#### 

ENAC-IT2

EPFL | Nicolas Crausaz

TPI – Canap-Gest

« Canap-gest » Application WEB de gestion de candidatures d’apprentissage epfl

# Sommaire

[Sommaire 1](#_Toc7520872)

[1 Analyse préliminaire 2](#_Toc7520873)

[1.1 Introduction 2](#_Toc7520874)

[1.2 Situation actuelle 2](#_Toc7520875)

[1.3 Objectif 4](#_Toc7520876)

[1.4 Travail préalable 5](#_Toc7520877)

[2 Analyse du projet 7](#_Toc7520878)

[2.1 Points importants du cahier des charges 7](#_Toc7520879)

[2.2 Faisabilité du projet 10](#_Toc7520880)

[2.3 Choix des technologies 11](#_Toc7520881)

[2.4 Planification initiale 11](#_Toc7520882)

[2.5 Matériel et ressources à disposition 11](#_Toc7520883)

[2.5.1 Global 11](#_Toc7520884)

[2.5.2 Services 11](#_Toc7520885)

[2.5.3 Dépendances - Description & Installation 11](#_Toc7520886)

[2.6 Versionning et sauvegardes 15](#_Toc7520887)

[3 Réalisation 16](#_Toc7520888)

[3.1 Modèle DB 16](#_Toc7520889)

[3.1.1 MCD 16](#_Toc7520890)

[3.1.2 MPD 17](#_Toc7520891)

[3.2 Structure de l’application 18](#_Toc7520892)

[3.3 Configuration 19](#_Toc7520893)

[3.4 Authentification 19](#_Toc7520894)

[3.5 Requêtes API 20](#_Toc7520895)

[3.6 Interface 23](#_Toc7520896)

[3.6.1 Structure 23](#_Toc7520897)

[3.6.2 Design 23](#_Toc7520898)

[3.6.3 Fonctionnalités 24](#_Toc7520899)

[3.6.4 Code 24](#_Toc7520900)

[3.7 Gestion des erreurs 24](#_Toc7520901)

[3.8 Tests 24](#_Toc7520902)

[4 Conclusion 25](#_Toc7520903)

[4.1 Atteinte des objectifs 25](#_Toc7520904)

[4.2 Planification 25](#_Toc7520905)

[4.3 Problèmes rencontrés 25](#_Toc7520906)

[4.4 Améliorations possibles 25](#_Toc7520907)

[4.5 Évolution 25](#_Toc7520908)

[5 Annexes 25](#_Toc7520909)

[5.1 Sources 25](#_Toc7520910)

[5.2 Compte-rendu des discussions 26](#_Toc7520911)

[5.3 Journal de travail 26](#_Toc7520912)

[5.4 Code source 26](#_Toc7520913)

[6 Glossaire 27](#_Toc7520914)

# Analyse préliminaire

## Introduction

Il y a deux ans, j’ai développé un formulaire permettant la postulation en ligne des potentiels apprentis, pour la formation apprentis EPFL. Ceci visait à remplacer le formulaire papier envoyé par poste qui était jusqu’à présent utilisé.

Ce formulaire permet de saisir diverses informations (personnelles, scolaire, professionnelles) et de joindre des fichiers annexes (cv, lettre de motivation etc.).

Le système est stable et fonctionne bien, cependant son utilisation comporte encore quelques contraintes. L’idée du projet « Canap-Gest» naît donc du principe de l’amélioration de ce système.

## Situation actuelle

La personne désirant postuler pour un apprentissage à l’EPFL se rend sur le formulaire, crée un compte au travers du service « compte guest » de l’EPFL, puis rempli ses informations et joint ses fichiers annexes.

Les résultats des postulations sont stockés sur un partage réseau réservé à cet effet, dans un dossier « nouvelles postulations ».

Le partage est structuré en 4 dossiers :

* Nouvelles : Résultats des postulations
* Valides : Dossiers valides
* Exclues : Dossiers non valides, incomplets, etc.
* Engagées : Dossiers retenu et prochainement engagés

Les responsables de la formation apprentis EPFL s’occupent de procéder à la sélection des dossiers et de transmettre les dossiers (en les déplaçant entre les dossiers parents) aux formateurs qui choisiront leurs futurs apprentis.

Les résultats des postulations suivent une structure de dossier spécifique, triée premièrement par métier, le dossier du postulant contient ensuite les fichiers annexes et les informations, celles-ci sont stockées dans un fichier JSON.

Le diagramme suivant récapitule la stratégie actuelle :



Figure 1: Fonctionnement actuel du système

Quelles sont donc les contraintes actuelles ?

* Les informations personnelles du postulant sont stockées dans un fichier JSON, il n’est donc pas facilement lisible sans utiliser quelconque outil pour faciliter son accès.
* Les responsables modifient directement les dossiers sur le partage, en cas de mauvaise manipulation (par exemple glisser-déposer au lieu de copier-coller) cela peut entrainer des problèmes (fichiers corrompus, permissions modifiées etc.).
* Nous ne disposons pas d’une vue globale sur les postulations.
* Nous ne disposons pas d’une manière de trier et comparer efficacement les postulations entre elles selon certains critères.

Des améliorations au niveau de la protection des données doivent également être appliquées, par exemple, dans l’état actuel, un candidat ne peut pas voir, modifier ou supprimer sa ou ses postulations effectuées, ce qui entraine des conflits avec la loi RGPD*.*

## Objectif

Le but principal serait donc de trouver des compromis pour éliminer les contraintes actuelles, il faut donc appliquer une solution permettant principalement de :

* Eviter la manipulation des fichiers de postulation directement sur le partage réseau.
* Améliorer l’accès et la facilité de traitement des données des candidats.
* Améliorer et faciliter le travail de sélection des candidatures valide par les responsables Formation Apprentis EPFL.
* Faciliter le travail de sélection et recrutement des postulants par les formateurs EPFL.

L’idée qui a été retenue pour obtenir ses résultats est de développer une interface de gestion centralisée, permettant aux responsables de la Formation Apprentis EPFL et des formateurs d’effectuer tout leur travail de tri/recrutement.

Pour ce faire, il a fallu avant tout réaliser les points suivants :

* Enregistrer les informations des candidatures (sans les fichiers) dans une base de données pour en faciliter l’accès ultérieur.
* Développer une interface web permettant l’administration des candidatures
* Développer une API permettant l’accès à la base de données, faisant le lien entre l’interface et la base de données.

## Travail préalable

Une partie du travail a été réalisée au préalable ; Premièrement j’ai créé et mis en production une base de données (DB), qui est hébergée sur un serveur MySQL de l’EPFL.

Ensuite, le formulaire à évidement dû être adapté pour pouvoir enregistrer les données en DB, et que le postulant puisse voir, éditer et supprimer ses postulations.

L’API pour l’accès aux données a également été développée en partie pour préparer l’implémentation des fonctionnalités de la future interface.

L’ajout de la base de données a déjà permis de combler la problématique du traitement des fichiers d’information directement sur le partage réseau et d’en simplifier sa structure, puisque que celui-ci ne sera utiliser que pour les fichiers annexes des candidatures et que l’accès direct au serveur ne sera plus utile.

Plusieurs discussions avec les responsables de la formation apprentis EPFL ont été entreprise et un questionnaire conçu et envoyé aux formateurs EPFL pour leur proposer des fonctionnalités et connaitre leur avis sur l’idée du projet et leurs éventuelles propositions de fonctionnalités.

Le chef de projet de ce projet s’est notamment basé sur les réponses de ce questionnaire pour établir les critères du cahier des charges et du projet.

Le diagramme suivant résume la situation :



Expliquer les rôles

# Analyse du projet

## Points importants du cahier des charges

Comme décrit auparavant, le projet consiste principalement à développer une application web offrant les caractéristiques suivantes, les fonctionnalités sont séparées entre les formateurs et les responsables.

**Fonctionnalités pour les responsables**

* **Gérer la liste des sites EPFL (lieux de formation)**

Il faudra ajouter une table « lieu » au modèle de la base de données actuel et d’ajouter des requêtes sur l’API pour l’ajout et l’édition des lieux.

L’interface permettra l’édition des lieux par les responsables.

* **Gérer la liste des professions et des places ouvertes**

Il faudra ici également ajouter plusieurs tables au modèle de la base de données actuel et d’ajouter des requêtes sur l’API pour l’ajout et l’édition des lieux.

Ces fonctionnalités sont communes à l’interface réalisée pour le point précédent.

* **Voir les candidatures, les valider, refuser ou les marquer comme incomplètes**

Au niveau de l’interface, il faudra ajouter une vue pour lister les postulations de manière globale et également de manière individuelle. Les vues devront avoir les fonctionnalités pour modifier le statut de la candidature.

Au niveau de l’API, les requêtes liées aux fonctionnalités ont déjà été implémentées, cependant un changement de la structure de la base de données, notamment l’ajout d’une table « statut » est envisageable, il faudra donc mettre à jour les fonctions de l’API.

* **Refuser des « vagues » de candidatures et supprimer leurs données**

La vue globale des candidatures devra offrir la possibilité de sélectionner plusieurs candidatures et de les supprimer en effectuant une seule action.

Lors d’un refus, le responsable contactera « manuellement » la personne concernée, puis ses données devront être totalement supprimées.

**Fonctionnalités pour les formateurs**

* **Lister les candidatures, globales et individuellement, voir les informations complètes et les fichiers annexes**

Les vues globales et individuelles devront s’adapter en fonction du rôle de l’utilisateur (responsable ou formateur) et afficher ou non des fonctionnalités.

La vue globale contiendra les informations personnelles les plus importantes d’une candidature. La vue individuelle regroupera toutes les informations liées (responsables, parcours scolaire, fichiers annexes etc.).

* **Identifier les nouvelles candidatures**

Il faudra rendre identifiables les nouvelles candidatures, par exemple en affichant le nombre de nouvelles candidatures depuis la dernière connexion de l’utilisateur. L’ajout d’un marqueur dans la vue globale peut également être ajouté pour améliorer la visibilité de ces candidatures.

* **Attribuer des notes aux candidats**

Un formateur pourra attribuer des notes aux candidatures afin qu’il puisse organiser ses candidats favoris. Un système de notation, par exemple de 1 à 5, pourra être mis en place.

Pour cela il faudra ajouter un table « note » à la base de données et ajouter les fonctionnalités nécessaires à l’API (ajout, modification et suppression).

* **Ajouter des commentaires aux candidats, privé ou partagés avec les autres formateurs**

Un formateur pourra commenter une candidature et choisir de partager ce commentaire avec les autres formateurs disposant de l’accès à cette même candidature, ou alors de restreindre la visibilité du commentaire à lui-même.

Ces commentaires s’afficheront sur la vue individuelle d’une candidature.

Des points techniques globaux seront également évalués et devront donc être mis en place :

* **Gestion de exceptions**

Les exceptions et erreurs devront être gérées à tous les niveaux de l’application, c’est-à-dire, au niveau des interactions avec la base de données, des erreurs de l’API (renvoi des messages d’erreur à l’interface) et également de l’interface côté client.

* **Connexion à la base de données au travers d’un compte de service n’ayant que le minimum de droits nécessaires**

Le compte de service utilisé par API pour se connecter à la base de données devra uniquement disposer des droits nécessaires aux interactions effectuées. Le script de création de cet utilisateur sera compris dans le script de création de la base de données.

* **Respect des directives web EPFL basiques**

Le choix du design général de l’application est libre, mais devra respecter des directives EPFL de base, c’est-à-dire d’avoir le logo EPFL en haut à gauche de la page et un pied de page comprenant le texte « © EPFL, tous droits réservés. »

* **L’application fonctionne sur les navigateurs modernes**

Les technologies utilisées devront être compatible avec les navigateurs modernes Firefox et Google Chrome. L’application devra fonctionner et s’afficher correctement sur ces deux navigateurs.

* **L’application n’est pas sensible aux attaques XSS et aux injections SQL**

*Continuer*

* **Le code respecte les conventions Vue.js**

Les convention Vue.js de catégorie A[[1]](#footnote-1) doivent être appliquées.

En cas d’exceptions, elles devront être documentées.

Afin de démontrer ces conventions dans un cas plus pratique, elles seront détaillées plus tard dans le rapport[[2]](#footnote-2).

Afin d’avoir plus de précisions sur certains points du cahier des charges, quelques questions ont été posées au chef de projet :

**Q : Est-ce que les responsables peuvent également attribuer des notes et commentaires sur les candidatures ?**

R : Non, ces fonctionnalités sont réservées aux formateurs, cependant les commentaires publics sont également visibles par les responsables.

**Q : Est-ce qu’il faut également mettre à jour les fonctionnalités du formulaire de candidatures suite aux modifications de la base de données**

R : Non, ces modifications ne sont pas comprises dans le projet et se feront ultérieurement.

**Q : Est-ce que les formateurs peuvent voir toutes les candidatures qu’importe leur statut ?**

R : Non, les formateurs ne peuvent voir uniquement les candidatures notées comme valides.

## Faisabilité du projet

Le projet dans son ensemble semble tout à fait réalisable, les prérequis techniques demandés (Vue.js, PHP, MySQL) sont acquis et ne devraient donc pas poser de problèmes ou bloquer la bonne progression du projet dans son ensemble.

Les points du cahier des charges sont assez précis, mais laissent tous de même une entière liberté au niveau de leur réalisation, les quelques questions posées au chef de projet ont permis de répondre aux dernières incertitudes sur ces points. Les fonctionnalités demandées semblent réalisables au niveau de mes connaissances et compétences techniques. Certaines de ces fonctionnalités sont d’ailleurs déjà partiellement implémentées, cependant beaucoup de modifications devront y être apportées.

Le temps à disposition est fixé à 88 heures sur 11 jours. J’estime que la charge de travail relativement correcte par rapport au temps à disposition, ce qui me laissera un minimum de marge d’erreur sur la planification du projet.

Je prévois d’anticiper au maximum le travail en cas d’avance sur la planification et en cas de problèmes, de ne pas rester bloquer trop longtemps sur la tâche qui pose problème. En cas de manque de temps je privilégierai la rédaction des documentations quitte à laisser une fonctionnalité de l’application partiellement ou non terminée.

## Choix des technologies

Les technologies principales de ce projet ont été choisies au préalable suite à des discussions avec le chef de projet. Au final le choix s’est porté vers PHP pour le backend (API), MySQL (DB) et Vue.js pour le frontend (interface).

L’utilisation d’un Framework PHP pour la réalisation de l’API à également été mise en avant et j’ai donc porté mon choix sur le micro-Framework Lumen. Le fait que Lumen fournisse uniquement les méthodes permettant la réalisation d’API est un argument qu’y m’a conforté dans le choix de ce Framework

Au niveau du frontend, le choix de Vue.js était très important car je dispose d’une expérience conséquente avec ce Framework, notamment grâce à la réalisation de divers projets de grandes envergures que j’ai eu l’occasion de réaliser avec cette technologie.

*Developer encore…, vuetify ?*

## Planification initiale

La planification initiale se trouve en annexe.

Afin de garder une trace des éventuels problèmes ou bugs dans mon application et également d’organiser mon travail, j’utiliserai la plateforme Trello qui me permet de créer des tableaux de tâches et d’y ajouter des remarques.

## Matériel et ressources à disposition

### Global

* Un PC sous Windows 10 Education
* Un IDE, Visual Studio Code
* Un client SQL, MySQLWorkbench
* Un serveur web local (WampServer)
* Un dépôt GIT

### Services

* Client Tequila EPFL

### Dépendances - Description & Installation

**Git :**

**WampServer :**

WampServer est une plateforme de développement web conçue pour Windows, elle permet d’héberger un serveur web tournant sur Apache, accompagné de MySQL et PHP.

La plateforme apporte également quelques raccourcis vers des actions de configuration du serveur, par exemple vHosts, configuration PHP etc.

Dans le cadre de ce projet, j’utiliserai PHP en version 7.2

Téléchargeable sur <http://www.wampserver.com/>

**Composer :**

Composer est un gestionnaire de paquet pour PHP, dans le cadre du projet il est utilisé pour installer le Framework Lumen et ses dépendances.

* Télécharger l’installeur sur : <https://getcomposer.org/doc/00-intro.md#installation-windows>
* Ajouter le chemin vers PHP à la variable d’environnement PATH

Pour valider l’installation, terminal :

* *composer*

**Lumen :**

Lumen est un micro-Framework basé sur le Framework Laravel, il réutilise en partie le fonctionnement de ce dernier, en retirant les éléments servant à la partie Vue. Lumen est donc parfaitement adapté pour réaliser une API, il est également très facile à prendre en main du fait de la simplicité de son code et de sa documentation et de sa grande communauté (principalement celle de Laravel). Il embarque par exemple des méthodes pour le routing, les middlewares, l’authentification, la validation etc.

Pour créer un projet Lumen avec une structure de base, dans un terminal :

* *composer create-project --prefer-dist laravel/lumen <nom\_projet>*

Puis pour démarrer l’application :

* *php -S localhost:8000 -t public*

Editer le fichier « .env » se trouvant à la racine du projet avec les informations demandées : APP\_KEY (Il suffit de générer une chaine de caractères pour cette valeur) et info de connexion à la DB.

**firebase/php-jwt :**

Librairie pour l’utilisation de tokens JWT avec PHP, qui permettent des transmettre des informations de manière sécurisée et de faire office d’authentification du client lors des requêtes à l’API.

Pour importer cette libraire, ouvrir un terminal dans le dossier du projet Lumen, puis :

* *composer require firebase/php-jwt*

Dans le fichier «. env », ajouter un champ « JWT\_SECRET », la valeur de ce champ sera la clé privée qui servira à crypter/décrypter les tokens JWT. Il suffit de générer une chaine de caractères pour cette valeur.

**Node.js :**

Node.js est un environnement JavaScript permettant d’interpréter du JavaScript côte serveur. Dans le cadre du projet, Node.js servira uniquement au développement, notamment pour installer les dépendances grâce à NPM, son gestionnaire de paquets inclus nativement.

* Télécharger et installer Node.js (v.11 actuellement) sur : <https://nodejs.org/en/>

**Vue.js**

*VUEJS*

Orienté composant

Revient en détail dans un autre point (Dev, code d’exemple, cycle de vie composant, evenement, v-model etc.)

**@vue/cli**

Outil en ligne de commande servant à générer des projet Vue.js avec une structure de base, en choisissant les dépendances souhaitées.

Il faut installer le CLI de manière globale :

* *npm install @vue/cli* ***-g***

L’argument -g permet d’installer le module de manière globale au système.

Une fois installé on peut donc l’utiliser depuis un prompt :

* *vue*

Pour créer un nouveau projet :

* *vue create <nom\_app>*

Choisir manuellement les paquets :

* Babel
* Vuex
* Router
* Linter / Formatter

Use history mode ? N

Il s'agit d'enlever le "#" dans l'url qui est affiché par vue-router. Si l’on choisit oui, il faudra avoir un .htaccess (ou autre) qui redirige toutes les requetes sur index.html

Linter / formatter config :

* ESLint + Standard config
* Lint on save

ESLint permet de valider la manière dont le code est écrit et d’assurer les bonnes pratiques en manière de code (indentation, code superflu etc.) en se basant sur les normes ECMAScript. Ici, j’utilise les standards officiels JavaScript[[3]](#footnote-3).

Les Frameworks JavaScript actuels utilisent la norme ECMAScript Version 6, appelée ES6, alors que les navigateurs ne supportent pas encore totalement cette norme, c’est pour que cela qu’il faut utiliser un transcompilateur (en JavaScript, paquet Babel installé précédemment) qui permet de convertir notre code vers la norme ES5, supportée par les navigateurs.

Configuration des paquets :

* In dedicated config files

Lors de la création du projet, un projet Git est initialisé et un commit initial est effectué.

Pour commencer le développement Vue.js, il faut installer les dépendances :

* *npm install*

Puis démarrer le serveur de développement :

* *npm run serve*

Cette commande va exécuter la fonction « serve » se trouvant dans le fichier *package.json,* situé à la racine du projet, qui va donc démarrer le serveur.

Lors du déploiement en production de l’application, il suffira d’exécuter la commande :

* *npm run build*

Cette commande va optimiser le code source en générant un simple fichier index.html ainsi que ses sources (JavaScript, CSS, et autres éventuels fichiers statiques) dans le dossier « dist »

*Développer le build*

**Vue-router**

**Vuex**

Fonctionnement des actions, mutations, getters etc. et interactions avec composants

« Vuex, c'est comme les lunettes : vous saurez quand vous en aurez besoin. »

**Vuetify**

* *vue add vuetify*

**Axios**

Axios est une librairie JavaScript / Node.js servant à effectuer des requêtes XMLHttpRequest (XHR)[[4]](#footnote-4) depuis un client vers des ressources serveurs

## Versionning et sauvegardes

J’utiliserai dans ce projet, le logiciel de gestion de version (VCS), très populaire, nommé Git.

Git permet de créer des versions, appelés **commit**, d’un ensemble de fichiers. Git permet également de créer des **branches**, permettant de travailler sur plusieurs états du projet en parallèle sans affecter la version stable, la branche par défaut est nommée **master**.

Un **commit** se compose d’un **hash** unique, d’un auteur et d’un message (optionnel mais évidemment très recommandé), le **hash** permet d’identifier le commit et d’y effectuer des actions, par exemple d’effectuer un retour en arrière dans les versions.

Git permet également de publier l’état de l’arbre des versions vers un serveur dédié **remote/repository**, dans le cadre de ce projet, il s’agit de c4science.ch, plateforme git dédiées aux universités suisses.

Exemple d’utilisation :

On initialise le repository :

* mkdir test-repo
* cd test-repo
* git init

On ajoute un nouveau document :

* echo hello >> readme.txt

On affiche l’état actuel des modifications

* git status

On « stage » les modifications

* git add \*

On « commit » :

* git commit -m "votre message"

On peut également ajouter un repository distant et y publier les modifications :

* git remote add origin "lien du repository"
* git push

J’effectuerai des commit / push de manière régulière, lors de modifications majeures et d’un état majoritairement stable de l’application et des documents.

Je garderai également une copie du projet à chaque publication vers c4science son mon espace personnel EPFL qui dispose d’une sauvegarde par heure.

# Réalisation

## Modèle DB

### MCD

Ajouter mcd

### MPD



## Structure de l’application

Comme expliqué précédemment, l’application web est séparée en plusieurs éléments ;

Le Frontend, qui comprend l’interface web coté client (Vue.js), et le Backend qui englobe l’API et la DB ainsi que tout autre service, par exemple le service d’authentification Tequila.

Ce type d’architecture logicielle est très utilisées dans les services web, celle-ci est appelé REST ou RESTful.

Les principes d’un architecture REST sont d’offrir des fonctions à tout ordinateurs ou clients web en leur fournissant des ressources uniquement textuelles, par exemple au format JSON, XML etc. Un service REST est dit **stateless**, c’est-à-dire qu’aucune connexion permanente n’est établie entre le client et le serveur et qu’aucune session n’est conservée du côté serveur, chaque requête est traitée indépendamment.



Les avantages principaux de l’utilisation d’architectures REST sont donc nombreux :

* Tout d’abord le fait que le client ne soit pas lié au reste de l’application sépare le code et permet donc la maintenance distincte sans conséquences sur le fonctionnement global, de ce fait il serait possible par exemple d’avoir plusieurs versions de l’API fonctionnant en même temps, sans incidence sur le fonctionnement des clients. Du fait de la séparation de l’application, plusieurs clients, par exemple une interface web et une application mobile, pourront utiliser la même API REST et donc réduire considérablement le temps de développement du service.
* La charge du serveur est inférieure à celle d’une architecture plus « standard », principalement due à l’absence de la gestion des états (sessions). Le service est donc plus simple et plus performant lors de grand nombre de requêtes simultanées. En cas de service à forte demande, une architecture REST est plus facilement adaptable pour une répartition sur plusieurs serveurs, vu qu’il n’y a pas de session à conserver sur un serveur spécifique ou à répliquer sur des serveurs multiples.
* REST fonctionne avec l’utilisation du protocole HTTP, ce qui permet d’en exploiter ces fonctionnalités, par exemple ses entêtes et **verbs**

## Configuration

Configuration Vue-cli (vue.config.js)

Ajout dans fichier host

Db

Api

User db

Locale vuetify

Etc.

Structure projet API et Vue

## Authentification

Afin de sécuriser les données et l’utilisation de l’API et interface, une authentification à été mise en place. Etant donnée qu’un API REST n’utilise pas de session, il n’y pas de moyen d’établir une connexion « permanente » entre l’API et le client, c’est pour cela qu’il faut authentifier chaque requête faite depuis les clients vers l’API.

J’ai donc choisi d’implémenter l’utilisation des **JSON Web Token (JWT)** (voir 3.4.2).

Lorsque qu’un client se rend sur l’interface, il envoie une requête GET sur une route spécifique pour la connexion à l’API, celle-ci va effectuer la connexion via le service EPFL Tequila qui en cas d’authentification valide renverra les données EPFL de l’utilisateur à l’API.

L’API va elle ensuite générer le **token JWT** contenant ces données, qu’elle enverra en tant que réponse au client. Le client stockera ce token et l’utilisera pour utiliser les requêtes nécessitant une authentification.



*Des lors un req auth se fait :::: continuer*

Un **token JWT** est une chaîne de caractères encodée qui se compose de trois parties distinctes. Prenons par exemple le **token JWT** suivant (dont la deuxième partie a été raccourcie) :

eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJsdzY3ODk3Nn0.upmPwSUPNZdJ-FYhtjHch3FTz\_r0nR9g-eEViFLVt5k

Ces trois parties sont séparées par un « . », la première est appelée « header », elle contient le type de token et l’algorithme d’encryptions utilisé pour le générer. Décryptée, cette chaine correspond à la structure suivante :

{

"typ": "JWT",

"alg": "HS256"

}

La deuxième est appelée « payload », elle contient les données, dans le cas de ce projet, la structure est la suivante :

{

"iss": "canap-gest",

"sub": "262544",

"tequila\_data": {

"firstname": "Nicolas Benjamin",

"name": "Crausaz",

"group": "…",

"user": "ncrausaz",

"sciper": "262544"

},

"permissions": […],

"role": "responsable",

"iat": 1553592576,

"exp": 1553678976

}

Les valeurs « iat » et « exp » sont obligatoires, elles représentantes le **timestamp** de génération et d’expiration. Lors de la validation du token, une vérification du timestamp d’expiration s’effectue, si la date est passée, l’authentification ne sera donc pas possible.

La dernière partie est nommée «signature », en effet elle contient un **hashage** **HMACSHA256** des données des parties précédentes (en **base64**) et de la clé d’encryption du token (qui est définie du côté serveur) :

HMACSHA256(

base64UrlEncode(header) + "." +

base64UrlEncode(payload),

<encryp\_key>

)

Expliquer les données tequila et les groupes epfl

## Requêtes API

Ci-dessous, la liste des requêtes disponibles sur l’API, groupées par « controller »

La base de l’URL est <http://localhost:8000>

Auth : Token obligatoire

Rôle : limitation des la route à certains rôles

Body : Contenu (données) obligatoires pour l’action de la requête, format x-www-form-urlencoded.

Retour au format json

Les routes marquées en VERT ont été ajoutée lors du projet, les autres routes ont été soit modifiées partiellement ou ne l’ont pas été.

**Authentification - AuthController**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Type | Route | Auth (token) ? | Rôle | Body | Description |
| GET | api/auth/login | Non | tous | - | Login via Tequila, retour token |
| GET | api/auth/logout | Non | tous | - | Déconnexion de Tequila |

**Utilisateur - UsersController**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Type | Route | Auth (token) ? | Rôle | Body | Description |
| GET | api/user | Oui | tous | - | Infos user connecté |
| GET | api/user/  permittedjobs | Oui | tous | - | Retourne les métiers visibles par l’user |
| GET | api/user/  hascommentedormarked | Oui | tous | - | Retourne les postulation commentées ou marquées par l’user |
| GET | api/user/data/{sciper} | Oui | tous | - | Retourne les infos LDAP d’un utilisateur selon son sciper |

**Postulations - ApplicantsController**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Type | Route | Auth (token) ? | Rôle | Body | Description |
| GET | api/applicants | Oui | tous | - | Retourne les postulations des tous les métiers accessibles (et non vides) |
| GET | api/applicants/  job/{job} | Oui | tous | - | Retourne les postulations d’un métier |
| GET | applicant/{id} | Oui | tous | - | Retourne toutes les infos d’une postulation |
| GET | applicant/{id}/  export | Oui | tous |  | Exporte et télécharger les données (json) d’une postulation |
| DELETE | applicant/{id} | Oui | resp | - | Supprime une postulation |

**Commentaires - CommentsController**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Type | Route | Auth (token) ? | Rôle | Body | Description |
| GET | api/applicant/  {id}/comments | Oui | tous | - | Retourne les commentaires d’une postulation, privés et publics |
| PUT | api/comment | Oui | tous | content,  is\_private,  applicant\_id | Ajoute un commentaire |
| PATCH | api/comment/  {id} | Oui | tous | content, is\_private | Edite un commentaire |
| DELETE | api/comment/{id} | Oui | tous | - | Supprime un commentaire |

**Notes – MarkersController**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Type | Route | Auth (token) ? | Rôle | Body | Description |
| GET | api/applicant/  {id}/marker | Oui | tous | - | Récupère la note de l’utilisateur sur une candidature |
| PUT | api/marker | Oui | tous | value, applicant\_id | Ajoute une note |
| PATCH | api/marker/{id} | Oui | tous | value | Edite une note |
| DELETE | api/marker/{id} | Oui | tous | - | Supprime une note |

**Statut - StatusController**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Type | Route | Auth (token) ? | Rôle | Body | Description |
| GET | api/status | Oui | tous | - | Récupère et télécharge un fichier |
| PATCH | api/status/  applicant/{id} | Oui | resp | status | Modifie le statut d’une postulation |

**Fichiers - FilesController**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Type | Route | Auth (token) ? | Rôle | Body | Description |
| GET | api/files/{id} | Oui | tous | - | Récupère et télécharge un fichier |

**Offres (places d’apprentissage) – PositionController**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Type | Route | Auth (token) ? | Rôle | Body | Description |
| GET | api/positions | Oui | resp | - | Récupère les places |
| GET | api/locations | Oui | resp | - | Récupère les lieux |
| GET | api/jobs | Oui | resp | - | Récupère les métiers |
| PUT | api/position | Oui | resp | position\_access\_group,  position\_spot\_number,  location\_id,  job\_id | Ajoute une place |
| PUT | api/location | Oui | resp | location\_site | Ajoute un lieu |
| PUT | api/job | Oui | resp | job\_full\_value | Ajoute un métier |
| PATCH | api/position/{id} | Oui | resp | position\_access\_group,  position\_spot\_number,  location\_id,  job\_id | Modifie une place |
| DELETE | api/position/{id} | Oui | resp | - | Supprime une place |

**Statistiques - StatsController**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Type | Route | Auth (token) ? | Rôle | Body | Description |
| GET | api/stats/total | Oui | tous | - | Récupère le nombre de postulation par métier |

## Interface

### Structure

L’interface affichera ou non des éléments en fonction du rôle de l’utilisateur connecté (formateur ou responsable)

Structure des pages et description, explication technique du fonctionnement

Routes

### Design

Couleurs basées sur <https://epfl-idevelop.github.io/elements/#/colors>

Montrer comment éditer le template Vuetify avec les nouvelles couleurs

Couleurs EPFL, logo et footer

Vuetify

### Fonctionnalités

Screen de l’interface avec explications

### Code

(Conventions Vue.js (A)

Les noms de composant devraient toujours être des mots multiples, à l’exception du composant racine App

La propriété data doit être une fonction

Les définitions de prop devraient être aussi détaillées que possible

Toujours utiliser key avec v-for

N’utilisez jamais v-if sur le même élément que v-for

Pour les applications, le style du niveau App au sommet et des composants de mises en page doivent être globaux, mais tous les autres styles des composants devraient être avec une portée limitée au composant

Utilisez toujours le préfixe $\_ pour les propriétés privées personnalisées dans un plugin, mixin, etc. Cela permet d’éviter les conflits avec le code d’autres développeurs. Il est également possible d’inclure un nom de portée (par ex. $\_yourPluginName\_)

)

Techno pricipales (vue, vuex, vuerouter)

Fonctionnement vuex

Composant vue exemple

## Gestion des erreurs

API

DB

Interface

SQL inject

XSS

## Tests

Décrire les tests effectués

Pas de tests unitaires, de tests « codés »

Tests sur navigateurs différents (Chrome et Firefox)

Test effectué -> résultat souhaité -> résultat obtenu

Comment je teste avec plusieurs groupes

Tests :

Acces, retour d’erreur / données, session expirée

# Conclusion

## Atteinte des objectifs

* Objectifs atteints / non-atteints
* Eléments restants

## Planification

* Comparaison du réel avec la planification initiale

## Problèmes rencontrés

* Points positifs / négatifs
* Difficultés particulières

Authentification

Promise

CORS

## Améliorations possibles

Amélioration état actuel (fin TPI)

## Évolution

* Suites possibles pour le projet (évolutions long terme)

# Annexes

## Sources

<https://nodejs.org/en/>

<https://lumen.laravel.com/>

<https://cli.vuejs.org/guide/>

<https://vuejs.org/>

<https://github.com/axios/axios>

<https://vuetifyjs.com/en/>

<https://jwt.io/>

<https://github.com/firebase/php-jwt>

<https://vuex.vuejs.org/fr/>

<https://php.net/>

Services :

<https://mockflow.com/>

<http://tequila.epfl.ch/>

<https://trello.com/>

<https://fr.wikipedia.org/wiki/Git>

<http://blog.pilotsystems.net/2012/septembre/les-api-rest>

## Compte-rendu des discussions

Discussions et échanges avec le formateurs et experts

Discussion initiale :

Rapport 2 fois / semaine

Réunion 2 : 26.04.2019 :

* Ajouter MCD
* Tests
* Relire les critères TPI
* Résumé en annexe

Discussion 26.04 dubois :

* Afficher plus de candidatures par défaut
* Trier par date par défaut

## Journal de travail

## Code source

L’entier du code source se trouve sur le repository c4science disponible ici :

<https://c4science.ch/source/canapgest/>

Pour disposer du code source en local, il suffit d’exécuter la commande suivante :

* *git clone* [*https://c4science.ch/source/canapgest.git*](https://c4science.ch/source/canapgest.git)

*+ script génération DB*

# Glossaire

A

API

application programming interface, interface permettant de fournir des services à d'autres applications 4, 5

C

Canap-Gest

Nom donné au projet, abréviation de "Candidature Apprentissage - Gestion" 2

CLI

Command-line interface 13

compte guest 2

E

ECMAScript

Normes JavaScript en matière de code 13

J

JSON

Format de données dérivé de la syntaxe des object JavaScript 2, 4

JWT

JSON Web Token 12

N

NPM

Node.js Packet Manager 12

R

RGPD

Règlement général sur la protection des données 4

T

transcompilateur

Converti un code source d'un language vers un autre 13

V

VCS

Version Control System 13

Git

Rest

Framework

<https://www.edoeb.admin.ch/edoeb/fr/home/documentation/bases-legales/Datenschutz%20-%20International/DSGVO.html>

1. https://fr.vuejs.org/v2/style-guide/index.html [↑](#footnote-ref-1)
2. Voir point 3.6.4 [↑](#footnote-ref-2)
3. <https://standardjs.com/> [↑](#footnote-ref-3)
4. <https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest> [↑](#footnote-ref-4)