

## 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**



**Étudiant :**

**Thibaud Alt**

**Travail proposé par :**

Guillaume Wägli  
Boulevard de Pérolles 20  
1700 Fribourg

**Enseignant responsable :**

Patrick Lachaize

**Année académique :**

2020-2021

Département TIC

Filière Informatique

Orientation Systèmes de gestion

Étudiant Thibaud, Alt

Enseignant responsable Patrick, Lachaize

### 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

Cliquez ou appuyez ici pour entrer du texte.

Étudiant :	Date et lieu :	Signature :
Alt Thibaud	.....	.....
Enseignant responsable :	Date et lieu :	Signature :
Lachaize Patrick	.....	.....
Nom de l'entreprise/institution :	Date et lieu :	Signature :
Guillaume Wägli	.....	.....

## 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

TODO

# Table des matières

<b>1</b>	<i>Introduction</i>	<b>9</b>
1.1	But du document	9
1.2	Problématique	9
1.3	Objectifs	9
1.4	Valeur ajoutée	10
<b>2</b>	<i>Cahier des charges</i>	<b>12</b>
2.1	Éléments généraux	12
2.2	Éléments d'études	13
2.3	Besoins fonctionnels	14
2.4	Besoins non fonctionnels	15
2.5	Extensions	15
<b>3</b>	<i>Étude de marché</i>	<b>17</b>
3.1	Cibles	17
3.2	Risques	17
<b>4</b>	<i>Analyse</i>	<b>20</b>
4.1	User stories	20
4.2	Graphisme et ergonomie	24
4.3	Réception des livrables, validation et paiement	31
4.4	Livraison et intégration continue	32
<b>5</b>	<i>JavaScript</i>	<b>33</b>
5.1	Applications web « <i>State-of-the-Art</i> »	33
5.2	Le JavaScript c'est quoi ?	33
5.3	Environnements d'exécutions	33
5.4	Frameworks <i>front-end</i>	34
5.5	Frameworks <i>back-end</i>	40
5.6	Tests technologiques	45
<b>6</b>	<i>Système de gestion de base de données</i>	<b>60</b>
6.1	Base de données relationnelle	60
6.2	Base de données orientée documents	63
6.3	Base de données orientée graphe	63
<b>7</b>	<i>Modélisation</i>	<b>68</b>
7.1	Diagrammes de classes	68

7.2	Diagrammes de cas d'utilisation.....	68
7.3	Diagrammes de séquence .....	68
<b>8</b>	<b>Planification.....</b>	<b>69</b>
8.1	Sprint N°1 : Développement de l'API.....	69
8.2	Sprint N°2 : Développement des interfaces utilisateur .....	69
8.3	Sprint N°3 : Intégration des interfaces et de l'API .....	69
8.4	Sprint N°4 : Mise en place des relations et recommandations .....	70
8.5	Sprint N°5 : Finalisation du projet .....	70
<b>9</b>	<b>Réalisation.....</b>	<b>71</b>
9.1	Tests .....	71
<b>10</b>	<b>Conclusion .....</b>	<b>76</b>
<b>11</b>	<b>Annexes.....</b>	<b>80</b>
11.1	Historique du développement web .....	80
11.2	Architectures logicielles .....	83
11.3	Frameworks .....	86
11.4	Solutions « stack » .....	89

# Glossaire

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

<b>PME</b>	Petite ou moyenne entreprise
<b>Repository</b>	Anglicisme de dépôt ou référentiel représentant un stockage centralisé et organisé de données
<b>Responsive</b>	Technique de conception de site web visant à offrir une consultation confortable sur toutes les tailles d'écrans
<b>SCRUM</b>	Cadre de développement de produits logiciels
<b>SGBDR</b>	Système de gestion de bases de données relationnelles
<b>SQL</b>	<i>Structured Query Language</i> , langage informatique normalisé servant à exploiter des bases de données relationnelles
<b>UI</b>	<i>User interface</i> , dispositif matériel ou logiciel permettant à un usager d'interagir avec un produit informatique
<b>UUID</b>	<i>Universally unique identifier</i> , système permettant d'identifier de façon unique une information
<b>UX</b>	<i>User experience</i> , qualité du vécu d'un utilisateur dans un environnement numérique ou physique
<b>XML</b>	<i>Extensible Markup Language</i> , langage de balisage extensible
<b>Wireframe</b>	Schéma de conception d'une interface graphique définissant les zones et les composants que celle-ci doit englober

# 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 directeur, 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. 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, *tout comme au 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 d'ainsi 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 mandants 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 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.
- 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, l'application web on attend au minimum 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'uniquelement 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 composé 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 nécessaire, 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 plus facile à mettre en place dans un premier temps, cependant l'utilisation éventuelle de microservice ne doit pas être et devra être prise en compte lors de la phase d'analyse et de conception.

#### 2.2.2.3 Gestion de dépendances

La gestion de dépendances sera réalisée avec *npm* qui est le gestionnaire de paquets officiel de Node.js si ce dernier est choisi lors de la phase d'analyse pour y développer la plateforme. Ce gestionnaire de paquet 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](https://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 ses 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	<b>Création d'un compte et authentification</b> 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	<b>Gestion de profil</b> 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	<b>Soumission de projets</b> 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	<b>Évaluation d'une équipe de développeurs</b> 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	<b>Soumission de candidature</b> 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	<b>Dépôt des livrables</b> 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	<b>Classement des équipes et des développeurs</b> 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	<b>Espace de travail</b> 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	<b>Discussions instantanées</b> En tant qu'utilisateur de la plateforme, je peux accéder via mon espace de travail à un système de discussions instantanées.
Epic 10	<b>Gestion des projets et des utilisateurs</b> 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	<b>Gestion de l'espace de connaissances</b>

	<p>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.</p>
Epic 12	<p><b>Intervention externe</b></p> <p>En tant qu'intervenant externe je dois, lorsqu'on me sollicite, pouvoir intervenir et apporter mon expertise sur un projet ou sur une situation.</p>

## 2.4 Besoins non fonctionnels

### 2.4.1 Contraintes dues à l'environnement

Ce projet étant nouveau et non lié à 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 n'est pas toujours facile de se souvenir quelle combinaison 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 bureau classique, il serait fort agréable pour l'utilisateur d'également disposer d'une interface sur ces 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. apportant des notions théoriques ainsi que différents savoirs nécessaires liés à la gestion de projets. 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 vulgariser, 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 ÉTUDE DE MARCHÉ

Le but de ce chapitre n'est pas de faire une étude de marché complète mais plutôt de faire un tour d'horizon des produits similaires existants, des possibilités, des risques et des opportunités.

### 3.1 Définission du marché

#### Caractéristiques générales du marché, taille, potentiel, segmentation

La zone géographique visée est, dans un premier temps, les pays européens et plus particulièrement la Suisse. Plus tard, il sera intéressant de réaliser une étude sur d'autres pays avec des fonctionnement managériaux plus différents comme l'Asie par exemple.

### 3.2 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 devraient 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 du personnel pour un 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.

### 3.3 L'offre

#### 3.3.1 Concurrence directe

*Trouver d'autres produits similaires et analyser leurs stratégies et leurs fonctionnalités.*

##### 3.3.1.1 *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. »*

*Site web :* [\*www.codeur.com\*](http://www.codeur.com)

##### 3.3.1.2 *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. »*

*Site web :* [\*www.freelance.com\*](http://www.freelance.com)

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

#### 3.3.2 Concurrence indirecte

Nous retrouvons comme type de concurrence indirect les entreprises spécialisées dans la réalisation de projets informatiques et qui offriraient des services similaires à la plateforme.

## 3.4 L'environnement

L'analyse de l'environnement et de son influence sur le marché peut être réalisé selon la méthode *PESTEL* qui reprend les facteurs macro-environnementaux pouvant influencer positivement ou négativement un projet.

### 3.4.1 Environnement politique

stabilité des gouvernements et des politiques, contexte politique, tendances fiscales... Quelle est la stabilité politique ? Existe-t-il des tensions particulières ? Quel est le régime en place ? Quelle est la politique en matière de fiscalité, de commerce, etc. ?

### 3.4.2 Environnement économique

cycle économique, taux de croissance, pouvoir d'achat, taux d'intérêt, monnaie, inflation, chômage... Quelle est la conjoncture économique actuelle ? Quel est le taux de chômage ? Quelle est le revenu disponible ? Quelle est son évolution ?

### 3.4.3 Environnement socio-culturel

démographie, composition socio-culturelle de la population et tendances, mobilité sociale, modes de consommation, éducation, travail, loisirs... Quelle est la culture ? Quelles sont les valeurs et les normes ? Quel est le niveau d'éducation ? Comment évolue la démographie ? Quelles sont les habitudes de consommation ?

### 3.4.4 Environnement technologique

politique publique de R&D, tendances d'innovation, dépenses privées de R&D... Quelles sont les évolutions technologiques à venir ? Sont-elles fréquentes ? Quels secteurs sont-ils concernés ?

### 3.4.5 Facteurs environnementaux

lois sur l'écologie et l'énergie, Quelle est la sensibilité aux enjeux du développement durable ? Quelles sont les mesures prises en faveur de l'environnement ? Quel traitement est réservé aux déchets ?

### 3.4.6 Environnement légal

lois, droit du travail, réglementation et normes de sécurité... Quelle est la législation qui encadre votre activité ? Comment peut-elle évoluer ? Quel est le rôle des pouvoirs publics ? Quel est le rôle des groupes d'influence et des organisations professionnelles ? Etc.

## 3.5 *Business Model Canvas*

Après avoir définis 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.

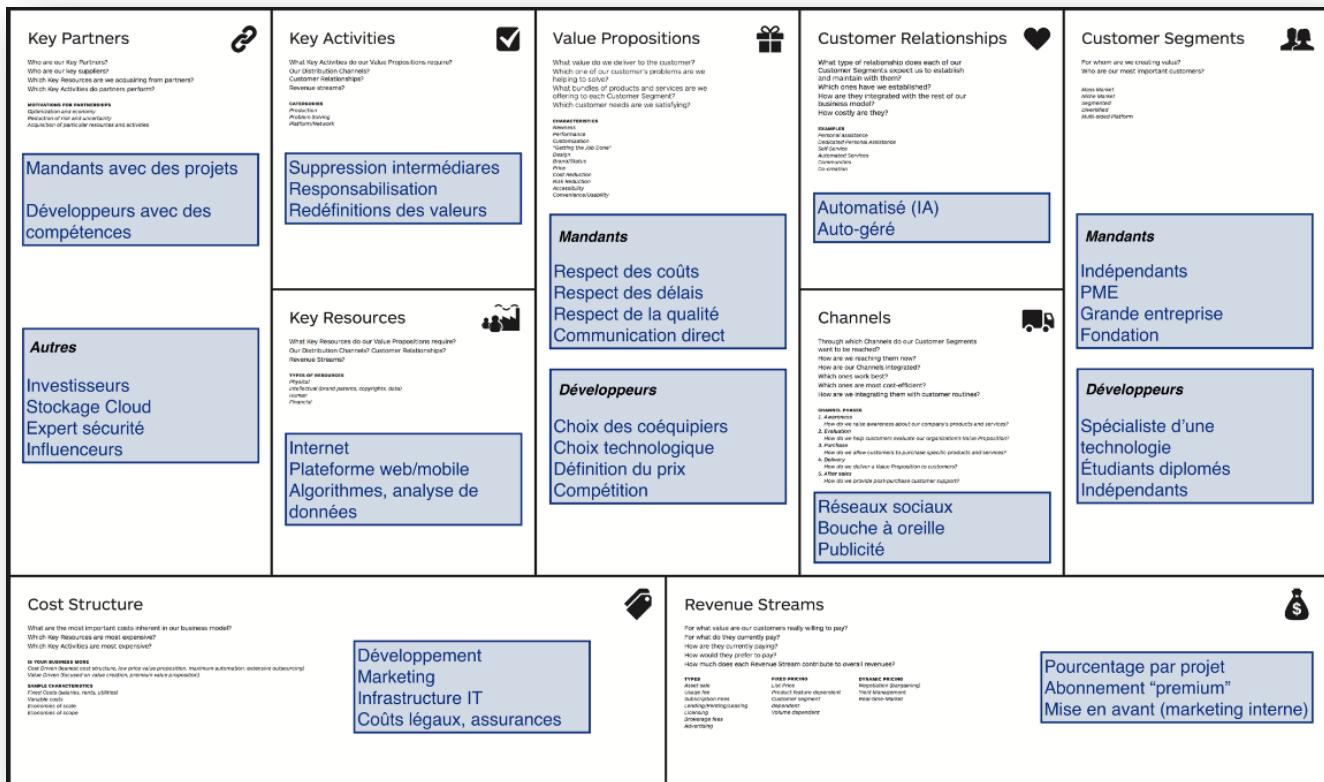


Figure 1 : « Business Model Canvas » de la plateforme à réaliser

## 4 ANALYSE

### 4.1 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.

#### 4.1.1 Crédation d'un compte et authentification

<i>Identifiant</i>	Epic 1
<i>Titre</i>	Création d'un compte et authentification
<i>Description</i>	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.
<i>Tâches associées</i>	<ol style="list-style-type: none"> <li>1. L'utilisateur doit pouvoir se créer un compte avec au minimum un nom, une adresse email et un mot de passe.</li> <li>2. L'utilisateur doit pouvoir se connecter et accéder à son compte.</li> <li>3. L'utilisateur doit pouvoir récupérer son mot de passe en cas d'oublis de celui-ci.</li> </ol>
<i>Priorisation</i>	3 / 3
<i>Complexité</i>	3 / 5
<i>Auteur</i>	Thibaud Alt

#### 4.1.2 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"> <li>1. L'utilisateur doit pouvoir ajouter/modifier/supprimer ses données de contacts (nom, prénom, adresse, etc.).</li> <li>2. L'utilisateur doit pouvoir modifier ses informations de connexions (email et mot de passe).</li> <li>3. L'utilisateur doit pouvoir ajouter/modifier/supprimer ses informations professionnelles (CV, langages connus, etc.).</li> <li>4. L'utilisateur doit pouvoir rejoindre et quitter une équipe de développeurs.</li> <li>5. L'utilisateur doit pouvoir créer une équipe de développeurs.</li> <li>6. L'utilisateur doit pouvoir gérer les informations de contacts d'une équipe de développeurs (adresse mail, informations bancaires, etc.).</li> <li>7. L'utilisateur doit pouvoir gérer les membres de son équipe de développeurs.</li> <li>8. L'utilisateur doit pouvoir transmettre la gestion et la responsabilité d'une équipe de développeurs à un utilisateur tiers.</li> </ol>
<i>Priorisation</i>	3 / 3

<i>Complexité</i>	2 / 5
<i>Auteur</i>	Thibaud Alt

#### 4.1.3 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"> <li>1. Le mandant doit pouvoir publier un projet à réaliser.</li> <li>2. Le mandant doit pouvoir modifier les informations liées à un nouveau projet (listes de fonctionnalités, délais).</li> <li>3. Le mandant doit pouvoir choisir une équipe de développeurs pour réaliser le projet.</li> <li>4. Le mandant doit pouvoir consulter les profils des équipes de développeurs.</li> <li>5. Le mandant doit pouvoir consulter les profils des développeurs.</li> </ol>
<i>Priorisation</i>	2 / 3
<i>Complexité</i>	2 / 5
<i>Auteur</i>	Thibaud Alt

#### 4.1.4 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"> <li>1. Le mandant doit pouvoir attribuer une note à une équipe de développeurs.</li> <li>2. Le mandant doit pouvoir écrire un rapport (feed-back) sur le déroulement d'un projet.</li> <li>3. Les notes et les rapports obtenus par une équipe de développeur doivent s'afficher sur le profil de celle-ci.</li> </ol>
<i>Priorisation</i>	2 / 3
<i>Complexité</i>	3 / 5
<i>Auteur</i>	Thibaud Alt

#### 4.1.5 Soumission de candidature

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

Complexité	2 / 5
Auteur	Thibaud Alt

#### 4.1.6 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"> <li>1. Une équipe de développeurs (via son responsable) doit pouvoir téléverser des documents électroniques de différents formats (« .pdf », « .docx », « .xlsx », etc.).</li> <li>2. Une équipe de développeurs (via son responsable) doit pouvoir téléverser des archives « .zip » de livrables du projet.</li> <li>3. Le mandant doit être notifié lorsqu'un téléchargement est effectué.</li> <li>4. Le mandant doit pouvoir consulter tous les documents et livrables liés au projet.</li> </ol>
Priorisation	3 / 3
Complexité	4 / 5
Auteur	Thibaud Alt

#### 4.1.7 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"> <li>1. Un visiteur doit pouvoir consulter une page de classement présentant les équipes de développeurs et comportant des options de tris et de filtres.</li> <li>2. Un visiteur doit pouvoir consulter le classement d'un développeur via sa page de profil.</li> </ol>
Priorisation	1 / 3
Complexité	4 / 5
Auteur	Thibaud Alt

#### 4.1.8 Espace de travail

Identifiant	Epic 8
Titre	Espace de travail
Description	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.
Tâches associées	<ol style="list-style-type: none"> <li>1. Un acteur du projet doit pouvoir ajouter différents types de fichiers (texte, images, etc.).</li> <li>2. Un acteur du projet doit pouvoir consulter toutes les ressources liées à un projet.</li> <li>3. Un acteur du projet doit pouvoir transmettre une information à tous les participants d'un projet.</li> <li>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.</li> </ol>

5. Le mandant doit pouvoir télécharger en tout temps les différents livrables.

*Priorisation* 2 / 3

*Complexité* 3 / 5

*Auteur* Thibaud Alt

#### 4.1.9 Discussions instantanées

*Identifiant* Epic 9

*Titre* Discussions instantanées

*Description* En tant qu'utilisateur de la plateforme, je peux accéder via mon espace de travail à un système de discussions instantanées.

*Tâches associées*

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.

*Priorisation* 1 / 3

*Complexité* 4 / 5

*Auteur* Thibaud Alt

#### 4.1.10 Gestion des projets et des utilisateurs

*Identifiant* Epic 10

*Titre* Gestion des projets et des utilisateurs

*Description* 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.

*Tâches associées*

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

*Priorisation* 1 / 3

*Complexité* 3 / 5

*Auteur* Thibaud Alt

#### 4.1.11 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*

1. En tant que modérateur de l'espace de connaissances, j'ai aux ressources publiques de tous les projets. Je peux les filtrer, les trier et les catégoriser.

	<ol style="list-style-type: none"> <li>2. En tant que modérateur de l'espace de connaissances, je peux ajouter une nouvelle ressource et la publier.</li> <li>3. En tant que modérateur de l'espace de connaissances, je peux modifier ou supprimer une ressource que j'ai publiée.</li> </ol>
Priorisation	1 / 3
Complexité	2 / 5
Auteur	Thibaud Alt

#### 4.1.12 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"> <li>1. Tout utilisateur doit pouvoir solliciter l'aider d'un intervenant externe</li> <li>2. Un intervenant externe doit pouvoir discuter, sous forme textuelle, avec un membre du projet.</li> <li>3. Un intervenant externe doit pouvoir discuter, sous forme textuelle, avec un groupe de membres du projet.</li> <li>4. Un intervenant externe doit pouvoir accéder aux ressources du projet</li> <li>5. Un intervenant externe doit pouvoir ajouter des ressources au projet (rapport d'expertise, PV de séance, etc.)</li> <li>6. Un intervenant externe doit pouvoir notifier un modérateur en cas de conflit qui sort de son domaine de compétences</li> </ol>
Priorisation	1 / 3
Complexité	2 / 5
Auteur	Thibaud Alt

## 4.2 Graphisme et ergonomie

### 4.2.1 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 de design « *flat patterns* » et minimalistes de la conception web qui sont les suivantes :

- 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 simples
- 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 souvent utilisées en aplats. Des icônes dessinées sous forme synthétique. Ainsi que des polices de caractères souvent originales et de taille importante dans les titrages.

Les sources d'inspirations pourraient être notamment le système d'exploitation *macOS*<sup>1</sup> à partir de sa version 10.10 et son dérivé *iOS*<sup>2</sup> depuis la version 7. L'interface graphique *d'Android* à partir de la version 5 et son « *Material Design*<sup>3</sup> ».

#### 4.2.1.1 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*<sup>4</sup> 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.

#### 4.2.1.2 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

<sup>1</sup> <https://developer.apple.com/design/human-interface-guidelines/macos/overview/themes/>

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

<sup>3</sup> <https://material.io/design/guidelines-overview>

<sup>4</sup> <https://www.pantone.com/eu/fr/>

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 semble 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](https://unsplash.com) regorge d'images de tout type et pourra être utilisé pour trouver des images pertinentes à intégrer.

#### 4.2.1.3 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.

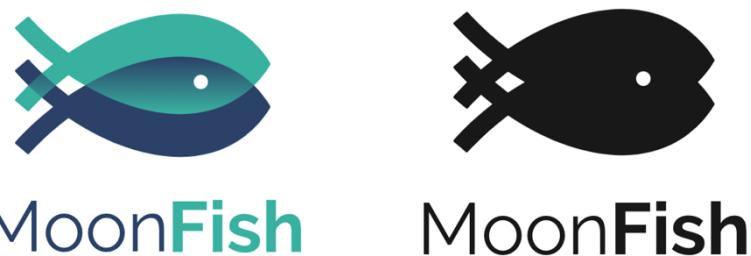


Figure 4 : Logotype et nom de la plateforme dans les couleurs définis et sa nuance en noir et blanc

Pour le nom de la plateforme, j'ai choisi d'utiliser le terme « *MoonFish* ». Celui-ci étant la traduction littérale du « poisson lune » français mais n'existant pas en anglais, j'ai trouvé intéressant de l'utiliser tel quel. Un poisson lune à 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.

#### 4.2.2 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.io](https://draw.io)<sup>5</sup>, en passant même par des applications dédiées telles que *Sketch*<sup>6</sup> ou *Adobe Photoshop*<sup>7</sup>. 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.

#### 4.2.3 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*<sup>8</sup> 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 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*<sup>9</sup>.

<sup>5</sup> <https://app.diagrams.net>

<sup>6</sup> <https://www.sketch.com>

<sup>7</sup> [https://www.adobe.com/ch\\_fr/products/photoshop.html](https://www.adobe.com/ch_fr/products/photoshop.html)

<sup>8</sup> <https://getbootstrap.com>

<sup>9</sup> <https://tailwindcss.com>

#### 4.2.3.1 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](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.

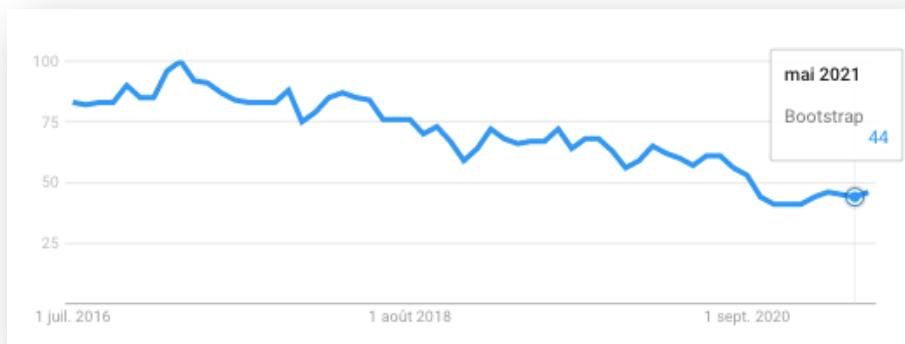


Figure 6 : Évolution du terme de recherche "Bootstrap" sur les 5 dernières années,  
<https://trends.google.fr/trends/explore?date=today%205-y&q=Bootstrap>

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*<sup>10</sup> 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<sup>e</sup> version sont : les nouveaux

<sup>10</sup> <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.

#### 4.2.3.2 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.

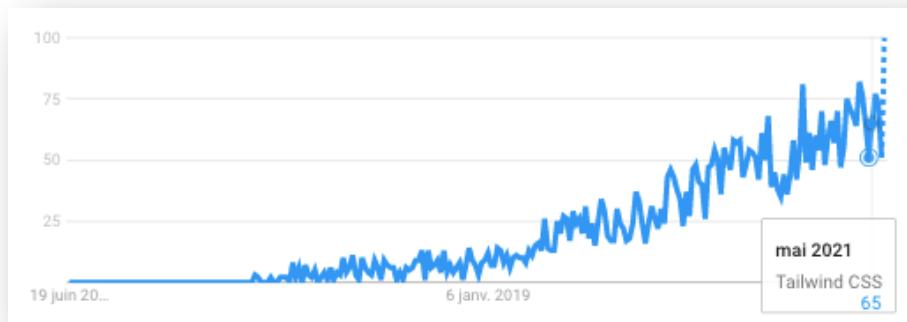


Figure 8 : Évolution du terme de recherche "Tailwind CSS" sur les 5 dernières années,  
<https://trends.google.fr/trends/explore?date=today%205-y&q=Tailwind%20CSS>

*Tailwind CSS*, via son arrivée récente, tire parti de toutes les fonctionnalités CSS moderne notamment par la prise en charge de la grille CSS, des dégradés composables, des sélecteurs d'état modernes, etc. De par son récent engouement 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 dédiée<sup>11</sup>, et le développement de celui-ci est prometteur.

<sup>11</sup> <https://github.com/tailwindlabs/tailwindcss>

#### 4.2.3.3 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* principalement utilisé aujourd'hui grâce à sa facilité d'utilisation, sa documentation et de nombreux « 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 relève surtout de 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.

#### 4.2.3.4 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.

#### 4.2.4 Maquettes

##### 4.2.4.1 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 des 3 parties distinctes présentées ci-dessous.

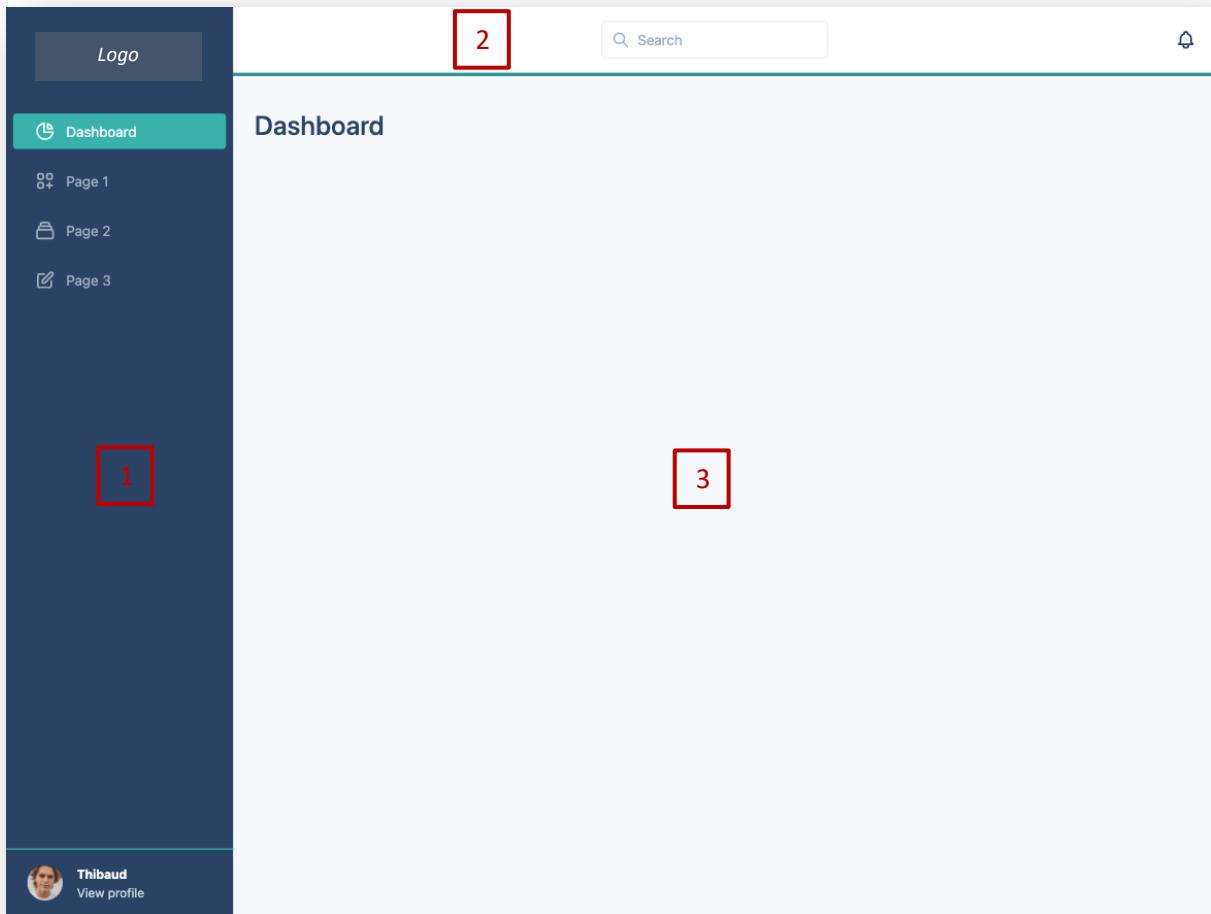


Figure 9 : Squelette de l'application

##### 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 des sous-menus de la section visitée. C'est également dans cette barre de navigation haute, sur la droite, qu'un résumé des notifications 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.

### 4.2.4.2 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.

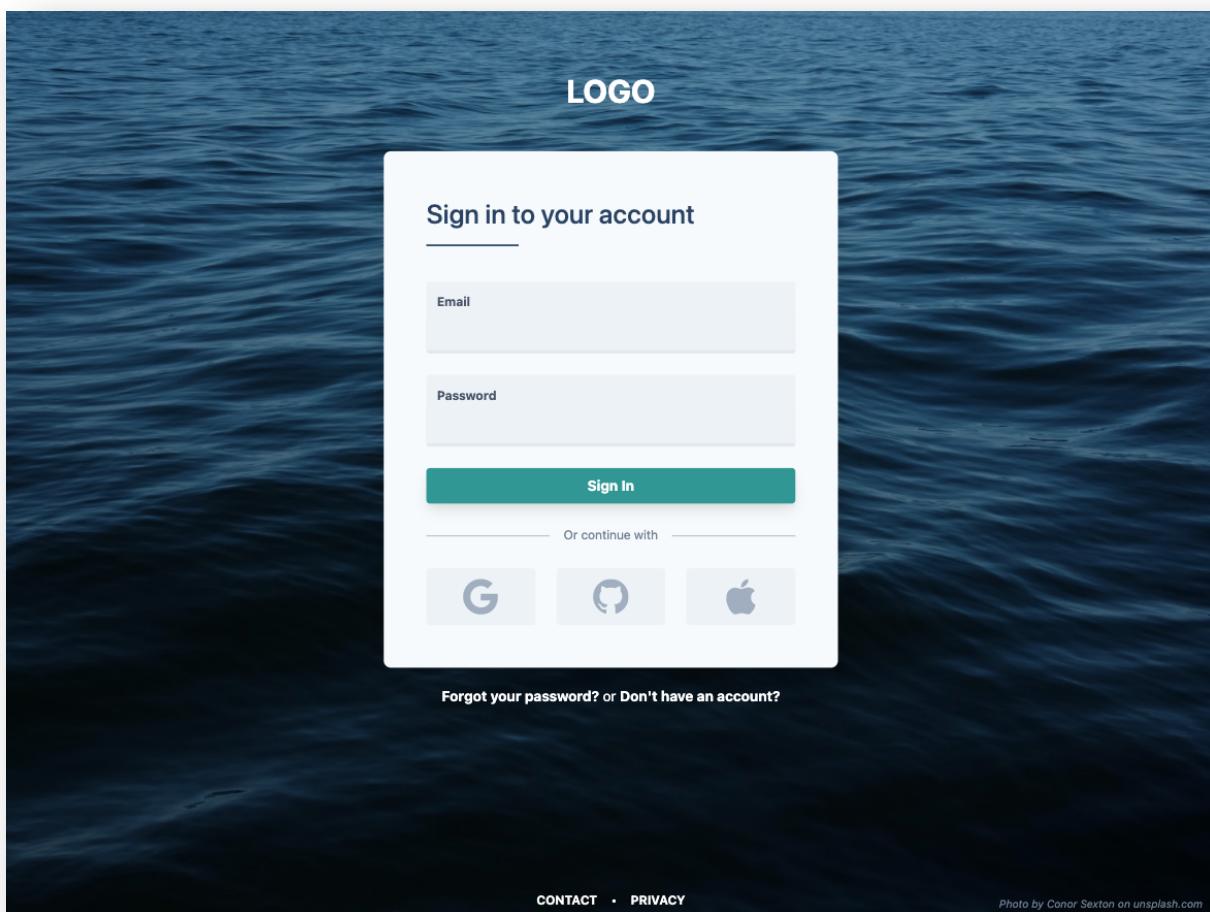


Figure 10 : Page d'entrée et d'identification

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.

## 4.3 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.

#### 4.3.1 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.

### 4.4 Livraison et intégration continue

#### 4.4.1 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*<sup>12</sup>. *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.

##### 4.4.1.1 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.netlify.app/>. Dorénavant, mon projet et ses évolutions seront donc visibles via cette URL.

---

<sup>12</sup> <https://www.netlify.com/>

## 5 JAVASCRIPT

### 5.1 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 que 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 développerais mon application.

### 5.2 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*<sup>13</sup>, 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 diffèrent considérablement dans leur conception.

#### 5.2.1 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 de mieux 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*<sup>14</sup>.



[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, la création de classes et d’interfaces, et l’import de modules. Tout cela en conservant l’approche non contraignante de JavaScript et en supportant la spécification *ECMAScript 6*.

### 5.3 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

<sup>13</sup> [https://developer.mozilla.org/fr/docs/Web/JavaScript/Language\\_Resources](https://developer.mozilla.org/fr/docs/Web/JavaScript/Language_Resources)

<sup>14</sup> <https://babeljs.io/>

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.

### 5.3.1 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](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*<sup>15</sup>, *Node.js* est utilisé sur 1,3% du web, ce qui signifie qu'il fait tourner au moins 20 millions de sites web.

#### 5.3.1.1 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.

### 5.3.2 Deno

*Deno* est également un environnement d'exécution, celui-ci 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 paquet.



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

#### 5.3.3 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.

## 5.4 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 principalement sécuritaires par l'ensemble des navigateurs web qui a commencé il y a plus de dix ans.

<sup>15</sup> <https://w3techs.com/technologies/details/ws-nodejs>

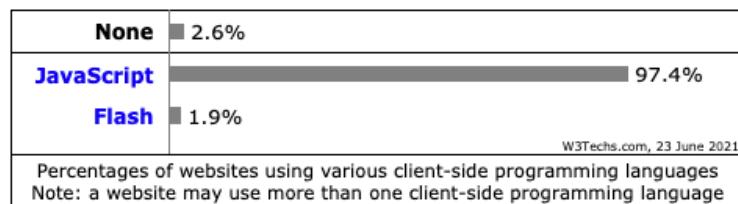


Figure 11 : Pourcentages de répartition des langages de programmation côté client utilisés par les sites web en juin 2021,  
[https://w3techs.com/technologies/overview/client\\_side\\_language](https://w3techs.com/technologies/overview/client_side_language)

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.

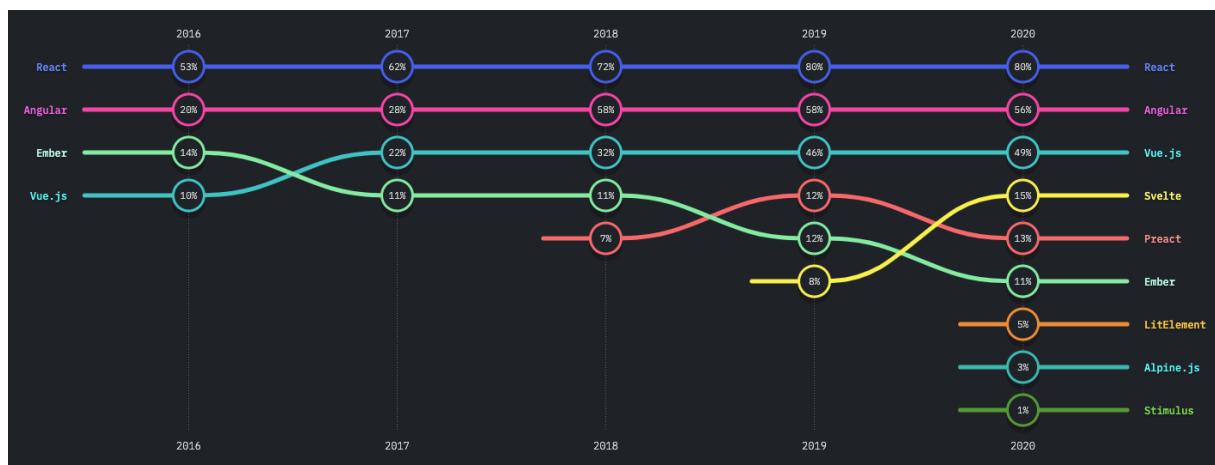


Figure 12 : Évolution d'utilisation des frameworks JavaScript front-end de 2016 à 2020,  
<https://2020.stateofjs.com/en-US/technologies/front-end-frameworks/>

#### 5.4.1 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](http://jquery.com)

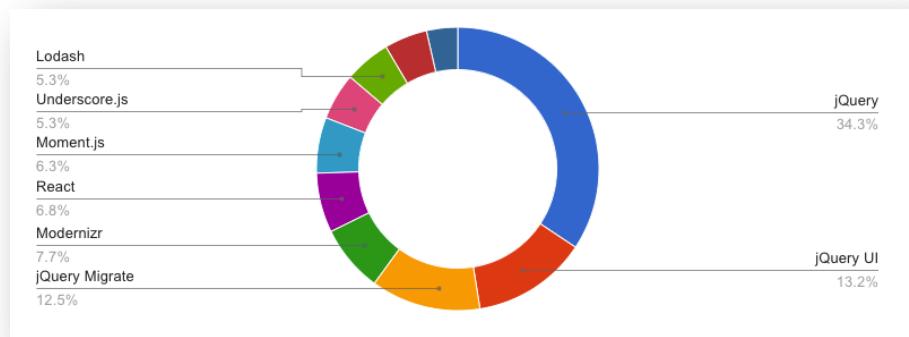


Figure 13 : Technologies de bibliothèques JavaScript les plus utilisées en se basant sur la part de marché en 2021,  
<https://www.wappalyzer.com/technologies/javascript-libraries/>

À l'époque de sa sortie, *jQuery* est très intéressant, car il offre de nouvelles possibilités aux programmateurs notamment en permettant de rendre des pages web jusqu'alors statiques, dynamiques 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.

#### 5.4.1.1 Avantages / Inconvénients

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

#### 5.4.1.2 Hello, World !

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <script crossorigin="anonymous"
        src="https://code.jquery.com/jquery-3.6.0.min.js"
        integrity="sha256-/xUj+3OJU5yExlq6GSYGSKh7tPXikynS7ogEvDej/m4="></script>
    <script>
        $(document).ready(function () {
            alert("Hello, World!");
        });
    </script>
</head>
<body>
</body>
</html>
```

## 5.4.2 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 dépendant 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.



[https://en.wikipedia.org/wiki/  
File:React-icon.svg](https://en.wikipedia.org/wiki/File:React-icon.svg)

- 2013 (8 ans)
- *Facebook & communauté*
- JavaScript
- Licence MIT
- [reactjs.org](http://reactjs.org)

### 5.4.2.1 Avantages / Inconvénients

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

### 5.4.2.2 Hello, World !

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

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

export default App;
```

## 5.4.3 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

#### 5.4.3.1 Avantages / Inconvénients

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

#### 5.4.3.2 Hello, World !

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

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

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

#### 5.4.4 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](http://vuejs.org)

#### 5.4.4.1 Avantages / Inconvénients

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

#### 5.4.4.2 Hello, World !

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

#### 5.4.5 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](https://svelte.dev)

#### 5.4.5.1 Avantages / Inconvénients

Avantages	Inconvénients
<ul style="list-style-type: none"> <li>- Moins de code, donc moins de bugs potentiels</li> <li>- Pas de DOM virtuel</li> <li>- Très réactif</li> <li>- Implémentation mobile possible</li> <li>- Courbe d'apprentissage minimale</li> <li>- Rapidité d'exécution</li> <li>- Peut être utilisé pour créer l'intégralité de l'application ou utilisé de manière incrémentale</li> <li>- Communauté amicale</li> </ul>	<ul style="list-style-type: none"> <li>- Faible part de marché</li> <li>- Pas de vérification de type</li> <li>- Confusion dans les noms des variables et de la syntaxe</li> <li>- Difficultés liées à un framework de type compilateur</li> <li>- Pas de soutien majeur</li> <li>- Petite communauté</li> </ul>

#### 5.4.5.2 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>
```

#### 5.4.6 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* ».



<https://preactjs.com/assets/app-icon.png>

- 2019 (2 ans)
- *Jason Miller*
- JavaScript
- Licence MIT
- [preactjs.com](https://preactjs.com)

*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.

#### 5.4.7 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*.



[https://en.wikipedia.org/wiki/File:Ember.js\\_Logo\\_and\\_Mascot.png](https://en.wikipedia.org/wiki/File:Ember.js_Logo_and_Mascot.png)

- 2011 (10 ans)
- *Ember Core Team*
- JavaScript
- Licence MIT
- [emberjs.com](https://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*.

### 5.5 Frameworks *back-end*

Bien que le JavaScript soit majoritairement utilisé pour réaliser la partie *front-end* des applications, il peut également être utilisée 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.

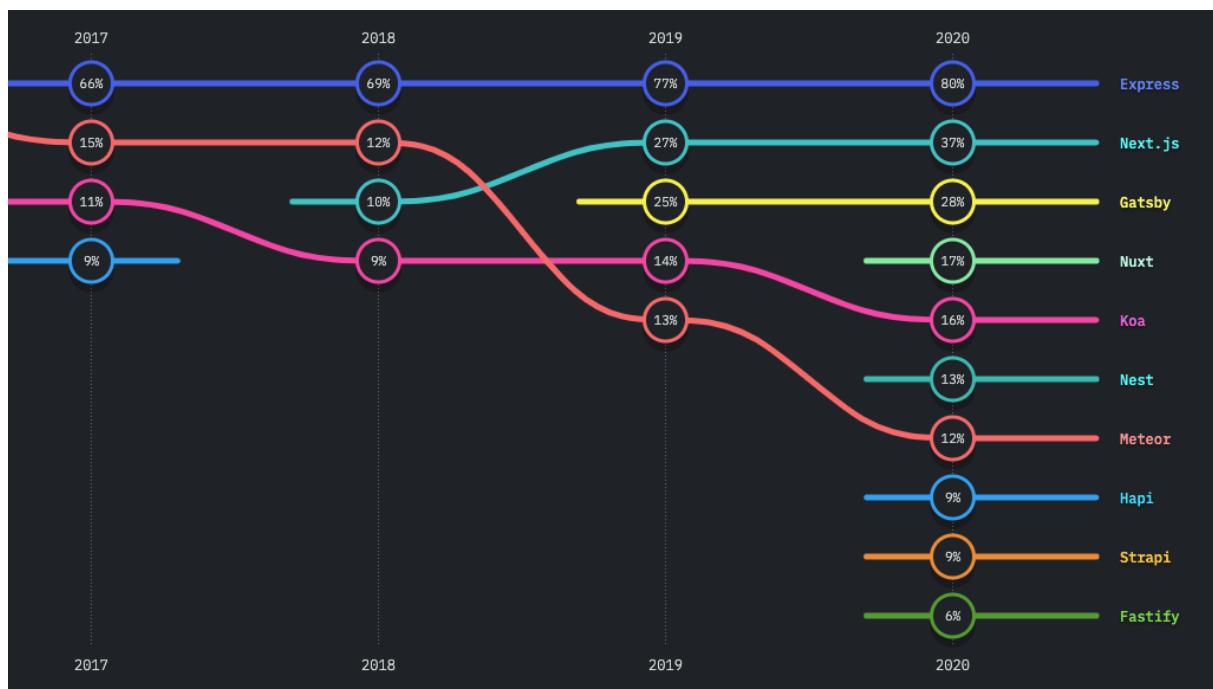


Figure 14 : Évolution d'utilisation des frameworks JavaScript back-end de 2017 à 2020,  
<https://2020.stateofjs.com/en-US/technologies/back-end-frameworks/>

### 5.5.1 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.

#### 5.5.1.1 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 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.

##### Inconvénients

- *La sécurité.* Il est de l'entièr 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 potentiellement difficile la compréhension et la maintenance du code.

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

- *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 simples, Express est assez facile à apprêhender pour les développeurs.

### 5.5.1.2 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')
});
```

### 5.5.2 Next.js

Basé sur *React.js*, *Next.js* a été créé par société néerlandaise *Vercel* en 2017. Les points forts de *Next* sont le rechargement 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, trouve avec *Next.js* de nouvelles fonctionnalités supplémentaires permettant d'être exécutées du côté serveur.

#### 5.5.2.1 Avantages / Inconvénients



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

- *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*.

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

### 5.5.2.2 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
```

### 5.5.3 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*, au contraire 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](http://gatsbyjs.com)

#### 5.5.3.1 Avantages / Inconvénients

##### Avantages

- *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.

##### Inconvénients

- *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.

### 5.5.3.2 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>
}
```

### 5.5.4 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.

#### 5.5.4.1 Avantages / Inconvénients



Avantages	Inconvénients
<ul style="list-style-type: none"> <li>- <i>La structure du projet.</i> Par défaut, le code est organisé par <i>Nuxt</i> dans une structure évolutive, logique et simple à comprendre.</li> <li>- <i>La communauté.</i> La communauté autour de <i>Nuxt.js</i> est grande et met à disposition une collection de plusieurs API, kits de démarrage, modules, bibliothèques, etc.</li> <li>- <i>Le fractionnement de code.</i> Une version statique de chaque page de l'application est générée par <i>Nuxt</i>. Par conséquent, le code JavaScript peut être divisé en plusieurs fichiers plus petits, améliorant ainsi la vitesse et les performances.</li> <li>- <i>L'installation et le développement rapides.</i> La majorité de la configuration initiale et de l'installation est prise en charge par <i>Nuxt.js</i>, ce qui permet de commencer à coder immédiatement.</li> </ul>	<ul style="list-style-type: none"> <li>- <i>L'intégration de bibliothèques.</i> L'intégration de bibliothèques et surtout de bibliothèques personnalisées avec <i>Nuxt</i> peut être longue et difficile.</li> <li>- <i>Les problèmes de débogages.</i> <i>Nuxt</i> génère de nombreux problèmes de débogage, ce qui peut s'avérer être assez frustrant.</li> </ul>

#### 5.5.4.2 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>
```

## 5.6 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, à entériné ma décision et à 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 leur framework *front-end* respectif, 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.

### 5.6.1 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'utiliserais 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.

### 5.6.2 Express.js

#### 5.6.2.1 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>)

#### 5.6.2.2 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 au 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();
```

#### 5.6.2.3 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
```

#### 5.6.2.4 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* <sup>16</sup> » et offre une prise en charge des transactions solide, la gestion des relations, un chargement rapide, etc.

<sup>16</sup> [https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global\\_Objects/Promise](https://developer.mozilla.org/fr/docs/Web/JavaScript/Reference/Global_Objects/Promise)

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.)

#### 5.6.2.5 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;
};
```

### 5.6.2.6 Interrogation avec Postman

Pour réaliser tous les appels à l'API facilement, j'utilise l'application cliente de *Postman*<sup>17</sup>. Celle-ci permet d'envoyer des requêtes, d'inspectez la réponse et de déboguer facilement. Par exemple, une fois la base de données initialisé et un utilisateur ajouté, nous pouvons obtenir la liste de tous les utilisateurs avec la requête suivante.

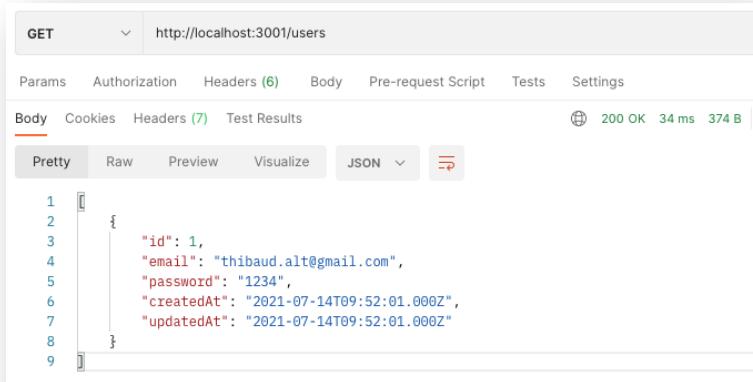


Figure 15 : Interrogation avec Postman du endpoint « /users »

### 5.6.3 React.js

#### 5.6.3.1 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>)

#### 5.6.3.2 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.

<sup>17</sup> <https://www.postman.com/product/api-client/>

### 5.6.3.3 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.

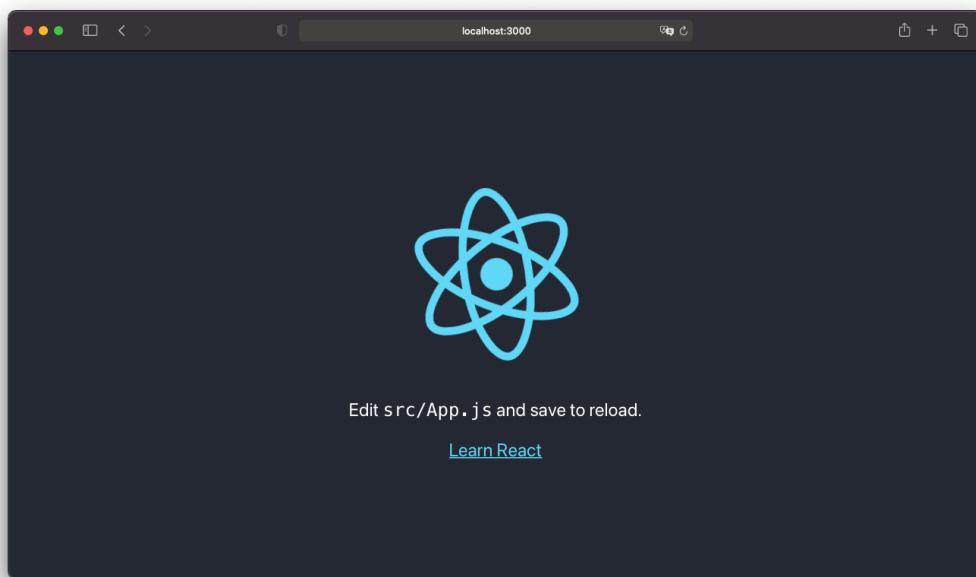


Figure 16 : Point de départ d'une application *React.js*

### 5.6.3.4 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 qui nous pourront par exemple accéder à la page de connexion en ajoutant « */login* » dans l'url.

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

### 5.6.3.5 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}>
  />
  ...
)
```

Un 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}>/</Header>
        <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/weevood/HEIG-VD\\_Travail-de-Bachelor/tree/main/4.Tests-Technos/React.js](https://github.com/weevood/HEIG-VD_Travail-de-Bachelor/tree/main/4.Tests-Technos/React.js).

## 5.6.4 Vue.js

### 5.6.4.1 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>)

### 5.6.4.2 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.

#### 5.6.4.3 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.

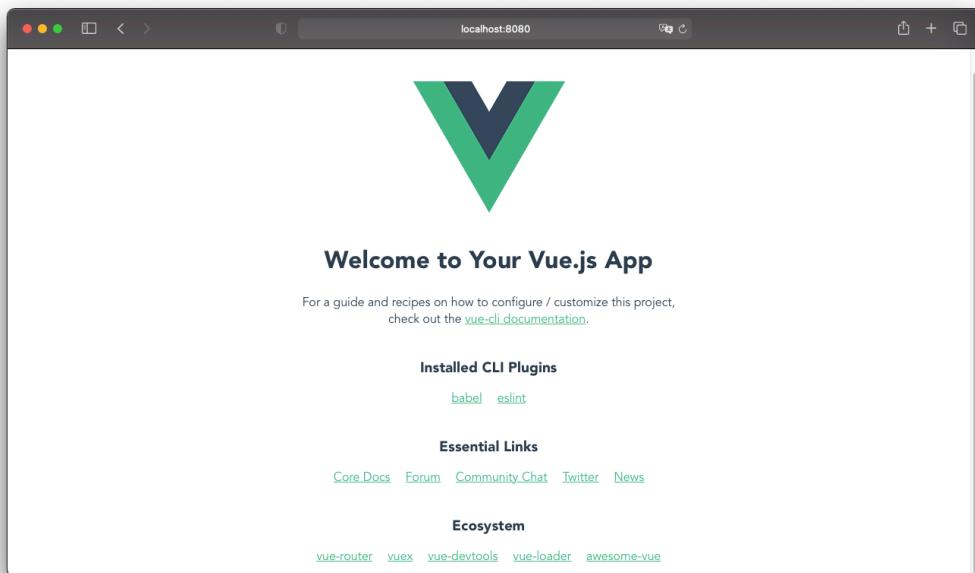


Figure 17 : Point de départ d'une application *Vue.js* fraîchement créée

#### 5.6.4.4 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
```

#### 5.6.4.5 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 appelé 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">
      <span>Sign In</span>
    </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 {
      loading: false,
      message: '',
      schema,
    };
  },
  methods: {
    handleLogin(user) {
      this.loading = true;

      this.$store.dispatch('auth/login', user).then(
        () => {
          this.$router.push('/profile');
        },
        (error) => {
          this.loading = false;
          this.message = (error.response && error.response.data && error.response.data.message)
            || error.message || error.toString();
        }
      );
    }
  }
};
</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 l'accède à sa page de profile.

```
// 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/weevoood/HEIG-VD\\_Travail-de-Bachelor/tree/main/4.Tests-Technos/Vue.js](https://github.com/weevoood/HEIG-VD_Travail-de-Bachelor/tree/main/4.Tests-Technos/Vue.js).

## 5.6.5 Angular

### 5.6.5.1 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>)

### 5.6.5.2 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* ».

#### 5.6.5.3 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*.

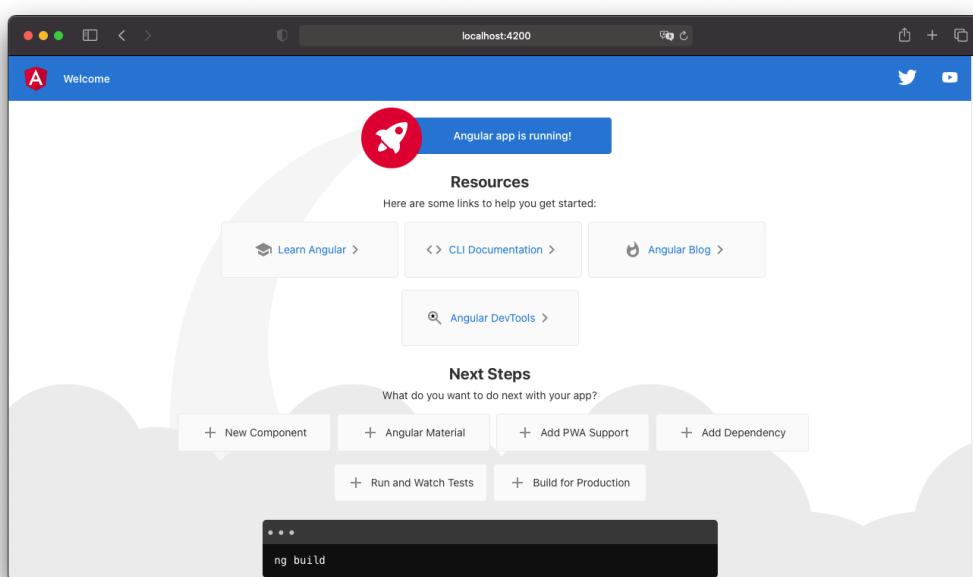


Figure 18 : Page d'accueil d'un projet Angular lors de sa création

#### 5.6.5.4 Développement de l'application de test

Avec *Angular*, pas besoins d'installer de dépendances car il embarque déjà toutes les fonctionnalités requises. Par sa nature orienté-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 auront 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 appelés 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.alertService.error(error);
          this.loading = false;
        }
      });
  }
}
```

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* mets 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(<string>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/weevood/HEIG-VD\\_Travail-de-Bachelor/tree/main/4.Tests-Technos/Angular](https://github.com/weevood/HEIG-VD_Travail-de-Bachelor/tree/main/4.Tests-Technos/Angular).

### 5.6.6 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 mieux 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 claire 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 confirme quand on observe et compare la taille des projets réalisés.

 <b>Vue.js</b> Modified: Today, 15:33	117.7 MB	 <b>Angular</b> Modified: Today, 15:26	321.7 MB	 <b>React.js</b> 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 applications 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 considéré comme plus compliqué, du fait des fonctionnalités initiales misent 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 dense et plus difficile.

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 établis 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

## 6 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.

### 6.1 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.

#### 6.1.1 MariaDB



[https://commons.wikimedia.org/  
wiki/File:MariaDB\\_colour\\_logo.svg](https://commons.wikimedia.org/wiki/File:MariaDB_colour_logo.svg)

*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.

##### 6.1.1.1 Installation

Pour simplifier l'installation et le déploiement ainsi que pour pouvoir recréer un environnement de développement facilement, j'ai choisi d'utiliser un conteneur *Docker*<sup>18</sup> 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
```

##### 6.1.2 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

<sup>18</sup> <https://www.docker.com/>

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'information pertinente directe à 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.

### 6.1.3 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.

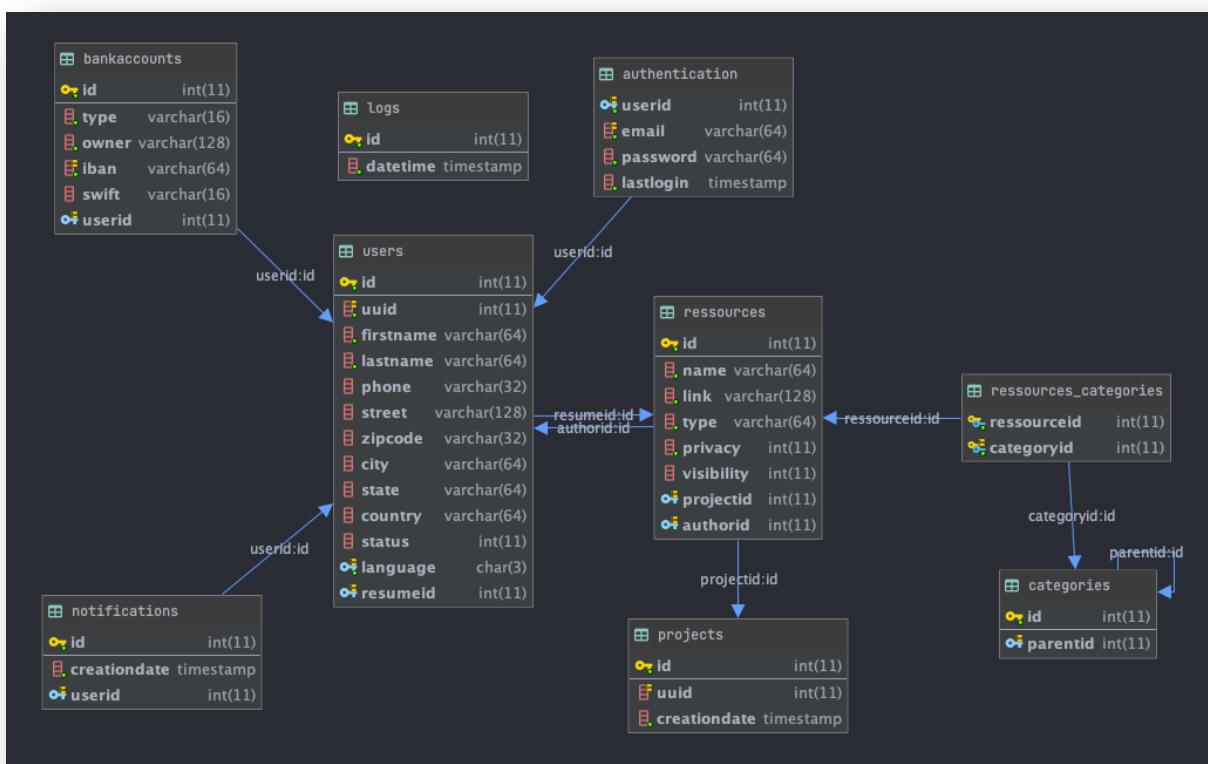


Figure 21 : Schéma initial de la base de données relationnelle

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

- Les tables « *authentication* » et « *bankaccounts* » sont volontairement séparés 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

#### 6.1.4 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).
2. 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 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 ce qui permet une mise en cache très simple.

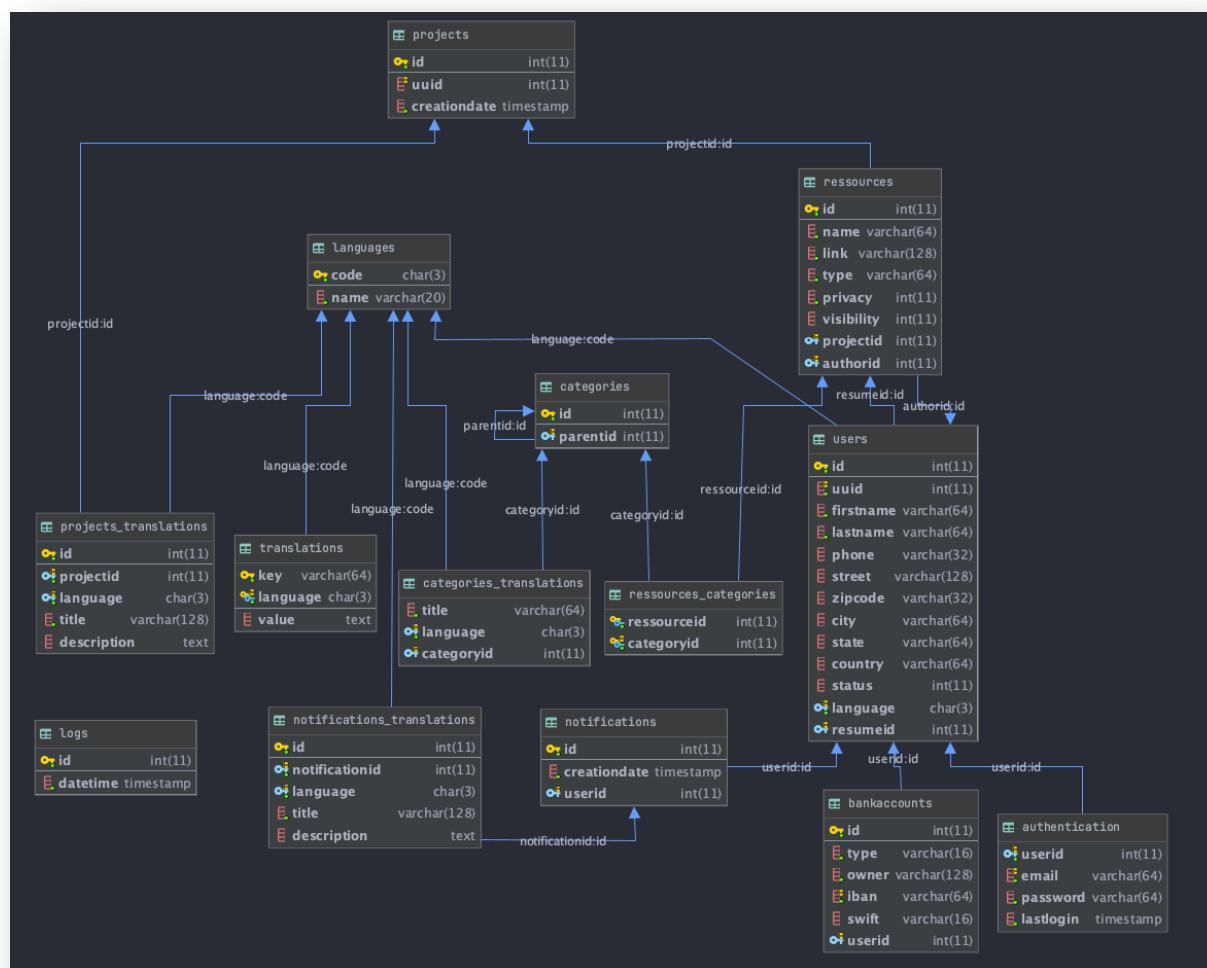


Figure 22 : Schéma de la base de données relationnelle incluant la gestion multilingue

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 leur table de traduction respective.
- 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.

## 6.2 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*.

### 6.2.1 MongoDB



*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*.

### 6.2.2 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.

## 6.3 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 noeuds 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 peu s'adapter au fur et à mesure à des modèles complexes.

### 6.3.1 Neo4j



[https://commons.wikimedia.org/wiki/File:Neo4j-logo\\_color.png](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 noeuds, notamment grâce

au pré calcul des jointures au moment de l'écriture des données. Les requêtes *Neo4j* utilisent le langage *Cypher*<sup>19</sup> qui se veut simple et efficace dans sa formulation.

#### 6.3.1.1 Installation

Tout comme pour *MariaDB*, la base de données *Neo4j* sera déployée dans un container *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>.

#### 6.3.2 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 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. Ou encore de proposer des projets à telle ou telle équipe de développeurs avec un ciblage de caractéristique et selon les différentes relations développées jusqu'alors.

#### 6.3.3 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 similarité. 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.

##### 6.3.3.1 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.

<b>uuid</b>	Identifiant unique universel permettant de retrouver l'entité du modèle relationnel correspondante
<b>tags</b>	Texte, étiquettes de caractéristiques discriminatoires (ex : domaines de compétences)

##### 6.3.3.2 Équipe

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

<sup>19</sup> <https://neo4j.com/developer/cypher/>

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.

<b>uuid</b>	Identifiant unique universel
<b>name</b>	Texte, nom de l'équipe
<b>status</b>	Texte, le statut actuel de l'équipe (actif, abandonnée, bannis)

#### 6.3.3.3 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 il se rapporte.

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.

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

#### 6.3.4 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-orientés). 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

#### 6.3.4.1 IS\_MEMBER\_OF

La première relation identifiée est la relation d'appartenance d'un utilisateur à une équipe, représenté ici par les arcs « *IS\_MEMBER\_OF* ». Ainsi un utilisateur peut faire partie d'une ou 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.

<i>since</i>	Date, depuis combien de temps
<i>isOwner</i>	Booléen, l'utilisateur est-il le propriétaire de l'équipe

#### 6.3.4.2 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>feedback</i>	Texte, rapport sur le déroulement si le projet est terminé

#### 6.3.4.3 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

#### 6.3.4.4 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

#### 6.3.4.5 ARBITRATES

La relation « *ARBITRATIVES* » 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> )

## 7 MODÉLISATION

7.1 Diagrammes de classes

7.2 Diagrammes de cas d'utilisation

7.3 Diagrammes de séquence

## 8 PLANIFICATION

La réalisation de l'application va être faite sur 24 jours. Celle-ci inclut le développement des fonctionnalités, les tests liés à ces développements et les résolutions de bugs. La planification sera divisée en 5 sprints selon la méthode Agile<sup>20</sup>.

Pour faciliter la gestion des tâches, réaliser le suivi et visualiser la progression, j'ai choisi d'utiliser le service en ligne *Trello*<sup>21</sup>. J'ai dans un premier temps reporté dans celui-ci les 5 *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 publiquement à l'adresse suivante : <https://trello.com/b/meyHR8e8/heig-vd-travail-de-bachelor>.

### 8.1 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"> <li>- Mise en place de la structure</li> <li>- Création des « <i>endpoints</i> »</li> <li>- Rédaction de tests unitaires</li> </ul>
Validation	Approbation de tous les tests unitaires définis

### 8.2 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"> <li>- Réaliser les différentes vues de l'interface utilisateur</li> <li>- Répondre aux cas d'utilisation identifiés</li> <li>- Réaliser les <i>Epics</i> de priorité 3 (<i>Epics</i> : 1, 2, 6)</li> <li>- Réaliser les <i>Epics</i> de priorité 2 (<i>Epics</i> : 3, 4, 5, 8)</li> <li>- Si le temps le permet, commencer la réalisation d'<i>Epics</i> de priorité 1</li> </ul>
Validation	Tests fonctionnels de navigation entre les pages, acceptation visuelle des vues

### 8.3 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"> <li>- Mettre en place la connexion entre les interfaces et l'API</li> <li>- Intégrer les différents <i>endpoints</i> de l'API aux interfaces</li> </ul>

<sup>20</sup> <https://scrumguides.org/index.html>

<sup>21</sup> <https://trello.com/>

<b>Validation</b>	Tester et valider l'intégration complète de l'application avec l'API
-------------------	--

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

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

## 8.5 Sprint N°5 : Finalisation du projet

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

## 9 RÉALISATION

### 9.1 Back-end avec *Express.js*

Lien postman ou pdf annexe ?

Code source dispo : lien Github

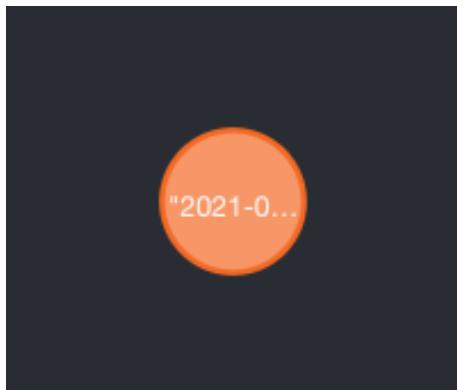
### 9.2 Front-end avec *Vue.js*

Code source dispo : lien Github

### 9.3 Recommandations sur *Neo4j* à l'aide de *Cypher*

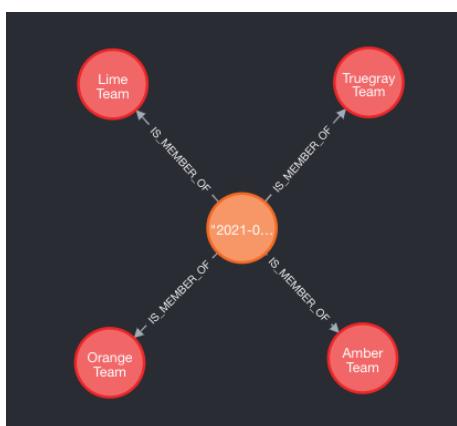
```
// Trouver un utilisateur via son uuid

MATCH (u:User {uuid: '1e93dbe8-501c-4b84-9b65-7e7c1fceb6f4'})
RETURN u
```



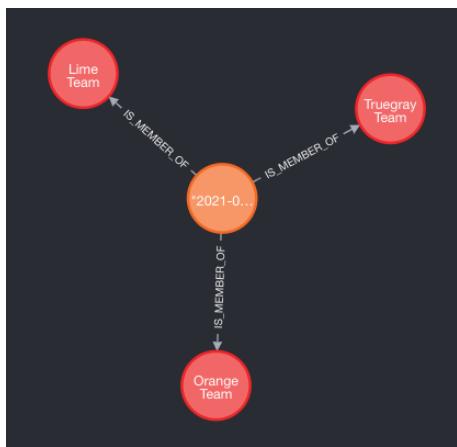
```
// Trouver tous les groupes dont fait partie un utilisateur

MATCH (u:User {uuid: '1e93dbe8-501c-4b84-9b65-7e7c1fceb6f4'})-[r:IS_MEMBER_OF]->(t:Team)
RETURN u, r, t
```



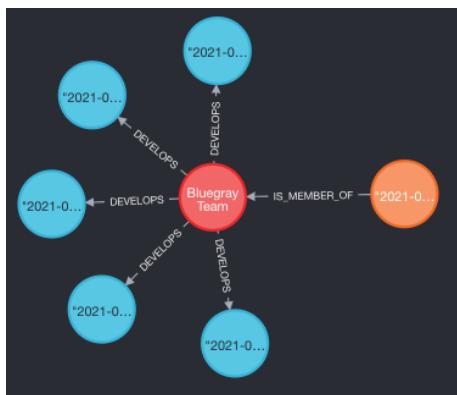
```
// Trouver tous les groupes d'un utilisateur avec une relation active
```

```
MATCH (u:User {uuid: '1e93dbe8-501c-4b84-9b65-7e7c1fceb6f4'})-[r:IS_MEMBER_OF {status: 2}]->(t:Team)
RETURN u, r, t
```



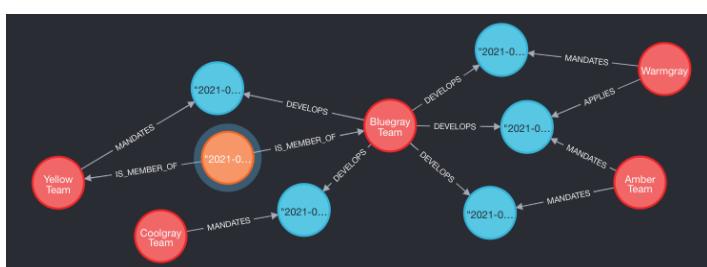
// Trouver tous les projets développés par un utilisateur

```
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
```



// 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
```



### 9.3.1 Recommandations N°1

Trouver les projets des domaines de compétences similaires (via leur tags) que les 3 meilleurs projets développés ayant obtenu 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": "2021-09-03T11:27:16.779000000+02:00", "deadline": "2020-10-05T07:58:19.989000000+02:00", "uuid": "8218d3fb-74e1-444b-9698-45d0a9aa", "status": 6, "tags": ["REBOL", "Clojure", "EuLisp"]}	4.9	["REBOL", "Clojure", "EuLisp"]
{"createdAt": "2021-09-03T11:27:16.779000000+02:00", "deadline": "2020-11-21T08:43:01.101000000+01:00", "uuid": "9ebf66bb-5694-4b70-bea4-c29b2ec5", "status": 6, "tags": ["RAPID", "Alma-0", "Euphoria"]}	3.14	["RAPID", "Alma-0", "Euphoria"]

```
// Trouver les 10 projets possédant les mêmes tags que les 3 meilleurs projets développés
MATCH (u:User {uuid: '24b4d8f8-9daa-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
WITH REDUCE(output = [], t IN tags | output + t) AS flatTags
MATCH (recs:Project)
WHERE recs.status = 4
    AND ANY(t IN apoc.convert.fromJsonList(recs.tags) WHERE t IN flatTags)
RETURN recs AS Recommendations
LIMIT 10
```



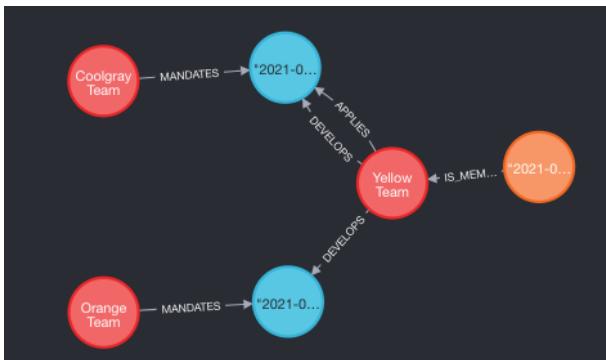
TODO : ne pas ressortir les projets déjà en lien ?

### 9.3.2 Recommandations N°2

Trouver les nouveaux projets proposés par les mêmes mandants que les projets déjà réalisés avec ceux-ci.

```
// 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
```



```
// 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": "2021-09-03T11:27:16.778000000+02:00", "deadline": "2021-05-08T15:21:05.315000000+02:00", "uuid": "064e866a-3440-426f-881b-2efef5c2ce7d", "status": 7, "tags": ["OptimJ", "LOLCODE", "Crystal"]}]
```

TODO : ne pas ressortir les projets déjà en lien ?

### 9.3.3 Recommandations N°3

Trouver les projets pour lesquels les équipes concurrentes ont postulé.

```
// 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
```



```
// 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
LIMIT 10
```

## 9.4 Tests

## 10 AMÉLIORATIONS

*cf rapport Loïc*

### 10.1 TODO 1

## 11 CONCLUSION

# Bibliographie

Wappalyzer. (Visité le 14.06.2021). *UI frameworks*. <https://www.wappalyzer.com/technologies/ui-frameworks/>

W<sup>3</sup>Techs. (Visité le 23.06.2021). *Technologies Overview*. <https://w3techs.com/technologies/>

The State of JavaScript Survey. (Visité le 23.06.2021). *Technologies*. <https://2020.stateofjs.com/en-US/technologies/>

Baumann A. (12 octobre 2016). Une brève histoire du web en 8 étapes. <https://apptitude.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](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](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>

## Annexes

## 12 ANNEXES

### 12.1 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/weevood/HEIG-VD\\_Travail-de-Bachelor](https://github.com/weevood/HEIG-VD_Travail-de-Bachelor).

#### 12.1.1 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*.

#### 12.1.2 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*.

#### 12.1.3 Maquettes *HTML*

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

## 12.2 Ressources externes

### 12.2.1 Documentation de l'API

La définition des routes de 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>.

### 12.2.2 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>.

### 12.2.3 Démonstration fonctionnelle

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

## 12.3 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*<sup>22</sup> ». 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.

<sup>22</sup> <http://info.cern.ch/Proposal.html>

3. L'*Hypertext Transfer Protocol*, abrégé HTML qui est le protocole de communication client-serveur permettant la récupération de ressources du web.

#### 12.3.1 Le HTML

La première page web d'internet<sup>23</sup> est 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.

#### 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 24 : Première page d'internet publiée en décembre 1990,  
<https://www.w3.org/History/19921103-hypertext/hypertext/WWW/TheProject.html>

#### 12.3.2 Le CSS

Il faudra attendre six ans pour que le CSS voie 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.

<sup>23</sup> <https://www.w3.org/History/19921103-hypertext/hypertext/WWW/TheProject.html>

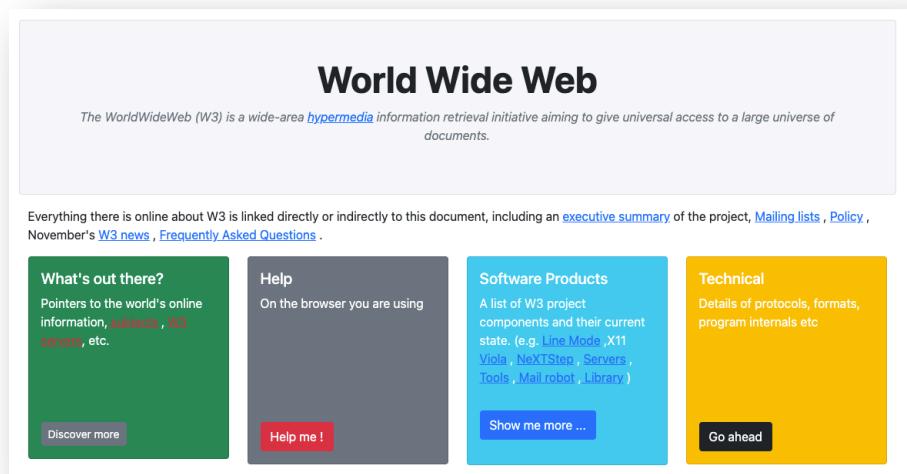


Figure 25 : Exemple de ce à quoi pourrait ressembler la première page HTML en lui appliquant quelques styles CSS

### 12.3.3 Le JS

Dès 1995, Brendan Eich pense et implémente le langage JavaScript, cependant il faudra attendre le concept de programmation AJAX, l'objet `XMLHttpRequest`<sup>24</sup> 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 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.

### 12.3.4 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 tel 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 la publiée sous-licence libre en 1995.

### 12.3.5 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

<sup>24</sup> <https://developer.mozilla.org/fr/docs/Web/API/XMLHttpRequest>

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 relationnelle, 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.

## 12.4 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 ces interactions. Il décrit comment un système informatique doit être conçu de manière à répondre aux spécifications.

### 12.4.1 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*).

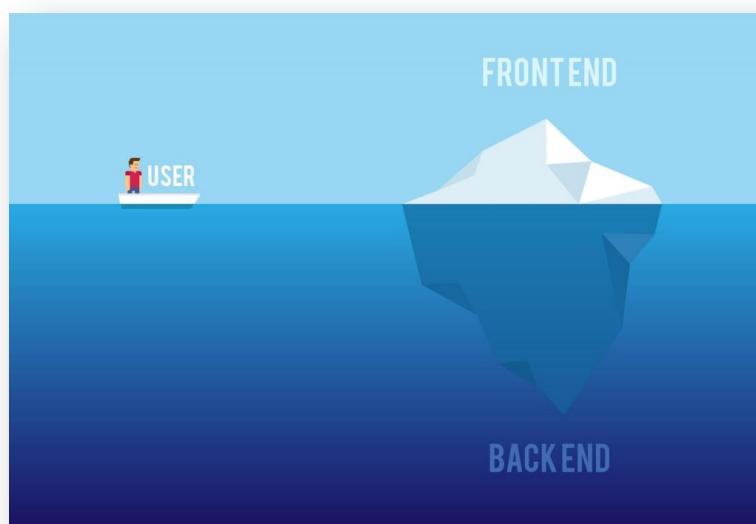


Figure 26 : Différences entre Back-end / Front-end et Full Stack, 11 février 2021,  
<https://www.leproductowner.com/fiches-metiers/backend-frontend-fullstack>

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, dû traiter des données, de vérifier des jetons de sécurités, 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.

#### 12.4.2 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 celle-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.

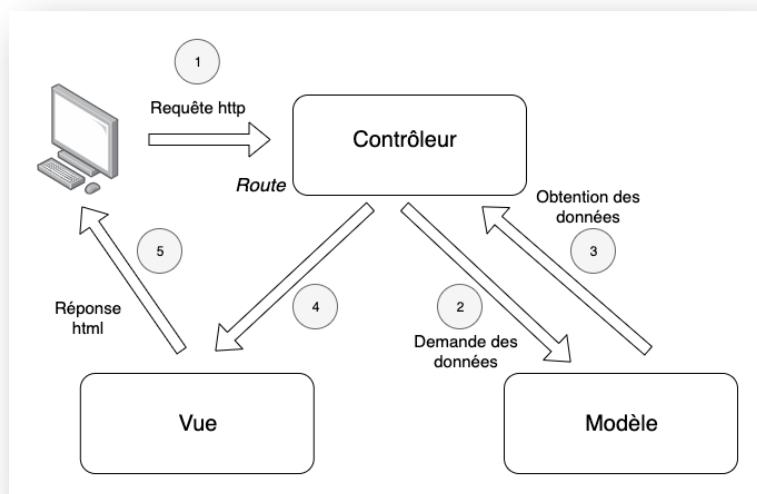


Figure 27 : Représentation des interactions entre le modèle, la vue et le contrôleur dans le cas d'une application web,  
[https://commons.wikimedia.org/wiki/File:Mod%C3%A8le-vue-contr%C3%B4leur\\_\(MVC\) - fr.png?uselang=fr](https://commons.wikimedia.org/wiki/File:Mod%C3%A8le-vue-contr%C3%B4leur_(MVC) - 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.

#### 12.4.3 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.

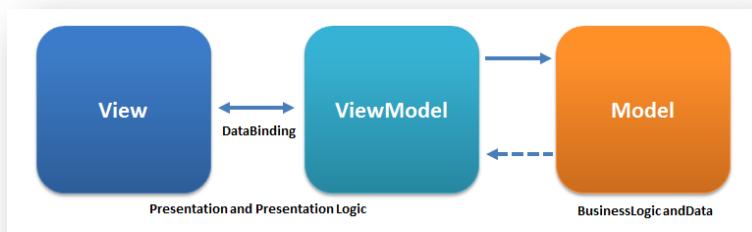


Figure 28 : Représentation de l'architecture MVVM et des interactions entre la vue, la vueModèle et le modèle,  
<https://en.wikipedia.org/wiki/File:MVVMPattern.png>

Dans cette architecture, les requêtes réalisées par l'utilisateur sur l'interface vont aller modifier une ou plusieurs données présentes dans le *viewModel*. Cette action peut provoquer un appel au code métier dans le *modèle*, qui va à son tour renvoyer une nouvelle donnée au *viewModel*. La *vue* ne sera pas changée, mais s'adaptera simplement pour afficher la ou les nouvelles données qui lui seront passées dynamiquement par le *viewModel*.

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*.

#### 12.4.4 Architecture trois tiers

L'architecture « *trois tiers* », « *trois niveaux* » ou « *trois couches* » 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 ; corresponds à l'affichage, à la restitution sur l'écran et au dialogue avec l'utilisateur
2. La couche de *traitement métier* des données ; corresponds 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 ; corresponds aux données qui sont destinées à être conservées sur une certaine durée et/ou de manière définitive.

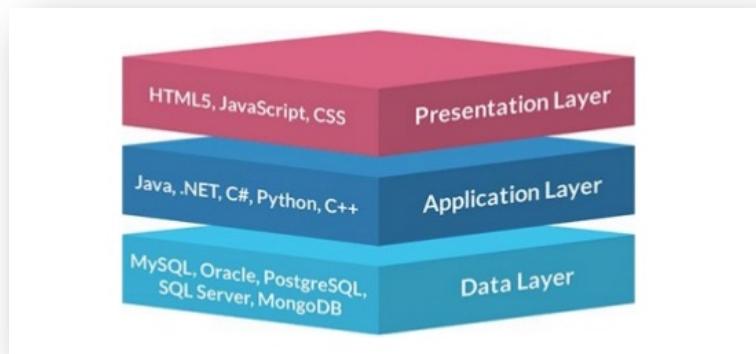


Figure 29 : Représentation de l'architecture 3-tier et des langages utilisés dans chacune des couches,  
<https://www.slideshare.net/TharinduWeerasinghe/multitier-designs-in-software>

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).

## 12.5 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 cauchemar pour les ingénieurs logiciels. En effet, ils devraient alors tout recréer à partir de zéro et ce 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.

### 12.5.1 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.

### 12.5.2 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*.

### 12.5.3 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.

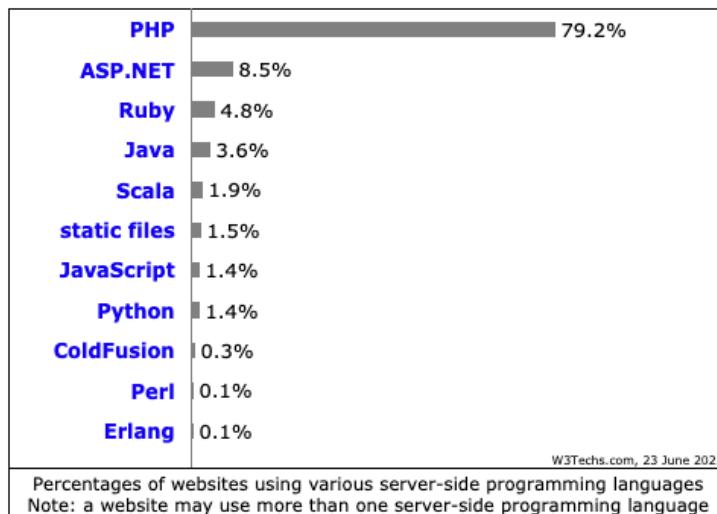


Figure 30 : Pourcentages de répartition des langages de programmation côté serveur utilisés par les sites web en juin 2021,  
[https://w3techs.com/technologies/overview/programming\\_language](https://w3techs.com/technologies/overview/programming_language)

Selon les données de W<sup>3</sup>Techs<sup>25</sup>, PHP est encore aujourd’hui le langage côté serveur le plus utilisé et de loin avec presque 80% de part de marché ! Ensuite APS.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<sup>26</sup> qui, toujours selon les données de W<sup>3</sup>Techs, occupe aujourd’hui presque un tiers du web (65% en juin 2021).

**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](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](https://en.wikipedia.org/wiki/File:Ruby_On_Rails_Logo.svg)

- 2004 (17 ans)
- Community
- Ruby
- Licence MIT
- rubyonrails.org

**Java**

**Scala**

**Python**

<sup>25</sup> <https://w3techs.com/>

<sup>26</sup> <https://wordpress.com/fr/>



[https://en.wikipedia.org/wiki/  
File:Spring\\_Framework\\_Logo\\_2018.svg](https://en.wikipedia.org/wiki/File:Spring_Framework_Logo_2018.svg)

- 2002 (19 ans)
- Pivotal Software
- Java
- Apache License 2.0
- [spring.io](http://spring.io)



[https://en.wikipedia.org/wiki/  
File:Play\\_Framework\\_logo.svg](https://en.wikipedia.org/wiki/File:Play_Framework_logo.svg)

- 2007 (14 ans)
- Lightbend & Zengularity
- Scala
- Apache License 2.0
- [playframework.com](http://playframework.com)



[https://en.wikipedia.org/wiki/  
File:Ruby\\_On\\_Rails\\_Logo.svg](https://en.wikipedia.org/wiki/File:Ruby_On_Rails_Logo.svg)

- 2005 (16 ans)
- Django Software Foundation
- Python
- 3-clause BSD
- [djangoproject.com](http://djangoproject.com)

## 12.5.4 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.

## 12.6 Solutions « *stack* »

Une « *pile de solutions* », plus connus 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* les plus célèbre est sous doute LAMP (*Linux*) et ses pendant MAMP (*macOS*) et WAMP (*Windows*).