, il suffit d'exécuter la commande suivante dans le dossier « *docker* » une fois *Docker* installé sur la machine hôte.

```
$ docker-compose up -d
```

L'interface de gestion *phpMyAdmin* est disponible en accédant à l'adresse : <http://localhost:8000>. Nous disposons également des deux commandes à dispositions permettant de faire un import ou un export des données de la base.

```
$ docker exec -i mariadb mysql -uroot -ppwd DB_NAME < db_dump.sql # Importation
$ docker exec -i mariadb mysql -uroot -ppwd DB_NAME > db_dump.sql # Exportation
```

5.1.2 Utilisations dans l'application

Ce modèle de bases de données relationnelles avec *MariaDB*, pourrait nous permettre de stocker les entités des différents acteurs de l'application et les relations présentent entre elles. Cependant les liens entre ces différentes entités vont très vite devenir nombreux et difficiles à gérer avec un modèle relationnel. C'est pourquoi j'ai choisi d'utiliser un modèle orienté graphe, décrit dans les points suivants, pour mieux gérer les relations entre les utilisateurs, les équipes et les projets.

Ce modèle relationnel, et plus particulièrement *MariaDB*, va me permettre de stocker les données qui n'entretiennent pas de multiples relations entre elles ou qui n'apportent pas d'informations pertinentes directes à

¹⁹ <https://www.docker.com/>

une relation. Ça sera notamment le cas des notifications, des logs de l'application, des informations d'authentifications des utilisateurs, des historiques des discussions, etc.

Certaines entités, comme les utilisateurs, pourront être représentées dans les deux modèles (relationnel et graphe). Mais les données stockées dans l'une ou l'autre base ne devront pas être redondantes. Pour définir si une propriété doit être stockée dans le modèle relationnel ou dans le modèle graphe, il faut se demander si celle-ci apporte une information pertinente aux relations. Si tel est le cas, il faut stocker l'information dans le modèle graphe, sinon il faut la stocker dans le modèle relationnel.

5.1.3 Schéma initial

Pour visualiser la base de données, j'ai décidé de réaliser un schéma initial simplifié omettant volontairement les aspects de multilinguisme. Ceux-ci seront ajoutés et présentés dans le point suivant.

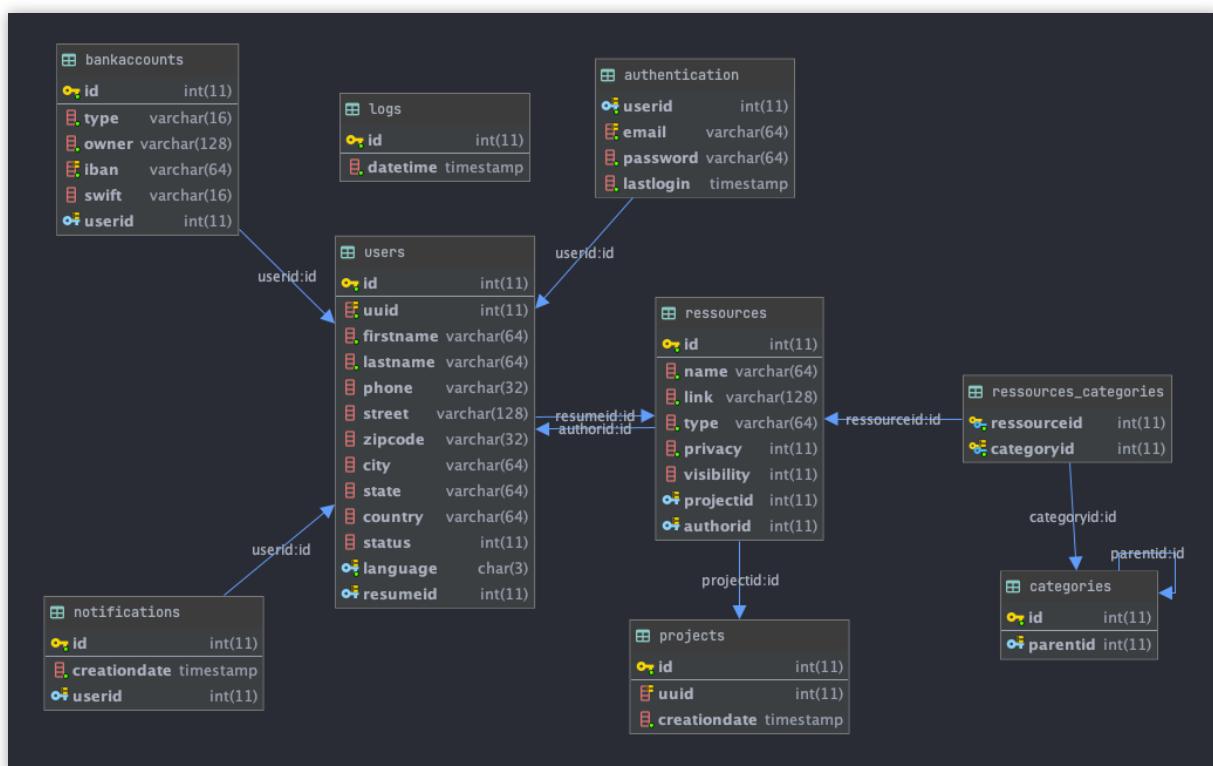


Figure 21 : Schéma initial de la base de données relationnelle

Dans ce schéma simplifié, les points importants à noter sont que :

- Les tables « *authentication* » et « *bankaccounts* » sont volontairement séparées de la table « *users* » dans le but de pouvoir facilement les remplacer par des microservices.
- Un utilisateur n'est pas relié à un projet, ces liens se feront via la base de données graphe
- Un utilisateur peut posséder un CV représenté sous forme de « *ressources* » et lié via son champ « *resumeid* » ; un projet peut référer plusieurs « *ressources* » (Cahier des charges, planning, rapport, etc.) et celles-ci peuvent avoir un auteur. Dans aucun cas, cette table « *ressources* » ne doit servir à créer des liens entre un utilisateur et un projet

5.1.4 Multilinguisme

Une fois le schéma de base identifié, il est nécessaire de lui ajouter une couche incluant la gestion multilingue. Pour se faire, j'ai choisi d'utiliser deux approches en parallèle.

1. La première est une approche par « ajout de table de traduction ». Celle-ci va me permettre de traduire les tables « *projects* » et « *notifications* ». Les avantages de cette approche sont : une approche relationnelle

propre et normalisée, l'ajout d'une nouvelle langue qui ne nécessite pas de modifications de schéma et l'interrogation relativement simple (une seule jointure sera requise).

- La seconde est une approche par « ajout d'une table clé-valeur unique » ; dans mon schéma la table « *translations* ». Celle-ci va me permettre de traduire tous les textes qui ne sont pas directement liés à un objet ; par exemple les éléments de l'interface graphique. Les avantages de cette approche sont liés au fait que l'ajout d'une nouvelle langue qui ne nécessite pas de modifications de schéma et que toutes les traductions sont stockées dans un seul endroit, ceci permettant une mise en cache très simple.

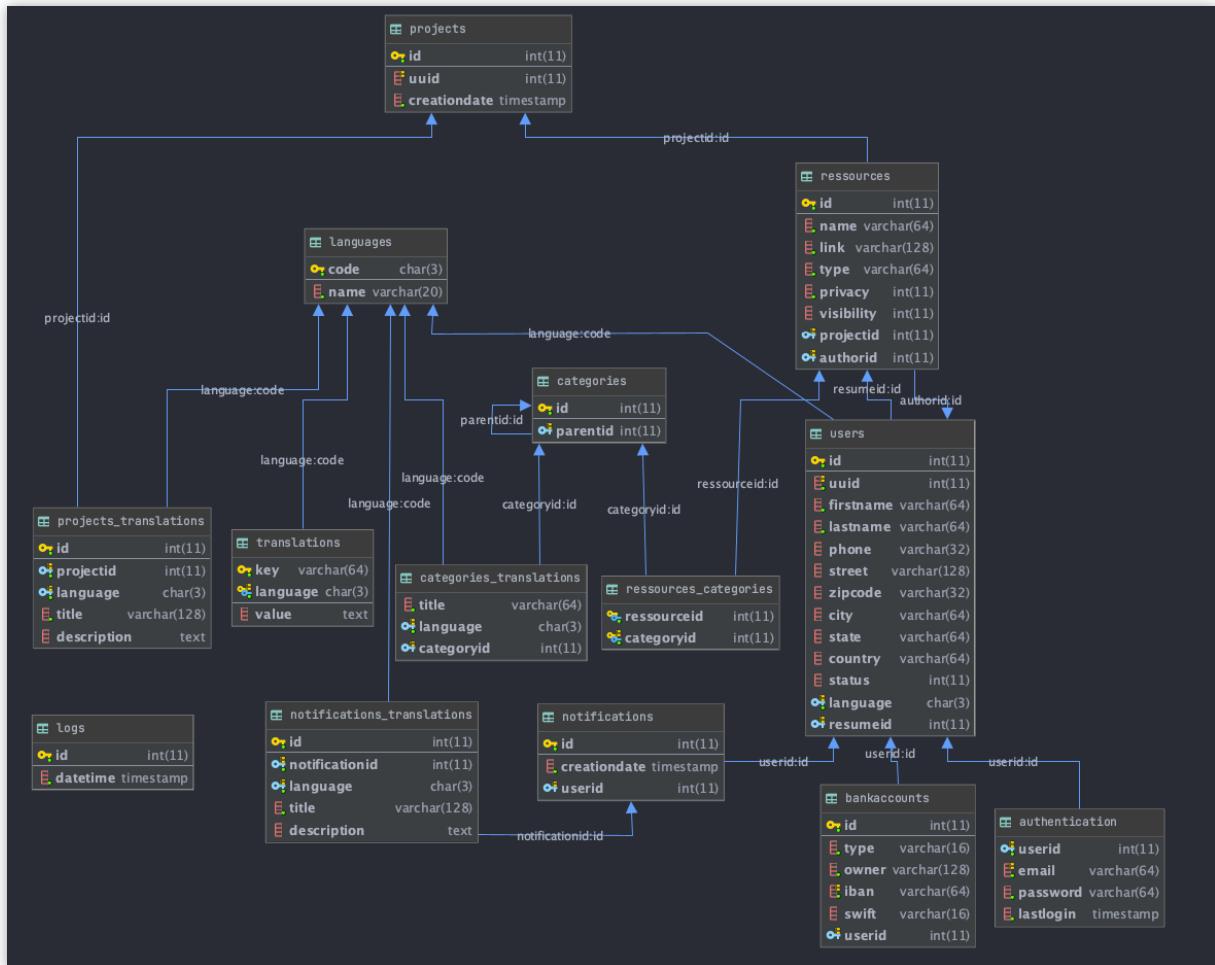


Figure 22 : Schéma de la base de données relationnelle incluant la gestion multilingue

Les points importants à noter après cet ajout de multilinguisme sont :

- En plus de la table « *users* », toutes les tables de traductions possèdent une relation avec la table « *languages* ». Cette dernière permettra de stocker toutes les langues sous la forme de couple : « *code ISO 639 – nom* ».
- Les titres et descriptions des projets et des notifications, sont déplacés dans leurs tables de traductions respectives.
- La table « *translations* » permettant de stocker les traductions d'ordre général, ne possède pas d'`id`. Sa clé primaire est formée avec le couple « *key – language* » qui se doit d'être unique.

5.2 Base de données orientée documents

Une base de données orientée documents est, comme son nom l'indique, un système de stockage conçu pour stocker, récupérer et gérer des documents. Ce modèle est également connu sous le terme de *semi-structuré*, dans lequel il n'y a pas de séparation nette entre les données et le schéma.

La principale différence entre les bases de données orientée documents et les bases de données relationnelles réside dans la manière de stocker et de récupérer les objets. Dans les bases de données relationnelles, les données sont stockées dans des tables distinctes définies et un seul objet peut être réparti sur plusieurs tables. Dans les bases de données orientées documents, les informations d'un objet sont stockées dans une seule instance de la base de données et chaque objet stocké peut ainsi être différent des autres. Cela permet d'éliminer des jointures pour reconstituer l'information ainsi que le mappage *objet-relationnel* lors du chargement des données.

Les objectifs d'une base de données orientée documents est la représentation des informations ayant des besoins de flexibilité, de richesse de la structure, d'autonomie ou de sérialisation. Un document peut être : une valeur atomique, une paire clé-valeur, un tableau de valeurs, un agrégat de paires clé-valeur ou une composition des possibilités précédentes. Il est souvent représenté avec du *XML* ou du *JSON*.

5.2.1 MongoDB



<https://commons.wikimedia.org/wiki/File:MongoDB-Logo.svg>

MongoDB est un système de gestion de base de données orientée documents utilisant des documents *JSON* avec des schémas facultatifs. *MongoDB* est développé depuis 2009 par *MongoDB Inc.* sous licence *SSPL*.

5.2.2 Utilisations dans l'application

MongoDB pourrait éventuellement être utilisé dans l'application pour la gestion des différentes traductions, pour stocker des ressources (*articles*, *fichiers*), ou pour d'autres besoins spécifiques.

5.3 Base de données orientée graphe

Les bases de données orientées graphe sont similaires aux bases de données orientées documents, mais ajoutent une couche de relation leur permettant de lier des documents et de les traverser plus rapidement. Elles utilisent la théorie des graphes (avec des nœuds reliés par des arcs) pour représenter et stocker les données. Les bases de données orientées graphes apportent des performances accrues pour traiter des données fortement connectées en évitant les nombreuses jointures très coûteuses qu'il faudrait mettre en place dans les bases de données relationnelles. De plus, la modélisation des bases de données orientée graphe est plus facile, car elle ne s'appuie pas sur un schéma rigide et peut s'adapter au fur et à mesure à des modèles complexes.

5.3.1 Neo4j



https://commons.wikimedia.org/wiki/File:Neo4j-logo_color.png

Neo4j est un système de gestion de base de données orientée graphe développé en Java depuis 2000 par la société suédo-Américaine *Neo Technology*. *Neo4j* est construite pour être extrêmement performante dans le traitement des liens entre les nœuds, notamment grâce au pré calcul des jointures au moment de l'écriture des données. Les requêtes *Neo4j* utilisent le langage *Cypher*²⁰ qui se veut simple et efficace dans sa formulation.

5.3.1.1 Installation

Tout comme pour *MariaDB*, la base de données *Neo4j* sera déployée dans un conteneur *Docker*. Pour ce faire, j'ai mis à jour le fichier « *docker-compose.yml* » avec les informations nécessaires à la mise en place de *Neo4j*.

```
$ docker-compose up -d
```

Une fois la commande précédente exécutée, la base de données *Neo4j* est disponible via un navigateur à l'adresse <http://localhost:7474/browser>.

5.3.2 Utilisations dans l'application

Dans la plateforme web qui va être développée, ce modèle correspond tout à fait à la gestion des différentes relations possibles entre les mandants, les développeurs, les projets, les experts, etc. C'est pourquoi, je vais utiliser une base de données orientée graphe, décrite ci-après, pour gérer ces relations qui peuvent très vite devenir

²⁰ <https://neo4j.com/developer/cypher/>

complexes. Une fois ces relations mises en place, il sera par exemple possible de proposer à un mandant d'autres équipes de développeurs semblables, si son équipe favorite n'est pas disponible. De plus, il sera possible de proposer des projets à telle ou telle équipe de développeurs avec un ciblage de caractéristiques et selon les différentes relations développées jusqu'alors.

5.3.3 Nœuds

Les différentes entités de l'application seront représentées sous la forme de nœuds. Les nœuds peuvent posséder une ou plusieurs propriétés sous la forme de « *clé - valeur* », ce qui dans un modèle relationnel correspondrait aux entrées d'une table. Les nœuds peuvent également être étiquetés permettant ainsi de les grouper par similarités. Par exemple un nœud « *utilisateur* » pourrait avoir les propriétés *nom*, *prénom*, *âge* et les étiquettes *développeur* et *expert*. Dans l'application, il y aura trois nœuds permettant de créer différentes relations.

5.3.3.1 Utilisateur

Le premier nœud est l'entité « *utilisateur* ». Celui-ci permet de représenter les différents types d'utilisateurs possibles, à savoir : les développeurs, les mandants, les modérateurs et les experts. Ces différents types seront attribués aux utilisateurs via les étiquettes et pourront être combinés. Un utilisateur pourra alors être développeur dans un projet A et mandant dans un projet B.

Les propriétés définies ici ne doivent pas être redondantes avec les propriétés définies dans le modèle relationnel. Pour définir si une propriété doit être stockée ici, il faut se demander si celle-ci apporte une information pertinente aux relations. De ce fait, les propriétés de cette entité « *utilisateur* » pouvant ajouter de l'information sont les suivantes.

uuid	Identifiant unique universel permettant de retrouver l'entité du modèle relationnel correspondante
tags	Texte, étiquettes de caractéristiques discriminatoires (ex : domaines de compétences)
createdAt	<i>DateTime</i> , date et heure de création de l'utilisateur

5.3.3.2 Équipe

La deuxième entité est une « *équipe* ». Celle-ci pourra être composée de 1 à plusieurs utilisateurs et permettra de regrouper ceux-ci notamment pour créer des équipes de développement ou des équipes d'utilisateurs représentants un mandant.

Les propriétés de cette entité « *équipe* » sont les suivantes. Dans le modèle relationnel, il n'y a pour l'instant pas d'entité « *équipe* », de ce fait tous les champs nécessaires à une équipe sont enregistrés ici en tant que propriété. Par souci d'évolutivité, j'ai choisi de tout de même ajouter un *uuid* à cette entité. Celui-ci permettra si besoin de la relier au modèle relationnel.

uuid	Identifiant unique universel
name	Texte, nom de l'équipe
color	Texte, une couleur associée à l'équipe
status	Texte, le statut actuel de l'équipe (actif, abandonnée, bannis)
createdAt	<i>DateTime</i> , date et heure de création de l'équipe

5.3.3.3 Projet

La troisième entité représentée sous forme de nœud est un « *projet* ». Cette entité permet de représenter les projets à réaliser, en cours de réalisation ou terminés. Ces nœuds projets pourront être étiquetés en fonction du ou des langages de programmation auxquels ils se rapportent.

Les propriétés de cette entité « *projet* » sont les suivantes : comme pour les utilisateurs, celles-ci ne doivent pas être redondantes avec les données stockées dans le modèle relationnel.

uuid	Identifiant unique universel permettant de retrouver l'entité du modèle relationnel correspondante
status	Texte, le statut actuel du projet (<i>préparation, validation, proposition, en cours, terminé, abandonné</i>)
deadline	Date, la date programmée de la fin du projet
tags	Texte, étiquettes de caractéristiques discriminatoires (ex : domaine d'application)
createdAt	<i>DateTime</i> , date et heure de création du projet

5.3.4 Relations

Maintenant que nous avons identifié les différents acteurs de l'application sous forme de nœuds, analysons les relations possibles et les arcs qui vont pouvoir être créés entre ces nœuds. Les relations possèdent également des propriétés et sont orientées (dans *Neo4j*, elles peuvent être traitées comme non-orientées). Par souci de simplification, les nœuds n'ont pas été étiquetés dans la représentation suivante et sont donc tous représentés dans leur type primaire.

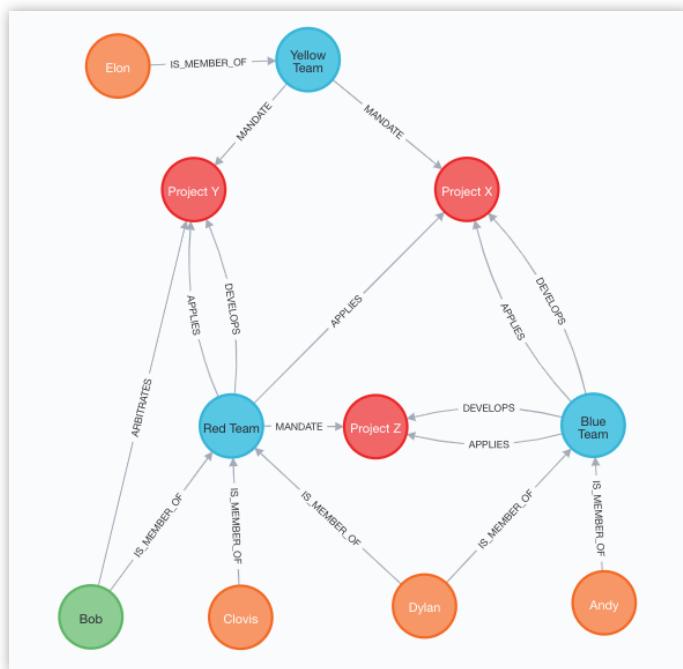


Figure 23 : Représentation simplifiée des différents nœuds et arcs possibles

5.3.4.1 IS_MEMBER_OF

La première relation identifiée est la relation d'appartenance d'un utilisateur à une équipe, représentée ici par les arcs « *IS_MEMBER_OF* ». Ainsi un utilisateur peut faire partie d'une ou de plusieurs équipes ; sur l'exemple c'est le cas de *Dylan* qui fait partie de l'équipe *rouge* et de l'équipe *bleue*. Les propriétés de cet arc sont les suivantes :

since	Date, depuis combien de temps
isOwner	Booléen, l'utilisateur est-il le propriétaire de l'équipe
status	Texte, le statut actuel de l'appartenance de l'utilisateur à l'équipe (actif, inactif, banni)

5.3.4.2 MANDATES

La relation « *MANDATES* » permet à une équipe de mandater un ou plusieurs projets. Dans l'exemple ci-dessus, l'équipe *jaune* mandate les projets *X* et *Y*. Il est tout à fait possible qu'une équipe développe un projet et en mandate un autre. C'est le cas de l'équipe rouge dans l'exemple précédent. Les propriétés de cet arc sont les suivantes :

<i>publishDate</i>	Date, date de publication du projet
<i>endDate</i>	Date, la date de fin si le projet est terminé
<i>mark</i>	Nombre, note attribuée par le mandant si le projet est terminé
<i>feed-back</i>	Texte, rapport sur le déroulement si le projet est terminé

5.3.4.3 APPLIES

La relation « *APPLIES* » permet à une équipe de développeurs de soumettre sa candidature pour un projet à réaliser. Les propriétés de cet arc sont les suivantes.

<i>date</i>	Date, date de soumission de la candidature
<i>price</i>	Nombre, prix proposé par l'équipe de développeurs
<i>specifications</i>	Lien, lien vers le cahier des charges proposé

5.3.4.4 DEVELOPS

La relation « *DEVELOPS* » permet de lier une équipe de développeurs à un ou plusieurs projets. Dans l'exemple ci-dessus, l'équipe *bleue* développe les projets X et Z. Cette relation est forcément créée en parallèle d'une relation « *APPLIES* » existante puisque les équipes doivent soumettre leur candidature avant d'être sélectionnées. Les propriétés de cet arc sont les suivantes.

<i>startDate</i>	Date, date d'attribution du projet à l'équipe
<i>endDate</i>	Date, la date de fin si le projet est terminé

5.3.4.5 ARBITRATES

La relation « *ARBITRATES* » pourra être établie entre un expert (utilisateur) et un projet lui permettant ainsi d'intervenir sur le projet en question. Les experts peuvent être externes ou alors intégrés à des équipes de développeurs. C'est le cas de *Bob* dans l'exemple. Les propriétés de cet arc sont les suivantes.

<i>date</i>	Date de l'intervention
<i>type</i>	Texte, type d'intervention (expertise, médiation, validation, etc.)
<i>status</i>	Texte, état de l'intervention (<i>en cours</i> , <i>en échec</i> , <i>terminé</i>)

6 PLANIFICATION

La réalisation de l'application va être faite sur 24 jours. Celle-ci inclut le développement des fonctionnalités, les tests liés et les résolutions de bugs. Cette planification sera divisée en cinq *sprints* selon la méthode *Agile*²¹.

Pour faciliter la gestion des tâches, réaliser le suivi et visualiser la progression, j'ai choisi d'utiliser le service en ligne *Trello*²². J'ai dans un premier temps reporté dans celui-ci les cinq *sprints* et leurs tâches principales. Tout au long de la réalisation, les tâches seront déplacées des colonnes de gauche vers celles de droite selon leur état d'avancement. Ce tableau est visible via l'adresse suivante : <https://trello.com/b/meyHR8e8/heig-vd-travail-de-bachelor>.

6.1 Sprint N°1 : Développement de l'API

Durée	2 semaines (6 jours). Du 28.07.21 au 30.07.21 et du 04.08.21 au 06.08.21
Goal	Déployer une API fonctionnelle permettant de réaliser toutes les opérations nécessaires aux différents <i>Epic</i> sélectionnés
Tâches	<ul style="list-style-type: none"> - Mise en place de la structure - Création des « <i>endpoints</i> » - Rédaction de tests unitaires
Validation	Approbation de tous les tests unitaires définis

6.2 Sprint N°2 : Développement des interfaces utilisateur

Durée	2 semaines (6 jours). Du 11.08.21 au 13.08.21 et du 18.08.2021 au 20.08.2021
Goal	Produire une application permettant aux utilisateurs de réaliser toutes les opérations nécessaires aux différents <i>Epic</i> sélectionnés
Tâches	<ul style="list-style-type: none"> - Réaliser les différentes vues de l'interface utilisateur - Répondre aux cas d'utilisation identifiés - Réaliser les <i>Epics</i> de priorité 3 (<i>Epics</i> : 1, 2, 6) - Réaliser les <i>Epics</i> de priorité 2 (<i>Epics</i> : 3, 4, 5, 8) - Si le temps le permet, commencer la réalisation d'<i>Epics</i> de priorité 1
Validation	Tests fonctionnels de navigation entre les pages, acceptation visuelle des vues

6.3 Sprint N°3 : Intégration des interfaces et de l'API

Durée	1 semaine (3 jours). Du 25.08.21 au 27.08.21
Goal	Mettre en relation l'API réalisé lors du premier sprint et les différentes interfaces réalisées lors du second sprint
Tâches	<ul style="list-style-type: none"> - Mettre en place la connexion entre les interfaces et l'API - Intégrer les différents <i>endpoints</i> de l'API aux interfaces
Validation	Tester et valider l'intégration complète de l'application avec l'API

²¹ <https://scrumguides.org/index.html>

²² <https://trello.com/>

6.4 Sprint N°4 : Mise en place des relations et recommandations

Durée	1 semaine (3 jours). <i>Du 01.09.21 au 03.09.21</i>
Goal	Créer des relations et tester le comportement de l'application par rapport à celles-ci
Tâches	<ul style="list-style-type: none">- Peupler la base de données avec des données provisoires- Créer des relations entre les différentes données- Mettre en place un algorithme de recommandations basique basé sur les relations- Documenter les tests réalisés
Validation	Tester l'application sans relations puis en ajoutant des relations et observer les changements des résultats obtenus

6.5 Sprint N°5 : Finalisation du projet

Durée	2 semaines (6 jours). <i>Du 08.09.21 au 10.09.21 et du 15.09.21 au 17.09.21</i>
Goal	Finaliser le projet et rédiger le rapport final
Tâches	<ul style="list-style-type: none">- Finaliser les éventuels développements non terminés- Tester l'application dans son ensemble et réaliser des ajustements- Documenter la réalisation et les tests
Validation	Présentation et démonstration de l'application finale au mandant

7 RÉALISATION

7.1 API avec *Express.js*

La réalisation de l'API *back-end* a été planifiée sur un sprint de six jours. Tous les points programmés ont pu être mis en place et ce, dans les délais. Après avoir introduit la structure du code source de cette API et les routes disponibles, je m'attarderai sur les points techniques les plus intéressants rencontrés durant le développement.

7.1.1 Structure du code

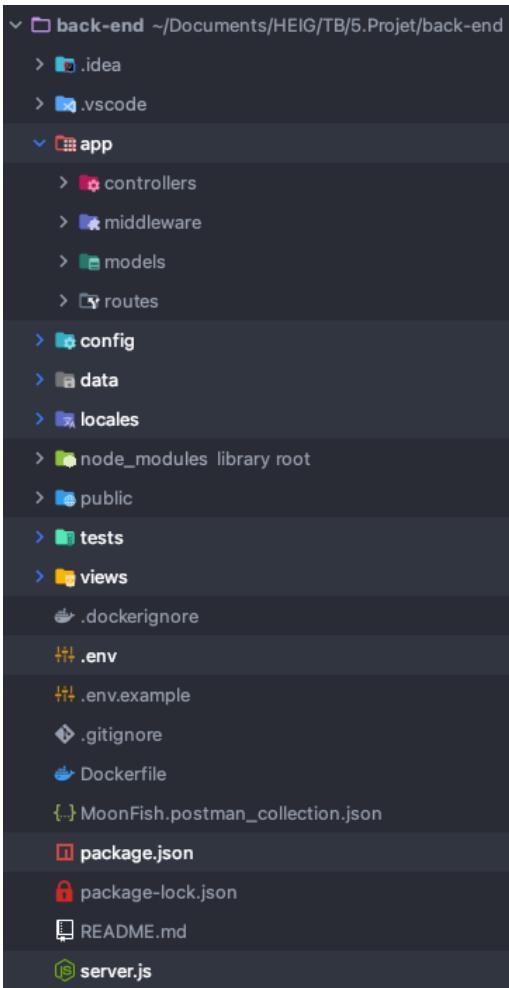


Figure 24 : Structure du code de l'API back-end avec Express

7.1.1.1 Les routes

Le dossier *routes* permet de définir tous les points d'entrées (chemins) de l'API. Il contient un fichier par domaine (*users*, *teams*, etc.) ainsi qu'un fichier *index*.

Le fichier *index* permet de parcourir l'ensemble des fichiers de domaines présents dans le dossier et d'automatiquement les charger dans l'application. De ce fait, lors de l'ajout d'un nouveau fichier, les routes créées et présentes dans celui-ci sont automatiquement chargées. Ce fichier *index* permet également de définir la route principale « / » et de renvoyer les requêtes invalides vers une page 404.

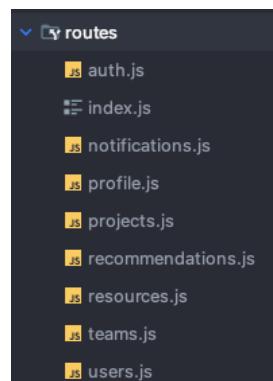


Figure 25 : Contenu du dossier « routes » de l'API, un fichier par domaine d'endpoints

Chacun de ces fichiers contient la définition d'une ou plusieurs routes. Cette définition est à chaque fois établie à l'aide de différents éléments présentés ci-dessous.

```
// Example : get all resources linked to a project
router.get(
  '/projects/:uuid/ressources', // 1. La méthode http (get, post, put, etc.)
  requireAuth, // 2. Le endpoint ou chemin d'accès, il s'agit de l'URL à interroger
  requiredRole(Role_USER), // 3. Si nécessaire, l'authentification vérifiée par le middleware
  trimRequest.all, // 4. Si nécessaire, le ou les rôles requis par accéder à la requête
  validateGetProject, // 5. Une fonction de « nettoyage » de données reçues
  getProjectTeams // 6. Si nécessaire, une fonction de validation des données reçues
) // 7. La fonction à exécuter en cas de validation des étapes précédentes
```

Lors de l'accès à un *endpoint*, les contrôleurs et le *middleware* exécutera les différentes vérifications requises (authentification, rôle, validité des données) puis va autoriser ou non l'exécution d'une fonction. Cette dernière fonction a pour rôle de retrouver, de formater et de rendre les données correspondantes demandées ; le cas échéant de retourner une erreur. Ci-dessous, la définition de toutes les routes du domaine « *projects* » interrogables.

```
// File : app/routes/projects.js

// Get all projects
router.get('/', requireAuth, requiredRole(Role_USER), trimRequest.all, getProjects)

// Get a project by uuid
router.get('/:uuid', requireAuth, requiredRole(Role_USER), trimRequest.all, validateGetProject, getProject)

// Get all project resources
router.get('/:uuid/resources', requireAuth, requiredRole(Role_USER), trimRequest.all, validateGetProject, getProjectResources)

// Get all project teams
router.get('/:uuid/teams', requireAuth, requiredRole(Role_USER), trimRequest.all, validateGetProject, getProjectTeams)

// Create a project (MANDATES)
router.post('/', requireAuth, requiredRole(Role_USER), trimRequest.all, validateCreateProject, createProject)

// Update a project by uuid
router.patch('/:uuid', requireAuth, requiredRole(Role_USER), trimRequest.all, validateUpdateProject, updateProject)

// Update a project status
router.patch('/:uuid/status/:status', requireAuth, requiredRole(Role_MODERATOR), trimRequest.all, validateProjectStatus, updateProjectStatus)

// Arbitrate a project (ARBITRATES)
router.put('/:uuid/arbitrate', requireAuth, requiredRole(Role_USER), trimRequest.all, validateArbitrateProject, arbitrateProject)

// Join a project (APPLIES)
router.put('/:uuid/apply', requireAuth, role(Role_USER), trimRequest.all, validateApplyProject, applyProject)

// Leave a project (DEVELOPS)
router.put('/:uuid/develop', requireAuth, role(Role_USER), trimRequest.all, validateDevelopProject, developProject)

// Note and feedback a project (MANDATES mark + feed-back)
router.put('/:uuid/feed-back', requireAuth, role(Role_USER), trimRequest.all, validateFeedbackProject, feedbackProject)

// Delete a project
router.delete('/:uuid', requireAuth, role(Role_ADMIN), trimRequest.all, validateDeleteProject, deleteProject)
```

L'ensemble des routes actuellement est résumé dans un tableau disponible dans les annexes de ce document.

7.1.1.2 Les contrôleurs

Les contrôleurs contiennent la logique de l'application et me permettent de répondre à une requête en combinant différents éléments (et/ou modèles) tout en ayant opéré les vérifications nécessaires (existence, validité des données) et en ayant formaté les données. Chaque domaine possède un dossier de contrôleurs. Dans ceux-ci nous pouvons en différencier trois types.

1. Tout d'abord les contrôleurs de validation. Ceux-ci permettent de définir les conditions des données à recevoir (paramètres, type et valeur de ceux-ci) pour que la requête soit acceptée. Si les données reçues sont valides, ce contrôleur laisse passer la requête au contrôleur suivant. Le cas échéant, elle bloque la requête et le signal par un message d'erreur de validation. Ces contrôleurs se présentent sous la forme d'un tableau constant contenant un ou plusieurs champs à valider, ainsi que les exigences pour chacun de ces champs. Dans l'exemple ci-dessous, seul le champ « `'id'` » est défini. Il doit être présent (`.exists()`) et ne pas être vide (`.not().isEmpty()`). Les

contrôleurs de validation font ensuite appel à la librairie « *express-validator* » (`check` et `validationResult`) permettant de vérifier si les conditions définies sont valides. Les fonctions « `validateResult`, `handleError` et `buildErrObject` » sont des *middlewares* sur lesquelles nous reviendrons plus tard.

<pre>// File : app/controllers/notifications/ validators/validateGetNotification.js const { check } = require('express-validator') /** * Validates the "get notification" request */ const validateGetNotification = [check('id') .exists() .withMessage('MISSING') .not().isEmpty() .withMessage('IS_EMPTY'), (req, res, next) => { validationResult(req, res, next) }]</pre>	<pre>// File : app/middleware/utils/validateResult.js const { validationResult } = require('express-validator') /** * Builds error for validation files * @param {Object} req - the request object * @param {Object} res - the response object * @param {Object} next - Navigate to the next function */ const validationResult = (req, res, next) => { try { validationResult(req).throw() ... return next() } catch (error) { return handleError(res, buildErrObject(422, error.array())) } }</pre>
---	---

2. Puis les contrôleurs d'entrés. Il s'agit de ceux référencés dans les routes. Il en existe donc un par route. Ceux-ci permettent de nettoyer les données reçues avec la requête (`matchedData`), de faire appel à d'autres contrôleurs pour retrouver les données (`findNotification`), de mettre en forme les données à renvoyer (`setNotificationInfo`) et de créer un message réponse (`res.status(200).json(...)`).

```
// File : app/controllers/notifications/getNotification.js

/**
 * Retrieve a single notification based on its id
 * @param {Object} req - request object
 * @param {Object} res - response object
 */
const getNotification = async (req, res) => {
  try {
    const data = matchedData(req)
    const notification = await findNotification(data.id)
    res.status(200).json(await setNotificationInfo(notification))
  } catch (error) {
    handleError(res, error)
  }
}
```

3. Et enfin les contrôleurs dits « *helpers* ». Ils peuvent être utilisés par les autres contrôleurs et sont dédiés à une tâche précise. Ils font appel aux *middlewares* (`getItem`) pour retrouver des données dans les bases et de ce fait fonctionnent avec le modèle des *promesses*²³JavaScript. Dans l'exemple ci-dessous, lors de la recherche d'une notification via son id, les traductions de celles-ci (stockées dans une seconde table) y sont appondues.

```
// File : app/controllers/notifications/helpers/findNotification.js

const Notification = mariadb.models.Notification
const NotificationTranslation = mariadb.models.NotificationTranslation

/**
 * Find a notification by its id and include the translations
 * @param {int} id - the notification's id
 * @param {string} lang - the translation to load
 */
const findNotification = (id = 0, lang = 'en') => {
  return new Promise(async (resolve, reject) => {
    try {
      resolve(await getItem(Notification, id, {
        include: [{ model: NotificationTranslation, where: { lang } }]
      }))
    } catch (error) {
      reject(buildErrObject(404, 'NOTIFICATION_DOES_NOT_EXIST'))
    }
  })
}
```

²³ https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise

7.1.1.3 Les middlewares

Les *middlewares* permettent la connexion entre les contrôleurs et les bases de données et servent donc de base à ceux-ci pour qu'ils puissent répondre aux requêtes. De plus, ils permettent également une généralisation du code, évitant ainsi les répétitions et apportent des classes d'utilitaires et de vérifications.

Je les ai divisés en quatre dossiers :

1. *auth* : Fonctions permettant de gérer les mots de passe (*checkPassword*, *encrypt*, *decrypt*, *hash*) et l'authentification.
2. *db* : Regroupant toutes les opérations CRUD sur les deux bases de données.
3. *emailer* : Fonctions permettant de construire et d'envoyer des emails.
4. *utils* : Fonctions utilitaires générales pouvant par exemple construire un message d'erreurs (*buildErrObject*), retrouver une adresse IP (*getIP*), convertir les éléments des requêtes en paramètres (*queryToParams*), etc.

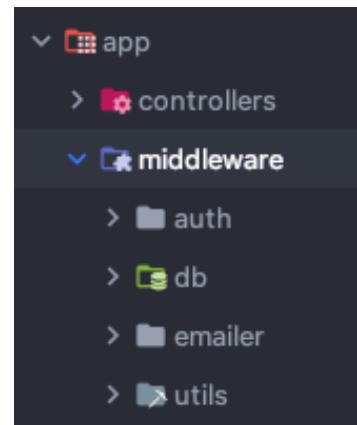


Figure 26 : Contenu du dossier « middleware » de l'API

Comme j'utilise deux bases de données, j'ai dû mettre en place les opérations CRUD pour chacune d'entre elles. Pour ce faire, j'ai utilisé « *Sequelize*²⁴ » et « *Neode*²⁵ » qui sont tous deux des ORM respectivement pour *MariaDB* et pour *Neo4j*, basés sur les *promesses* JavaScript. Ci-dessous, la récupération de projets dans la base de données relationnelle (en arrière-plan l'ORM utilisera l'opération SQL « *SELECT* ») et dans la base de données graphes (en arrière-plan l'ORM utilisera l'opération Cypher « *MATCH* »). Chaque fonction fait appel à une autre fonction *middleware* utilitaire (*queryToOptions* et *queryToParams*) permettant de mettre en forme et d'éventuellement ajouter des valeurs par défaut aux options reçues par la requête. Elles utilisent ensuite le modèle passé en paramètre pour réaliser l'opération de façon différente. Pour *Sequelize*, le modèle contient des fonctions (*findAll*) interrogeables directement auxquelles il est possible de passer des options (filtre, ordre, limites, etc.). Alors que pour *Neode*, il faut interroger une instance (*neo4j.all*) en lui passant comme paramètre le modèle, puis les différents paramètres.

<pre>// File : app/middleware/db/getItems.js /** * Get items from MariaDB database * matching the options * * @param {Object} model - the Sequelize model * @param {Object} options - build and query options */ const getItems = async (model, options = {}) => { return new Promise(async (resolve, reject) => { options = await queryToOptions(options) model.findAll(options) .then(async item => { resolve(item) }) .catch(error => { reject(error) }) }) } </pre>	<pre>// File : app/middleware/db/getNodes.js /** * Get nodes from Neo4j database * matching the query options * * @param {string} model - the Neo4j model * @param {Object} options - build and query options */ const getNodes = async (model, options = {}) => { return new Promise(async (resolve, reject) => { const params = await queryToParams(options) await neo4j.all(model, params.filters, params.orders, params.limit, params.offset) .then(async collection => { if (collection.length) resolve(collection._values) else resolve([]) }) .catch(error => { reject(error) }) }) } </pre>
---	---

En plus des opérations CRUD, pour la base de données graphe *Neo4j*, j'ai créé des *middlewares* liés aux relations entre les nœuds. Ces opérations n'étant pas totalement prises en charge par l'ORM « *Neode* », j'ai préféré construire directement des requêtes *Cypher*. Dans l'exemple ci-dessous, la fonction *getNodeRelations* permet de retrouver toutes les relations et tous les nœuds liés en partant d'un nœud précis. Une fois la requête *Cypher* exécutée, celle-ci nous

²⁴ <http://sequelize.org/>

²⁵ <https://github.com/adam-cowley/neode>

retourne un certain nombre de « *records* » que j'enregistre dans deux tableaux : un tableau contenant les nœuds et un tableau contenant les relations. Ces tableaux sont alors encapsulés dans un objet puis retournés.

```
// File : app/middleware/db/getNodeRelations.js

/**
 * Get all nodes relations from database
 *
 * @param {string} model - the Neo4j model
 * @param {uuid} uuid - the node uuid
 * @param {string|array} relation - a specific relation (IS_MEMBER_OF)
 */
const getNodeRelations = async (model, uuid, relation = '') => {
  return new Promise(async (resolve, reject) => {
    ...
    await neo4j.cypher(`MATCH (a:${model} {uuid: '${uuid}'})-[r:${relation}]-(b) RETURN b, r`)
      .then(async res => {
        if (res.records.length) {
          let nodes = [], relations = []
          for (const record of res.records) {
            nodes.push(record.get('b'))
            relations.push(record.get('r'))
          }
          resolve({ nodes, relations })
        } else resolve([])
      })
      .catch(error => { reject(error) })
  })
}
```

La requête *Cypher* réalisée ici est plutôt simple, elle se décompose comme suit :

1. `MATCH (a:${model} {uuid: '${uuid}'})` : L'opérateur `MATCH` permet de sélectionner les nœuds répondants à certaines conditions. Ici on définit un modèle et on lui assigne l'alias `a`, ce modèle doit contenir l'uuid passé via l'interpolation (`${uuid}`). Par l'unicité de nos uuid, on s'assure ici que l'on aura un seul nœud en retour.
2. `-[r:${relation}]` : Une fois le nœud identifié par `MATCH`, on lui demande toutes ses relations en leur assignant l'alias `r`. Les relations doivent correspondre au ou aux type(s) passés par l'interpolation (`${relation}`).
3. `-(b)` : Finalement, de ces relations trouvées, on demande les nœuds attachés sans condition en leur assignant l'alias `b`.
4. `RETURN b, r` : Cette dernière partie retourne les nœuds trouvés (`b`) et les arcs (`r`) par lesquels ils sont reliés.

Une requête construite pour retrouver toutes les équipes dont un utilisateur fait partie serait alors la suivante. A noter que cette requête d'exemple retourne en plus le nœud initial. Et ce uniquement dans le but d'obtenir une représentation visuelle du résultat, inutile, donc omise dans la fonction « `getNodeRelations` ».

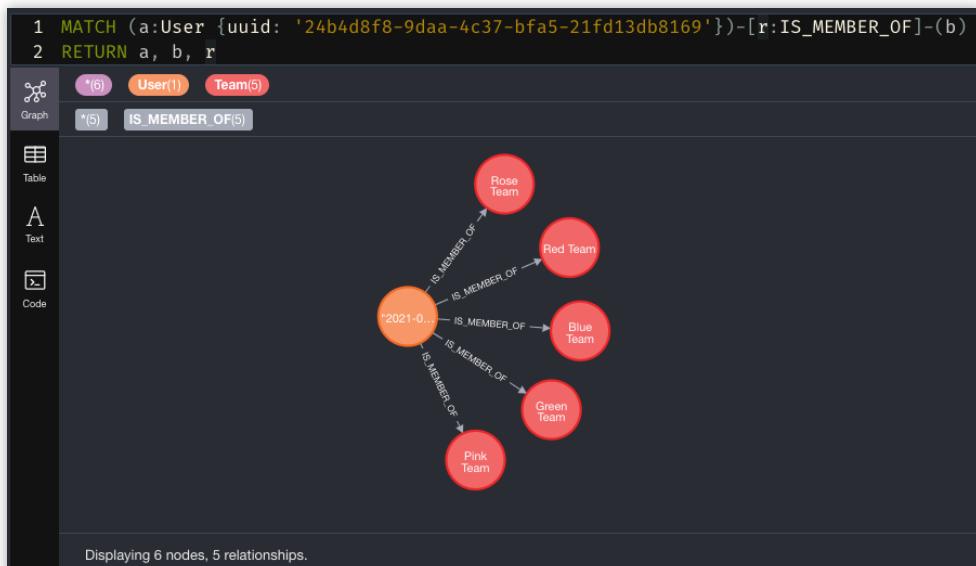


Figure 27 : Exemple de résultat d'une requête Cypher construite par les middlewares

7.1.1.4 Les modèles

Les modèles de données sont des représentations syntaxiques de données présentes dans une base de données. Du fait que j'utilise deux bases de données différentes (*Neo4j* et *MariaDB*), certains modèles doivent être définis dans les deux systèmes. C'est le cas pour le modèle « utilisateur » ainsi que pour le modèle « projet » qui ont tous deux de multiples propriétés. Les autres modèles (ressources, notifications, etc.) sont uniquement définis pour *MariaDB* à l'exception du modèle « équipe ». En effet, celui-ci est, quant à lui, uniquement défini pour *Neo4j*, car il ne possède pour l'instant que quelques propriétés. Si par la suite, le besoin se faisait sentir d'ajouter plus de propriétés à ce modèle, il serait très simple d'en ajouter une représentation *MariaDB*.

La définition d'un modèle pour *MariaDB* avec « *Sequelize* » se fait sous la forme d'un JSON. Le champ « `name` » définit le nom de la table dans la base et les champs attribuent les différentes colonnes. Pour chaque colonne, il est nécessaire de définir un type ; puis il est possible de gérer l'unicité et la nullité. La clé primaire est ajoutée par défaut si elle n'est pas définie. Il est également possible de définir un champ comme clé primaire à l'aide de « `primaryKey:true` ». Pour chaque colonne, le champ « `validate` » permet de contraindre les données à respecter une définition, par exemple une expression régulière, une taille, un format, etc.

```
// File : app/models/mariadb/User.js

module.exports = {
  name: 'users',
  attributes: {
    uuid: {
      type: DataTypes.STRING(36),
      validate: { isUUID: 4, len: 36 },
      allowNull: false,
      unique: true
    },
    firstName: {
      type: DataTypes.STRING(64),
      validate: { is: /^[\\w\\-\\s]+$/ , max: 64 },
      allowNull: false
    },
    lastName: {
      type: DataTypes.STRING(64),
      validate: { is: /^[\\w\\-\\s]+$/ , max: 64 },
      allowNull: false
    },
    ...
  },
};
```

La définition des relations est faite lorsque tous les modèles sont créés. Il est nécessaire de toujours créer les relations dans les deux sens, en indiquant la clé étrangère et les clés primaires. Par exemple, pour créer le lien entre les utilisateurs et leurs comptes bancaires, on procède comme suit.

```
// File : app/models/mariadb/index.js

// BankAccounts => User
User.hasMany(BankAccount, {foreignKey: 'userId', sourceKey: 'id'})
BankAccount.belongsTo(User, {foreignKey: 'userId', targetKey: 'id'})
```

La définition d'un modèle pour *Neo4j* avec la librairie « *neo4j* » se fait également sous la forme d'un JSON. Chaque première entrée définit une propriété (ci-dessous *uuid*, *tags* et *createdAt*) ou une relation (ci-dessous *isMemberOf* et *arbitrates*). Pour chacune des propriétés, il est nécessaire de déterminer un type et il est possible de la rendre requise ainsi que de lui attribuer une valeur par défaut. La propriété « *uuid* » est désignée comme clé primaire à l'aide de « `primary:true` ». Les relations sont définies par le type « `'relationship'` » puis doivent être nommées via l'attribut « `relationship` » et doivent définir un modèle cible avec « `target` ». Elles peuvent alors recevoir des propriétés. Les définitions de celles-ci sont identiques aux propriétés évoquées auparavant.

```
// File : app/models/neo4j/User.js

module.exports = {

  // Définition des propriétés
```

```

uuid: { primary: true, type: 'uuid', required: true },
tags: 'string',
createdAt: { type: 'datetime', default: () => new Date },

// Définition des relations
isMemberOf: {
  type: 'relationship',
  relationship: RELATION_IS_MEMBER_OF,
  target: 'Team',
  direction: 'out',
  properties: {
    since: { type: 'datetime', required: true, : () => new Date },
    isOwner: { type: 'boolean', required: true, default: false },
    status: { type: 'integer', required: true, default: STATUS_INACTIVE }
  }
},
arbitrates: { ... }
};

```

La liaison des données entre les deux bases se fait à l'aide d'un « *uuid* » : un système d'identifiants uniques universels. Chaque entrée de la base de données possède donc un identifiant généré lors de sa création. Celui-ci est identique pour un même élément dans les bases *Neo4j* et *MariaDB*.

7.1.2 Séquence de bout en bout

Pour mieux illustrer un appel de bout en bout depuis l'accès à l'URL jusqu'à la réponse JSON, prenons l'exemple suivant. Je désire connaître les données d'un projet spécifique dont je connais l'*uuid*. De ce fait, j'exécute une requête sur l'URL « {{server}}/projects/55d498d5-58bf-4873-920b-ffffa5db641a1 ». Pour cet exemple, on considère que je suis déjà authentifié (le système d'authentification est détaillé au chapitre suivant).

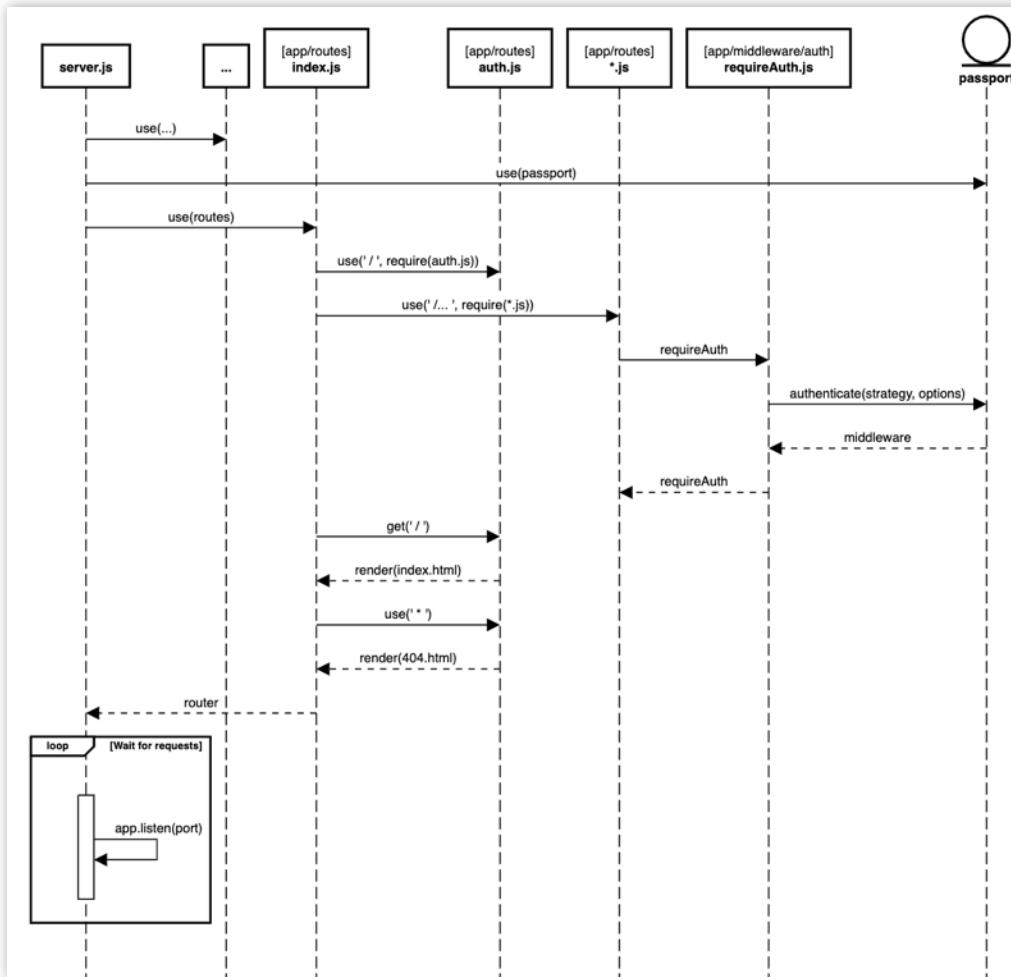


Figure 28 : Diagramme de séquence du lancement de l'API se terminant par l'attente de requêtes,
version originale disponible en annexe dans le fichier « Diagrammes/API-Sequence-start.png »

Avant même d'accéder à cet *endpoint*, le serveur doit être lancé. Lors de ce lancement, le fichier « *server.js* » configure différents éléments (représenté par « *use(...)* » par souci de simplification dans le schéma suivant) et charge notamment toutes les routes définies dans les différents fichiers contenus dans le dossier « *app/routes* ». Il initialise également l'authentification à l'aide de la librairie « *passport* ». Je reviendrai sur celle-ci dans le chapitre suivant. Une fois toutes les routes chargées (y compris l'*index* et la page 404), le serveur se met en écoute sur le port défini et est prêt à recevoir des requêtes.

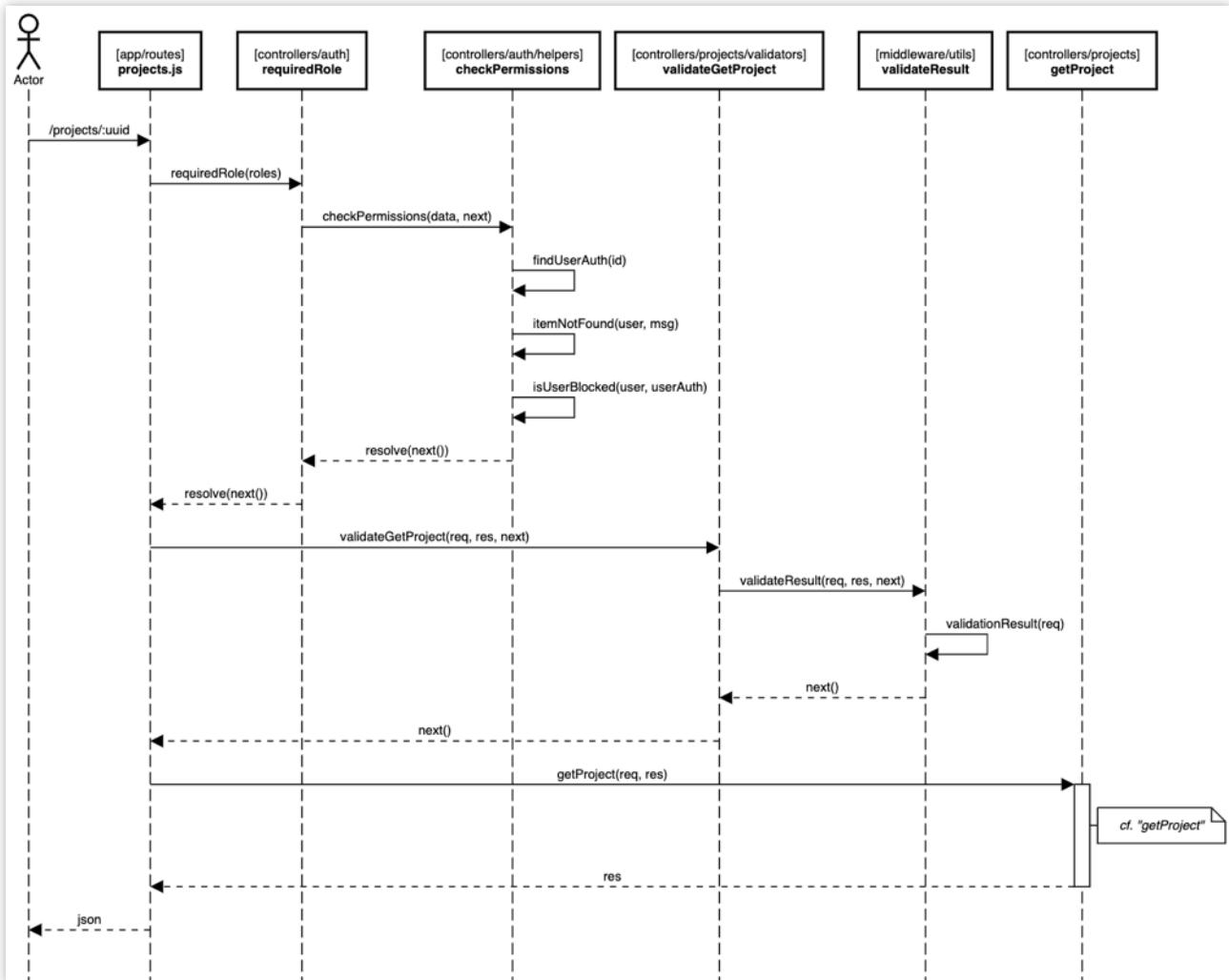


Figure 29 : Diagramme de séquence du traitement de l'API lors d'un accès au endpoint « `/projects/:uuid` »,
version originale disponible en annexe dans le fichier « *Diagrammes/API-Sequence-getProject.png* »

Pour commencer, le fichier « *projects.js* », contenant la route interrogée, procède à la vérification du rôle de l'utilisateur à l'aide des fonctions « *requiredRole* » et « *checkPermissions* ». Ces dernières cherchent l'utilisateur connecté, confirment qu'il est existant et actif puis vérifient qu'il possède bien le rôle requis. Une fois ces vérifications faites, un test de validation des éléments de la requête est effectué via les méthodes « *validateGetProject* » et « *validateResult* ». Si aucune erreur n'est détectée et que la validation et la vérification sont correctes, le fichier « *projects.js* » exécute la fonction « *getProject* ».

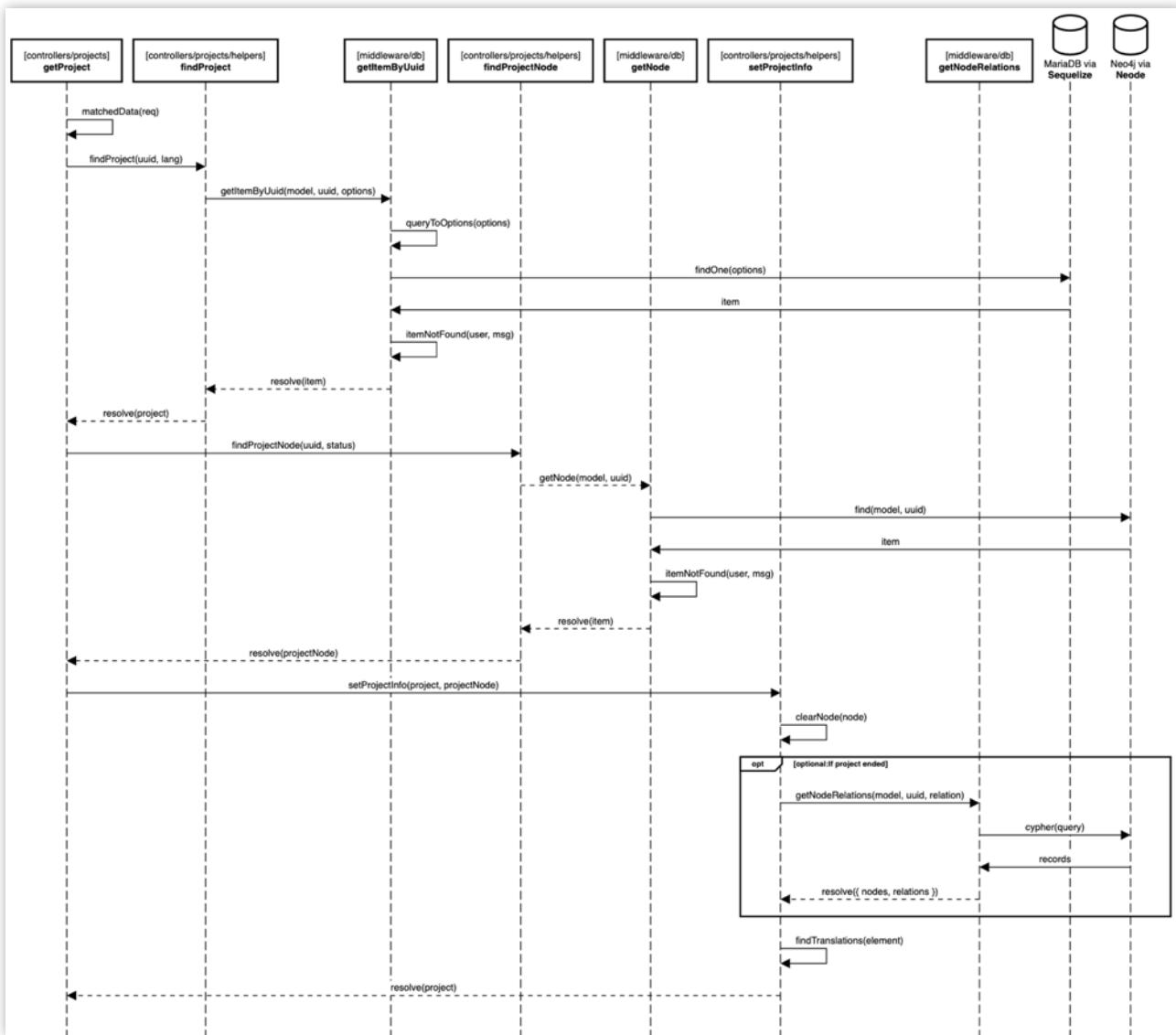


Figure 30 : Diagramme de séquence du traitement de l'API lors d'un appel à la fonction « *getProject* » ,
version complète disponible en annexe dans le fichier « *Diagrammes/API-Sequence-getProject.png* »

La fonction « *getProject* » est alors exécutée et commence par nettoyer les données reçues en ne gardant que celles spécifiées lors de la validation grâce à la fonction « *matchedData* ». Puis, comme les données d'un projet sont réparties entre les deux bases de données, la fonction va les interroger tour à tour avant de combiner les résultats. C'est d'abord le cas avec « *findProject* » qui interroge la base de données *MariaDB* via *Sequelize* et la fonction de *middleware* « *getItemByUuid* », tout en configurant les paramètres à transmettre (*queryToOptions*) et en effectuant les opérations de vérification (*itemNotFound*). Puis c'est au tour de *Neo4j* d'être interrogé par « *findProjectNode* » via *Neode* et la fonction *middleware* « *getNode* ». Cette dernière réalise également une vérification d'existence. Finalement la fonction « *setProjectInfo* » permet de combiner les données trouvées dans les 2 bases de données, de les nettoyer, d'y apprendre les traductions nécessaires et éventuellement d'y ajouter les relations entre les nœuds.

7.1.3 Authentication

Toutes les requêtes peuvent être protégées et nécessitent une authentification. J'ai géré cette authentification avec la librairie « *Passport.js*²⁶ » ainsi que des jetons de type *JWT*²⁷. On peut découper l'authentification pour un utilisateur en 3 étapes distinctes :

²⁶ <http://www.passportjs.org/>

²⁷ <https://jwt.io/>

1. L'enregistrement de l'utilisateur (réalisé une seule fois) : c'est cette action qui a pour effet de sauvegarder l'utilisateur dans les bases de données ainsi que de vérifier la validité de son adresse mail.
2. La connexion de l'utilisateur (réalisé à chaque nouvelle connexion) : cette action vérifie l'identité de l'utilisateur (email / mot de passe) et lui fournit un *token* lui permettant de réaliser des requêtes protégées.
3. La réinitialisation du mot de passe de l'utilisateur (réalisé à chaque fois que celui-ci l'oublie !) : cette action vérifie à nouveau l'email de l'utilisateur puis lui permet de choisir un nouveau mot de passe.

7.1.3.1 Enregistrement d'un utilisateur

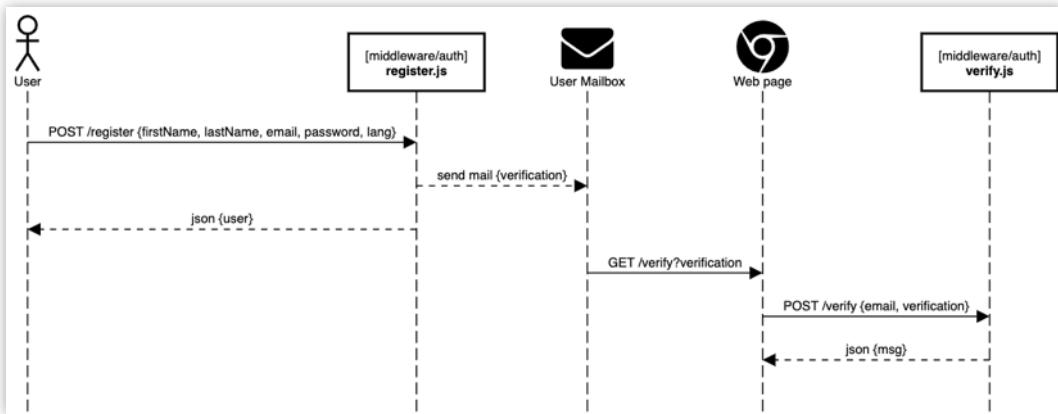


Figure 31 : Diagramme présentant les étapes de l'enregistrement pour un utilisateur, version originale disponible en annexe dans le fichier « Diagrammes/User-Registration.png »

L'enregistrement d'un utilisateur dans l'application se déroule en deux étapes pour celui-ci. Tout d'abord, l'utilisateur fait une demande d'enregistrement (via un formulaire web) contenant son nom, son prénom, son adresse email et un mot de passe. L'API vérifie alors que l'adresse n'existe pas déjà puis, si tel est le cas, envoie un email à l'adresse renseigné avec un *token* de vérification. L'utilisateur doit alors se rendre dans sa boîte mail et cliquer sur le lien reçu. Celui-ci contient le *token* de vérification. Ce lien le renvoie vers l'application et son compte est de ce fait vérifié. Dès lors, l'utilisateur peut se connecter à l'application.

Tant qu'un compte n'est pas vérifié, l'utilisateur ne peut accéder aux requêtes protégées. De ce fait, on s'assure de la validité de l'adresse email de l'utilisateur.

7.1.3.2 Connexion d'un utilisateur

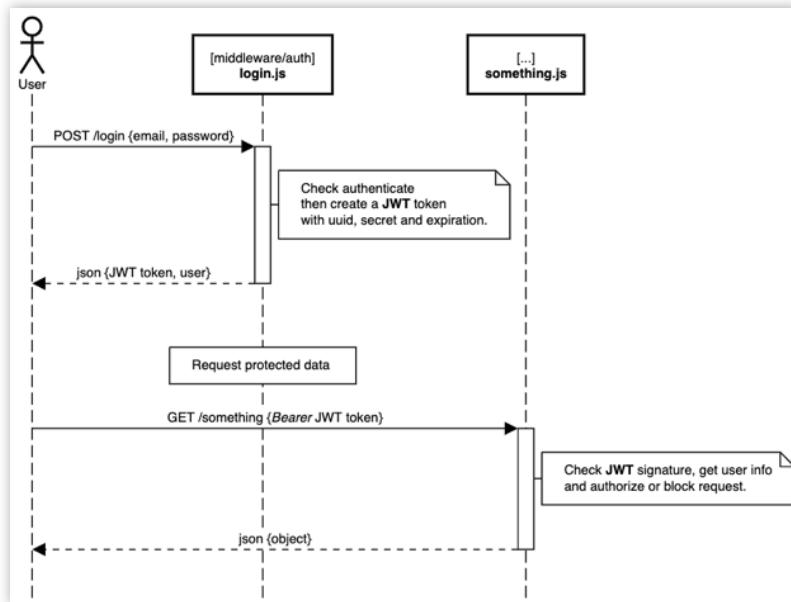


Figure 32 : Diagramme présentant les étapes de connexion d'un utilisateur et l'obtention d'un token, version originale disponible en annexe dans le fichier « Diagrammes/User-Login.png »

Pour accéder aux requêtes protégées, l'utilisateur doit au préalable se connecter avec son couple email / mot de passe. De ce fait, son identité peut être vérifiée et les requêtes approuvées. Cependant il serait trop contraignant de devoir se reconnecter à chaque nouvelle requête, c'est pourquoi j'ai mis en place un système de *token*. Lors de la première connexion de l'utilisateur, si celui-ci fournit un couple email / mot de passe correct, un *token* est généré qu'il pourra alors utiliser lors des requêtes suivantes. Celui-ci lui sert d'identité aux yeux de l'API et permet donc de valider et de personnaliser ses requêtes.

Pour décomposer toutes les actions entreprises lors de la connexion d'un utilisateur, j'ai réalisé le diagramme de séquence ci-dessous. Sur celui-ci, j'ai volontairement compressé certaines parties dans un souci de lisibilité. Le schéma complet est disponible en annexe.

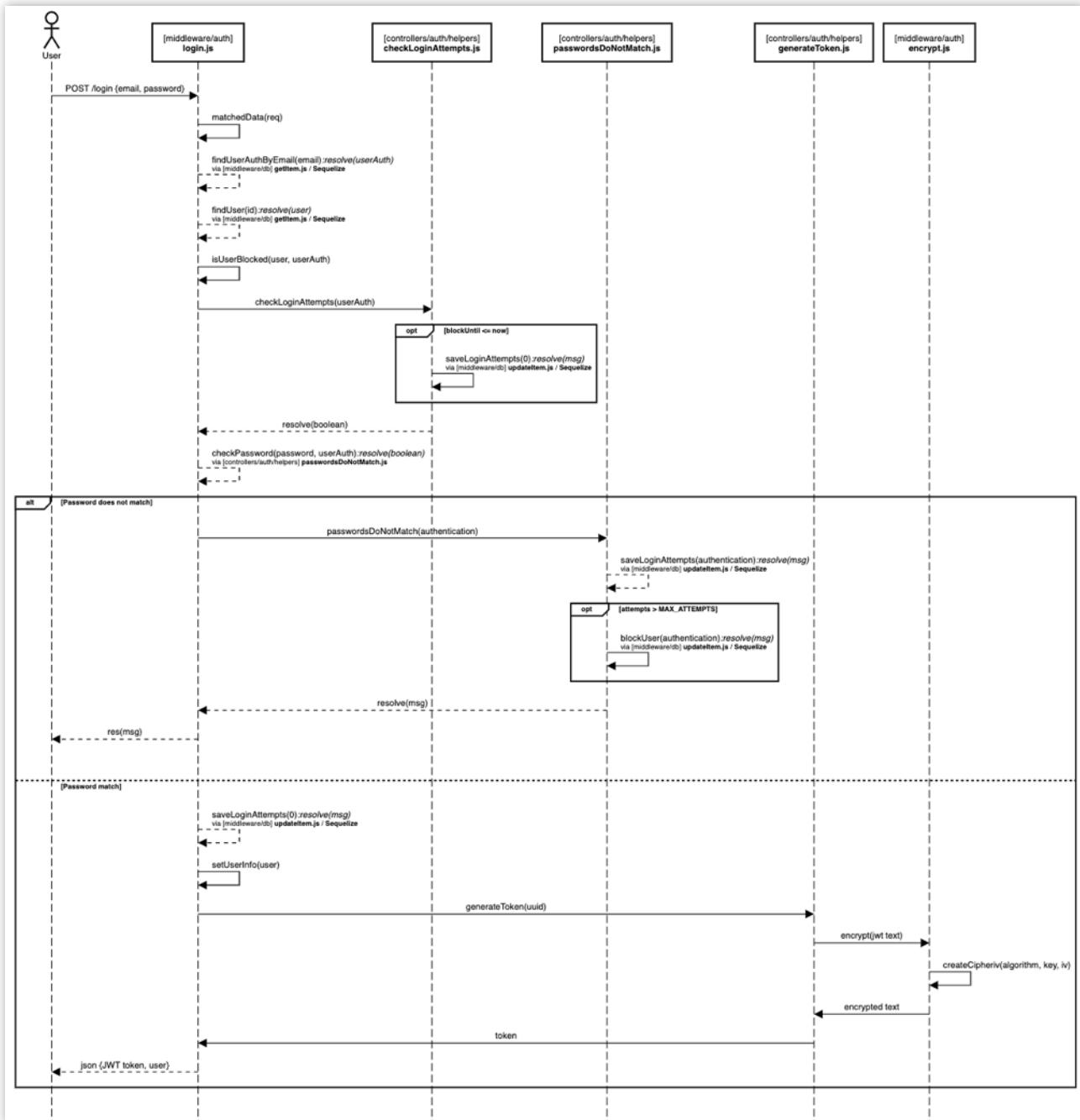


Figure 33 : Diagramme présentant les étapes de connexion d'un utilisateur et l'obtention d'un token,
version complète disponible en annexe dans le fichier « Diagrammes/API-User-Login-Full.png »

Le contrôleur « *login.js* » commence par récupérer les informations de l'utilisateur dans la base de données via son email (*findUserAuthByEmail* et *findUser*) puis vérifie si le statut de celui-ci est correct (*isUserBlocked*). Ensuite, il

vérifie le nombre précédent de connexions échouées (*checkLoginAttempts*) et les remets à zéro si le temps de blocage défini est dépassé (*saveLoginAttempts*). Il vérifie la correspondance du mot de passe reçu lors de la requête et du mot de passe, stocké sous la forme d'un *hash*, en base de données (*checkPassword*). À ce moment-là, deux scénarios se présentent :

1. Le mot de passe est incorrect

Le contrôleur va alors enregistrer une nouvelle tentative de connexion (*saveLoginAttempts*) et éventuellement bloquer l'utilisateur (*blockUser*) si le nombre de tentatives maximal est atteint. Le contrôleur retourne alors un message d'erreur et la requête est terminée.

2. Le mot de passe est correct

Dans ce cas-là, le nombre de tentatives de connexion (*saveLoginAttempts*) est remis à zéro, puis les informations de l'utilisateur sont formatées en prévision de leur retransmission (*setUserInfo*). Le contrôleur « *generateToken* » est alors interrogé en lui transmettant l'*uuid* de l'utilisateur.

```
// File : app/controllers/auth/helpers/generateToken.js

/**
 * Generates a token with uuid, secret and expiration
 *
 * @param {uuid} uuid - the user's uuid
 */
const generateToken = (uuid) => {
  try {
    // Gets expiration time
    const expiration = Math.floor(Date.now() / 1000) + 60 * process.env.JWT_EXPIRATION_IN_MINUTES
    // returns signed and encrypted token
    return encrypt(jwt.sign({ data: { _id: uuid }, exp: expiration }, process.env.JWT_SECRET))
  } catch (error) { throw error }
}
```

Celui-ci compose un temps d'expiration depuis un nombre de minutes défini dans la configuration (*JWT_EXPIRATION_IN_MINUTES*). Il utilise ensuite la librairie « *jsonwebtoken*²⁸ » en lui fournissant l'*uuid* de l'utilisateur, l'expiration et une phrase secrète également définie dans la configuration (*JWT_SECRET*) pour créer un *JsonWebToken* sous la forme d'une chaîne de caractère. Avant de retourner le *token* créé, celui-ci est encrypté à l'aide de la fonction « *encrypt* » qui lui ajoute un sel depuis la configuration (*JWT_SALT*).

```
// File : .env
...
JWT_SECRET=MyUltraSecurePasswordIWantForgetToChange
JWT_SALT=MySaltINeedToGenerate
JWT_EXPIRATION_IN_MINUTES=4320
...
```

Le fichier « *.env* » contenant les constantes est différent pour chaque environnement (développement, production, etc.), celles-ci peuvent alors être adaptées en fonction et aux besoins.

7.1.3.3 Accès à une requête protégée

Toute requête définie comme protégée et nécessitant d'être authentifiée dans les routes (*requireAuth*) est interceptée par la librairie « *passport* » avant même d'arriver dans le contrôleur « *requiredRole* ». C'est à ce moment-là que le *token* devant accompagner la requête est vérifié et validé ou rejeté.

```
// File : config/passport.js
const passport = require('passport')
const JwtStrategy = require('passport-jwt').Strategy
const { decrypt } = require('../app/middleware/auth/decrypt')
const { findUserByUuid } = require('../app/controllers/users/helpers')
```

²⁸ <https://www.npmjs.com/package/jsonwebtoken>

```

/**
 * Extracts token from request
 *
 * @param {Object} req - request object
 * @returns {string|null} token - decrypted token
 */
const jwtExtractor = (req) => {
  let token = null

  // Extracts token from headers, body or query
  if (req.headers.authorization) { token = req.headers.authorization.replace('Bearer ', '').trim() }
  else if (req.body.token) { token = req.body.token.trim() }
  else if (req.query.token) { token = req.query.token.trim() }

  // Decrypts token
  if (token) { token = decrypt(token) }

  return token
}

/**
 * Options object for JWT middleware
 */
const jwtOptions = {
  jwtFromRequest: jwtExtractor,
  secretOrKey: process.env.JWT_SECRET
}

/**
 * Login with JWT middleware
 */
const jwtLogin = new JwtStrategy(jwtOptions, (payload, done) => {
  findUserByUuid(payload.data._id)
    .then(async (user) => { return user ? done(null, user) : done(null, false) })
    .catch(error => { return done(error, false) })
})

// Intercept all protected requests
passport.use('jwt', jwtLogin)

```

Le *token* JWT est extrait de la requête (*jwtExtractor*), puis est décrypté à l'aide du sel (*JWT_SECRET*). Il est alors passé à la librairie « *passport* » avec la phrase secrète (*JWT_SECRET*). Celle-ci le vérifie, puis nous renvoie les données contenues (l'*uuid* de l'utilisateur) avec lesquels il est possible de retrouver l'utilisateur effectuant la requête (*findUserByUuid*).

7.1.3.4 Demande de réinitialisation de mot de passe

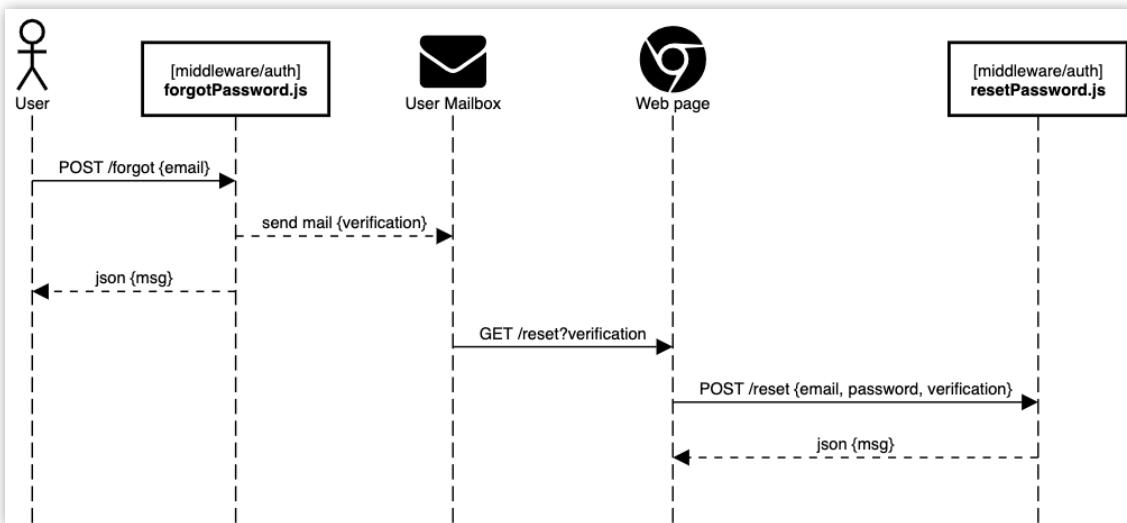


Figure 34 : Diagramme présentant les étapes de réinitialisation du mot de passe d'un utilisateur, version originale disponible en annexe dans le fichier « Diagrammes/User-ForgotPassword.png »

La demande de réinitialisation de mot de passe fonctionne sur le même principe que l'enregistrement d'un utilisateur, en deux étapes. Celui-ci commence par faire une demande de réinitialisation (via un formulaire web). Cette demande génère un *token* qui lui est transmis par mail à l'adresse fournie lors de son enregistrement.

L'utilisateur doit alors se rendre dans sa boîte mail et cliquer sur le lien reçu. Ce lien, contenant le *token* de vérification, le renvoie sur un formulaire lui permettant de modifier son mot de passe.

7.1.4 Chargement des données initiales

Lors du lancement du serveur en mode production, les bases de données sont vidées puis des données initiales sont chargées dans celles-ci. Pour ce faire, j'ai créé des données aléatoires pour chacun des modèles à l'aide de la librairie *faker.js*²⁹. Cette dernière permet de générer différentes données comme des noms, des descriptions, des URLs, des extensions de fichiers, etc. Dans l'exemple ci-dessous un certain nombre de projets (*NB_OF_SEEDS*) sont créés chacun avec des *translations* et éventuellement avec des *ressources*. J'utilise un maximum de variables aléatoires, de ce fait chaque initialisation de base de données est différente et permet de nouveaux scénarios.

```
// File : data/projects.js

for (let i = 1; i <= NB_OF_SEEDS; i++) {
    ...
    for (let j = 0; j <= random(0, 2); j++) {
        resources.push({
            name: (!j && status > PROJECT_STATUS_ONGOING) ? '[Feedback] ' : '') + faker.system.commonFileName(),
            link: faker.internet.url(), // Génération d'un URL avec faker.js
            type: faker.system.commonFileType(), // Génération d'une extension de fichier avec faker.js
            privacy: random(0, 1) ? 'public' : 'private',
            authorId: random(0, NB_OF_SEEDS)
        })
    }
    projects.push({
        resources,
        project: {
            id: i,
            uuid: uuid.v4(), // Génération de l'uuid unique de l'objet
            status: status,
            deadline: status <= PROJECT_STATUS_ONGOING ? faker.date.future() : faker.date.past(),
            tags: JSON.stringify(tags)
        },
        trans: [
            {
                projectId: i,
                lang: 'en',
                title: 'Project ' + faker.lorem.word(),
                description: faker.commerce.productDescription()
            }
        ]
    })
}
}
```

Une fois les données créées, celles-ci sont ajoutées dans la base de données graphe *Neo4j* et dans la base de données relationnelle *MariaDB*. Certains modèles sont uniquement ajoutés dans l'une ou dans l'autre base. Pour les modèles présents dans les deux bases, un *uuid* par objet est généré lors de l'opération précédente. Ceux-ci sont alors ajoutés aux objets dans les deux bases et permettront d'établir les connexions.

```
// File : data/projects.js

projects.forEach((project) => {
    try {
        // Création du nœud projet dans Neo4j
        neo4j.create('Project', project.project)

        // Création du projet dans MariaDB
        Project.create(project.project)
            .then((item) => {
                // Création des éventuelles ressources liées
                if (project.ressources) project.ressources.forEach(resource => { ... })
            })
            .then(() => {
                // Création des traductions du projet
                for (const trans of project.trans) Trans.create(trans)
            })
    } catch (error) { console.error(error) }
})
```

²⁹ <https://www.npmjs.com/package/faker>

7.1.5 Tests unitaires

En plus des tests fonctionnels réalisés à l'aide du client de *Postman*³⁰ au cours de développement de l'API, j'ai décrit tout un ensemble de tests unitaires. Pour cela, j'ai utilisé le framework de test JavaScript « *Mocha*³¹ » permettant de réaliser des tests asynchrones, tout en générant des rapports au format *HTML*. Un test « *Mocha* » se définit comme suit : on définit le *endpoint* à interroger en lui passant les autorisations nécessaires puis on décrit la réponse attendue. Dans l'exemple ci-dessous, le statut de la réponse doit valoir 200 ; le contenu de celle-ci doit être un tableau contenant un objet et cet objet doit contenir les propriétés « `'title'`, `'description'`, `'priority'` ». Si un seul des éléments décrit n'est pas respecté le test échoue. Le cas échéant, le test est considéré comme réussi.

```
// File : data/notifications.js
...
it('it should GET all the notifications', (done) => {
  chai.request(server)
    .get('/notifications') // endpoint to examine
    .set('Authorization', `Bearer ${token}`) // add needed authorization
    .end((err, res) => {
      res.should.have.status(200) // response status must be equals to 200
      res.body.should.be.an('array') // response must be an array
      res.body[0].should.be.an('object') // this array must contains an object
      res.body[0].should.include.keys('title', 'description', 'priority') // this object must have properties
        « title, description and priority »
      done() // If so, test passed
    })
})
})
```

Pour chaque *endpoint* j'ai alors décrit une série de plusieurs tests permettant de valider celui-ci. Par exemple pour les méthodes GET en interrogeant la route sans authentification puis avec l'authentification. Pour les méthodes POST en tentant d'insérer deux fois la même adresse mail ou encore pour les méthodes PUT en essayant de supprimer des relations inexistantes.

```
// File : tests/team.js
describe('***** TEAMS *****', () => {
  describe('POST /login', () => { ... })
  describe('GET /teams', () => { ... })
  describe('POST /teams', () => { ... })
  describe('GET /teams/:uuid', () => { ... })
  describe('PATCH /teams/:uuid', () => { ... })
  describe('PATCH /teams/:uuid/status/:status', () => { ... })
  describe('PUT /teams/:uuid/leave & /teams/:uuid/join', () => {
    const uuid = createdTeams.slice(-1).pop()

    it('it should DELETE A RELATION between team and user', (done) => {
      chai.request(server)
        .put(`/teams/${uuid}/leave`).set('Authorization', `Bearer ${token}`)
        .end((error, res) => {
          res.should.have.status(200)
          res.body.should.be.a('object')
          res.body.should.have.property('msg').eql('TEAM_LEAVE')
          done()
        })
    })
    it('it should NOT LEAVE a team before join it', (done) => {
      chai.request(server)
        .put(`/teams/${uuid}/leave`).set('Authorization', `Bearer ${token}`)
        .end((error, res) => {
          res.should.have.status(403)
          res.body.should.be.a('object')
          res.body.should.have.property('error')
          res.body.error.should.have.property('msg').eql('USER_NOT_IN_TEAM')
          done()
        })
    })
  })
})
```

³⁰ <https://www.postman.com/product/api-client/>

³¹ <https://mochajs.org/>

```

        })
    })
it('it should ADD A RELATION between team and user', (done) => {
  chai.request(server)
    .put(`/teams/${uuid}/join`).set('Authorization', `Bearer ${token}`)
    .end((error, res) => { ... })
})
it('it should NOT ADD an existing relation between team and user', (done) => {
  chai.request(server)
    .put(`/teams/${uuid}/join`).set('Authorization', `Bearer ${token}`)
    .end((error, res) => { ... })
})
})
})
describe(`DELETE /teams/:uuid`, () => { ... })
...
})
}

```

Dans l'exemple ci-dessus pour les routes « `PUT /teams/:uuid/leave & /teams/:uuid/join` », je réalise quatre tests en série. Tout d'abord la relation entre l'utilisateur et l'équipe est existante et doit donc être supprimée (`DELETE A RELATION`). Puis celle-ci n'existant plus, la tentative suivante de suppression doit échouer (`NOT LEAVE`). Idem pour la création de la relation qui fonctionne lorsque celle-ci n'existe pas (`ADD A RELATION`), puis qui doit échouer quand celle-ci est déjà existante (`NOT ADD`). Une fois ces tests rédigés, il est possible de les exécuter et d'en observer le résultat directement dans un IDE ou sous la forme d'un rapport `HTML`.

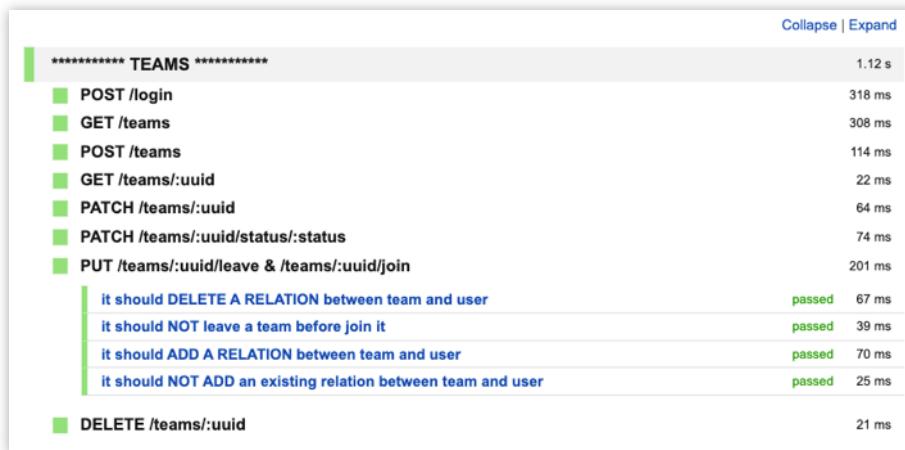


Figure 35 : Rapports de résultats des tests unitaires effectués sur le endpoint « `/teams` »

Les rapports de résultats de tous les tests sont disponibles dans les annexes dans le dossier « `Annexes/Tests-unitaires/Test_Results_*.html` ».

7.1.6 Déploiement

Le déploiement de cette API *back-end*, ne fut pas une tâche aisée. Je me suis d'abord tourné vers des services en ligne comme *Heroku*³² ou *Hidora*³³, mais aucune des solutions proposées par ces services ne convenait à mon architecture incluant le serveur *Express* et deux bases de données. Dès lors et pour bénéficier d'une configuration souple et sur mesure, j'ai décidé d'héberger mon *API* sur mon serveur *Synology*. Pour faciliter le déploiement, j'ai mis en place différents conteneurs *Docker* via la configuration « `docker-compose` » ci-dessous. Celle-ci met en relation, via le réseau « `moonfish` », quatre conteneurs :

1. « `moonfish-backend` » : Le conteneur *Express* contenant l'*API*
2. « `moonfish-mariadb` » : Le conteneur exécutant la base de données *MariaDB*
3. « `moonfish-phpmyadmin` » : Permettant d'administrer *MariaDB*
4. « `moonfish-neo4j` » : Le conteneur exécutant la base de données *Neo4j*

³² <https://www.heroku.com/>

³³ <https://hidora.io/fr/>

```
// File : docker/docker-compose.yml

version: "3.0"
services:

  moonfish-backend:
    container_name: moonfish-backend
    restart: always # Always restart the container if it stops
    build:
      context: ../back-end/
      target: dev
    volumes: ../back-end:/app
    command: sh -c "sleep 60 && npm run ${EXPRESS_CMD}" # 'npm run start' : for production mode,
               # 'npm run dev' : for development mode, etc
    ports: 3000:3000 # Ports mapping
    depends_on: # Wait on databases before start
      - mariadb
      - neo4j
    environment:
      NODE_ENV: ${NODE_ENV} # 'development' or 'production' mode

  mariadb:
    image: mariadb
    container_name: moonfish-mariadb
    restart: always # Always restart the container if it stops
    ports: 3306:3306 # Ports mapping
    volumes: mariadb-data:/var/lib/mysql
    environment:
      MARIADB_RANDOM_ROOT_PASSWORD: 1
      MARIADB_DATABASE: moonfish
      MARIADB_USER: moonfish
      MARIADB_PASSWORD: password

  phpmyadmin:
    image: phpmyadmin
    container_name: moonfish-phpmyadmin
    restart: unless-stopped # Similar to always, except that when the container is stopped
                       # it is not restarted even after Docker daemon restarts.
    ports: 8000:80 # Ports mapping
    environment:
      - PMA_ARBITRARY=1
      - PMA_HOST=mariadb
    depends_on: mariadb # Wait on mariadb before start

  neo4j:
    image: neo4j
    container_name: moonfish-neo4j
    restart: always # Always restart the container if it stops
    hostname: neo4j
    ports: # Ports mapping
      - 7474:7474
      - 7687:7687
    volumes:
      - ./neo4j/conf:/conf
      - ./neo4j/data:/data
      - ./neo4j/import:/import
      - ./neo4j/logs:/logs
      - ./neo4j/plugins:/plugins
    environment:
      NEO4J_AUTH: neo4j/password # username (must be neo4j) / password

  volumes:
    mariadb-data: # This is where MariaDB data will be stored, even if container is restarted, data will be there.

networks:
  default: # This is the shared network between the fourth containers
  name: moonfish
```

Bien que paraissant simple une fois réalisé, la rédaction de ce fichier fut laborieuse pour obtenir quatre conteneurs entièrement fonctionnels. L'avantage est que dès lors, une seule commande suffit pour lancer l'API, celle-ci est la suivante :

```
$ docker-compose up
```

De plus, il est possible de forcer les bases de données à se réinitialiser et à charger les données initiales en ajoutant un argument comme suit :

```
$ docker-compose --env-file .env.drop up
```

Une fois la commande exécutée, les conteneurs vont démarrer et s'initialiser, se configurer et charger les différentes données. Ils sont prêts après environ une minute si les bases de données sont réinitialisées et l'API peut alors répondre aux requêtes reçues.

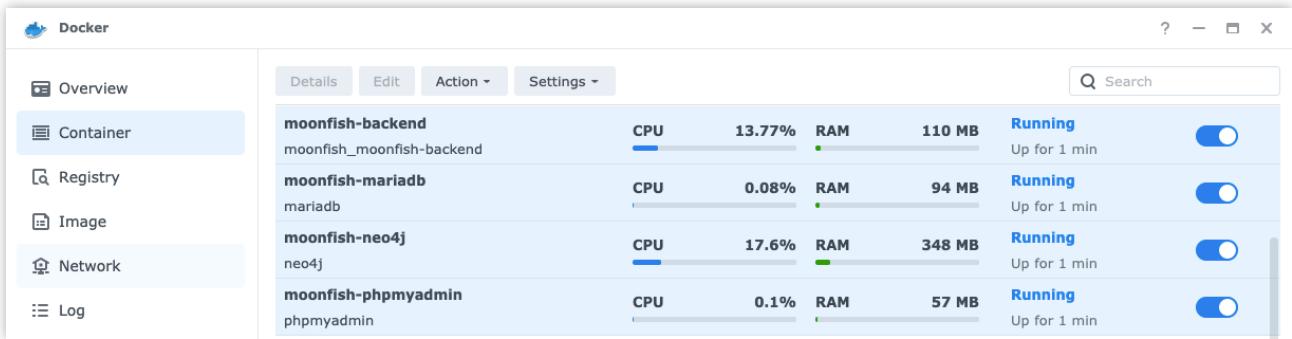


Figure 36 : Affichage des quatre conteneurs Docker de l'API MoonFish s'exécutant sur un serveur Synology

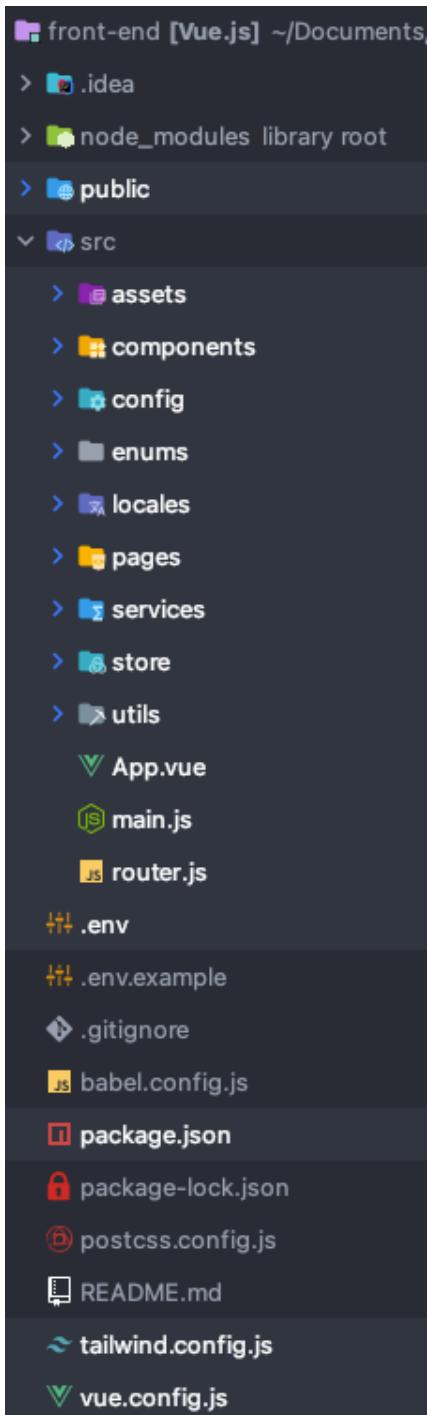
7.1.7 Code source et documentation annexe

Le code source compilable de cette partie *back-end* de l'application est disponible sur le *repository GitHub* dans le dossier *5.Projet/back-end*.

Du plus, la définition et la documentation des routes disponibles dans l'API « *MoonFish - Express.js REST API with JWT* » réalisé à l'aide de *Postman* est consultable directement en ligne via l'adresse suivante : <https://documenter.getpostman.com/view/8210926/TzseHmCz>.

7.2 Front-end avec *Vue.js*

7.2.1 Structure du code



Mon projet *Vue.js* est structuré via l’arborescence présentée ci-contre. Les dossiers et fichiers intéressants sont les suivants :

- *src* : Dossier contenant l’index HTML principal de l’application qui sera interprété par le navigateur, ainsi que certaines ressources statiques (*favicon.ico*).
 - *src* : Dossier contenant les sources principales du projet :
 - *assets* : Polices de caractères, images, styles supplémentaires, icônes *font-awesome*, etc.
 - *components* : Composants *Vue.js* utilisés tout au long de l’application. Contient un sous-dossier *layout* pour les composants de dispositions complexes (*Sidebar*, *Navigation*, etc.) et un sous-dossier *ui* pour les petits éléments (boutons, bar de recherche, etc.)
 - *config* : Définitions des configurations des librairies (*i18n*) et des constantes.
 - *enums* : Types énumérés
 - *locales* : Dossier contenant les fichiers JSON de traductions.
 - *pages* : Dossier contenant un sous-dossier par pages d’entrées de l’application (*projects*, *teams*, etc.).
 - *services* : Dossier permettant de stocker les services notamment les connexions à l’API, la gestion de l’authentification et du cache.
 - *store* : Gestion de l’authentification et de la session utilisateur avec *Vuex*.
 - *utils* : Fonctions d’ordre utilitaires (*capitalize*, *clean*, etc.).
 - *App.vue* : Template de base chargé par *Vue.js* sur toutes les pages permettant d’encapsuler les autres pages.
 - *main.js* : Ce fichier est le point d’entrée de l’application *Vue.js*.
 - *router.js* : Fichier de définition des routes via « *vue-router* » et de liaison avec les « *pages* ».
 - *.env* : Variables d’environnements
 - *package.json* : Fichier contenant tous les détails, scripts et dépendances du projet *npm*.
 - *tailwind.config.js* : Fichier de configuration de *tailwindcss* (thème, couleurs, plug-ins, polices, etc.).
 - *vue.config.js* : Fichier de configuration de *Vue.js* (serveur, plug-ins, etc.).
- Les éléments restants sont respectivement : le dossier de configuration de l’IDE (*.idea*), le dossier *node_modules* contenant tous les modules *Node.js* nécessaires, le fichier d’exemple de la configuration de l’environnement, de *Git*, et de *npm*.

Figure 37 : Structure du code de l’application front-end avec *Vue.js*

7.2.2 Squelette, navigation, notifications et Dashboard

Pour accéder à l’application, il est nécessaire de se créer un compte puis de se connecter à l’aide de celui-ci. Cette connexion se fait via l’interface décrite dans le chapitre « *Maquettage* ». J’ai ensuite voulu la navigation et l’ergonomie de l’application simple et efficace. Cette expérience utilisateur est également décrite dans le chapitre « *Maquettage* ». Pour rappel elle se compose de trois zones différentes :

1. Un volet latéral sur la gauche dont le contenu est identique sur toutes les pages du site. Il présente le logo de l'application, le menu de navigation entre les sections principales et le profil de l'utilisateur actuellement connecté.
2. Le haut de page qui comporte un menu de recherche ainsi qu'un résumé des notifications actuelles.
3. Le contenu dynamique a proprement dit de la page, zone dans laquelle il y a la possibilité de faire défiler de haut en bas si le contenu est plus grand que la hauteur de l'écran.

7.2.2.1 Dashboard

Une fois connecté, l'utilisateur est directement dirigé sur son *Dashboard*, celui-ci lui présentant alors en son centre une recommandation de projets ainsi qu'un historique des activités passées. Le volet de navigation sur la gauche présente pour chaque utilisateur trois menus principaux en plus du *Dashboard* : *Projects*, *Teams* et *Resources*.

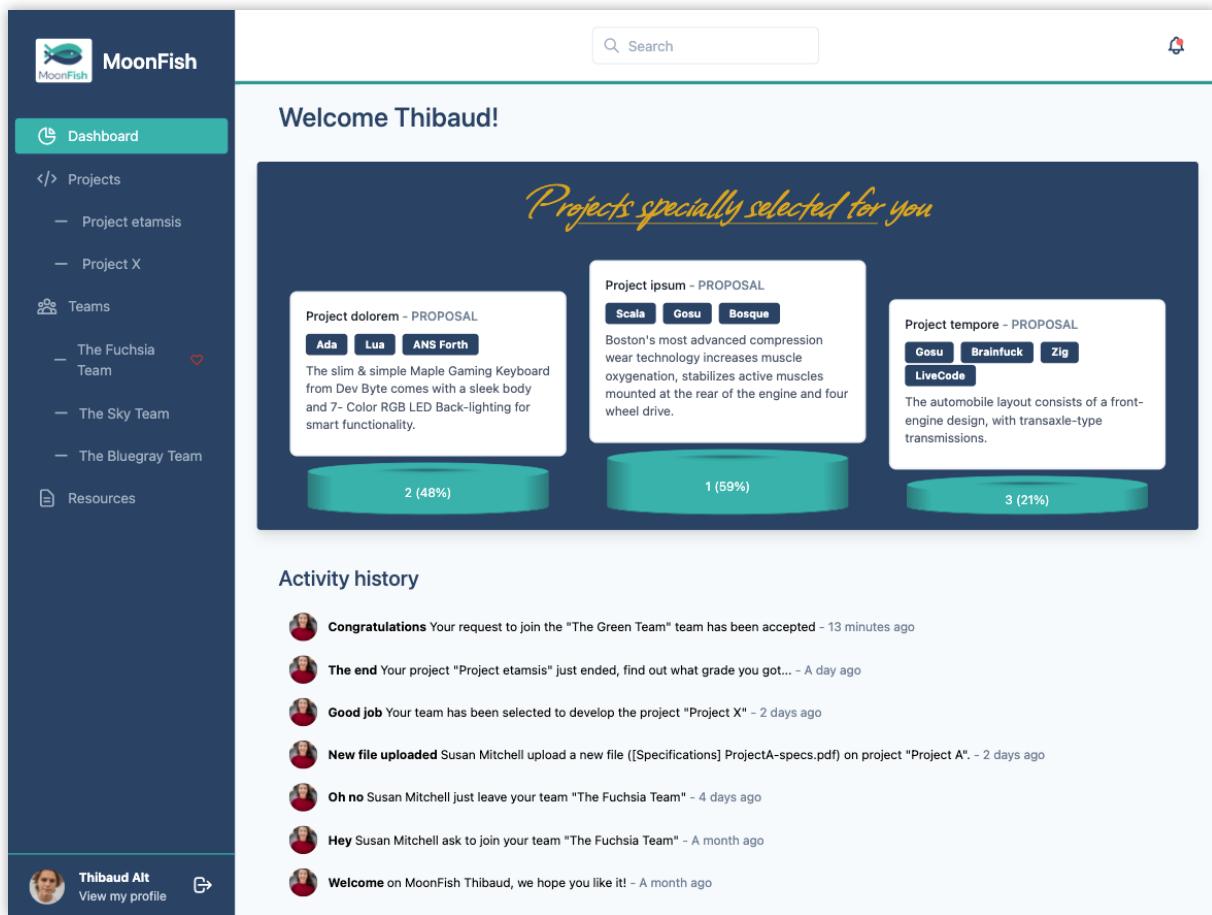


Figure 38 : Présentation du Dashboard d'accueil exposant les recommandations de projets et l'historique des activités

Sous les menus principaux *Projects* et *Teams*, s'affichent les équipes desquelles l'utilisateur connecté fait partie (ici « *The Fushia Team* », « *The Sky Team* » et « *The Bluegray Team* ») ainsi que les projets en proposition et en cours de réalisation (ici « *Project etamsis* » et « *Project X* »). On peut observer un cœur à côté de l'équipe « *The Fushia Team* », signifiant que l'utilisateur en est son propriétaire.

Pour afficher cette première page, plusieurs composants *Vue.js* sont combinés comme suit. Tout d'abord une fois le fichier « *public/index.html* » chargé par le navigateur celui-ci inclut la vue initiale « *App.vue* ». Cette vue initiale va composer la page à l'aide des composants décrits dans le schéma suivant, et ce pour toutes les pages de l'application.

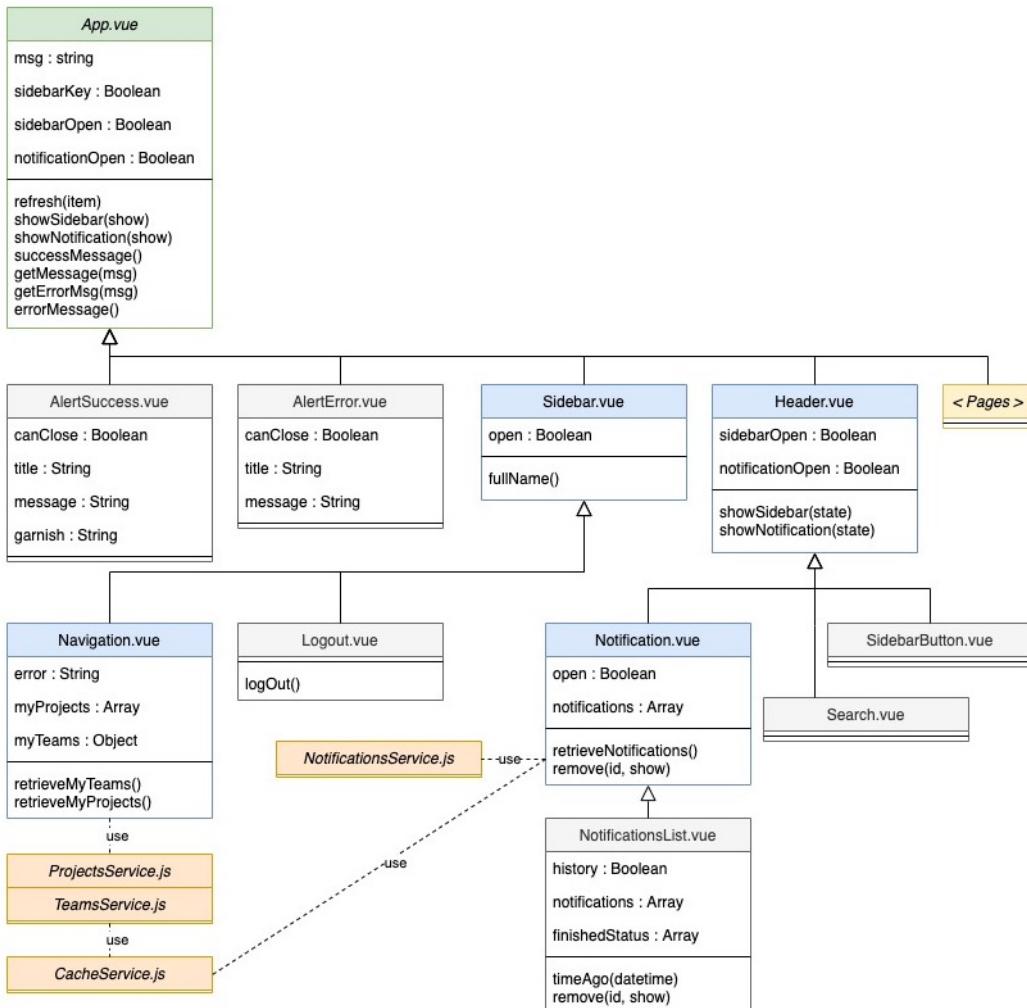


Figure 39 : Diagramme des classes chargées sur chacune des pages de l'application et formant ainsi son squelette, version originale disponible en annexe dans le fichier « Diagrammes/Vue-App.jpg »

Ci-contre et ci-dessus : en vert est représenté la classe initiale « `App.vue` », en bleu sont représentés les composants dits complexes du dossier « `components/layout` » représentant des compositions d'autres composants, en gris les composants simples du dossier « `components/ui` », en orange les services faisant appel à l'API et en jaune les pages. Ces pages sont les éléments dynamiques qui changent lors de la navigation et qui sont gérés par le fichier « `router.js` ». Pour la page `Dashboard`, voici les éléments supplémentaires qui sont chargés (avec en violet les composants de librairies externes).

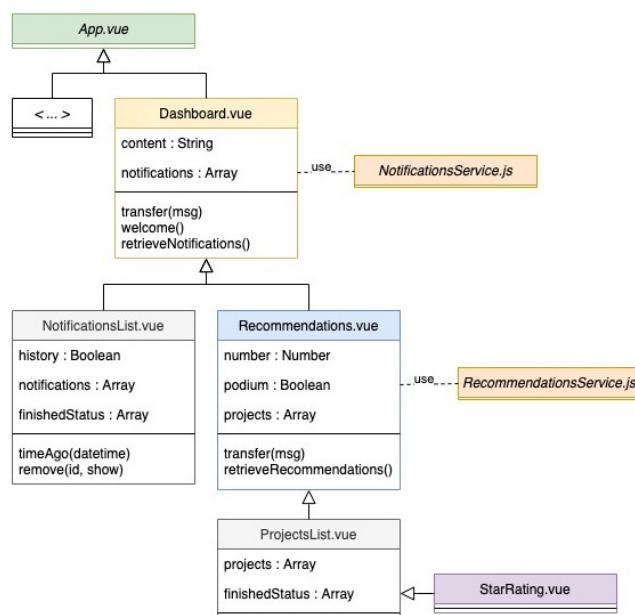


Figure 40 : Diagramme des classes supplémentaires intégrées lors du chargement de la page `Dashboard`, disponible en annexe dans le fichier « Diagrammes/Vue-Dashboard.jpg »

7.2.2.2 Gestion des notifications

Un des points intéressants et illustrant la plupart des fonctionnements des composants *Vue.js*, est la récupération des notifications et les implications que celles-ci apportent. Pour commencer, dès que le composant « *Notification.vue* » est monté, il se met à récupérer, toutes les minutes, les notifications auprès de l'*API* via les fonctions « `mounted()` » et « `retrieveNotifications()` ».

```
// File : src/components/Layout/Notification.vue
<template>
  <div class="flex items-center">
    <div class="relative">
      <button @click="$emit('show-notification', !open)" class="..."> ... </button>
      <div :class="open ? 'block' : 'hidden'" @click="$emit('show-notification', false)" class="..."></div>
      <div :class="open ? 'inline' : 'hidden'">
        <NotificationsList :notifications="notifications" @remove="remove"/>
      </div>
    </div>
  </div>
</template>
<script>
  data() { return { notifications: [] }; },
  mounted() {
    this.retrieveNotifications();
    let _this = this;
    setInterval(function() {
      _this.retrieveNotifications()
    }, 60 * 1000); // Look for new notifications every minute
  },
  methods: {
    ...
    async retrieveNotifications() {
      const oldLength = this.notifications.length
      this.notifications = await request(NotificationService.getMine(), this);
      if (this.notifications.length > oldLength)
      {
        CacheService.flush(); // Remove all cached keys (myTeams & myProjects)
        this.$emit('refresh'); // Refresh the sidebar component
      }
    },
  }
</script>
```

La fonction « `retrieveNotifications()` » charge via l'*API* les notifications, les transmet au composant « *NotificationsList* » qui se charge de les afficher, puis détermine si de nouvelles notifications sont disponibles. Si tel est le cas, elle fait un appel au service « *CacheService* » lui demandant de vider le cache et notamment les clés « *myTeams* » et « *myProjects* » stockées dans le *localStorage*³⁴ et contenant les équipes et les projets auxquels l'utilisateur est actuellement lié. Les notifications reçues étant la plupart du temps liées à des changements de statuts (ajout dans une équipe, acceptation ou finalisation d'un projet, etc.), le renouvellement du cache est pertinent à ce moment-là. Une fois le cache vidé, le volet de navigation gauche doit peut-être ajouter et/ou retirer des équipes et/ou des projets de ses sous-menus. Pour ce faire, le composant « *Notification.vue* » émet un événement à son composant parent via la fonction « `this.$emit('refresh')` ». Ce message sera retransmis jusqu'au composant incluant le composant à rafraîchir, ici « *App.vue* ». Ce dernier traitera alors le rafraîchissement, en obligeant le composant « *Sidebar.vue* » à se recharger, ce qui aura pour effet de rafraîchir l'interface et de repeupler le cache.

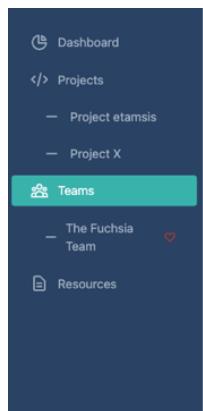
```
// File : src/components/Layout/Notification.vue
async remove(id, show) {
  this.notifications.some(function(notif, i) {
    if (notif.id === id) {
      this.notifications.splice(i, 1); // Find the notification and remove it from the list
      return true; // Exit the loop once the notification is found
    }
  }, this);
  this.$emit('show-notification', show);
  await request(NotificationService.delete(id), this) // Delete the notification via the API
}
```

³⁴ <https://developer.mozilla.org/fr/docs/Web/API/Window/localStorage>

La deuxième fonction de ce composant illustre l'intérêt des applications dites réactives. Dans le menu des notifications, il est possible d'en supprimer une, une fois celle-ci lue par l'utilisateur via un bouton spécifique. Lors du clic sur ce bouton, la notification est immédiatement retirée de la liste et disparaît donc visuellement. Puis une requête est faite sur l'*API* pour effectivement effectuer l'action de suppression de la notification dans la base de données. Ces applications réactives permettent donc d'effectuer des actions très rapidement sans la nécessité, dans certaines mesures, d'attendre un retour de traitement d'une *API*.

7.2.3 Gestion des équipes

La gestion des équipes s'opère via deux types de pages. Premièrement la page « *Teams* » présentant une liste de toutes les équipes actives classées par leur note actuelle (note moyenne des projets terminés), ainsi qu'un bouton permettant de créer une nouvelle équipe.



Teams

The Green Team - 5 members ★★★★★

The Violet Team - 5 members ★★★★★

The Amber Team - 6 members ★★★★★

The Fuchsia Team - 5 members ★★★★★

The Rose Team - 2 members

Create a new team

Figure 41 : Liste de toutes les équipes présentent dans l'application

Create a new team

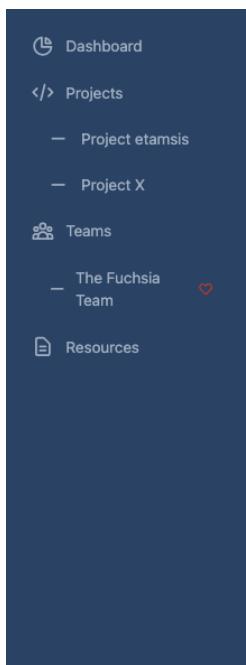
Name Name is required!

Color Color is required!

Save

Figure 42 : Formulaire de création d'une équipe

Chaque équipe peut être alors sélectionnée et cela redirige vers la seconde page de gestion ; la gestion de cette équipe elle-même. Cette page présente l'équipe, sa note actuelle, ses membres et les projets avec lesquels elle entretient un lien. Si l'utilisateur ne fait pas partie de l'équipe, il peut la rejoindre, respectivement la quitter s'il en fait partie. Là aussi, les membres et les projets peuvent être sélectionnés et redirigent vers leurs pages respectives.



The Orange Team

★★★★★ 3.2

Lore ipsum dolor sit amet, consectetur adipiscing elit. Vivamus est massa, interdum non laoreet quis, vehicula eget ligula. Mauris a est metus. Aliquam auctor est non nunc tempus, non vehicula justo gravida. Morbi nec sollicitudin magna. Morbi iaculis iaculis erat a fermentum. Cras vitae ipsum urna. Mauris ac mi vehicula.

Members

Preston Brakus - Owner +82 04 490 19 27

Ernestine Ferry +83 94 161 10 44

Projects

Project dolores - ENDED Elixir R Seed7 Finished on: 24.11.2020 ★★★★★ Feedback →

Project quasi - ONGOING Oxygen Raku Factor Ergonomic executive chair upholstered in bonded black leather and PVC padded seat and back for all-day comfort and support

Project omnis - ENDED ANS Forth Processing Finished on: 24.03.2021 ★★★★★ Feedback →

Figure 43 : Page de présentation d'une équipe dont l'utilisateur ne fait pas partie

Si l'utilisateur est le propriétaire de l'équipe, il peut, en plus de l'affichage, modifier les informations de l'équipe et y gérer les membres en acceptant les nouvelles requêtes, en en bannissant certains ou en léguant les droits à un autre membre de l'équipe.

Figure 44 : Page de présentation d'une équipe dont l'utilisateur est le propriétaire

Ces deux pages de gestion des équipes partagent plusieurs fonctionnalités (rejoindre / quitter une équipe, formulaire d'ajout et de modification) et de ce fait, les mêmes fichiers de Vue peuvent être utilisés comme illustré dans les diagrammes suivants.

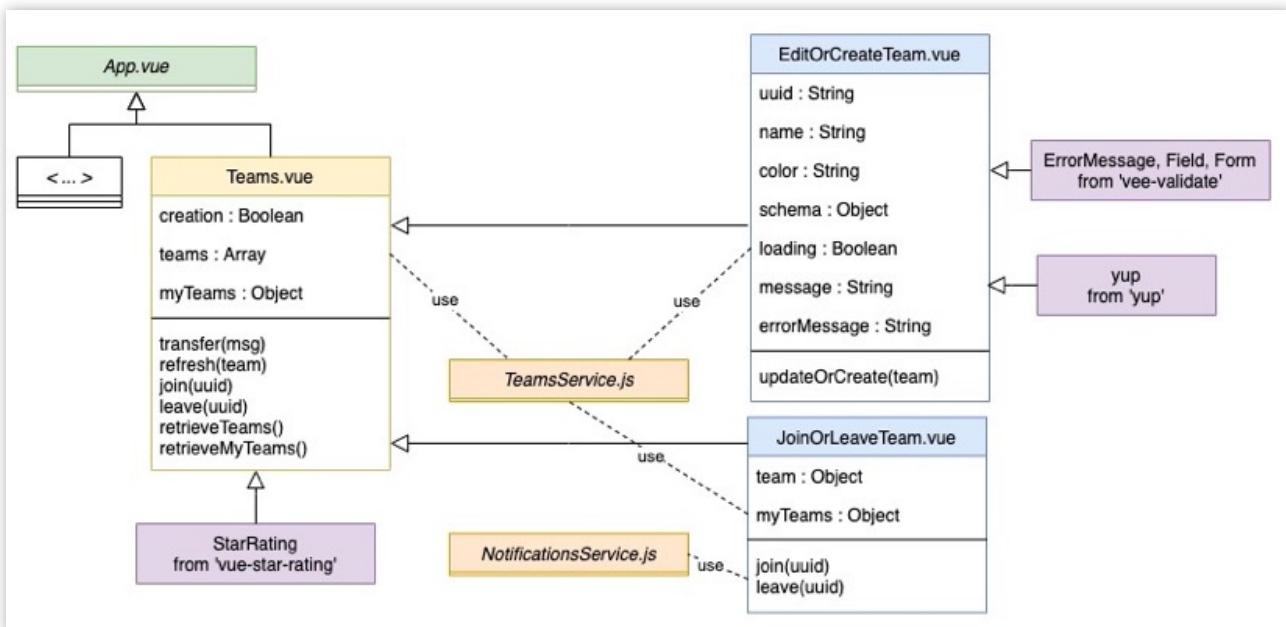


Figure 45 : Diagramme des classes intégrées lors du chargement de la page présentant toutes les équipes, version originale disponible en annexe dans le fichier « Diagrammes/Vue-Teams.jpg »

L'avantage de structurer une application avec des composants est justement cette possibilité de réutilisation et de partage du code illustré ici par les composants « *EditOrCreateTeam.vue* » et « *JoinOrLeaveTeam.vue* ».

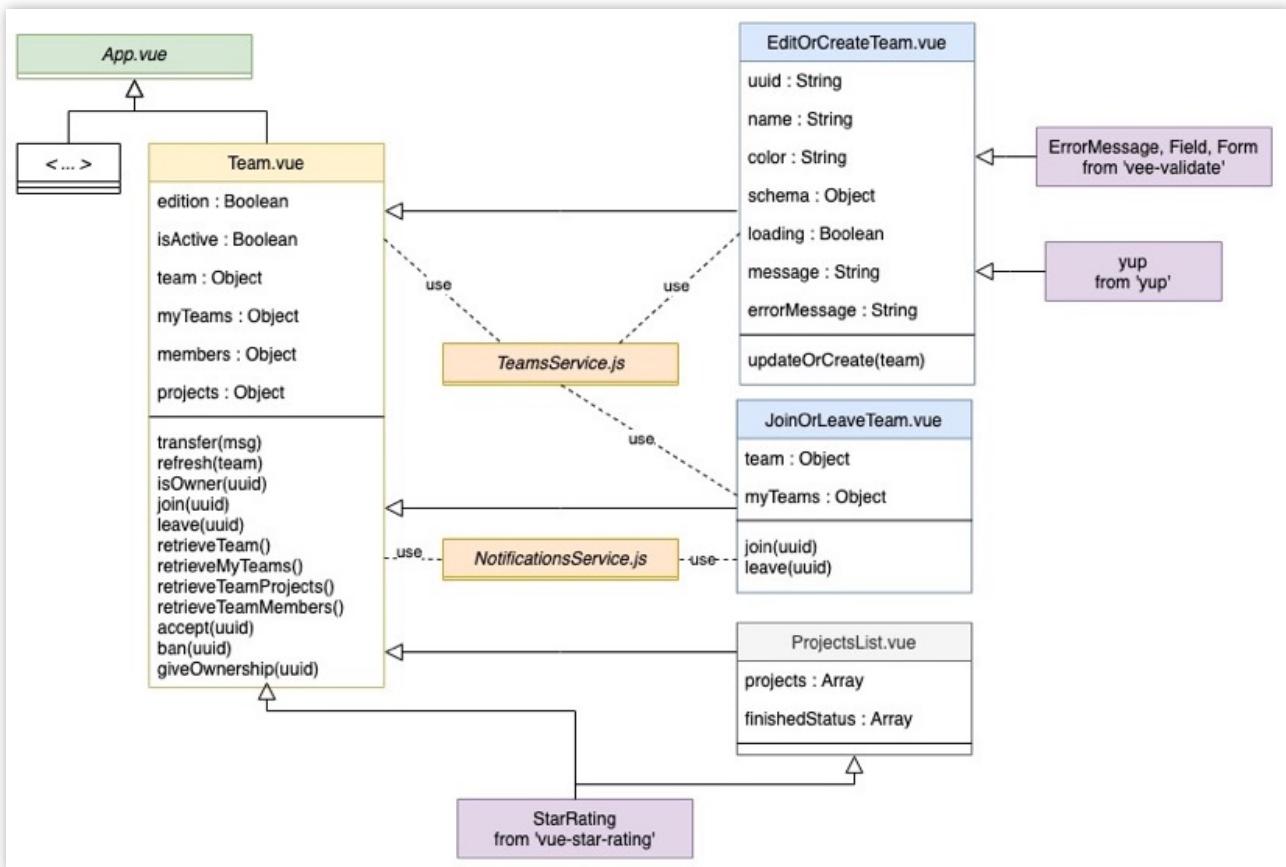


Figure 46 : Diagramme des classes intégrées lors du chargement de la page de présentation d'une équipe, version originale disponible en annexe dans le fichier « Diagrammes/Vue-Team.jpg »

Le composant « `EditOrCreateTeam.vue` » est un formulaire permettant de créer ou de modifier une équipe. Celui-ci est d'abord déclaré dans le `template` avec les champs le composant. Lors de la soumission de celui-ci, les valeurs entrées seront vérifiées grâce au `schema` les décrivant. Si certaines données sont invalides, les messages d'erreurs stockées dans l'objet « `errors` » sont affichés et il est impossible de valider le formulaire. Une fois toutes les erreurs corrigées, la fonction `updateOrCreate` est exécutée.

```

// File : src/components/Layout/EditOrCreateTeam.vue
<template>
    ...
    <Form class="flex flex-col" @submit="updateOrCreate" :validation-schema="schema" v-slot="{ errors }">
        <Field hidden id="uuid" name="uuid" type="text" :value="uuid"/>
        <div class="mb-6 pt-3 rounded bg-gray-200">
            <label class="..." for="name">{{ $t('name') }}</label>
            <Field id="name" name="name" type="text" :value="name" class="..." />
            <ErrorMessage name="name" class="..." />
        </div>
        <div class="mb-6 pt-3 rounded bg-gray-200">
            <label class="..." for="color">{{ $t('color') }}</label>
            <Field id="color" name="color" type="text" :value="color" class="..." />
            <ErrorMessage name="color" class="..." />
        </div>
        <button :disabled="loading || Object.keys(errors).length" class="...">
            <svg v-show="loading" class="..." viewBox="0 0 24 24">...</svg>
            <span>{{ $t('save') }}</span>
        </button>
    </Form>
    </section>
</template>

<script>
...
export default {

```

```
...
  props: { // Define received properties from parent (in edition case only)
    uuid: { type: String, default: '' },
    name: { type: String, default: '' },
    color: { type: String, default: '' }
  },
  data() {
    const schema = yup.object().shape({
      name: yup.string() // Define the team name format and error message
        .required(this.$t('required', { item: this.$t('name') }))
        .max(60, this.$t('maxChars', { max: 60 })),
      color: yup.string() // Define the team color format and error message
        .required(this.$t('required', { item: this.$t('color') }))
        .max(60, this.$t('maxChars', { max: 60 }))
    });
    return {
      schema,
      loading: false,
      message: this.$route.query.message,
      errorMessage: this.$route.query.error,
    };
  },
  methods: {
    async updateOrCreate(team) {
      this.loading = true;
      team.color = team.color.toLowerCase();
      if (team.uuid) { await request(TeamsService.update(team), this) } // Update existing team
      else { team = await request(TeamsService.create(team), this) } // Create a new team
      this.$emit('done', team);
    }
  }
}
</script>
```

La fonction `updateOrCreate` définit alors s'il s'agit d'une édition ou d'une création, appelle la fonction du service `TeamsService` correspondante, puis confirme l'action à son parent en lui retournant l'objet créé ou modifié via « `this.$emit('done', team)` ».

7.2.3.1 Traductions

La syntaxe « `{{ $t('XXX.YYY') }}` » utilisée à plusieurs reprises dans toutes les vues de l'application permet de rendre les différents *templates* traduisibles très facilement. En effet, plutôt que d'écrire directement le mot ou la phrase à afficher, on utilise un mot clé. La valeur de ce mot clé est alors définie dans un fichier *YAML* et ce pour toutes les langues disponibles.

// File : src/Locales/en.yml	// File : src/Locales/fr.yml
# Authentication	# Authentication

```
Forgot:
  title: "Get a new password"
  successful: "Your reset link his on the way..."
  submit: "Redeem a password reset link"
  forgot: "Forgot your password?"
```

```
Forgot:
  title: "Obtenir un nouveau mot de passe"
  successful: "Votre lien de réinitialisation est en route..."
  submit: "Demander un lien de réinitialisation"
  forgot: "Mot de passe oublié ?"
```

De ce fait, lors d'un changement de langue, l'application charge le fichier *YAML* de la langue actuelle comprenant tous les mots-clés définissant ainsi les traductions et le texte à afficher. Cela est géré par la librairie « *vue-i18n* » ainsi que par le fichier de configuration « *src/config/i18n.js* ».

7.2.4 Gestion des projets

La gestion des projets s'opère également via deux types de pages. Premièrement la page « *Projects* » présentant une liste paginée de tous les projets et un filtre permettant d'afficher uniquement les projets « en proposition », ainsi qu'un bouton permettant de créer un nouveau projet. Les projets identifiés par une étoile sont les projets avec lesquels l'utilisateur à un lien.

The screenshot shows a 'Projects' page with a sidebar on the left containing navigation links: Dashboard, Projects (selected), Teams, and Resources. The main area displays a grid of project cards. Each card includes the project name, status, tags, a brief description, and a deadline. A 'Create a new project' button is located in the top right corner.

Project Name	Status	Tags	Description	Deadline
Project eos - ENDED	ENDED	Zig, DarkBasic, Nim, AMOS BASIC	Boston's most advanced compression wear technology increases muscle oxygenation, stabilizes active muscles	Deadline: 03.07.2021
Project qui - ONGOING	ONGOING	Transcript, Swift, J	The Apollotech B340 is an affordable wireless mouse with reliable connectivity, 12 months battery life and modern design	Deadline: 31.07.2022
Coronavirus - ENDED	ENDED	.NET	communiqué du 11 mars 2020	Deadline: 25.09.2021
Project ut - PROPOSAL	PROPOSAL	LiveCode, Clojure	Boston's most advanced compression wear technology increases muscle oxygenation, stabilizes active muscles	Deadline: 16.09.2022
Project aut - ABANDONED	ABANDONED	Kotlin	New range of formal shirts are designed keeping you in mind. With fits and styling that will make you stand apart	Deadline: 17.09.2021
Project dolores - ENDED	ENDED	Elixir, R, Seed7	The Apollotech B340 is an affordable wireless mouse with reliable connectivity, 12 months battery life and modern design	Deadline: 12.01.2021
Project velit - ABANDONED	ABANDONED	S-Lang, Q	The automobile layout consists of a front-engine design, with transaxle-type transmissions mounted at the rear of the engine and four wheel drive	Deadline: 16.09.2021
Project ea - ENDED	ENDED	Dafny, Crystal	The Apollotech B340 is an affordable wireless mouse with reliable connectivity, 12 months battery life and modern design	Deadline: 10.03.2021
Project ut - ONGOING	ONGOING	Gosu, NetRexx, Fortress, K	Andy shoes are designed to keep in mind durability as well as trends, the most stylish range of shoes & sandals	Deadline: 16.08.2022

Figure 47 : Liste paginée présentant un échantillon de tous les projets présents dans l'application

N'importe quel projet peut alors être sélectionné et redirige vers la seconde page de gestion : la gestion d'un projet lui-même. Cette page présente différents éléments évoluant au fur et à mesure du statut du projet. Certains éléments sont tout de même toujours présents. Il s'agit du nom, du statut, des tags et de la description du projet, de la *deadline* ou de la date de finalisation, des équipes candidates (ou plus tard dévelopeuses), de l'équipe mandataire et des ressources liées.

The screenshot shows a 'Project tempore' page with a sidebar on the left containing navigation links: Dashboard, Projects (selected), Teams, and Resources. The main area displays a detailed project view. It includes a 'PROPOSAL' section with tags (Gosu, Brainfuck, Zig, LiveCode) and a description: 'The slim & simple Maple Gaming Keyboard from Dev Byte comes with a sleek body and 7- Color RGB LED Back-lighting for smart functionality'. Below this is a button: 'Apply for this project with « The Fuchsia Team »'. The 'Teams' section shows two teams: 'The Bluegray Team - APPLIES \$ 65761 See specifications →' and 'The Sky Team - MANDATES'. The 'Resources' section lists two files: '[Specifications] Bluegray-Team_Proposition.pdf - PUBLIC' and 'Project_tempore_long_description.docx - PUBLIC', each with a 'Download the file' link.

Figure 48 : Page de présentation d'un projet « en cours de proposition »

Si l'utilisateur fait partie d'équipes et que le projet est « en cours de proposition », un bouton est affiché permettant à une équipe de proposer sa candidature pour la réalisation du projet. Une fois ce bouton cliqué, l'utilisateur est invité à entrer un prix et à téléverser un fichier décrivant sa candidature. Fichier qui s'ajoutera alors aux ressources liées du projet.

Au contraire, si l'utilisateur est le mandant du projet, il peut naviguer entre les candidatures et leurs spécifications, puis en sélectionner une et ainsi faire passer le projet au statut de développement. Il peut également modifier les données liées au projet. Une fois le projet finalisé, l'équipe mandante via son responsable, peut assigner une note au projet et y lier un *feed-back* sous la forme d'un fichier. Le projet est alors clôturé.

The screenshot shows a user interface for managing projects. On the left is a sidebar with navigation links: Dashboard, Projects (with 'Project etamsis' selected), Teams, and Resources. The main content area is titled 'Project etamsis' and shows it is an 'ONGOING' project in '.NET'. It lists 'Project etamsis' three times with a deadline of '01.10.2021'. Below this is a 'Teams' section showing 'The Fuchsia Team - DEVELOPS' and 'The Fuchsia Team - MANDATES'. Under 'Resources', there is a rating of 2.52 based on 5 stars, a file upload area with a placeholder 'Drag 'n' drop some files here, or click to select files', and two download links: '[Specifications] The-big-team-candidate.pdf - PUBLIC' and 'Project-etamsis.pdf - PUBLIC'.

Figure 49 : Page de présentation d'un projet « en développement » pour lequel le mandant est en train d'assigner une note

Les deux pages de gestion des projets partagent également plusieurs fonctionnalités comme le formulaire d'ajout et de modification et de ce fait, les mêmes composants *Vue* peuvent être utilisés comme illustré dans les deux diagrammes suivants.

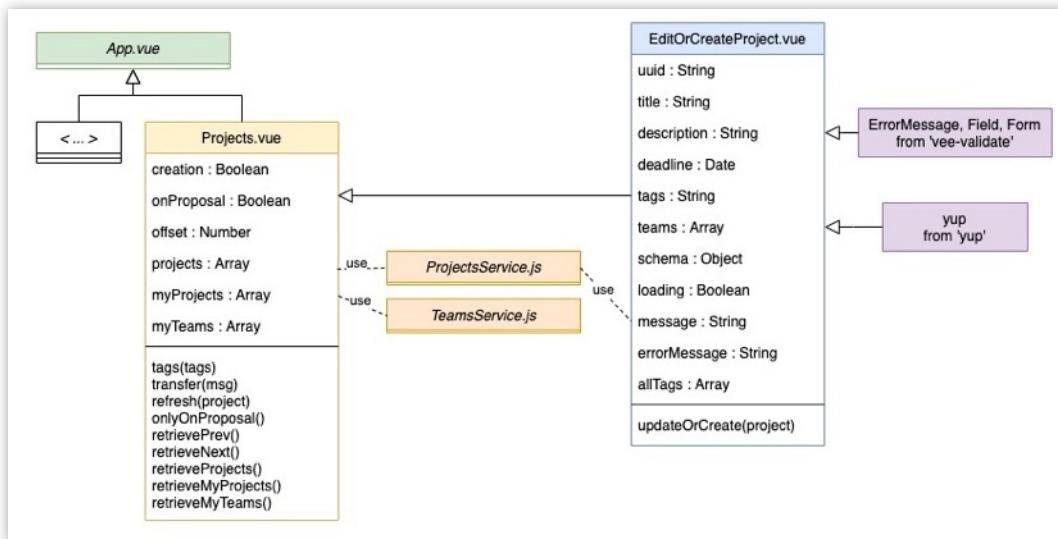


Figure 50 : Diagramme des classes intégrées lors du chargement de la page présentant tous les projets, version originale disponible en annexe dans le fichier « Diagrammes/Vue-Projects.jpg »

L'avantage de structurer une application avec des composants est à nouveau illustré ici par le composant « *EditOrCreateProject.vue* ».

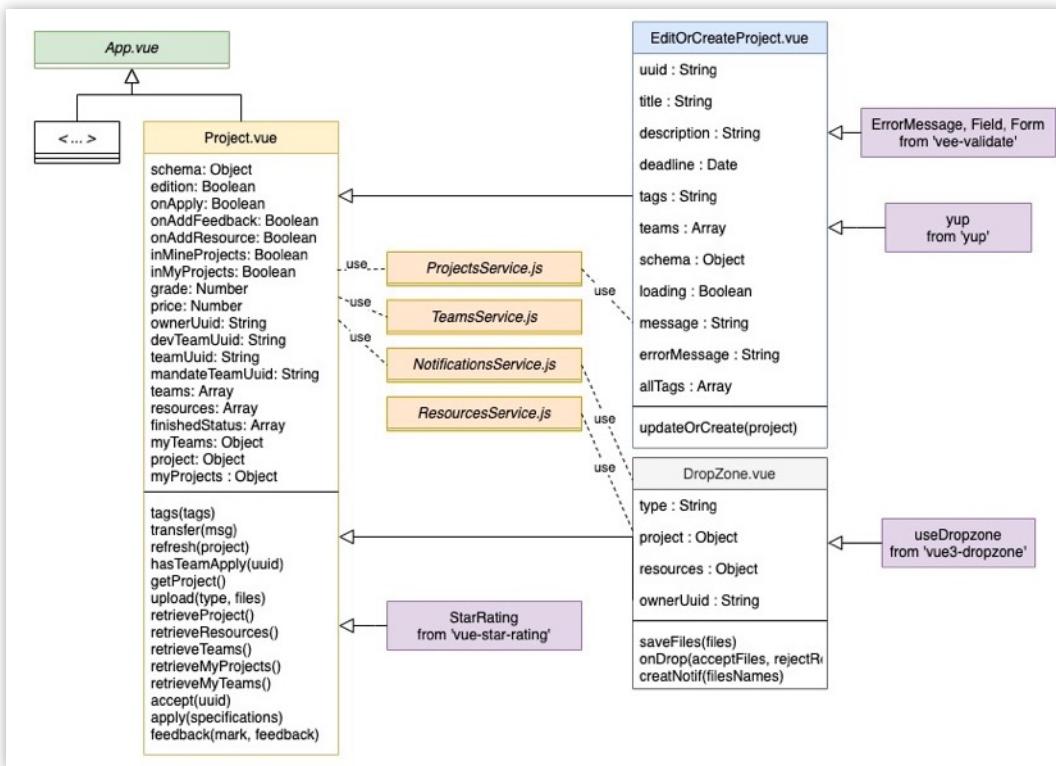


Figure 51 : Diagramme des classes intégrées lors du chargement de la page de présentation d'un projet, version originale disponible en annexe dans le fichier « Diagrammes/Vue-Project.jpg »

Les vues restantes (`Profile.vue`, `User.vue`, `Resources.vue`, etc.) fonctionnant sur le même principe que celles présentées précédemment, je ne vais pas m'y attarder ici.

7.2.5 Déploiement

Le déploiement de cette partie *front-end* ayant déjà étant mis en place et configuré lors de la phase d'analyse sur *Netlify*, il s'est fait automatiquement à chaque *commit*. Le seul élément que j'ai dû ajouter a été la variable d'environnement « `VUE_APP_API_BASE_URL` » permettant à mon application d'exécuter les requêtes API sur mon serveur *Synology*.

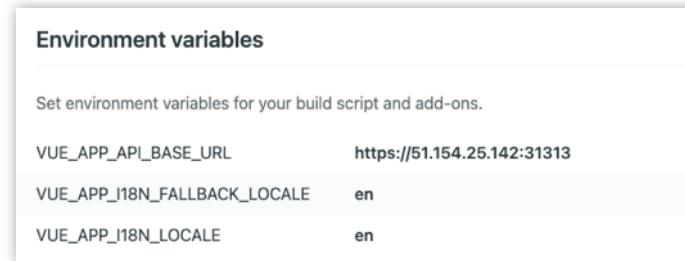


Figure 52 : Configuration des variables d'environnement au sein même du service en ligne « Netlify »

Pour que les requêtes faites depuis mon application sur mon serveur *Synology* soient acceptées, j'ai dû mapper et ouvrir les ports de celui-ci. De plus, les requêtes étant faites en *HTTPS*, il m'a fallu générer un certificat *SSL* et l'ajouter sur mon serveur *Synology*. De ce fait, les requêtes sont acceptées et transmises de façon chiffrée. Je n'ai eu aucun problème avec ce service de déploiement *Netlify* et en suis très satisfait.

7.2.6 Code source

Le code source exécutable de cette partie *front-end* de l'application réalisé avec *Vue.js* est disponible dans le dossier `5.Projet/front-end`. Pour s'exécuter correctement, elle nécessite évidemment une connexion avec l'API.

7.3 Recommandations sur Neo4j à l'aide de Cypher

Je vais créer trois façons différentes de retrouver des projets à recommander à un utilisateur et ce à un instant « *t* » donné. Ces trois manières partent toujours du nœud de l'utilisateur puis utilisent les relations que celui-ci possède avec ces équipes pour identifier des projets à recommander. De ce fait, lorsque les relations de l'utilisateur changeront, mais également lorsque les relations de ses équipes avec les projets changeront, les recommandations seront adaptées. Les trois comportements que j'ai imaginés sont les suivants, la réalisation de chacun d'entre eux est décrite plus en détail dans les sous-chapitres ultérieurs :

1. Recommander des projets de domaines de compétences similaires (via leur tags) en partant des trois projets développés ayant obtenu les meilleures notes.
2. Recommander les nouveaux projets proposés par les mêmes mandants que les projets déjà réalisés avec ceux-ci.
3. Recommander des projets pour lesquels les équipes concurrentes ont postulé.

Pour les réaliser, je vais utiliser *Cypher*³⁵ qui est le langage de requêtes développé et utilisé pour les bases de données *Neo4j*. Une introduction de ce language et la construction d'une requête pas à pas est disponible dans les annexes.

7.3.1 Recommandations N°1

Le but de cette première recommandation est de retrouver les projets des domaines de compétences similaires (via leur tags) en partant des trois projets développés ayant obtenus les meilleures notes. Pour ce faire, je pars de la requête créée pas à pas et j'y ajoute la contrainte de retrouver les projets terminés (*p.status = 6*). Je retourne alors non plus un graphe, mais un tableau contenant les nœuds projets, les notes reçues pour ceux-ci et les tags. Ce tableau est trié par ordre décroissant des notes et est limité aux trois projets ayant les meilleures notes.

```
// Trouver les 3 meilleurs projets développés et terminés par les équipes de l'utilisateur
MATCH (u:User {uuid: '34c5b2f6-28c8-4f52-b7d5-dc188a9d053b'})-[rmo:IS_MEMBER_OF {status: 2}]->(t:Team)
  -[rd:DEVELOPS]->(p:Project)<-[rm:MANDATES]-(m:Team)
WHERE p.status = 6
RETURN p AS Project, rm.mark AS Mark, apoc.convert.fromJsonList(p.tags) AS Tags
ORDER BY rm.mark DESC LIMIT 3
```

"Project"	"Mark"	"Tags"
{"createdAt": "2021-09-10T14:44:13.12400000+02:00", "deadline": "2020-10-14T03:21:09.54800000+02:00", "uuid": "d6c009e5-f6b1-430d-81ad-59b21b6eaf00", "status": 6, "tags": "[\"Hack\", \"Transcript\", \"Gambas\"]"}	4.98	[{"Hack", "Transcript", "Gambas"}]
{"createdAt": "2021-09-10T14:44:13.12400000+02:00", "deadline": "2020-12-13T18:22:23.15300000+01:00", "uuid": "alff606c-b446-484a-9651-916ee3882d35", "status": 6, "tags": "[\"Fantom\", \"Perl\", \"Oxygene\", \"Opa\"]"}	3.14	[{"Fantom", "Perl", "Oxygene", "Opa"}]
{"createdAt": "2021-09-10T14:44:13.12400000+02:00", "deadline": "2021-07-13T17:27:43.51200000+02:00", "uuid": "b9504f2e-51ed-4cc3-83c7-fb745dcac585", "tags": "[\"ActionScript\", \"Mercury\"]", "status": 6}	2.72	[{"ActionScript", "Mercury"}]

Il faut maintenant repartir de cette liste de tags trouvée. Les tableaux de tags sont combinés, et de ce fait, tous les tags trouvés à cette étape sont considérés avec le même poids (*fLatTags*). Je recherche alors tous les projets possédant au minimum un des tags de mon tableau et ayant le statut « proposition » (*recs.status = 4*) en limitant ce nouveau résultat à 10 projets.

```
// Trouver tous les mandants des projets développés par un utilisateur
MATCH (u:User {uuid: '24b4d8f8-9daaa-4c37-bfa5-21fd13db8169'})-[rmo:IS_MEMBER_OF {status: 2}]->(t:Team)
  -[rd:DEVELOPS]->(p:Project)<-[rm:MANDATES]-(m:Team)
WHERE p.status = 6
WITH p
ORDER BY rm.mark DESC LIMIT 3
WITH COLLECT(apoc.convert.fromJsonList(p.tags)) AS tags
```

³⁵ <https://neo4j.com/developer/cypher/>

```
WITH REDUCE(output = [], t IN tags | output + t) AS flatTags
MATCH (recs:Project)
WHERE recs.status = 4 AND p <> recs AND any(t IN apoc.convert.fromJsonList(recs.tags) WHERE t IN flatTags)
RETURN recs AS Recommendations LIMIT 10
```

```
["Recommendations"]
[{"createdAt": "2021-09-10T14:44:13.126000000+02:00", "deadline": "2021-11-27T12:46:13.837000000+01:00", "uuid": "6352a271-b487-4ebc-92aa-f456b1f206fb", "tags": ["PHP", "Transcript", "Rust"], "status": 4}
[{"createdAt": "2021-09-10T14:44:13.127000000+02:00", "deadline": "2022-02-06T13:13:02.729000000+01:00", "uuid": "91d77c67-e4ee-4d56-ad1c-7e50aa3dd3b3", "tags": ["Ceylon", "Gambas"], "status": 4}
[{"createdAt": "2021-09-10T14:44:13.129000000+02:00", "deadline": "2022-06-17T01:58:18.091000000+02:00", "uuid": "84684ea7-f467-4b8b-82e4-00df585f23eb", "tags": ["NetRexx", "Transcript"], "status": 4}
[{"createdAt": "2021-09-10T14:44:13.129000000+02:00", "deadline": "2022-07-11T13:21:34.472000000+02:00", "uuid": "7ed28074-1b96-48ad-b1ce-726648cd613", "tags": ["Fantom", "Groovy"], "status": 4}]
```

Une fois cette requête réalisée, je l'intègre dans un nouveau contrôleur de mon API et lui ajoute les variables dynamiquement. J'exécute alors cette requête à l'aide de l'ORM « *Neode* » avant d'en traiter le résultat. Je procéderai ainsi pour mes trois requêtes de recommandations.

```
// File : app/controllers/recommendations/helpers/findProjectsRecosByTags.js

/**
 * Find recommended projects by tags similarities
 *
 * @param {uuid} uuid - the main project uuid
 * @param {int} limit - the maximum number of recommendations to retrieve
 */
const findProjectsRecosByTags = (uuid, limit = 10) => {
  return new Promise(async (resolve, reject) => {
    const query = `MATCH (u:User {uuid: '${uuid}'})-[rmo:IS_MEMBER_OF {status: ${STATUS_ACTIVE}}]` +
      `-(t:Team)-[rd:DEVELOPS]->(p:Project)<- [rm:MANDATES]-(m:Team)` +
      `WHERE p.status = ${PROJECT_STATUS_ENDED} WITH p ORDER BY rm.mark DESC` +
      `LIMIT ${REF_PROJECTS_LIMIT}` +
      `WITH COLLECT(apoc.convert.fromJsonList(p.tags)) AS tags` +
      `WITH REDUCE(output = [], t IN tags | output + t) AS flatTags` +
      `MATCH (recs:Project)` +
      `WHERE recs.status = ${PROJECT_STATUS_PROPOSAL}` +
      `AND any(t IN apoc.convert.fromJsonList(recs.tags) WHERE t IN flatTags)` +
      `RETURN recs AS recommendations, flatTags LIMIT ${limit}`

    await neo4j.cypher(query, {})
      .then(async res => {
        let projects = []
        if (res.records.length) {
          const flatTags = res.records[0].get('flatTags')
          for (const record of res.records) { ... }
          // Sort projects by number of matching tags
          projects.sort((a, b) => (a.nbOfMatchingTags > b.nbOfMatchingTags) ? -1 : 1)
        }
        resolve(projects)
      })
      .catch(error => { reject(error) })
  })
}
```

7.3.2 Recommandations N°2

La seconde recommandation a pour but de retrouver les nouveaux projets proposés par les mêmes mandants que les projets déjà réalisés avec ceux-ci. Pour ce faire, je pars à nouveau de la requête permettant de retrouver les mandants, mais j'y ajoute cette fois-ci la contrainte de retrouver les projets en cours ou terminés (`p.status IN [5,6]`).

```
// Trouver Les mandants des projets actuellement développés ou terminés par Les équipes de L'utilisateur
```

```

MATCH (u:User {uuid: '34c5b2f6-28c8-4f52-b7d5-dc188a9d053b'})  

  -[rmo:IS_MEMBER_OF {status: 2}]->(t:Team)  

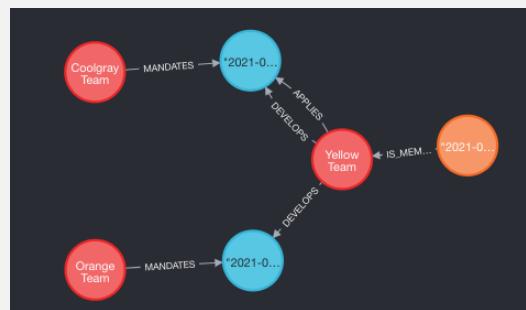
  -[rd:DEVELOPS]->(p:Project)<-[rm:MANDATES]-(m:Team)  

WHERE p.status IN [5,6]  

RETURN u, rmo, t, rd, p, rm, m  

ORDER BY rm.mark DESC LIMIT 10

```



De ces équipes **m**, je cherche tous les projets mandatés par celles-ci ayant le statut « proposition » et n'étant pas déjà en lien avec l'utilisateur, en limitant les résultats à 10 projets.

```

// Trouver les mandants des projets actuellement développés ou terminés par les équipes de l'utilisateur

MATCH (u:User {uuid: '34c5b2f6-28c8-4f52-b7d5-dc188a9d053b'})  

  -[rmo:IS_MEMBER_OF {status: 2}]->(t:Team)  

  -[rd:DEVELOPS]->(p:Project)<-[rm:MANDATES]-(m:Team)  

  -[rm2:MANDATES]->(recs:Project)  

WHERE p.status IN [5,6] AND p <> recs AND recs.status = 4  

RETURN recs AS Recommendations LIMIT 10

```

```

["Recommendations"]
[{"createdAt": "2021-09-10T14:44:13.12400000+02:00", "deadline": "2021-10-26T01:18:07.97700000+02:00", "uuid": "49a2d948-d66c-4959-bc0d-23b0a1e74484", "tags": "[\"J\", \"LiveScript\", \"Io\"]", "status": 4}
]

```

7.3.3 Recommandations N°3

La troisième et dernière recommandation consiste à trouver les projets en proposition pour lesquels les équipes concurrentes ont postulé. Pour ce faire, je commence par rechercher les équipes concurrentes de l'utilisateur.

```

// Trouver les équipes concurrentes via les projets actuellement développés ou terminés

MATCH (u:User {uuid: '34c5b2f6-28c8-4f52-b7d5-dc188a9d053b'})  

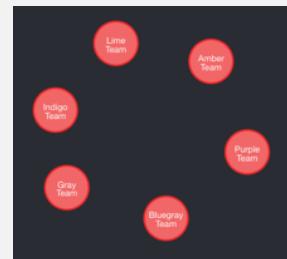
  -[rmo:IS_MEMBER_OF {status: 2}]->(t:Team)  

  -[rd:DEVELOPS]->(p:Project)<-[ra:APPLIES]-(a:Team)  

WHERE p.status IN [5,6] AND t <> a  

RETURN a LIMIT 10

```



Une fois ces équipes concurrentes identifiées, il est plutôt facile de trouver les projets ayant le statut « en propositions » et pour lesquelles ces équipes ont postulées. On se limite à nouveau aux dix premiers résultats.

```

// Trouver les projets pour lesquels les équipes concurrentes ont postulées

MATCH (u:User {uuid: '34c5b2f6-28c8-4f52-b7d5-dc188a9d053b'})  

  -[rmo:IS_MEMBER_OF {status: 2}]->(t:Team)  

  -[rd:DEVELOPS]->(p:Project)<-[ra:APPLIES]-(a:Team)  

  -[ra2:APPLIES]->(recs:Project)  

WHERE p.status IN [5,6] AND t <> a AND p <> recs AND recs.status = 4  

RETURN recs AS Recommendations LIMIT 10

```

```

["Recommendations"]
[{"createdAt": "2021-09-10T14:44:13.12400000+02:00", "deadline": "2021-10-26T01:18:07.97700000+02:00", "uuid": "49a2d948-d66c-4959-bc0d-23b0a1e74484", "tags": "[\"J\", \"LiveScript\", \"Io\"]", "status": 4}
]
[{"createdAt": "2021-09-10T14:44:13.12400000+02:00", "deadline": "2021-10-26T01:18:07.97700000+02:00", "uuid": "49a2d948-d66c-4959-bc0d-23b0a1e74484", "tags": "[\"J\", \"LiveScript\", \"Io\"]", "status": 4}
]

```

7.3.4 Combinaison et résultat

J'ai intégré ces trois recommandations dans mon API en créant un *endpoint* pour chacune d'entre elles. Cependant, voulant obtenir pour un utilisateur une combinaison de projets présents dans une ou plusieurs recommandations, j'ai créé un *endpoint* supplémentaire permettant d'utiliser et de combiner les trois autres. C'est celui-ci qui est interrogé par mon application *Vue.js*.

```
// File : app/routes/recommendations.js

// Get projects recommendations based on other teams applies
router.get('/projects/applies', requireAuth, requiredRole(Role_USER), trimRequest.all, getProjectsRecosByApplies)

// Get projects recommendations based on mandates
router.get('/projects/mandates', requireAuth, requiredRole(Role_USER), trimRequest.all, getProjectsRecosByMandates)

// Get projects recommendations by tags similarities
router.get('/projects/tags', requireAuth, requiredRole(Role_USER), trimRequest.all, getProjectsRecosByTags)

// Get a set of projects recommendations (combination of applies, mandates and tags)
router.get('/projects', requireAuth, requiredRole(Role_USER), trimRequest.all, getProjectsRecommendations)
```

Le contrôleur « `getProjectsRecommendations` » interroge à tour de rôle les contrôleurs « `findProjectsRecosByApplies` », « `findProjectsRecosByMandates` » et « `findProjectsRecosByTags` » et retrouve ainsi trois listes pouvant contenir des projets.

```
// File : app/routes/recommendations.js

/**
 * Get projects recommendations by tags similarities when called by route
 *
 * @param {Object} req - request object
 * @param {Object} res - response object
 */
const getProjectsRecommendations = async (req, res) => {
  try {
    let recommendedProjects = {}
    const limit = req.query.limit ? parseInt(req.query.limit) : 10
    const user = await findUserByUuid(req.user.uuid)
    const percentage = (100 / NB_OF_RECOMMENDATIONS)

    await findProjectsRecosByApplies(user.uuid, limit).then((projects) => { ... })
    await findProjectsRecosByMandates(user.uuid, limit).then((projects) => { ... })
    await findProjectsRecosByTags(user.uuid, limit).then((projects) => { ... })

    recommendedProjects = Object.values(recommendedProjects)
    recommendedProjects.sort((a, b) => (a.recommendedAt > b.recommendedAt) ? -1 : 1)
    res.status(200).json(recommendedProjects.slice(0, limit))
  } catch (error) { handleError(res, error) }
}
```

Chacune des listes de projets est alors parcourue et assigne à ses projets un pourcentage de recommandation allant de quelques pourcents à 33.3% ($100\% \text{ divisé par } 3 \text{ recommandations possibles}$) selon différents critères. Chaque projet pouvant être retrouvé dans plusieurs listes, les pourcentages de ceux-ci s'additionnent pour former un pourcentage de recommandation final.

Prenons un exemple avec les listes de projets retrouvés par *Cypher* suivantes :

Contrôleur	Projets
<code>findProjectsRecosByMandates</code>	A, B, C
<code>findProjectsRecosByTags</code>	A, C
<code>findProjectsRecosByApplies</code>	C, D

- Le pourcentage assigné pour une recommandation par « *mandates* » est lié à la note obtenue lors de la réalisation du projet précédent ; note pouvant aller de 0.1 à 5.0. De ce fait, un projet ayant obtenu une note de 2.6 obtiendra un pourcentage de 17.3%. Un projet ayant obtenu une note de 4.3 obtiendra un pourcentage de 28.6%, etc. Dans notre exemple le projet A (3.6) obtient 24.0%, le projet B (4.9) 32.6% et le projet C (4.3) 28.6%.

- Le pourcentage assigné pour une recommandation par « *tags* » est proportionnel au nombre de tags similaires avec une limite haute de cinq tags. De ce fait, un projet recommandé ayant une similarité d'un seul tag obtiendra un pourcentage de 6.6%. Un projet recommandé ayant une similarité de deux tags obtiendra un pourcentage de 13.3% et ainsi de suite, jusqu'à un projet ayant une similarité de cinq tags qui obtiendra un pourcentage de 33.3%. Dans notre exemple le projet A (3 tags) obtient 20.0% et le projet C (1 tag) 6.6%.
- Le pourcentage assigné pour une recommandation par « *applies* » fonctionne sur le même principe qu'une recommandation par « *tags* » avec une limite haute de dix « *apply* ». De ce fait, un projet recommandé ayant seul « *apply* » obtiendra un pourcentage de 3.3%, un projet recommandé ayant une similarité de six « *apply* » obtiendra un pourcentage de 20.0%, et ainsi de suite jusqu'à un projet ayant une similarité de dix « *apply* » qui obtiendra un pourcentage de 33.3%. Dans notre exemple le projet C (4 *applies*) obtient 13.3% et le projet D (8 *applies*) 26.6%.

Le contrôleur « `getProjectsRecommendations` » combine les trois listes de projets et retourne finalement un tableau de projets (triés par ordre décroissant de pourcentage de recommandation) pouvant être affiché par l'application *Vue.js* à l'utilisateur. Dans notre exemple, cette liste finale se composera donc comme suit.

Projet	Mandates	Tags	Applies	Recommandation
C	28.6	6.6	13.3	48.5 %
A	24.0	20.0	-	44.0 %
B	32.6	-	-	32.6 %
D	-	-	26.6	26.6 %

L'application *Vue.js* formate alors les données reçues et affiche sur le « *Dashboard* » de l'utilisateur les trois projets actuellement recommandés avec le taux de pourcentage de la recommandation. Ceux-ci vont évidemment changer au fur et à mesure des actions de l'utilisateur (finalisation d'un projet, postulation à un nouveau projet, etc.).

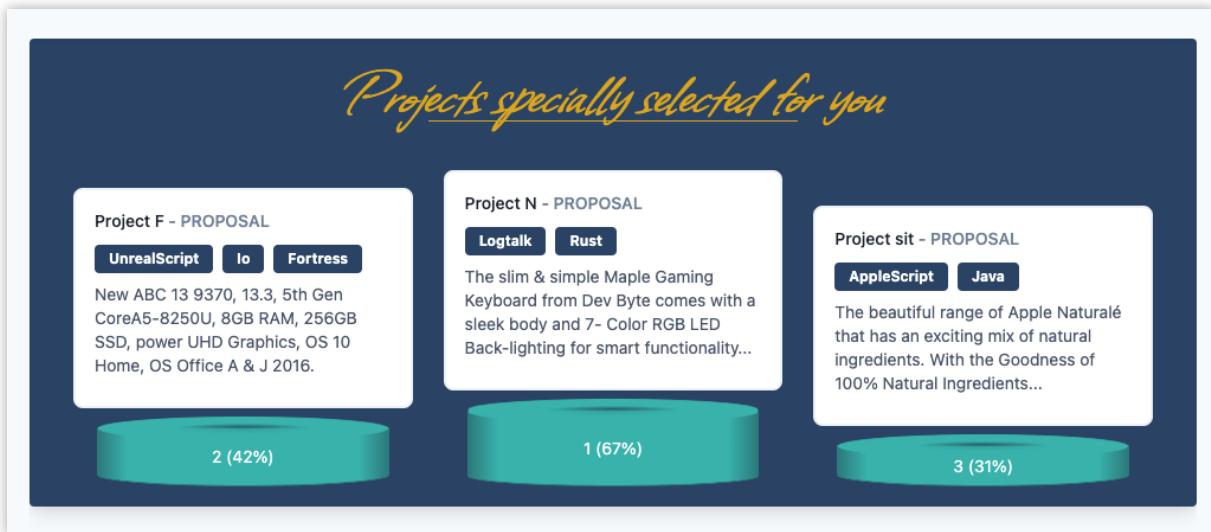


Figure 53 : Résultat de recommandations de projets s'affichant sur le Dashboard d'un utilisateur

7.4 Tests de bout en bout

Plusieurs tests de validation des fonctionnalités du point de vue d'un utilisateur permettant de confirmer que le système fonctionne comme prévu ont été réalisés. Pour ce faire, j'ai principalement utilisé *macOS Big Sur* comme système d'exploitation avec le navigateur *Google Chrome* en version 93. L'environnement utilisé pour ces tests a été la version mise en ligne sur *Netlify* interrogeant l'API hébergé sur mon serveur *Synology*. Tous les résultats de ces tests sont disponibles dans le fichier annexe « *Tests_End-to-end.xlsx* ».

8 AMÉLIORATIONS

Du fait que l'application n'est actuellement qu'à sa toute première version, de nombreuses améliorations devraient encore être mises en place et développées. Cela permettrait de fortifier les fonctionnalités mises en place, la sécurité et l'expérience utilisateur de manière générale.

8.1 Back-end

8.1.1 Meilleure gestion des erreurs et des codes d'erreur

Actuellement, j'utilise principalement le code d'erreur « *422 Unprocessable Entity* » pour créer les réponses des requêtes ayant généré un problème. Il serait judicieux de mieux analyser les sources d'erreurs et d'ainsi mieux utiliser les normes des codes de réponse *HTTP*.

8.1.2 Ajout des catégories aux ressources

Ma base de données relationnelles permet d'ajouter diverses catégories aux ressources. Ces catégories n'ont pour l'instant pas été mises en place et devrait l'être via *l'API*.

8.1.3 Ajout d'autres notifications

Le système d'ajout et de traitement des notifications développées est pratique et performant. De ce fait, il serait judicieux d'ajouter des notifications pour la plupart des actions effectuées par les utilisateurs et d'améliorer ainsi l'expérience et l'historique des activités.

8.1.4 Améliorations des algorithmes de recommandations

Les algorithmes de recommandations mis en place sont des algorithmes basiques. Avec plus de temps, j'aurais voulu les améliorer et notamment y intégrer certains éléments de la théorie des graphes. De plus, le langage *Cypher* permet certainement des requêtes avancées beaucoup plus performantes.

8.1.5 Mise en place d'un système de *logs* efficaces

Actuellement, aucun élément (mis à part les consoles) nous permet de moniter les erreurs rencontrées par les utilisateurs sur *l'API* ou sur l'application *front-end*. Il serait judicieux de mettre en place un système de sauvegarde de *logs* sous la forme de fichiers textes et ce sur un volume persistant.

8.2 Front-end

8.2.1 Forum et espace de travail

La mise en place d'un forum intégré dans l'espace de travail de chacun des projets notamment décrit dans l'épic 8 : « *Un acteur du projet doit pouvoir transmettre une information à tous les participants d'un projet.* », a volontairement été omise. Avec tous les outils de communication présents aujourd'hui (*Microsoft Teams*, *Slack*, etc.), il faudrait analyser ce que ce forum pourrait véritablement mettre en place comme valeur ajoutée.

8.2.2 Champs de recherche sur tout le site

Le champ de recherche présent sur l'en-tête de toutes les pages devrait permettre d'effectuer une recherche sur l'ensemble des éléments de l'application comme : les utilisateurs, les équipes, les projets, les ressources, mais également les réglages, la gestion du profil, les notifications, etc.

8.2.3 Sécuriser les requêtes

Un important travail de sécurité devrait être fait avant la mise en production. Par exemple, le fait d'ajouter des *tokens* de validation des requêtes permettant de valider que celles-ci ont bien été envoyées depuis le *front-end* et non pas forgées (éviter les *cross-site request forgery*).

8.2.4 Amélioration de la mise en cache des requêtes

Actuellement, seules les requêtes permettant de retrouver les équipes et les projets dont un utilisateur fait partie sont mises en cache (*MyProjects*, *MyTeams*). Il serait judicieux d'analyser quelles autres requêtes pourraient être mises en cache et pour combien de temps.

9 CONCLUSION

Tout au long de ce travail de Bachelor clôturant mes études, j'ai pu enrichir mes connaissances des technologies web et plus principalement du langage *JavaScript*. A l'issu de ce travail, je remarque néanmoins que le *JavaScript* reste un langage complexe et long à mettre en place et ce, même au travers des différents frameworks que j'ai pu explorer. De plus, au travers de ce projet j'ai aussi pu tester et utiliser le déploiement avec *Docker* dans une situation pratique. Cette prise en main ne fut pas une étape particulièrement aisée, mais celle-ci m'a apporté de nouvelles connaissances très intéressantes à exploiter dans le monde professionnel. J'ai également beaucoup apprécié la découverte de *Neo4j* et des bases de données orientées graphe, ainsi que le langage de requêtes *Cypher* dont la syntaxe et l'utilisation me plaisent fortement.

Ce projet a été vraiment très intéressant à mener et à développer pour moi. Il reste certes de nombreuses heures de travail à réaliser pour ajouter des fonctionnalités, mais le résultat que j'ai pu obtenir lors de cette première version est d'après moi plutôt satisfaisant. J'ai par ailleurs bien pu suivre ma planification établie au préalable et tous les points inclus dans celle-ci ont été réalisés. L'élaboration de ce travail, en parallèle de mon emploi n'a pas été facile tous les jours pour moi et j'aurais souhaité avoir plus de temps pour pouvoir mettre en place les améliorations décrites ainsi que les extensions citées ci-dessus.

Arrivé au terme de ce long travail, je tire un bilan positif de cette expérience qui m'a permis d'avoir une meilleure vision des frameworks *JavaScript* présents dans le monde professionnel d'aujourd'hui. Durant l'élaboration de ce travail, j'ai été confronté à des problèmes qui me semblaient complexes à première vue, mais que j'ai pu résoudre en les divisant et en les traitant par incrémentation. Ce projet, pour lequel je suis sorti de ma zone de confort, m'a permis d'opérer des choix ambitieux et d'acquérir de nouvelles compétences qui me seront, je l'espère, très utiles pour mon futur.

En définitive, découvrir et expérimenter la façon de représenter et de stocker les données dans des bases orientées graphe ont vivement consolidé mon envie de me former davantage dans la gestion et l'analyse des données. La partie déploiement avec *Docker* a attisé ma curiosité et m'a fait prendre conscience qu'il serait judicieux d'explorer plus en profondeur le domaine du « *cloud computing* » et des architectures microservices. C'est pourquoi, lors de mes projets futurs, j'analyserai encore plus les possibilités offertes par ces différents domaines afin d'en tirer tous leurs avantages.

Glossaire

Ajax	Méthode permettant d'effectuer des requêtes à un serveur et d'en afficher les résultats sans avoir besoin de recharger une page web complète
API	<i>Application Programming Interface</i> , ensemble normalisé de méthodes servant de façade par laquelle un logiciel peut offrir des services à d'autres logiciels
Back-end	Couche représentant l'accès aux données d'un logiciel, travail réalisé par le serveur
Callback	Une fonction de rappel (<i>callback</i>) est une fonction qui est passée en argument à une autre fonction. Cette autre fonction peut alors <i>rappeler</i> la fonction argument à un moment donné.
Commit	Anglicisme désignant l'enregistrement effectif d'une transaction dans un système de révision de fichier
CSS	<i>Cascading Style Sheets</i> , langage informatique décrivant la présentation d'une page web
DOM	<i>Document Object Model</i> , interface de programmation normalisée permettant à des scripts d'examiner et de modifier le contenu d'une page web
Endpoint	Points de contact de la communication lors de l'interaction avec une <i>API</i>
Framework	Ensemble de composants structurels servant à créer les fondations d'un logiciel
Front-end	Couche de présentation, travail réalisé par le client
JavaScript	Langage de programmation de scripts principalement employé dans les pages web
JSON	<i>JavaScript Object Notation</i> , format de données textuelles permettant de représenter de l'information structurée
GIT	Logiciel de gestion de versions décentralisé
Hello, World!	Programme informatique très simple permettant d'illustrer la syntaxe de base d'un langage de programmation
HTML	<i>Hypertext Markup Language</i> , langage de balisage conçu pour représenter les pages web
HTTP	<i>Hypertext Transfer Protocol</i> , protocole de communication client-serveur développé pour le World Wide Web
HTTPS	Variante sécurisée du protocole HTTP à travers l'usage des protocoles spécifiques (TLS).
JS	Abréviation de <i>JavaScript</i>
JSON	<i>JavaScript Object Notation</i> , format de données textuelles permettant de représenter de l'information sous forme structurée
NPM	Gestionnaire de paquets officiel de Node.js
Open Source	Logiciel respectant les principes de libre accès au code source et de libre redistribution
ORM	<i>Object–Relational Mapping</i> , interface entre un programme applicatif et une base de données relationnelle
PHP	<i>Hypertext Preprocessor</i> , langage de programmation libre, principalement utilisé pour produire des pages web dynamiques
PME	Petite ou moyenne entreprise

Repository	Anglicisme de dépôt ou référentiel représentant un stockage centralisé et organisé de données
Responsive	Technique de conception de site web visant à offrir une consultation confortable sur toutes les tailles d'écrans
SCRUM	Cadre de développement de produits logiciels
SEO	<i>Search Engine Optimisation</i> , ensemble de techniques visant à améliorer le positionnement d'un site web dans les résultats de recherche d'un moteur de recherche
SGBDR	Système de gestion de bases de données relationnelles
SQL	<i>Structured Query Language</i> , langage informatique normalisé servant à exploiter des bases de données relationnelles
UI	<i>User interface</i> , dispositif matériel ou logiciel permettant à un usager d'interagir avec un produit informatique
UUID	<i>Universally unique identifier</i> , système permettant d'identifier de façon unique une information
UX	<i>User experience</i> , qualité du vécu d'un utilisateur dans un environnement numérique ou physique
XML	<i>Extensible Markup Language</i> , langage de balisage extensible
Wireframe	Schéma de conception d'une interface graphique définissant les zones et les composants que celle-ci doit englober
YAML	<i>Yet Another Markup Language</i> , format de représentation de données par sérialisation, alternative à JSON

Bibliographie

Wappalyzer. (Visité le 14.06.2021). *UI frameworks*.

<https://www.wappalyzer.com/technologies/ui-frameworks/>

W³Techs. (Visité le 23.06.2021). *Technologies Overview*.

<https://w3techs.com/technologies/>

The State of JavaScript Survey. (Visité le 23.06.2021). *Front-end Frameworks*.

<https://2020.stateofjs.com/en-US/technologies/front-end-frameworks/>

The State of JavaScript Survey. (Visité le 23.06.2021). *Back-end Frameworks*.

<https://2020.stateofjs.com/en-US/technologies/back-end-frameworks/>

Baumann A. (12 octobre 2016). Une brève histoire du web en 8 étapes.

<https://applitude.ch/digital-insights/une-histoire-du-web/>

Historique du web. (Visité le 23.06.2021).

https://www.editions-ellipses.fr/index.php?controller=attachment&id_attachment=29972

Design patterns. (Visité le 24.06.2021). *The Catalog of Design Patterns*.

<https://refactoring.guru/design-patterns/catalog>

Frossard J. (2019). *Éléments d'architecture logicielle*.

https://www.epai-ict.ch/ict-modules/assets/M120_Architecture.pdf

Sutherland J. & Schwaber K. (2020, novembre). *The Scrum Guide*.

<https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf>

Rectorat HES-SO. *Eléments et démarche pour une analyse environnementale à la HES-SO*.

<https://www.hes-so.ch/data/documents/Brochure-Guide-theorique-Elements-et-demarche-pour-analyse-environnementale-HES-SO-6085.pdf>

Shaumik Daityari. (Visité le 08.07.2021). *Angular vs React vs Vue: Which Framework to Choose in 2021*.

<https://www.codeinwp.com/blog/angular-vs-vue-vs-react/>

Aris Pattakos. (Visité le 08.07.2021). *Angular vs React vs Vue 2021*.

<https://athemes.com/guides/angular-vs-react-vs-vue/>

Fireship. *Vue.js Explained in 100 Seconds ; Angular in 100 Seconds ; React in 100 Seconds*.

<https://www.youtube.com/watch?v=nhBVL41-Cw> ;

<https://www.youtube.com/watch?v=Ata9cSC2WpM> ;

<https://www.youtube.com/watch?v=Tn6-Plqc4UM>

Rlogical Techsoft.Pvt.Ltd. (Visité le 09.07.2021). *4 JavaScript Frameworks You Should Consider For Your Next Web App*.

<https://javascript.plainenglish.io/4-javascript-frameworks-you-should-consider-for-your-next-web-app-6feceefc0ae0>

Krunal Shah. (Visité le 14.07.2021). *PHP vs Nodejs: What To Choose in 2021*.

<https://www.thirdrocktechkno.com/blog/php-vs-nodejs-what-to-choose-in-2021/>

Fiverr. (Visité le 21.07.2021).

<https://fr.fiverr.com/>

Talents Connection. (Visité le 21.07.2021).

<https://www.talentsconnection.ch/>

Frédéric Lelièvre. (Visité le 21.07.2021). *Un site Web aide les indépendants spécialisés dans l'informatique et la communication à trouver des mandats.*

<https://www.letemps.ch/economie/un-site-web-aide-independants-specialises-linformatique-communication-trouver-mandats>

BezKoder. (Visité le 30.07.2021). *Vue 3 Authentication with JWT, Vuex, Axios and Vue Router.*

<https://www.bezkoder.com/vue-3-authentication-jwt/>

Andreas Löw. (Visité le 12.08.2021). *How to translate your Vue.js application with vue-i18n.*

<https://www.codeandweb.com/babeledit/tutorials/how-to-translate-your-vue-app-with-vue-i18n>

neo4j.com. (Visité le 01.09.2021). *Build a Cypher Recommendation Engine.*

<https://neo4j.com/developer/cypher/guide-build-a-recommendation-engine/>

nodejs.org. (Visité le 16.09.2021). *Dockerizing a Node.js web app.*

<https://nodejs.org/de/docs/guides/nodejs-docker-webapp/>

Geshan Manandhar. (Visité le 16.09.2021). *Use Node.js with Docker and Docker Compose to improve DX.*

<https://blog.logrocket.com/node-js-docker-improve-dx/>

Guillaume Robez. (Visité le 02.10.2021). *48 plateformes freelance pour trouver des missions en 2021.*

<https://independant.io/plateforme-freelance/>

Steven Wagner. (Visité le 02.10.2021). *Swico anticipe la croissance du marché TIC en Suisse.*

<https://www.ictjournal.ch/etudes/2020-01-21/swico-anticipe-la-croissance-du-marche-tic-en-suisse>

Liste des figures

Figure 1 : Idée de « Business Model Canvas » pouvant servir de base pour la plateforme web à réaliser.....	18
Figure 2 : Couleur primaire (PANTONE 534 CP) et couleur secondaire (PANTONE 2398 CP), https://www.pantone.com/eu/fr/color-finder/534-CP , https://www.pantone.com/eu/fr/color-finder/2398-CP	23
Figure 3 : Palette de nuances de gris choisie pour la réalisation de l'application, https://tailwindcss.com/docs/customizing-colors	23
Figure 4 : Logotype et nom de la plateforme dans les couleurs définies et sa nuance en noir et blanc.....	24
Figure 5 : Logo de Bootstrap depuis sa version 5, https://commons.wikimedia.org/wiki/File:Bootstrap_logo.svg	25
Figure 6 : Évolution du terme de recherche "Bootstrap" sur les 5 dernières années, https://trends.google.fr/trends/explore?date=today%205-y&q=Bootstrap	25
Figure 7 : Logo de Tailwind CSS, https://tailwindcss.com/brand/	26
Figure 8 : Évolution du terme de recherche " Tailwind CSS" sur les 5 dernières années, https://trends.google.fr/trends/explore?date=today%205-y&q=Tailwind%20CSS	26
Figure 9 : Maquette représentant le squelette de l'application	28
Figure 10 : Maquette représentant la page d'entrée et d'identification de l'application	29
Figure 11 : Pourcentages de répartition des langages de programmation côté client utilisés par les sites web en juin 2021, https://w3techs.com/technologies/overview/client_side_language	32
Figure 12 : Évolution d'utilisation des frameworks JavaScript front-end de 2016 à 2020, https://2020.stateofjs.com/en-US/technologies/front-end-frameworks/	33
Figure 13 : Technologies de bibliothèques JavaScript les plus utilisées en se basant sur la part de marché en 2021, https://www.wappalyzer.com/technologies/javascript-libraries/	33
Figure 14 : Évolution d'utilisation "(l'utiliserait à nouveau + ne l'utiliserait plus) / total" des frameworks JavaScript back-end de 2017 à 2020, https://2020.stateofjs.com/en-US/technologies/back-end-frameworks/	38
Figure 15 : Interrogation avec Postman du endpoint « /users »	45
Figure 16 : Point de départ d'une application React.js	45
Figure 17 : Point de départ d'une application Vue.js fraîchement créée	48
Figure 18 : Page d'accueil d'un projet Angular lors de sa création	51
Figure 19 : Comparaison du poids de chacun des projets réalisés.....	53
Figure 20 : Choix final des frameworks JavaScript back-end et front-end	54
Figure 21 : Schéma initial de la base de données relationnelle	56
Figure 22 : Schéma de la base de données relationnelle incluant la gestion multilingue.....	57
Figure 23 : Représentation simplifiée des différents nœuds et arcs possibles	60
Figure 24 : Structure du code de l'API back-end avec Express.....	64
Figure 25 : Contenu du dossier « routes » de l'API, un fichier par domaine d'endpoints	64
Figure 26 : Contenu du dossier « middleware » de l'API	67
Figure 27 : Exemple de résultat d'une requête Cypher construite par les middlewares	68
Figure 28 : Diagramme de séquence du lancement de l'API se terminant par l'attente de requêtes, version originale disponible en annexe dans le fichier « Diagrammes/API-Sequence-start.png »	70
Figure 29 : Diagramme de séquence du traitement de l'API lors d'un accès au endpoint « /projects/:uuid », version originale disponible en annexe dans le fichier « Diagrammes/API-Sequence-getProject.png »	71
Figure 30 : Diagramme de séquence du traitement de l'API lors d'un appel à la fonction « getProject », version complète disponible en annexe dans le fichier « Diagrammes/API-Sequence-getProject.png »	72
Figure 31 : Diagramme présentant les étapes de l'enregistrement pour un utilisateur, version originale disponible en annexe dans le fichier « Diagrammes/User-Registration.png ».....	73
Figure 32 : Diagramme présentant les étapes de connexion d'un utilisateur et l'obtention d'un token, version originale disponible en annexe dans le fichier « Diagrammes/User-Login.png »	73
Figure 33 : Diagramme présentant les étapes de connexion d'un utilisateur et l'obtention d'un token, version complète disponible en annexe dans le fichier « Diagrammes/API-User-Login-Full.png »	74
Figure 34 : Diagramme présentant les étapes de réinitialisation du mot de passe d'un utilisateur, version originale disponible en annexe dans le fichier « Diagrammes/User-ForgotPassword.png ».....	76
Figure 35 : Rapports de résultats des tests unitaires effectués sur le endpoint « /teams »	79
Figure 36 : Affichage des quatre conteneurs Docker de l'API MoonFish s'exécutant sur un serveur Synology	81
Figure 37 : Structure du code de l'application front-end avec Vue.js	82

Figure 38 : Présentation du Dashboard d'accueil exposant les recommandations de projets et l'historique des activités	83
Figure 39 : Diagramme des classes chargées sur chacune des pages de l'application et formant ainsi son squelette, version originale disponible en annexe dans le fichier « Diagrammes/Vue-App.jpg »	84
Figure 40 : Diagramme des classes supplémentaires intégrées lors du chargement de la page Dashboard, disponible en annexe dans le fichier « Diagrammes/Vue-Dashboard.jpg »	84
Figure 41 : Liste de toutes les équipes présentent dans l'application	86
Figure 42 : Formulaire de création d'une équipe	86
Figure 43 : Page de présentation d'une équipe dont l'utilisateur ne fait pas partie.....	86
Figure 44 : Page de présentation d'une équipe dont l'utilisateur est le propriétaire.....	87
Figure 45 : Diagramme des classes intégrées lors du chargement de la page présentant toutes les équipes, version originale disponible en annexe dans le fichier « Diagrammes/Vue-Teams.jpg »	87
Figure 46 : Diagramme des classes intégrées lors du chargement de la page de présentation d'une équipe, version originale disponible en annexe dans le fichier « Diagrammes/Vue-Team.jpg »	88
Figure 47 : Liste paginée présentant un échantillon de tous les projets présents dans l'application.....	90
Figure 48 : Page de présentation d'un projet « en cours de proposition »	90
Figure 49 : Page de présentation d'un projet « en développement » pour lequel le mandant est en train d'assigner une note	91
Figure 50 : Diagramme des classes intégrées lors du chargement de la page présentant tous les projets, version originale disponible en annexe dans le fichier « Diagrammes/Vue-Projects.jpg »	91
Figure 51 : Diagramme des classes intégrées lors du chargement de la page de présentation d'un projet, version originale disponible en annexe dans le fichier « Diagrammes/Vue-Project.jpg »	92
Figure 52 : Configuration des variables d'environnement au sein même du service en ligne « Netlify »	92
Figure 53 : Résultat de recommandations de projets s'affichant sur le Dashboard d'un utilisateur	97
Figure 54 : Première page d'internet publiée en décembre 1990, https://www.w3.org/History/19921103-hypertext/hypertext/WWW/TheProject.html	111
Figure 55 : Exemple de ce à quoi pourrait ressembler la première page HTML en lui appliquant quelques styles CSS.....	111
Figure 56 : Différences entre Back-end / Front-end et Full Stack, 11 février 2021, https://www.leproductowner.com/fiches-metiers/backend-frontend-fullstack	113
Figure 57 : Représentation des interactions entre le modèle, la vue et le contrôleur dans le cas d'une application web, https://commons.wikimedia.org/wiki/File:Mod%C3%A8le-vue-contr%C3%B4leur_(MVC)_-_fr.png ?uselang=fr	114
Figure 58 : Représentation de l'architecture MVVM et des interactions entre la vue, la vueModèle et le modèle, https://en.wikipedia.org/wiki/File:MVVMPattern.png	115
Figure 59 : Représentation de l'architecture 3-tier et des langages utilisés dans chacune des couches, https://www.slideshare.net/TharinduWeerasinghe/multitier-designs-in-software	115
Figure 60 : Pourcentages de répartition des langages de programmation côté serveur utilisés par les sites web en juin 2021, https://w3techs.com/technologies/overview/programming_language	117

Annexes

10 ANNEXES

10.1 Repository GitHub

Un *repository GitHub* privé dédié au projet et contenant toutes les ressources de celui-ci est disponible à l'adresse suivante : https://github.com/weevoed/HEIG-VD_Travail-de-Bachelor.

10.1.1 Code source

- Le code source compilable de la partie *back-end* de l'application réalisé avec *Express* est disponible dans le dossier *5.Projet/back-end*.
- Le code source exécutable de la partie *front-end* de l'application réalisé avec *Vue.js* est disponible dans le dossier *5.Projet/front-end*.
- Le code source du conteneur *Docker* utilisant *docker-compose* est consultable dans le dossier *5.Projet/docker*.

10.1.2 Journal de travail

Mon planning ainsi qu'un journal de travail, sous la forme d'un tableur Excel, est consultable dans le fichier *TB_Planning_Alt-Thibaud.xlsx*.

10.1.3 Maquettes HTML

Des maquettes HTML statiques ont été réalisées ; les sources de celles-ci sont disponibles dans le dossier *3.Maquettes*

10.2 Ressources externes

10.2.1 Documentation de l'API

La définition et la documentation des routes disponibles dans l'API « *MoonFish - Express.js REST API with JWT* » réalisé à l'aide de *Postman* sont consultables directement en ligne via l'adresse suivante :

<https://documenter.getpostman.com/view/8210926/TzseHmCz>.

Un résumé de l'ensemble des routes actuellement définies et interrogables est présenté dans le tableau suivant :

	Méthode	Chemin	Action
Authentication	POST	/auth/login	Connecte l'utilisateur
	POST	/auth/register	Enregistre l'utilisateur
	POST	/auth/verify	Vérifie l'adresse mail d'un utilisateur
	POST	/auth/forgot	Permet la demande de réinitialisation de mot de passe
	POST	/auth/reset	Réinitialise le mot de passe de l'utilisateur
	GET	/auth/token	Renvoie un token à l'utilisateur
Notifications	GET	/notifications	Get all notifications
	GET	/notifications/:id	Get a notification by id
	POST	/notifications	Create a notification
	PATCH	/notifications/:id	Update a notification by id
	DELETE	/notifications/:id	Delete a notification (Mark as read instead of permanent deletion)
Profile	GET	/profile	Get profile route
	GET	/profile/bankaccounts	Get a user bank accounts
	GET	/profile/notifications	Get a user notifications
	GET	/profile/projects	Get all user projects
	GET	/profile/tags	Get my tags
	GET	/profile/teams	Get all user teams
	PATCH	/profile	Update profile route
	PATCH	/profile/password	Change password route
	PATCH	/profile/resume/:resumeld	Update resume reference
Projects	GET	/projects	Get all projects
	GET	/projects/:uuid	Get a project by uuid
	GET	/projects/:uuid/resources	Get all project resources
	GET	/projects/:uuid/teams	Get all project teams
	POST	/projects	Create a project (MANDATES)
	PATCH	/projects/:uuid	Update a project by uuid
	PATCH	/projects/:uuid/status/:status	Update a project status
	PUT	/projects/:uuid/arbitrate	Arbitrate a project (ARBITRATES)
	PUT	/projects/:uuid/apply	Join a project (APPLIES)
	PUT	/projects/:uuid/develop	Leave a project (DEVELOPS)

Recommendations	PUT	/projects/:uuid/feed-back	Note and feedback a project (MANDATES mark + feedback)
	DELETE	/projects/:uuid	Delete a project
Resources	GET	/recommendations/projects	Get a set of projects recommendations (combination of applies, mandates and tags)
	GET	/recommendations/projects/applies	Get projects recommendations based on other teams applies
	GET	/recommendations/projects/mandates	Get projects recommendations based on mandates
	GET	/recommendations/projects/tags	Get projects recommendations by tags similarities
Teams	GET	/resources	Get all resources
	GET	/resources/:id	Get a resource by id
	POST	/resources	Create a resource
	PATCH	/resources/:id	Update a resource by id
	DELETE	/resources/:id	Delete a resource
Teams	GET	/teams	Get all teams
	GET	/teams/:uuid	Get a team by uuid
	GET	/teams/:uuid/users	Get all team members
	GET	/teams/:uuid/projects	Get all team projects
	POST	/teams	Create a team
	PATCH	/teams/:uuid	Update a team by uuid
	PATCH	/teams/:uuid/status/:status	Update a team status
	PATCH	/teams/:uuid/users/:uuid	Accept a user by uuid (update IS_MEMBER_OF status)
	PUT	/teams/:uuid/join	Join a team (IS_MEMBER_OF)
	PUT	/teams/:uuid/leave	Leave a team (IS_MEMBER_OF)
Users	DELETE	/teams/:uuid	Delete a team
	GET	/users	Get all users
	GET	/users/:uuid	Get a user by id
	GET	/users/:uuid/bankaccounts	Get all user bank accounts
	GET	/users/:uuid/notifications	Get all user notifications
	GET	/users/:uuid/projects	Get all user projects
	GET	/users/:uuid/resources	Get all user resources
	GET	/users/:uuid/teams	Get all user teams
	PATCH	/users/:uuid	Update a user by id
	PATCH	/users/:uuid/ban	Ban a user by id
Users	PATCH	/users/:uuid/roles/:role	Assign a role to user
	DELETE	/users/:uuid	Delete a user

10.2.2 Suivi du projet

Pour les suivis des différentes tâches des sprints, j'ai utilisé un tableau *Trello* et l'ai mis à jour tout au long du projet. Celui-ci est disponible via l'adresse suivante : <https://trello.com/b/meyHR8e8/heig-vd-travail-de-bachelor>.

10.2.3 Démonstration fonctionnelle

Une démonstration fonctionnelle de l'état actuel du projet peut être consultée via l'URL suivant : <https://heig-tb-moonfish.netlify.app>.

10.3 Réalisation de requêtes avec le langage Cypher

10.3.1 Introduction

Cypher³⁶ est le langage de requêtes développé et utilisé pour les bases de données *Neo4j*, comme comparaison Cypher correspond au langage *SQL* pour les bases de données *MySQL*. Le langage Cypher est orienté graphe et se veut simple et efficace dans la formulation et la syntaxe des différentes requêtes qu'il permet (interrogation, mise à jour, etc.). Aujourd'hui, il est considéré comme un langage efficace et tire principalement sa force du fait de sa conception et de sa syntaxe simple qui se veut d'une grande aide pour ses utilisateurs.

Le principe de Cypher est qu'il définit un ensemble de fonctions, notamment les fonctions *MATCH* et *WHERE*. Ces deux fonctions principales, ressemblantes aux *SELECT* et *WHERE SQL*, permettent de décrire le modèle de recherche tout en y ajoutant certaines contraintes. Cypher définit différentes autres fonctions telles que *ORDER BY*, *LIMIT*, *WITH*, etc. permettant, en les combinant, de créer efficacement différentes requêtes d'interrogations.

³⁶ <https://neo4j.com/developer/cypher/>

10.3.2 Réalisation de requêtes pas-à-pas

Comme le langage *Cypher* est assez nouveau pour moi, je vais construire mes requêtes en y ajoutant des éléments au fur et à mesure. Commençons très simplement par retrouver un utilisateur via le modèle *User* avec son *uuid* comme filtre. En retournant l'utilisateur, j'obtiens le nœud de celui-ci.

```
// Trouver un utilisateur via son uuid
MATCH (u:User {uuid: '1e93dbe8-501c-4b84-9b65-7e7c1fceb6f4'})  
RETURN u
```

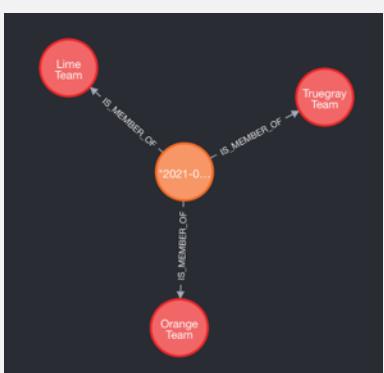
Depuis cet utilisateur, nous pouvons retrouver les équipes dont il fait partie via la relation *IS_MEMBER_OF*. En retournant ces trois éléments (utilisateur, relation, équipe), j'obtiens un graphe représentant visuellement ma requête.

```
// Trouver tous les groupes dont fait partie un utilisateur
MATCH (u:User {uuid: '1e93dbe8-501c-4b84-9b65-7e7c1fceb6f4'})  
-[r:IS_MEMBER_OF]->(t:Team)  
RETURN u, r, t
```



Tout comme j'ai ajouté un filtre sur le modèle *User*, je peux également ajouter un filtre sur la relation *IS_MEMBER_OF*. Celui-ci me permettant de conserver uniquement les équipes actives.

```
// Trouver tous les groupes d'un utilisateur avec une relation active
MATCH (u:User {uuid: '1e93dbe8-501c-4b84-9b65-7e7c1fceb6f4'})  
-[r:IS_MEMBER_OF {status: 2}]->(t:Team)  
RETURN u, r, t
```



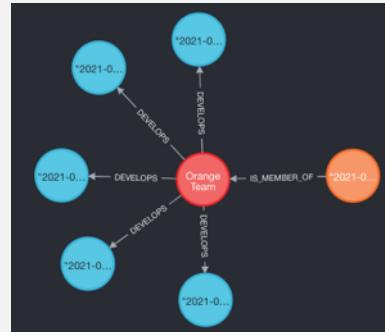
Depuis ces équipes, je peux maintenant relier les projets développés par celles-ci via la relation *DEVELOPS*. En ajoutant à mes valeurs de retour la nouvelle relation et les nœuds projets, j'obtiens un nouveau graphe. Noter que pour cette requête, l'*uuid* demandé a changé et le nouvel utilisateur fait à présent partie d'un seul groupe.

```
// Trouver tous les projets développés par un utilisateur
```

```

MATCH (u:User {uuid: '34c5b2f6-28c8-4f52-b7d5-dc188a9d053b'})-[:rmo:IS_MEMBER_OF {status: 2}]->(t:Team)-[:rd:DEVELOPS]->(p:Project)
RETURN u, rmo, t, rd, p

```

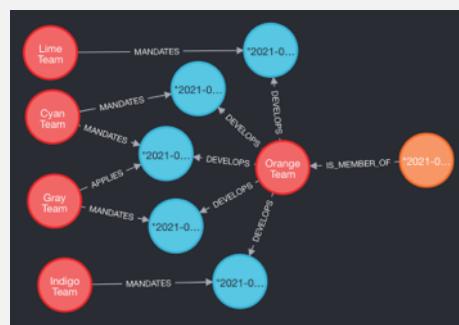


De ces projets, je peux repartir et demander quelle équipe les a mandatés. J'obtiens alors plusieurs nœuds équipes ayant des relations différentes avec les projets. Syntaxiquement, cette dernière relation `MANDATES`, a une flèche pointant de droite à gauche vers les projets trouvés. Cela est logique et se confirme visuellement sur le graphique. En gardant alors uniquement les nœuds « `m` », j'obtiens toutes les équipes mandantes des projets de l'utilisateur en cours.

```

// Trouver tous les mandants des projets développés par un utilisateur
MATCH (u:User {uuid: '34c5b2f6-28c8-4f52-b7d5-dc188a9d053b'})-[:rmo:IS_MEMBER_OF {status: 2}]->(t:Team)-[:rd:DEVELOPS]->(p:Project)-[:rm:MANDATES]-(m:Team)
RETURN u, rmo, t, rd, p, rm, m

```



De là, je suis prêt à construire mes trois requêtes de recommandations.

10.4 Historique du développement web

L'origine du web remonte jusqu'en 1989, année où l'informaticien Timothy John Berners-Lee travaillant au CERN publia un document intitulé « *Information Management : A Proposal*³⁷ ». À cette époque, Sir Berners-Lee cherchait une solution pour faciliter le partage d'informations entre ingénieurs et il la trouva en combinant internet (à cette époque de nombreux ordinateurs étaient déjà interconnectés) avec une autre technologie émergente : *Hypertext*. Au courant de l'année 1990, Sir Berners-Lee décrira trois des technologies fondamentales du web encore utilisées aujourd'hui, il s'agit de :

1. Le *HyperText Markup Language*, abrégé HTML et qui est le langage de balisage et de formatage brut conçu pour représenter des pages web.
2. Le *Uniform Resource Identifier* qui définit une courte chaîne de caractères identifiant une ressource sur un réseau et dont la syntaxe est normalisée.
3. L'*Hypertext Transfer Protocol*, abrégé HTTP qui est le protocole de communication client-serveur permettant la récupération de ressources du web.

10.4.1 Le HTML

La première page web d'internet ³⁸est mise en ligne en décembre 1990. Celle-ci est plutôt brute et contient uniquement de l'information et des liens de manière schématisée et ce de par le fait que le seul langage disponible à cette époque est le HTML.

³⁷ <http://info.cern.ch/Proposal.html>

³⁸ <https://www.w3.org/History/19921103-hypertext/hypertext/WWW/TheProject.html>

World Wide Web

The WorldWideWeb (W3) is a wide-area [hypermedia](#) information retrieval initiative aiming to give universal access to a large universe of documents.

Everything there is online about W3 is linked directly or indirectly to this document, including an [executive summary](#) of the project, [Mailing lists](#), [Policy](#), November's [W3 news](#), [Frequently Asked Questions](#).

[What's out there?](#)

Pointers to the world's online information, [subjects](#), [W3 servers](#), etc.

[Help](#)

on the browser you are using

[Software Products](#)

A list of W3 project components and their current state. (e.g. [Line Mode](#), [X11](#), [Viola](#), [NeXTStep](#), [Servers](#), [Tools](#), [Mail robot](#), [Library](#))

[Technical](#)

Details of protocols, formats, program internals etc

[Bibliography](#)

Paper documentation on W3 and references.

[People](#)

A list of some people involved in the project.

[History](#)

A summary of the history of the project.

[How can I help ?](#)

If you would like to support the web..

[Getting code](#)

Getting the code by [anonymous FTP](#), etc.

Figure 54 : Première page d'internet publiée en décembre 1990,
<https://www.w3.org/History/19921103-hypertext/hypertext/WWW/TheProject.html>

10.4.2 Le CSS

Il faudra attendre six ans pour que le CSS voit le jour. L'ajout majeur de ces feuilles de style en cascade est de séparer le contenu de la mise en forme d'un site web. Le CSS permet de modifier le rendu brut d'un document HTML et ainsi d'améliorer l'aspect visuel des données présentées par ce document. Le « *CSS level 1* » puis ses versions successives ouvriront la voie de l'intégration et de l'évolution des mises en forme des pages web que l'on connaît aujourd'hui.

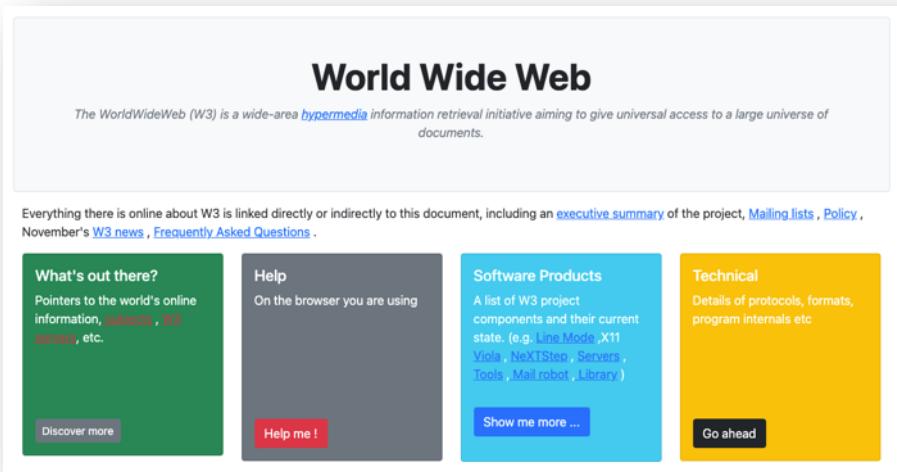


Figure 55 : Exemple de ce à quoi pourrait ressembler la première page HTML en lui appliquant quelques styles CSS

10.4.3 Le JS

Dès 1995, Brendan Eich pense et implémente le langage JavaScript. Cependant il faudra attendre le concept de programmation AJAX, l'objet `XMLHttpRequest`³⁹ et ses requêtes asynchrones pour que celui-ci se démocratise. L'utilisation des requêtes AJAX, notamment sur le site *Gmail* (l'un des tout premiers sites web dynamiques), fut

³⁹ <https://developer.mozilla.org/fr/docs/Web/API/XMLHttpRequest>

particulièrement appréciée par les utilisateurs. Ceux-ci trouvaient ces sites plus fluide, plus dynamique et donc plus agréable à utiliser.

Dès lors, les 3 piliers du web qui sont *HTML*, *CSS* et *JS* étaient lancés. Ils vont alors évoluer jusqu'à nos jours où ils forment toujours la structure, le style et les interactions de nos sites internet. Aujourd'hui, les sites et applications web utilisent, pour la plupart, des *frameworks* pour être conçus et maintenus plus facilement. Nous reviendrons plus en détail sur les *frameworks* dans les points suivants.

10.4.4 Le PHP

Les pages web en *HTML / CSS / JS* sont dites « statiques », ce qui signifie que son contenu est fixe, qu'il ne peut pas varier et qu'il est le même pour tous les utilisateurs. Mais très vite, dès 1993, le besoin de pouvoir interagir avec l'utilisateur ainsi que de pouvoir générer des pages spécifiques et « à la demande » apparaît. Plus tard, on qualifiera ces pages web de « dynamique », car leur contenu peut quant à lui varier en fonction de différentes informations telles que l'heure actuelle, le nom de l'utilisateur, la position géographique, un formulaire spécifique rempli par l'utilisateur, etc.

Pour traiter ces interactions et ces informations, il a fallu inventer un langage de programmation serveur. C'est ainsi qu'en 1994, le programmeur canadien *Rasmus Lerdorf* a créé la première version du PHP pour « *Personal Home Page* ». Monsieur Lerdorf voulait conserver les traces des visiteurs qui venaient consulter son CV publié sur sa page internet personnelle. Pour ce faire, il a enrichi une bibliothèque logicielle en langage C puis l'a publié sous-licence libre en 1995.

10.4.5 MySQL

Finalement, il a fallu trouver un moyen de sauvegarder facilement des données afin de pouvoir les exploiter rapidement. Il serait tout à fait possible et cela a été fait dans un premier temps, de conserver des données sous forme de fichier de textes brut sur le serveur. Cependant cette pratique n'est pas viable, car elle devient très vite inefficace lorsque la taille des données à traiter augmente. C'est pourquoi la programmation de sites web a été conçue et adaptée à l'utilisation de structures de gestion de données : les SGBDR.

Au début des années 1995, le finlandais *Michael Widenius* crée le logiciel de gestion de bases de données relationnelles le plus répandu dans le monde encore actuellement : MySQL. MySQL, comme tout système de base de données relationnelles, organise les données en plusieurs tables de données dans lesquelles les types de données sont clairement définis et peuvent être liés les uns aux autres. Dès lors, les sites web peuvent alors tirer parti du langage SQL utilisé pour créer, modifier et extraire des données de base de données relationnelles. Ils peuvent alors par exemple contrôler l'accès des utilisateurs et les droits qui leur sont concédés, stocker des informations spécifiques sur tel ou tel utilisateur, etc.

En 2009, Michael Widenius créer une dérive de MySQL pour continuer son développement *Open Source*, sous le nom de MariaDB.

10.5 Architectures logicielles

L'architecture logicielle permet de décrire au travers de modèles, de schémas et d'une manière symbolique, les différents éléments que composent un système informatique ainsi que ses interactions. Il décrit comment un système informatique doit être conçu de manière à répondre aux spécifications.

10.5.1 Front-end / Back-end / Full Stack

Les concepts de « *Front-end* », « *Back-end* », et « *Full Stack* » sont parfois mélangés et/ou difficiles à saisir dans une application web. Chacun de ces termes ne se réfère pas à un langage précis, mais plutôt à un groupe de langages utilisés conjointement dans le but de réaliser une partie de l'application. De ce fait, la partie *front-end* peut être résumé par « *ce que l'utilisateur voit* » sur son écran et « *ce avec quoi il interagit directement* ». La partie *back-end*, quant à elle, correspond alors à « *tout ce qui se passe en arrière-plan* » pour que l'utilisateur puisse interagir correctement avec une application. Ce concept est illustré par l'image suivante, l'utilisateur depuis son bateau voit la partie émergée de l'iceberg. Celui-ci pour exister et flotter se compose non seulement de sa partie émergée (*front-end*), mais également de sa partie immergée (*back-end*).

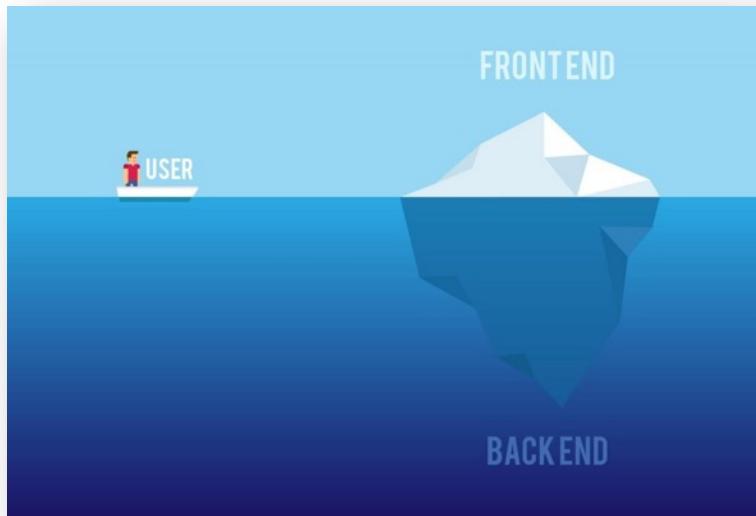


Figure 56 : Différences entre Back-end / Front-end et Full Stack, 11 février 2021,
<https://www.leproductowner.com/fiches-metiers/backend-frontend-fullstack>

En se plaçant du point de vue de l'utilisateur, le *front-end* désigne donc tout ce que celui-ci voit au premier plan, par exemple des boutons, des liens, des formulaires, des images, des vidéos, etc. Pour afficher, styliser et gérer ces éléments, il faut des langages frontaux comme le HTML, le CSS et le JS ; ceux-ci sont exécutés directement sur la machine du client. Pour faire fonctionner tous ces éléments et interpréter toutes les actions de l'utilisateur, le *back-end* entre en jeu et s'occupe alors d'administrer la soumission d'un formulaire, de traiter des données, de vérifier des jetons de sécurité, etc. Les langages de programmation permettant de faire de telles actions sont par exemple PHP, Java, Python ou même JavaScript ; ils sont exécutés sur le serveur où l'application est hébergée.

Finalement, le terme *full-stack* désigne l'ensemble du *front-end* et du *back-end*. Ainsi, il se réfère généralement aux développeurs maîtrisant les deux aspects essentiels au bon fonctionnement d'une application et les différents langages s'y rattachant.

10.5.2 MVC : Modèle – vue – contrôleur

Le motif d'architecture logicielle « *MVC* » pour « *Modèle – Vue – Contrôleur* » a été lancé en 1978. Il était alors principalement destiné aux interfaces graphiques des applications web. Dans ce modèle, les données de l'application, l'interface utilisateur et la logique métier sont divisées en trois composants distincts ayant chacune des responsabilités différentes.

1. *Le modèle* : il contient les données à afficher.

Les modèles représentent la structure des données, leur définition ainsi que les fonctions qui leur sont propres (validation, lecture et enregistrement). Ils sont complètement décorrélés du code métier et de l'affichage, de ce fait la modification de la logique et/ou de l'interface n'affecte en rien la structure de ces *modèles* de données.

2. *La vue* : elle contient la présentation de l'interface graphique.

Les vues représentent les parties visibles de l'interface graphique à présenter au client qui fait une requête. Elles se servent des données provenant des *modèles* pour afficher des éléments visuels comme des diagrammes, des formulaires, des boutons, etc. À nouveau, l'isolement du code de l'interface avec la logique métier et avec les données permet de faire des modifications sur celles-ci sans avoir à se soucier de la structure des données ou du fonctionnement de la logique.

3. *Le contrôleur*, il contient la logique concernant les actions effectuées par l'utilisateur.

Les contrôleurs sont au cœur de la logique métier de l'application puisqu'ils se situent entre les *vues* et les *modèles*. Les requêtes faites par un client depuis l'interface graphique vont être dirigées vers un *contrôleur*. Celui-ci sera

chargé de manipuler les données en interrogeant les *modèles*, de les traiter par rapport au besoin, et d'informer les *vues* de répondre au client avec de nouveaux éléments.

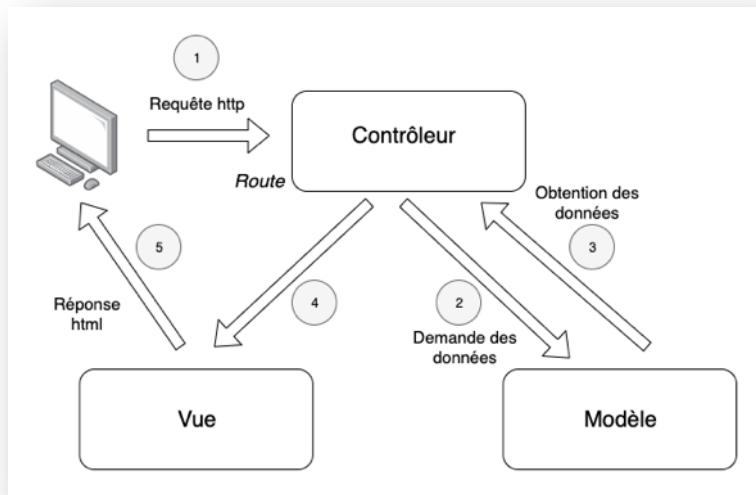


Figure 57 : Représentation des interactions entre le modèle, la vue et le contrôleur dans le cas d'une application web,
[https://commons.wikimedia.org/wiki/File:Mod%C3%A8le-vue-contr%C3%B4leur_\(MVC\) - fr.png?uselang=fr](https://commons.wikimedia.org/wiki/File:Mod%C3%A8le-vue-contr%C3%B4leur_(MVC) - fr.png?uselang=fr)

Le flux de traitement imposé par le modèle MVC est représenté par le schéma précédent. Il s'opère de telle façon à ce que 1) une requête envoyée depuis la *vue* est analysée par le *contrôleur* 2) le *contrôleur* demande au *modèle* approprié d'effectuer des traitements 3) le *contrôleur* obtient des données en retour 4) le *contrôleur* notifie la *vue* que la requête est traitée 5) la *vue* notifiée affiche le résultat du traitement.

Le cycle « *demande => mise à jour => affichage* » mis en place par ce modèle correspond très bien aux applications web, son principal avantage étant que chacun des composants peut être modifié indépendamment, ce qui améliore la maintenabilité de l'application. La majorité des frameworks web actuels se basent, utilisent et implémentent ce modèle d'architecture, mais nous y reviendront dans le chapitre suivant.

10.5.3 MVVM : Modèle – vue – vue modèle

Le modèle d'architecture « *MVVM* » pour « *Modèle – Vue – Vue modèle* » est apparu en 2004 et a été créé par Microsoft pour son framework .NET. Comme pour le modèle MVC, cette méthode permet de séparer la vue de la logique et de l'accès aux données. Cependant, la différence se trouve au niveau du *ViewModel* qui contrairement au *contrôleur* de l'architecture MVC sert de lien bidirectionnel entre l'interface. Cette méthode est appelée « *data binding* ».

Dans cette architecture, les modèles et les vues sont également divisés et peuvent être modifiés séparément. Le modèle *MVVM* se compose de trois parties distinctes :

1. Le *modèle* ; logique de travail avec les données et description des données fondamentales requises pour que l'application fonctionne.
2. La *vue* ; l'interface graphique (fenêtres, listes, boutons, etc.)
3. Le *viewModel* ; abstraction de la vue et conteneur pour les données du modèle. Il contient un modèle converti en vue ainsi que les commandes que la vue peut utiliser pour influencer le modèle.

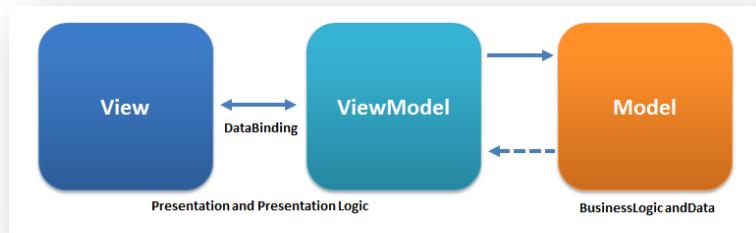


Figure 58 : Représentation de l'architecture MVVM et des interactions entre la vue, la vueModèle et le modèle,
<https://en.wikipedia.org/wiki/File:MVVMPattern.png>

Dans cette architecture, les requêtes réalisées par l'utilisateur sur l'interface vont aller modifier une ou plusieurs données présentes dans le *vueModèle*. Cette action peut provoquer un appel au code métier dans le *modèle*, qui va à son tour renvoyer une nouvelle donnée au *vueModèle*. La *vue* ne sera pas changée, mais s'adaptera simplement pour afficher la ou les nouvelles données qui lui seront passées dynamiquement par le *vueModèle*.

De ce fait, on comprend bien que les architectures *MVC* et *MVVM* sont fortement semblables. La principale différence de l'architecture *MVVM* réside donc dans le fait que les actions de l'utilisateur entraînent des modifications des données du modèle et cette communication est dite bidirectionnelle entre la *vue* et le *modèle*.

10.5.4 Architecture trois tiers

L'architecture « *trois niveaux* », « *trois couches* » ou « *trois couleurs* » se réfère toujours à l'application d'un modèle d'architecture plus général ; le « *multi-tiers* » divisé en trois niveaux distincts. Cette architecture, basée sur l'environnement client – serveur vise à modéliser une application comme un empilement de trois couches logicielles dont le rôle est clairement défini. Elle se compose donc des trois couches suivantes :

1. La couche de *présentation* des données ; correspond à l'affichage, à la restitution sur l'écran et au dialogue avec l'utilisateur
2. La couche de *traitement métier* des données ; correspond à la mise en œuvre de l'ensemble des règles, de la gestion et de la logique applicative
3. La couche d'*accès* aux données persistantes ; correspond aux données qui sont destinées à être conservées sur une certaine durée et/ou de manière définitive.

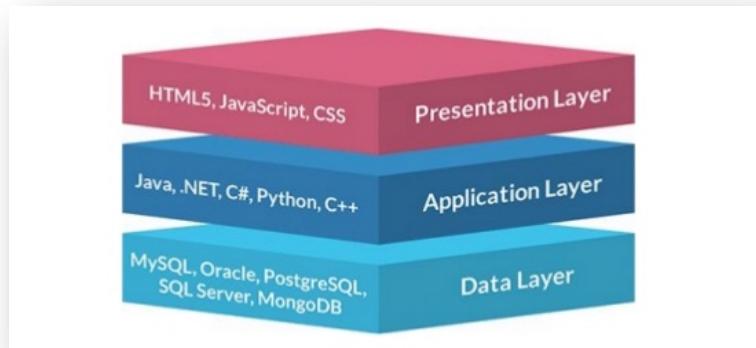


Figure 59 : Représentation de l'architecture 3-tier et des langages utilisés dans chacune des couches,
<https://www.slideshare.net/TharinduWeerasinghe/multitier-designs-in-software>

Dans cette approche, les différentes couches communiquent entre elles au travers d'un « *modèle d'échange* » et chacune d'entre elles met à disposition un ensemble de services pour les autres couches. Ces services sont mis à disposition des couches adjacentes et il est par conséquent interdit d'invoquer les services d'une couche plus basse que la couche immédiatement inférieure ou plus haute que la couche immédiatement supérieure. Il s'agit là de la principale différence avec les modèles *MVC* et *MVVM* dans lesquels les *modèles* pouvaient, d'une certaine façon,

directement communiquer avec les *vues*. L'architecture trois tiers impose donc de toujours repasser par cette couche applicative intermédiaire et son flux de contrôle traverse le système de haut en bas (les couches supérieures sont toujours source d'interactions alors que les couches inférieures ne font que répondre à des requêtes).

10.6 Frameworks

Depuis de nombreuses années, d'innombrables frameworks ont vu le jour pour créer toutes sortes d'applications web. Leur but initial est de simplifier le processus de développement, d'augmenter la flexibilité et de réduire les délais de mise sur le marché. Sans l'utilisation de frameworks, le développement web moderne serait un véritable cauchemard pour les ingénieurs logiciels. En effet, ils devraient alors tout recréer à partir de zéro et ce à chaque nouveau projet (logique métier, options de sécurité, gestion de navigation, etc.).

C'est pourquoi, en 2021, la quasi-totalité des développeurs utilisent des frameworks comme surcouche de langage pour créer et mener à bien leur projet.

10.6.1 Qu'est-ce qu'un framework ?

Le framework c'est la *boîte à outils* du développeur. Il met à disposition du développeur un ensemble de modules de programmation, d'outils et de bibliothèques prêts à l'emploi lui permettant de construire son application. Ils permettent également de mettre un cadre, un squelette et de dicter les règles de construction des architectures des applications, des API, des interfaces, etc.

En plus du fait que les frameworks simplifient la création et le maintien de projets web, ils ont de nombreux avantages. On peut notamment relever les avantages suivants tant au niveau économique que technique :

- *Le développement est accéléré*

Les frameworks évitent aux programmeurs de devoir réinventer la roue en effectuant des tâches basiques depuis zéro lors du démarrage d'un projet. La possibilité d'utiliser des modèles et des outils pré-écrits pour créer rapidement la base d'un projet permet d'économiser un temps considérable. De ce fait, les développeurs peuvent mieux se concentrer sur les détails spécifiques du projet et ainsi mieux garantir sa qualité finale.

- *Le gain de fiabilité et de sécurité*

Les composants prêts à l'emploi et mis à disposition par les frameworks, ont été créés et améliorés par une communauté de milliers de développeurs. Ils ont donc été testés et éprouvés dans de nombreux scénarios possibles. En les utilisant, les développeurs évitent de nombreux bugs et s'assurent de créer une solution stable, fiable et sécurisée dans un délai plus court.

- *Le respect des meilleures pratiques*

Les méthodologies des frameworks intègrent généralement les meilleures pratiques d'ingénierie logicielle reconnue actuellement. En suivant les règles proposées par les frameworks, les développeurs évitent de nombreux obstacles de conception et cela permet d'éliminer des bugs en amont.

- *La simplification de la maintenance et des développements futurs*

Les frameworks définissent une structure unifiée pour le développement, de sorte que les applications basées sur ceux-ci soient plus faciles à maintenir et à améliorer. N'importe quel développeur peut facilement comprendre un projet développé avec un framework qu'il maîtrise sans connaître en détail le projet. Il lui est alors facile d'ajouter des fonctionnalités ou apporter des modifications de manière transparente.

- *Un gain de performance*

Les projets basés sur des frameworks ont tendance à fonctionner beaucoup plus rapidement et à assurer une montée en charge plus élevée. Ce qui est crucial pour des solutions informatiques modernes qui se doivent d'être polyvalente et extensible.

10.6.2 Séparation des préoccupations

Comme décrit précédemment, la quasi-totalité des applications web modernes créées aujourd'hui peuvent se décomposer en deux parties distinctes.

1. La première, du côté client, qu'on appelle le « *front-end* ». Cette partie peut être résumée par « *ce que l'utilisateur voit* ».
2. La seconde, du côté serveur, qu'on appelle le « *back-end* » et qui peut être résumé par « *ce qui se passe en arrière-plan (under the hood)* »

De ce fait, il existe des frameworks *front-end* et des frameworks *back-end*, permettant ainsi de réaliser chaque partie en tirant partie des avantages détaillés ci-dessus.

Les frameworks ***back-end*** sont responsables de la partie dite cachée d'un site web ou d'une application s'exécutant directement sur le serveur. Pour y accéder, la plupart des applications utilisent des interfaces de programmations (API) mises à disposition par les frameworks *back-end*. Ils s'occupent du fonctionnement du serveur et d'accès à la base de données, de la logique métier et de l'architecture, des protocoles de routage, de la sécurité des données, des options d'autorisation et des droits d'accès, etc. Les frameworks *back-end* peuvent être basés sur différents langages de programmation tels que *PHP*, *.NET*, *Ruby*, *Python*, *Java*, *JavaScript*, etc.

Les frameworks ***front-end*** vont permettre aux développeurs de réaliser l'interface utilisateur d'une application ou d'un site web. Notamment de gérer les multiples interactions que l'application mettra à disposition des utilisateurs finaux d'un point de vue de l'affichage. Mais également de la conception *UX* et *UI*, des modèles, de l'optimisation, du référencement, etc. Ils sont basés sur des langages de balisage et de programmation dits frontaux tels que le *HTML*, le *CSS* et le *JavaScript*.

10.6.3 Frameworks *back-end*

Aujourd'hui, il existe de nombreux langages de programmation *back-end* et tout autant, voire plus, de frameworks *back-end*. Il m'est donc impossible de tous les lister, c'est pourquoi j'ai choisi un framework par langage de programmation parmi les plus utilisés du marché en 2021 pour établir une vue d'ensemble.

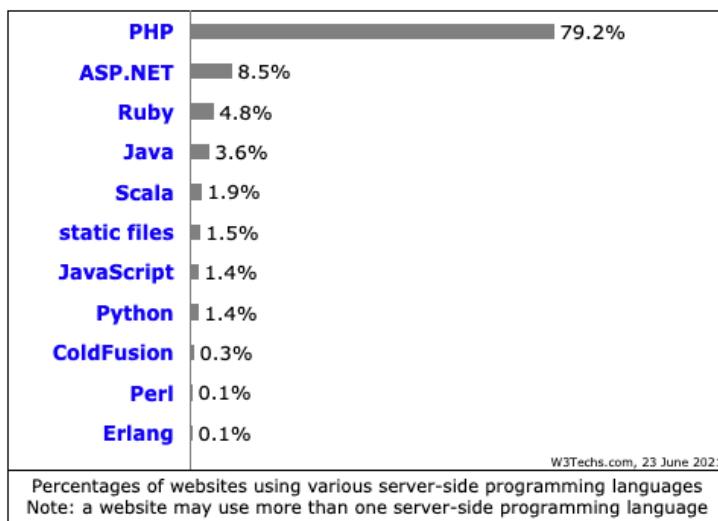


Figure 60 : Pourcentages de répartition des langages de programmation côté serveur utilisés par les sites web en juin 2021,
https://w3techs.com/technologies/overview/programming_language

Selon les données de W³Techs⁴⁰, *PHP* est encore aujourd'hui le langage côté serveur le plus utilisé et de loin avec presque 80% de part de marché ! Ensuite *ASP.NET* occupe la seconde place avec 8.5% du marché. Les 12.3% restants sont majoritairement occupés par *Ruby* (4.8%) et *Java* (3.6%) puis par *JavaScript* et *Python* tous deux à 1.4%. Il est intéressant de noter que 1.5% des sites web actuels sont encore réalisés avec des fichiers statiques et donc sans utiliser de langage de programmation du côté serveur. La grande part de marché qu'occupe *PHP* est principalement dû au nombreux CMS se basant sur cette technologie et notamment à *WordPress*⁴¹ qui, toujours selon les données de W³Techs, occupe aujourd'hui presque un tiers du web (65% en juin 2021).

⁴⁰ <https://w3techs.com/>

⁴¹ <https://wordpress.com/fr/>

PHP



<https://en.wikipedia.org/wiki/File:Laravel.svg>

- 2011 (10 ans)
- *Taylor Otwell*
- PHP
- Licence MIT
- laravel.com

.NET Languages



https://blog.soat.fr/wp-content/uploads/2015/11/asp.net_.jpg

- 2002 (19 ans)
- Microsoft
- .NET Languages
- Apache License 2.0
- dotnet.microsoft.com

Ruby



https://en.wikipedia.org/wiki/File:Ruby_On_Rails_Logo.svg

- 2004 (17 ans)
- *Community*
- Ruby
- Licence MIT
- rubyonrails.org

Java



https://en.wikipedia.org/wiki/File:Spring_Framework_Logo_2018.svg

- 2002 (19 ans)
- *Pivotal Software*
- Java
- Apache License 2.0
- spring.io

Scala



https://en.wikipedia.org/wiki/File:Play_Framework_logo.svg

- 2007 (14 ans)
- *Lightbend & Zengularity*
- Scala
- Apache License 2.0
- playframework.com

Python



https://en.wikipedia.org/wiki/File:Django_Logo.svg

- 2005 (16 ans)
- *Django Software Foundation*
- Python
- 3-clause BSD
- djangoproject.com

10.6.4 Frameworks *front-end*

Le langage JavaScript est aujourd’hui l’unique langage disponible pour réaliser la partie *front-end* d’une application. De ce fait, d’innombrables frameworks existent chacun étant plus ou moins adapté et conçu pour tel ou tel type d’applications et fournissant plus ou moins de fonctionnalités. Parmi eux, certains prennent le dessus et sont alors majoritairement utilisés par les développeurs. Ces différents frameworks *front-end* ont été décrit dans le chapitre consacré au JavaScript.

10.7 Solutions « *stack* »

Une « *pile de solutions* », plus connue sous sa terminologie anglaise « *solution stack* » est un ensemble de sous-systèmes et/ou de composants logiciels nécessaires pour créer une plate-forme informatique complète. Les applications créées s’exécutent alors « sur » la plate-forme résultante et aucun logiciel supplémentaire n’est nécessaire pour les prendre en charge.

Certains des composants disponibles sont si souvent choisis ensemble par les développeurs qu’ils en deviennent des « *stack* » connus et reconnus et possède alors un nom. En règle générale, le nom donné est un acronyme représentant les composants individuels. Une des solutions de *stack* la plus célèbre est sous doute LAMP (*Linux*) et ses pendant MAMP (*macOS*) et WAMP (*Windows*).