PRO3600

Livrable 3

Gamification du module informatique

Ce document contient :
- le contexte du produit ;
- une analyse des besoins ;
- un cahier des charges ;
- une spécification fonctionnelle ;
- la conception préliminaire.
- la conception détaillée ;
- le bilan technique ;
- le bilan de la gestion du projet ;
- un manuel utilisateur ;
- un manuel d'exploitation des tests ;

Membres du groupe	Apport au document
MAZEAU Samson	Création, rédaction et relecture
NADAL SANTA Lorenzo	Rédaction et mise en page
OULD-AKLOUCHE Juliette	Rédaction
GENT Thibaut	
LARISSON Anthony	
TRAHAY François (*tuteur*)	Révision

Table des matières

1 Introduction

Le présent document est le dernier livrable du module PRO3600, il a été précédé de deux autres livrables (le prototype et le pré-rapport) et sera défendu par une soutenance. Ce document produit sur LibreOffice contient un cahier des charges, la spécification fonctionnelle, une conception préliminaire de l'application, la conception détaillée, un bilan technique, un bilan sur la gestion du projet et le manuel utilisateur.

Ce livrable qui fait office de rapport final a pour vocation de dresser une analyse globale du projet. Délivré avec ce document, un logiciel et un lot de tests dont l'exploitation est décrite dans les parties correspondantes.

2 Cahier des charges

Afin de répondre efficacement à la demande de notre client : le département informatique des écoles IMT-BS et Telecom SudParis représenté par François TRAHAY (tuteur du groupe), nous présenterons dans cette partie le contexte de la demande, la description et l'analyse des besoins, les différents produits du projet ainsi que leur fonction, les critères d'acceptabilité et de réception, les contraintes de délais et techniques.

2.1 Contexte de la demande

L'ensemble des ressources des enseignements des écoles sont disponibles sur la plateforme Moodle. Cette dernière permet aux élèves de déposer leurs devoirs que les professeurs noteront ultérieurement, télécharger les éléments de cours, les fascicules d'exercices ou toutes ressources mises à disposition par l'enseignant.

En général, le format des séances de cours d'un module est le suivant :

- Un cours sur une notion spécifique (ex : Les tables de hachages en Java) ;
- L'application des connaissances en séance de travaux pratiques.

Souhaitant moderniser et appuyer l'enseignement sur une plateforme plus adaptée aux besoins de l'apprentissage des sciences informatiques le client désire acquérir une application web qui se positionnera en support au format de cours et encouragera la pratique du code aux étudiants.

2.2 Description de la demande

2.2.1 Besoins fonctionnels

L'application du client doit remplir les objectifs suivants :

- La « gamification » de l'enseignement en ajoutant le gain de points d'expérience par la validation des projets ;
- La mise en place d'une « roadmap » du module ;
- La correction automatique des exercices des élèves ;
- La visualisation des données concernant les élèves disponible pour le corps professoral ;
- L'ajout/suppression des projets/correction par l'enseignant;

2.2.2 Besoins techniques

Le client n'ayant pas spécifié de besoin technique, les différents choix sont à la charge de l'équipe qui réalisera le projet. Les contraintes techniques seront présentées et justifiées dans la partie : .

2.2.3 Échelons

Pour répondre à la demande formulée par le client, l'équipe projet a rendu trois livrables à savoir:

- le pré-rapport (14 février);
- le prototype (16 mars);
- le présent document qui fait office de rapport (18 mai);

Également, une preuve de la réalisation sera démontrée en séance lors de la soutenance le 2 juin 2020.

2.3 Contraintes

2.3.1 Contraintes de délais

Le projet représente 50h de travail par membre de l'équipe, soit 250 heures dont 10 % (25 heures) seront attribuées à la gestion de projet. Le projet prendra fin lors de la soutenance du 2 juin 2020.

2.3.2 Contraintes techniques

Conformément à la partie 2.2.2 Besoins techniques, le projet doit seulement répondre aux contraintes techniques de réalisation qu'un projet de cette envergure nécessite.

3 Spécification fonctionnelle

Afin de répondre à la demande du client, le groupe-projet (ci-après dénommé l'équipe) devra réaliser complètement la plateforme web qui sera constituée des éléments suivants :

- **une partie client** (front-end);
- **une partie serveur** (back-end);
- **une API Rest** ;

3.1 Client

Le visuel de l'application est à charge de l'équipe qui devra respecter les contraintes suivantes :

- l'application devra être ergonomique ;
- l'application devra s'adapter aux différents appareils qui s'y connectent ;
- l'application devra être accessible (au sens des règles W3C).

3.2 Utilisateurs

Il existe différents types d'utilisateur :

- le visiteur :
    - il aura seulement accès à l'enregistrement ou l'authentification ;
- l'élève :
    - il disposera des droits de lecture et écriture sur ses propres données ;
    - il disposera des droits de lecture des données ouvertes par le professeur ;
- le professeur
    - il disposera des droits de lecture des données de ses élèves ;
    - il pourra ajouter/supprimer des projets au cursus ;
    - il pourra définir l'accès à certaine donnée de sa classe.

L'utilisateur devra pouvoir s'enregistrer et s'authentifier. L'enregistrement sera possible seulement pour les adresses courriel pourvu d'un nom de domaine provenant des écoles.

3.3 Menu

Un menu sera présent sur la page des utilisateurs authentifiés. Il sera composé des rubriques suivantes:

- Accueil ;
- Roadmap ;

- Informations personnelles;

3.4 Rubrique : Accueil

Cette rubrique est composée des éléments suivants:

- une photo de l'élève ;
- la barre de progression de l'élève ;
- ses projets/exercices en cours ;
- les données ouvertes par le professeur sur la classe ;
- ses derniers projets validés ;

3.5 Rubrique : Roadmap

Cette rubrique contient l'arborescence des projets. On distingue par couleur sur l'arborescence 3 types de projets :

- les projets validés en vert ;
- les projets non accessibles grisés ;
- les projets en cours ou non validés sans couleur ;

l'utilisateur peut naviguer sur la plateforme en cliquant sur les projets pour consulter leurs informations ;

L'utilisateur spécifique « professeur » a accès à des fonctionnalités d'ajout, de modification ou de suppression de projet. Il peut également modifier les droits de lecture des projets pour les élèves.

3.6 Rubrique : Informations personnelles

Cette rubrique permet aux utilisateurs de modifier/consulter leurs informations personnelles comme :

- l'adresse courriel ;
- le mot de passe ;
- une photo ;
- l'année de promotion ;
- la date de naissance ;
- des informations complémentaires.

3.7 L'élément « projet »

Un « projet » est entièrement défini par :

- un intitulé
- un fichier de correction ;
- un fichier PDF descriptif des attentes du projet

La page associée à un projet fournit l'ensemble de ces données. Un professeur peut également joindre

- les notions pratiquées ;
- des ressources pédagogiques ;
- une courte description des attentes;

Sur la page projet les élèves ont accès aux différentes données telles que le pourcentage de réussite, la moyenne etc.

4 Conception préliminaire

Nous avons choisi de découper l'application en deux parties :

- Le client (front-end)
- Le serveur (back-end)

Ce choix de découpage va nous permettre de travailler en parallèle sur les deux parties.

4.1 Technologies

Lors de la conception préliminaire nous avions fait le choix d'utiliser les technologies suivantes :

Pour la partie client :

- HTML5|CSS3, Javascript ;
- Frameworks : Bootstrap et Angular2+;

Bootstrap nous permettra de développer la partie client de manière responsive. De plus Angular2+ est un Framework JS libre développé par Google soutenu par la communauté open source.

Pour la partie serveur :

- PHP, SQL ;
- Frameworks : Symfony, MySQL ou PostgreSQL ;

De même que précédemment, l'utilisation de ces technologies est répandue, ce qui permet l'accès à une documentation dense.

Cependant au fil du projet nous avons été amené à utiliser node.js et le framework express plutôt que php car il était plus pertinent pour nous de mieux approfondir javascript plutôt que d'alourdir notre socle minimum de connaissance pour démarrer la conception du projet.

4.2 Front-end

Nous avons fait le choix de développer une application mono-page (Single-Page-Application, SPA). Elle sera composée d'un menu fixe (et responsive) dont les rubriques seront : *Accueil*, *Roadmap* et *Informations personnelles*. Une SPA permet de ne pas rafraîchir tout la page pour mettre à jour les données mais seulement les parties concernées.

L'architecture d'une application Angular est composée entre autres de :

- *Module* : Déclaration qui permet de regrouper les différents composants de l'application ;
- *Components*: Composants de l'application;
- *Services and dependency injection*: Ensemble de méthodes qui répondent aux besoins fonctionnels de l'application ;

Nous avons divisé la partie client en deux parties :

- *une partie publique* : ouverte à tous les visiteurs ;
- *une partie privée* : ouverte aux utilisateurs inscrits et authentifiés;

Voici une ébauche de l'arborescence de la partie client :

```
app/
|- app.module.ts
|- app-routing.module.ts
|- public/
        |- inscription/
                |- inscription.module.ts
                |- inscription.service.ts
                |- inscription.html
                |- style/
                        inscription.css
        |- auth/
                |- auth.module.ts
                |- auth.service.ts
                |- auth.html
                |- style/
                        |- auth.css
                        |- src/
                |- index.ts
        |- about/
                |- about.module.ts
                |- about.service.ts
                |- about.html
                |- style/
                        |- about.css
|- protected/
        |- menu/
                |- menu.module.ts
                |- style/
                        |- menu.css
                |- menu.html
                |- index.ts
        |- dashboard
                |-
                ...
        |- roadmap
                |-
                ...
        |- project
                |-
                ...
        |- settings
```

La qualité modulaire d'Angular nous permet d'imaginer la partie client sous différents modules comme ceux présentés au dessus.

Nous pourrons également dans un second temps développer une partie administration pour les utilisateurs qui seraient en charge de l'application.

Voici une illustration des relations entre les différents éléments du framework Angular :

4.3 Back-end

Afin de faire le liant entre la partie client et la partie serveur nous allons développer une API Restful qui permettra d'accéder aux données par la méthode HTTP.
L'architecture du back-end reposera sur le même développement modulaire que le front-end:

- un module d'inscription;
- un module d'authentification;
- un module qui traite les projets;
- un module de correction;
- un module qui offre l'accès aux données de l'utilisateur ;

4.4 Le prototype

Comme décrit dans la partie 2.3 Échelons, nous avons proposé un prototype de notre application le 16 mars. Le prototype répondait bien à son cahier des charges (fixe avec notre tuteur) à savoir:

- Une application décomposée en trois parties (front-end, back-end et base de données);
- Les trois composants échangent les données relatives aux fonctionnalités;
- L'application client permet à un utilisateur de s'enregistrer, de se connecter et de s'inscrire à un projet.

Le prototype découlant des choix que nous avions effectués lors de la rédaction du pré-rapport, il nous semble pertinent aujourd'hui de l'intégrer à la partie conception préliminaire car il est une version préliminaire du logiciel final.

Le prototype nous a permis de mettre en évidence trois points majeurs bloquants:

- la difficulté de développer une application qui regroupe trois composants sur des environnements différents (dans notre cas nous développons simultanément sur Ubuntu, Windows et MacOS);
- Les bugs liés aux différences de versions entre les composants selon le système d'exploitation (exemple MySQL server ne fonctionne pas pareil sur Windows ou sur Ubuntu où il y a un problème de connexion entre MySQL server et MySQL workbench);
- La contrainte de devoir ouvrir deux terminals (ou terminaux ?), exécuter les commandes propres au lancement des parties, devoir importer manuellement les données dans MySQL workbench.

Nous en avons conclu que pour rendre un logiciel utilisable et parfaitement exploitable dans les conditions où nous l'avons développé il nous fallait isoler les différentes parties de l'environnement hôte et rendre plus automatisé l'exécution de notre livrable. C'est pourquoi nous

avons utilisé Docker qui fait l'objet de la sous partie suivante et la partie 5 conception détaillée "Dockerisation" du projet.

4.5 Docker

Docker est une technologie qui permet de créer des containers pour empaqueter une application et ses dépendances dans un conteneur isolé et qui peut être exécuté sur n'importe quel serveur.

Nous avons fait le choix d'utiliser Docker suite à l'analyse du prototype mais également pour satisfaire au mieux les contraintes qu'engendre le module de correction c'est à dire:

- récupération d'un script dont on ne connaît pas le contenu;
- exécution de ce script;
- restitution des informations qui donnent suite à l'exécution.

L'exécution d'un script dont on ne sait pas s'il est malveillant ne peut pas se faire dans le même environnement que le serveur back-end pour des raisons de sécurité. Nous avons donc choisi de créer une image Docker pour chaque fichier reçu, d'exécuter le fichier dans le container (donc isolé des autres environnements), de récupérer l'output puis de supprimer le container, son image, et le fichier. C'était le meilleur moyen avec nos connaissances de répondre à plusieurs problèmes en utilisant un minimum de technologie.

Vous trouverez plus d'information sur notre utilisation de Docker dans la partie suivante et également plus d'information concernant le module de correction dans la partie conception détaillée du back-end.

5 Conception détaillée - “Dockerisation”

Nous avons choisi d’utiliser cette technologie aux deux tiers du module pour plusieurs raisons qui ont été détaillées dans les parties précédentes.
Cette partie est divisée en trois parties. Dans la première nous évoquerons l’architecture globale de notre application et comment la technologie Docker est venue s'insérer dans notre projet. Les deux parties suivantes traiteront le Dockerfile et de Docker-compose de notre application.

5.1 Architecture globale

Comme évoqué dans les parties précédentes, notre application est divisée en trois parties distinctes comme l’illustre l’architecture du livrable final:

```
.
├── front-end
├── back-end
├── database
└── docker-compose.yml
3 directories, 1 file
```

Chacune de ces parties interagit différemment avec les deux autres composants selon le système d’exploitation et sa version. Nous avons choisi d’utiliser Docker pour pouvoir isoler nos composants du système hôte et ainsi rendre l’interaction dépendante seulement des configurations que nous avions établis dans les Dockerfile (fichiers qui définissent les règles de construction de l’image du container).

Nous avons donc créé pour chacune des parties de notre application une image propre avec seulement ses dépendances et des règles spécifiques selon le composant. Ainsi chaque répertoire (front-end, back-end et database) contient un fichier “Dockerfile” qui est exploité par le docker-compose qui coordonne le tout selon les règles qui y sont définies.

Nous étudierons un Dockerfile dans la partie suivante et enfin le docker-compose dans celle qui suivra.

5.2 Dockerfile - exemple front-end

Nous allons commenter le Dockerfile que nous avons crée pour la partie “front-end”.

```
### STAGE 1: Build ###
FROM node:10.20.1-alpine3.11
WORKDIR /app

# install and cache app dependencies
COPY package.json /app/package.json
RUN npm install
RUN npm install -g @angular/cli
COPY . .

EXPOSE 4200
CMD ["ng","serve","--host", "0.0.0.0"]
```

Le *Dockerfile* se découpe en deux étapes :

- première étape est la construction de l’image:
    - le mot clé *“FROM” p*ermet de définir l’image de base sur laquelle notre application va fonctionner. Ici nous avons choisi node.10.20.1-alpine3.11 car node 10.20.1 est une version stable que nous avons également pu réutiliser pour l’image du back-end. De plus Alpine Linux est une distribution Linux ultra-légère ce qui permet d’avoir une taille d’image correcte avec un minimum de fonctionnalité basique.
    - Le mot clé *WORKDIR* permet de définir le répertoire ou nous allons lancer les commandes suivantes. Il est tout à fait possible de changer de WORKDIR dans un même dockerfile selon le besoin. Ce n’a pas été le cas dans notre projet.
- seconde étape qui illustre l’installation des dépendances:
    - le mot clé *“COPY”* qui prélève les packages.json qui définissent l’ensemble des librairies que le projet utilise;
    - la commande *RUN* npm install permet d’installer ces librairies en se basant uniquement sur ce qui est décrit dans le package.json;
    - *COPY . .* permet de copier les fichiers dans le répertoire du Dockerfile a l'exception de ceux stipulés dans .dockerignore (nous y reviendrons);
    - *EXPOSE* permet de définir le port sur lequel va fonctionner notre image;
    - *CMD* définit la commande qui lance l’application;

L’utilisation d’un *.dockerignore* est importante car cela permet de copier les fichiers comme le répertoire *node_modules* (qui contient l’ensemble des dépendances) et d’autre fichier inutiles pour l’image comme les fichiers *.git*.

5.3 Docker-compose

Dans cette partie nous allons commenter le docker-compose qui permet de coordonner le lancement des trois containers.

```
version: '3'
services:
   db:
      build: ./database
   back-end:
      build: ./back-end
      ports:
         - "3000:3000"
      volumes:
         - /var/run/docker.sock:/var/run/docker.sock
      depends_on:
         - db
      restart: on-failure
   front-end:
      build: ./authservice
      ports:
         - "4200:4200"
      depends_on:
         - back-end
```

Les arguments importants de notre docker-compose sont:

- L'attribut *"depends_on"* permet de définir une dépendance entre deux containers. Dans les règles du service Back-end nous observons l'option restart: on-failure qui permet au container du back-end de se relancer s'il n'arrive pas à se connecter a la base de données (ce qui arrive systématiquement le temps que le base de données s'initialise);
- L'attribut *"volumes: - /var/run/docker.sock:/var/run/docker.sock"* permet au container Back-end d'avoir accès au Docker Daemon de l'hôte. Cela permet au container Back-end de lancer des containers en parallèle de l'application au même niveau que lui. Cette méthode s'appelle Docker-out-of-Docker (Mot clé: DooD).

6 Conception détaillée - côté client

Nous avons voulu réaliser l'application la plus simple et la plus esthétique possible afin que l'utilisation soit des plus agréables. Nous allons, dans cette partie, la décrire succinctement et avec précision.

6.1 Technologies

Nous avons travaillé tout du long du projet avec le framework *Angular*.
Néanmoins, nous avons nourri le projet de nombreuses librairies et technologies compatibles avec ce framework.

Pour l'esthétisme de notre application nous avons choisi d'utiliser *Nebular.* C'est une librairie facile d'utilisation et qui décore notre application de manière harmonieuse. Cependant, nous verrons plus loin que son utilisation fut incompatible avec d'autres librairies telle que *D3.js.*

Les graphiques des projets et le diagramme en étoile dans la partie toggle ont été réalisé avec *Chart.js*. C'est une librairie qui comporte de nombreuses méthodes qui nous ont permis de lier, efficacement et simplement, les variables des différents graphiques avec les données du back-end. Par exemple nous avions eu du mal à lier les données du taux de réussite de chaque projet aux coordonnées du graphique que vous pouvez voir dans l'onglet "voir le projet " de Roadmap. Car les attributions étaient asynchrones. Ce qui nous a valu quelques surprises.

Nous avons voulu utiliser *D3.js* afin de réaliser une roadmap interactive, "zoomable", afin que l'utilisateur puisse chercher les projets qui l'intéressent en fonction du module et du niveau de difficulté. Mais la formation à cet outil était trop fastidieuse et complexe. De plus nous avons voulu intégrer quelques exemples simples dans notre application mais *D3.js* est incompatible avec *Nebular*. Les deux composants entrent en conflit lors de leur affichage simultané. L'un s'affiche au dessus et masque l'autre. Nous avons donc abandonné cette alternative et nous avons opté pour un composant *Nebular* pour créer la Roadmap intéractive.

6.2 Architecture de l'application

Dans cette partie nous allons évoquer dans un premier temps la structure de l'application et son évolution depuis le prototype. Nous illustrerons ensuite les évolutions de l'interface graphique.

Enfin nous conclurons sur l'étude de l'architecture des composants Angular et des liens entre les différents documents qu'un composant nécessite.

6.2.1 Architecture du front-end

Voici la structure du prototype dans le répertoire ./src/app

```
.
├── app-routing.module.ts
├── app.component.css
├── app.component.html
├── app.component.spec.ts
├── app.component.ts
├── app.module.ts
├── auth.guard.spec.ts
├── auth.guard.ts
├── data-user
│   ├── data-user.component.css
│   ├── data-user.component.html
│   ├── data-user.component.spec.ts
│   ├── data-user.component.ts
│   ├── projet.spec.ts
│   └── projet.ts
├── login
│   ├── login.component.css
│   ├── login.component.html
│   ├── login.component.spec.ts
│   └── login.component.ts
├── login.guard.spec.ts
├── login.guard.ts
├── register
│   ├── register.component.css
│   ├── register.component.html
│   ├── register.component.spec.ts
│   └── register.component.ts
├── roadmap
│   ├── modal
│   ├── projet
│   ├── roadmap.component.css
│   ├── roadmap.component.html
│   ├── roadmap.component.spec.ts
│   └── roadmap.component.ts
├── services
│   ├── auth.service.spec.ts
│   ├── auth.service.ts
│   ├── projet.service.spec.ts
│   ├── projet.service.ts
│   ├── token-interceptor.service.spec.ts
│   └── token-interceptor.service.ts
└── settings
    ├── settings.component.css
    ├── settings.component.html
    ├── settings.component.spec.ts
    └── settings.component.ts
8 directories, 38 files
```

Les principaux documents sont :

- les différents répertoires composants (roadmap, settings, login, data-user);
- le répertoire services qui contient l'ensemble des appels vers l'API du back-end;
- les fichiers de types "guards".

Nous avons constaté que cette organisation n'allait pas nous permettre d'évoluer sur le long terme et nous avons choisi de modifier la structure afin de séparer distinctement les composants disponibles lorsque l'utilisateur est connecté de ceux disponibles lorsqu'il s'agit de visiteur non identifié et de créer des répertoires utiles comme interface et guards qui regroupe les fichiers correspondants comme l'illustre l'architecture de l'application client délivrée :

```
├── app-routing.module.ts
├── app.component.css
├── app.component.html
├── app.component.spec.ts
├── app.component.ts
├── app.module.ts
├── application
│   ├── application.component.css
│   ├── application.component.html
│   ├── application.component.spec.ts
│   ├── application.component.ts
│   ├── data-user
│   ├── home
│   ├── projets
│   ├── roadmap
│   └── settings
├── auth
│   ├── auth.component.css
│   ├── auth.component.html
│   ├── auth.component.spec.ts
│   ├── auth.component.ts
│   ├── login
│   └── register
├── guards
│   ├── auth.guard.spec.ts
│   ├── auth.guard.ts
│   ├── login.guard.spec.ts
│   └── login.guard.ts
├── interfaces
│   ├── historique.ts
│   ├── projet.ts
│   └── user.ts
├── not-found
│   ├── not-found.component.css
│   ├── not-found.component.html
│   ├── not-found.component.spec.ts
│   └── not-found.component.ts
└── services
    ├── auth.service.spec.ts
    ├── auth.service.ts
    ├── projet.service.spec.ts
    ├── projet.service.ts
    ├── settings.service.spec.ts
    ├── settings.service.ts
    ├── token-interceptor.service.spec.ts
    └── token-interceptor.service.ts
13 directories, 33 files
```

Nous avons donc créé deux composants auth et application afin de regrouper les composants existants en sous composants de ses composants globaux. Cette architecture nous permet donc de mettre à disposition les composants selon l'utilisateur (authentifié ou non) comme le montre l'extrait du fichier app.component.html

```
<nb-layout *ngIf="!_authService.loggedIn()" center>
  <nb-layout-column>
    <app-auth>
      </app-auth>
  </nb-layout-column>
  <nb-layout-footer>A student projet from Telecom SudParis</nb-layout-footer>
</nb-layout>
<app-application *ngIf="_authService.loggedIn()"  ></app-application>
```

Les conditions sur l'opérateur *ngIf permettent dans un cas de générer le composant app-auth dans l'autre de générer l'application.

6.2.2 Interface graphique

Notre objectif était de segmenter notre application autour de 3 rubriques essentielles. La roadmap où tous les projets sont rassemblés. Les settings où l'utilisateur peut changer son mot de passe et sa photo de profil. Enfin la page d'accueil qui recueille toutes les informations essentielles sur les projets et l'utilisateur.
Voici des captures d'écran de nos trois composants qui façonnent notre application :

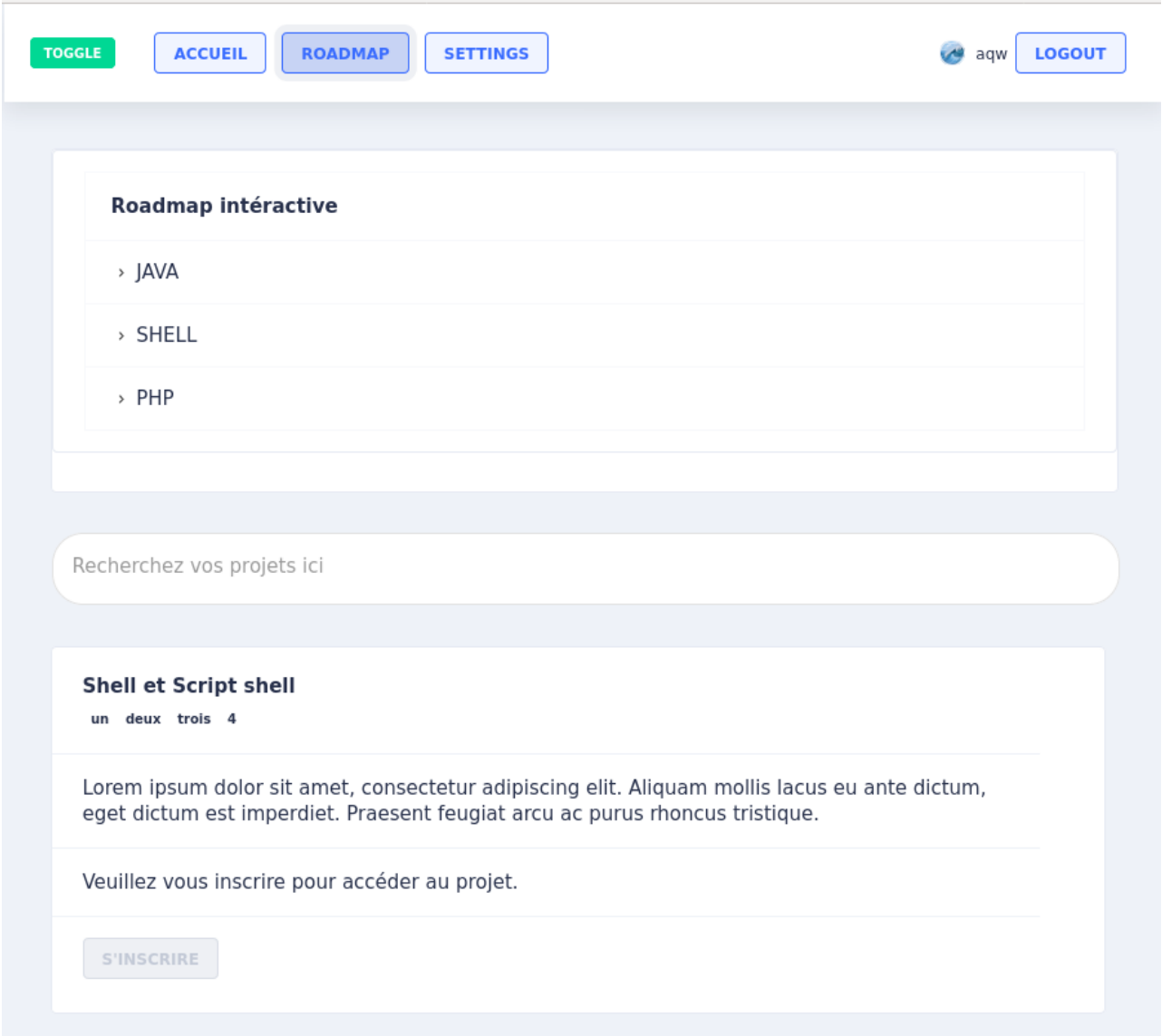


Figure 1: Aspect visuel de la roadmap

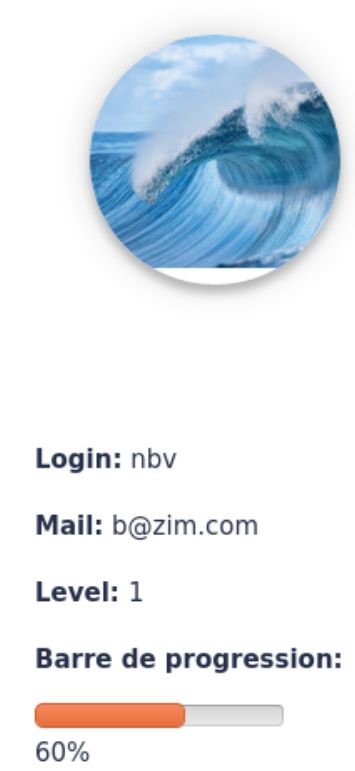


Figure 2: Aspect visuel du composant "data-user"

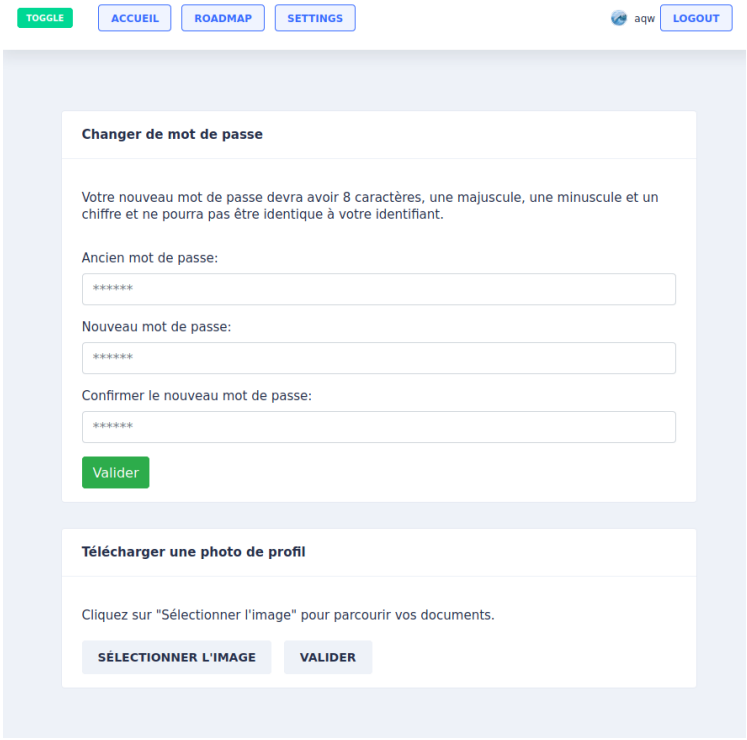


Figure 3: Aspect visuel du composant "settings"

6.3 Architecture des composants

Nous allons détailler ici l'architecture interne d'un composant Angular.
Lorsque l'on entre dans le terminal la commande suivante:
ng generate component [nom du composant],
Angular créer un composant qui comporte par défaut quatre fichiers qui ont chacun une utilité propre :

- .CSS:
  C'est le module qui est dédié comme son nom l'indique au langage CSS.

- .HTML:

C'est dans ce module que le langage html se situe. La propriété de "binding" permet de lier des données présentes dans .TS pour les afficher directement dans le fichier html.

- .TS.SPEC:

Angular a prévu un fichier où des tests peuvent être réalisés. Au cours du projet nous n'avons jamais utilisé ce fichier car il n'est pas nécessaire lorsqu'on code une application Web.

- .TS:
  C'est le fichier principal du composant. Ce composant se suffit à lui-même pour générer la page web. Il peut (potentiellement) contenir le css et le html du composant. C'est ici que sont déclarées et importées les fonctions écrites en TypeScript. Les modules utiles pour le composant sont importés au début du fichier:

  @Component possède trois arguments qui référence le composant parmi tous les composants existants:
  - selector
    permet de nommer le dossier html du composant. C'est grâce à cet argument que le dossier html du composant peut être implanté dans d'autres fichiers html d'autres composants.

  - templateUrl
    Cet attribut peut contenir:
    - Tout le contenu du fichier html (mais son écriture peut être lourde et rapidement illisible)

- Le lien vers le fichier html du composant.

- styleUrls

  Cet attribut peut contenir :

  - Tout le contenu du fichier css (mais son écriture peut être lourde et rapidement illisible)
  - Le lien vers le fichier css du composant.

Chaque fichier .TS exporte une classe qui détient la même syntaxe qu'une classe JavaScript qui contient une déclaration des variables utiles pour le composant, un constructeur et des déclarations de fonctions dont la syntaxe diffère légèrement d'un fichier JavaScript.

6.4 Authentification

Dans cette sous partie nous allons traiter le cas de l'authentification dans le front-end qui est en lien avec les parties 7.4 authentification dans le back-end.

6.4.1 Les formulaires

Les composants Login et Register permettent à l'utilisateur de s'enregistrer et de se connecter par le biais d'un formulaire L'utilisateur entre son nom d'utilisateur, puis son adresse mail enfin son mot de passe avec une demande de confirmation de mot de passe.
Lorsque le back-end confirme que l'utilisateur est bien un utilisateur enregistré dans la base de données, le "body" de la réponse contient le jeton créé par le back-end.
Nous stockons ce jeton dans le local storage afin de garantir l'authenticité tout au long de la navigation de l'utilisateur.

6.4.2 Token-interceptor service

Afin d'assurer l'authenticité de l'utilisateur lorsque ce dernier fait des requêtes vers le back-end nous avons mis en place un service qui intercepte les requêtes http et qui insère dans le header de la requête le token stocké dans le Local Storage comme suit l'extrait de code provenant de *../services/ token-interceptor.service.ts ligne 12*.

```
intercept(req, next)
  {
    let authService = this.injector.get(AuthService);
    let tokenizedReq = req.clone(
     {
       setHeaders:
       {
         Authorization: `Bearer ${authService.getToken()}`
       }
     });
```

```
    return next.handle(tokenizedReq);
}
```

Cette fonction permet donc au back-end de pouvoir assurer si le jeton dans le header est bien authentique et d'en conclure donc sur les données à délivrer.

6.5 Principales fonctionnalités

Nous allons dans cette partie détailler les principales fonctionnalités des composants que nous avons créés dans notre application.

6.5.1 Roadmap

L'onglet Roadmap contient l'ensemble des projets disponibles et une vue d'ensemble de ces projets dans une Roadmap intéractive.

6.5.2 Roadmap interactive

La roadmap interactive en haut de page rassemble tous les projets par module et par niveau de difficulté. Le niveau "novice" est propre aux utilisateurs ayant un niveau compris entre 1 à 2. Le niveau "intermédiaire" est propre aux utilisateurs ayant un niveau compris entre 2 et 3. Le niveau "Avancé" est propre aux utilisateurs ayant un niveau compris entre 3 et plus. Le plus de cette roadmap est la recherche rapide d'un projet en fonction de son module et de son niveau de difficulté. L'utilisateur entre ensuite le nom du projet dans la barre de recherche qui se trouve en dessous.
Malheureusement, le temps nous manquait pour traiter les résultats de la fonction *tri_lvrequis*. Dans sa forme actuelle la roadmap est codée en "dur", ce qui n'est pas le résultat attendu.
La fonction:

```
tri_lvrequis(moduleId : number, lvRequis : number, projets : Projet[]) : Projet[] {
    var tab : Projet[] = [];

    for(var proj of projets) {
      if (proj.lvlrequis === lvRequis && proj.moduleId === moduleId){
          tab.push(proj);
      }
    }

    return tab;
  };
```

permet de classer les projets par module et par difficultés, ce qui simplifie l'affichage.

Cependant, pour que la roadmap soit réellement interactive, il aurait fallu créer une fonction dans le sous composant *"tree.ts"* qui aurait permis de lier les données du back-end avec la roadmap intéractive. Car le squelette de la roadmap interactive contient 3 matrices (Une pour chaque module), dans chaque matrice il y a 3 autres sous-matrices (une pour chaque niveau de difficulté), dans chaque sous-matrice il s'y trouve 1 sous-sous-matrice qui contient les noms des projets concernés. Il aurait donc fallu créer une fonction qui initialise une matrice triple et insère au fur et à mesure les nom des projets grâce à la fonction *tri_lvrequis*. On aurait sûrement "découper" cette fonction en 3 sous-fonctions qui auraient eu pour but de créer un seul type de matrice à la fois. Mais le manque de temps ne nous a pas permis de le réaliser.

Malgré cet échec il nous a semblé pertinent de laisser cette roadmap telle qu'elle est pour recevoir des remarques ou pouvoir discuter de la façon dont on aurait pu la coder afin qu'elle soit réellement interactive.

6.5.2.1 La barre de recherche

Des tags sont associés à chacun des projets pour faciliter la recherche en fonction des points d'intérêts de chaque utilisateur. Nous avons choisi de limiter leur nombre à quatre pour éviter de surcharger la page d'informations.

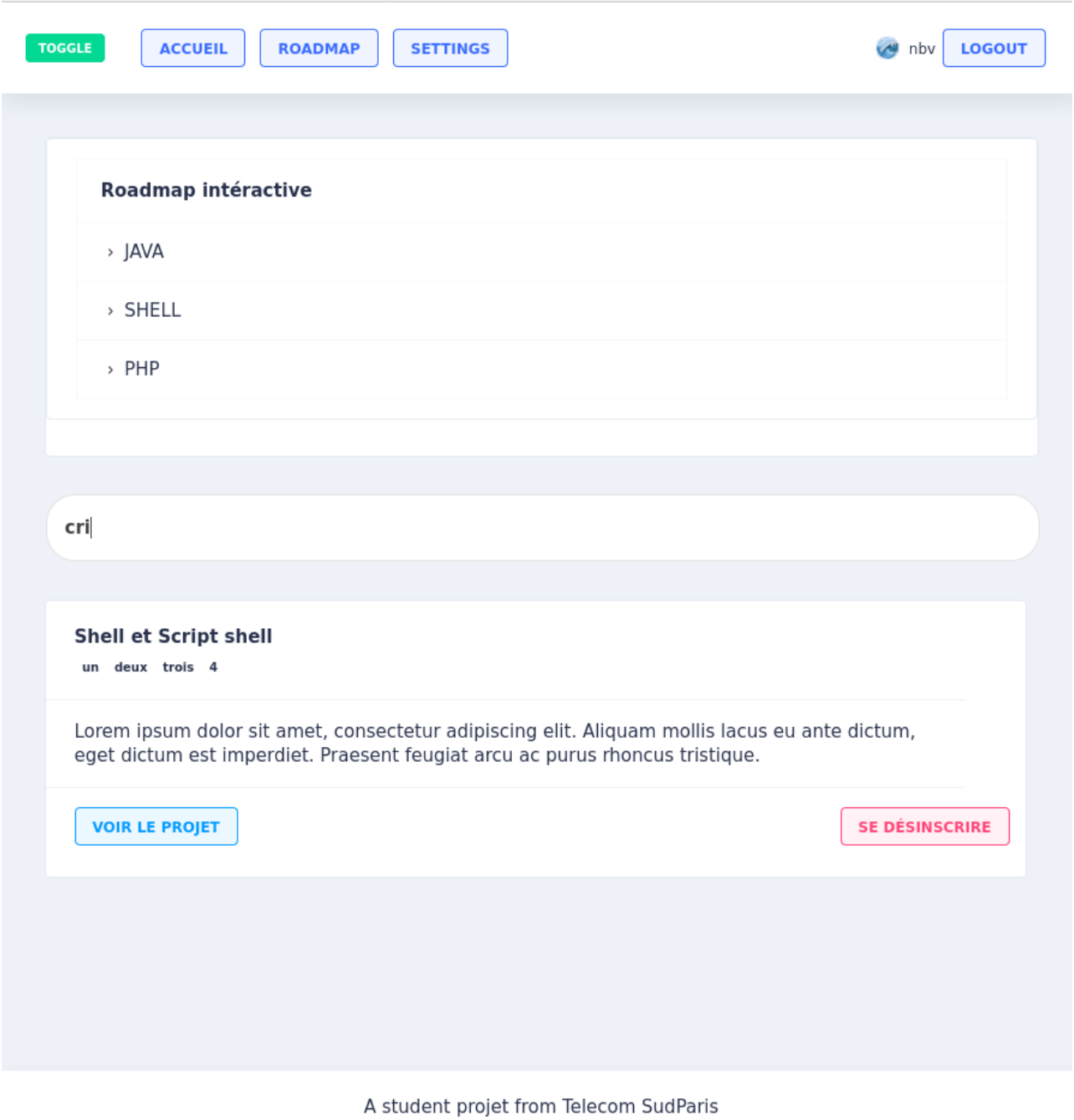


Figure 4: Exemple d'utilisation de la barre de recherche

Comme vous le voyez dans la capture d'écran ci-dessus, il est possible de saisir du texte dans la barre de recherche qui est directement liée au nom des projets et leurs tags.

Un module Angular offre cette fonctionnalité peut être importé comme suit :

```
import { Ng2SearchPipeModule } from 'ng2-search-filter';
```

Il est appelé et directement utilisé dans le code HTML comme le montre l'extrait du fichier *roadmap.component.html:*

```
<div class="ok" *ngFor="let projet of projetS | filter : searchText" >
```

ng2-search-filter contient une fonction *get_searchText(searchText)* qui prend en argument *searchText* (le string entré par l'utilisateur) et renvoie cet argument à la fonction compare(). La fonction compare() renvoie (dans notre cas) le projet dont son nom ou ses tags contiennent le string searchText. *compare()* utilise une fonction *Pipe()* qui compare la chaîne de caractère que l'utilisateur rentre dans la barre de recherche avec tous les attributs de string de l'objet *ProjetS* du composant *Roadmap.ts*. En l'occurrence ici, seulement les attributs *Tags* et *nom* (cf Partie 8.1, table PROJETS).

6.5.2.2 L'inscription au projet

Les boutons "s'inscrire" et "se désinscrire" sont directement liés au serveur.
Le bouton "s'inscrire" est directement lié à la fonction:

```
Register(){
      this._projetService.get_registered(this.projet.idprojets)
      .subscribe(
         res => {
            console.log(res);
            if (res.status = 200)
            {
               this.projet.isRegistered = true;
            }
         },
         err => console.log(err)
      )
   };
```

qui permet de mettre à jour le booléen *isRegistered* à la valeur "true" (dans le back-end) pour que l'étudiant puisse accéder au projet. Il y a ensuite un ajout de valeur dans la table

REGISTEREDUSER dans la base de données (cf partie 8.1) à savoir le nom de l'utilisateur, sa date d'inscription et sa future note lorsqu'il s'inscrit via ce bouton.
Le principe est le même pour le bouton "se désinscrire" mais il provoque l'effet inverse.

6.5.2.3 le composant "projets"

L'onglet "Voir le projet" dans chaque projet est une page indépendante rassemblant toutes les informations utiles sur un projet:

Une description générale du projet ;

- Le nom de l'encadrant en charge de ce projet (pour les éventuelles questions) ;
- La possibilité de déposer son projet afin d'être corrigé par le système de correction géré par le back-end. ;
- Un graĥique rassemblant le nombre de personne ayant obtenu un certain pourcentage de réussite (représentant la note) sur ce projet lors de leur première ou seconde tentative ;
- Enfin la note obtenue par l'élève sur ce projet ;

6.5.3 Guards

Le module Guards est présent dans tous les projets Angular. Ce module implémente une fonction *CanActivate()* qui est directement lié au routeur via le module *app-routing.module*. Dans *app-routing.module* l'objet *route* contient de nombreuses voies d'accès qui permettent de rediriger ou non l'utilisateur vers d'autres services: service de connexion, service d'authentification, à la roadmap, au settings,etc. Nous avons mis un exemple de voie ci-dessous:

```
{
        path: 'login',
        component: AuthComponent,
        canActivate: [LoginGuard]
  },
```

Cette voie permet l'accès au service de connexion de notre application. Le droit d'accès est donc donné par la fonction *CanActivate()*.
Dans notre projet nous avons souhaité utiliser ce composant Guard pour mettre des conditions sur l'accès à notre site. Différencier le cas où un utilisateur n'est pas authentifié, au cas où il est authentifié mais qu'il ne s'est pas encore connecté.

Notre composant Guards se décompose en deux sous-composants:

- login.guard
- auth.guard

Chacun de ses sous-composant implémente une fonction *CanActivate()* mais chacune d'elle retourne un accès différent.

6.5.3.1 Login guard

Ce composants implémente une fonction *CanActivate():*

```
canActivate(): boolean {
        if (this._authService.loggedIn())
           return (false);
        else
        {
           return (true);
        }
    }
```

qui retourne un booléen sur l'accès au service de connexion. Elle permet de rediriger un utilisateur déjà authentifié et connecté directement à la page d'accueil de notre site. A l'inverse, elle dirige l'utilisateur qui est authentifié mais qui ne s'est pas encore connecté au formulaire de connexion .

6.5.3.2 Auth guard

Ce composant implémente une fonction *CanActivate()*:

```
canActivate() : boolean {
        if (this._authService.loggedIn())
        {
           return true;
        }
        else
        {
           this._router.navigate(['login']);
           return false;
        }
     }
```

qui retourne un booléen sur l'accès au service d'authentification. Si l'utilisateur s'est déjà authentifié, ce guard permet de ne pas rediriger l'utilisateur vers l'authentification. A l'inverse, elle redirige l'utilisateur qui n'est pas encore authentifié vers le service d'authentification.

6.5.4 Autres fonctionnalités

6.5.4.1 Projets

Le composant *projets* dans Angular correspond à la page accessible via le bouton “voir le projet” dans la roadmap.
La principale fonctionnalité de cette page est le dépôt du fichier afin de le corriger (cf partie 7.3).

Cette page est spécifique à l’Id d’un projet combiné avec les informations de l’utilisateur propre à ce projet (note, nombre de tentative) (cf 8.1 table PROJETS). Elle affiche les informations propres à chaque projet.
La fonction:

```
onFileChanged(event) {
      this.selectedFile = event.target.files[0];
      var fullpath = this.selectedFile.name;
      var backslash = fullpath.lastIndexOf("\\");
      var filename = fullpath.substr(backslash + 1);

      var confirm_message = confirm("Files selected for import are \n Zip File: " + filename + "\n\nDo you want to proceed?");
}
```

permet de déposer un fichier et renvoie un message de confirmation lorsque le fichier à bien été déposé.

6.5.4.2 Settings

Dans l’onglet Settings, nous avons intégré une possibilité d’un changement de mot de passe par l’utilisateur et une “customization” de la photo de profil de son compte. Nous estimons ces deux outils indispensables pour une utilisation agréable et personnelle de notre application. De plus nous avons imposé pour le nouveau mot de passe la présence de certains caractères pour qu’il soit valide.

6.5.5 Page d’accueil

La page d'accueil correspond au composant *home* d’Angular.
Elle rassemble quelques données utiles à nos yeux pour concilier l’utile et l’agréable.

7 Conception détaillée - back-end

Lors de la rédaction du pré-rapport nous pensions utiliser php et le framework Symfony pour le développement du back-end mais comme évoqué dans la partie 4.1 Conception préliminaire - Technologies, nous avons préféré utiliser node.js qui est un framework javascript.

Dans cette partie nous évoquerons les différentes technologies qui sont utilisées lors du fonctionnement du back-end, nous évoquerons également les librairies sur lesquelles nous nous sommes appuyés. Nous montrerons l'évolution de l'architecture du back-end depuis le prototype vers sa version finale. Nous traiterons le cas particulier du module de correction qui est la fonctionnalité principale de notre application. Cette partie se conclura par l'ensemble des endpoints de l'API.

7.1 Technologies

Nous avons utilisé l'ensemble des technologies suivantes :

- *Node.js* (framework: express);
- *Docker* (pour le module de correction);
- *Shell* (Script pour la correction);

Node.js et le framework express sont le socle technologique de notre serveur back-end. L'utilisation de Javascript dans nos composants front-end et back-end fait sens car le dimensionnement du module (50 heures par étudiant) ne nous permettait pas de consacrer un surplus de temps à l'autoformation. Vous trouverez plus d'informations sur ces points en partie gestion de projet.

Nous traiterons l'utilisation de *Docker* et *Shell* en partie Module de correction.

Voici une liste des principales librairies qui nous ont permis de réaliser notre projet:

- *bcrypt* est une librairie qui utilise la fonction de hachage du même nom, nous l'avons utilisée pour chiffrer les mots de passe des utilisateurs avant de les stocker dans la base de données.
- *jsonwebtoken* est un standard ouvert qui permet l'échange de jetons entre plusieurs parties. Nous avons donc créé pour chaque utilisateur un jeton transmis à l'application client et stocké dans le local storage lors de la connexion.
- *shell.js* est une librairie qui permet d'exécuter des commandes shell;

- *Multer* qui nous permet d'enregistrer un fichier.

7.2 Architecture

L'architecture du prototype était la suivante:

```
.
├── README.md
├── connection_db.js
├── package-lock.json
├── package.json
├── photos
│   └── anonymous-user.png
├── routes
│   └── api.js
└── server.js
2 directories, 7 files
```

les principaux fichiers sont:

- *server.js* qui contient les directives de configuration concernant le serveur back-end;
- *connection_db.js* contient les directives de connexion à la base de données;
- *./routes/api.js* contient l'ensemble des fonctionnalités de l'application.

L'architecture finale du back-end :

```
├── Dockerfile
├── README.md
├── connection_db.js
├── models
│   └── user.js
├── node_modules
├── package-lock.json
├── package.json
├── routes
│   └── api.js
├── server.js
├── services
│   ├── ProjetService.js
│   ├── SettingsService.js
│   ├── UserService.js
│   ├── correction
│   └── tokenverify.js
└── src
    └── photos
245 directories, 12 files
```

Nous avons choisi de rendre l'application plus modulaire en transformant en profondeur l'application.

Pour cela nous avons :

- réduit *./routes/api.js* à contenir seulement les endpoints de l'API. c'est à dire presque l'ensemble de toutes les méthodes get, post, delete etc..
- créer un répertoire services qui contient toutes les fonctions de l'application séparées selon leur utilité en 5 fichiers:
    - *ProjetService.js* contient l'ensemble des fonctions liées au traitement de projet;
    - *SettingsService.js* contient les fonctions qui permettent à l'utilisateur de modifier ses données (mot de passe, photo de profil);
    - *UserService.js* contient les fonctions qui traitent les données de l'utilisateur (inscription, connexion, etc..);
    - *tokenverify.js* contient les fonctions qui assurent de l'authenticité de l'utilisateur;
    - *./correction/CorrectionService.js* contient toutes les fonctions liées à la correction d'un projet;
- Exporter les fonctions des services pour les rendre utilisables dans toute l'application;

Cela nous a permis de pouvoir tester des fonctions au cas par cas sans avoir à faire un appel à l'API et d'organiser le code en le rendant plus persistant.

7.3 Module de correction

le module de correction est la fonctionnalité phare du projet car c'est elle qui permet à un utilisateur de se faire corriger, d'attribuer des points et de lui permettre d'évoluer vers d'autres projets.

7.3.1 Architecture du module

Dans le répertoire ./services/correction nous avons les fichiers suivants:

```
├── CorrectionService.js
├── correct.sh
├── debian_container
│   └── dockerfile
├── src
├── test.sh
└── tmp
```

Voici une description des différents fichiers:

- *CorrectionService.js* contient les fonctions que le back-end utilisent pour mener à bien la correction;
- *debian_container/dockerfile* contient l'ensemble des directives permettant de construire des images sur la base debian pour chaque fichier qui doit être exécuté pour la correction;

- *correct.sh* est le fichier shell exécuter par le back-end qui permet de lancer le container Docker où va se dérouler la correction;
- *./tmp* est le répertoire où sont stockés les fichiers des utilisateurs lorsqu'ils sont reçus par le back-end;
- ./src contient tous les fichiers de corrections pour les projets.

7.3.2 Traitement par le serveur node.js

Lorsqu'un utilisateur souhaite envoyer son projet le front-end fait un appel à l'API vers l'url http://localhost/3000/projet/upload_projet/:id où id correspond à l'identifiant du projet dans la base de données comme le montre l'extrait de la partie recevant l'appel depuis le front-end (CorrectionService.js 55)

```
router.post('/upload_projet/:id', tkn.verifyToken, upload.single('myFile'), async (req, res) => {
    let id = req.params.id;
    let filename = req.nameuploadedfile;
    let data = await correction(id, filename, req.courriel);

    if(data === false)
    {
        res.status(409).send();
    }
    else
    {
        res.status(200).send(data);
    }
});
```

La requête va passer dans deux fonctions avant d'exécuter le bloc de code:

- *tkn.verifyToken* (*./services/tokenverify*) est la fonction qui vérifie l'authenticité de l'utilisateur;
- *upload.single* est une fonction de la libraire *multer* qui permet d'enregistrer le fichier reçu sous l'attribut 'MyFile" dans la requête;

la fonction utilise les mots clés async/await qui empêche le caractère asynchrone et qui permet d'attendre que la fonction correction(idprojet, filename, courriel) ait été exécutée.

```
function correction(idprojet, filename, courriel) {
    return new Promise(async resolve => {
        let old_note = await get_old_note(idprojet, courriel);
        let FileCorrection = await get_correction_file(idprojet);
        let pathRes = "./services/correction/tmp/" + FileCorrection + ".res";
        let iddepots = await get_id_depot(filename);
        let pathExec = './services/correction/correct.sh ' + filename + " " + FileCorrection + " > " + pathRes;

        if (iddepots === false) {
            console.log("iddepots non valide")
```

```
            resolve(false)
        }
        else if (FileCorrection === false) {
            console.log("Fichier de correction non valide")
            resolve(false)
        }
        else if (old_note === false) {
            console.log("erreur old_note")
            resolve(false)
        }
        else {
            pathExec = './services/correction/correct.sh ' + filename + " " + FileCorrection + " > " + pathRes;
            spawn('sh', [pathExec], {
                stdio:'ignore',
                detached: true,
                shell: true
            })
            setTimeout(process_result, 15000, pathRes, iddepots, old_note, courriel, idprojet)
            resolve(true);
        }
    });
}
```

cette fonction fait appel à d'autres fonctions comme:

- *get_old_note(idprojet, courriel);*
    - permet de récupérer la plus haute note obtenue;
- *get_correction_file(idprojet);*
    - permet d'obtenir le fichier de correction qui correspond au projet;
- *get_id_depot(filename);*
    - permet de récupérer l'id de la correction en cours;
- *process_result(pathRes, iddepots, old_note, courriel, idprojet);*
    - permet de traiter les données une fois que la correction a été effectuée.

La fonction correction récupère dans un premier temps toutes les données dont elle a besoin pour mener à son terme la correction. Ensuite elle exécute par le biais de spawn le fichier *./correct.sh* que nous étudierons dans la partie suivante. Le fichier *correct.sh* est éxecuté en mode détaché ce qui permet de ne pas attendre qu'il soit exécuté pour passer aux prochaines instructions.
Nous avons estimé que le temps nécessaire pour effectuer une correction était bien inférieur à 15 secondes, c'est pour cela que nous lançons par le biais de la fonction *setTimeout* la fonction *process_result()* 15 secondes après l'exécution de *correct.sh*. Nous avons fait ce choix pour permettre de ne pas monopoliser le serveur pendant le temps d'une correction (environ 8s).
Cela permet à l'utilisateur de continuer à utiliser l'application et aux composants back-end et front-end de communiquer durant le déroulement de l'application.
Une version initiale de *correction()* attendait que toute la correction soit effectuée pour retourner la note obtenue vers le front-end ce qui ne permettait plus l'utilisation de l'interface client.

7.3.3 Traitement par correct.sh

Extrait du fichier *correct.sh* ligne 20 a 35.

```
## ------  CREATION DE L'IMAGE DOCKER ------ ##
cp $pathFile/"$1" $pathContainer
cp $pathCorrectionFile/"$2" $pathContainer
docker image build $pathContainer -t $1 > tmp

## ------  SUPPRESSION DU FICHIER A TESTER DANS LE BACK-END ------ ##
rm $pathContainer/"$1"
rm $pathContainer/"$2"

## ------  LANCEMENT DU CONTAINER DEBIAN AVEC LE FICHIER A TESTER------ ##
docker run --name "$1" -d "$1" > tmp
id=$(docker ps --format="{{.ID}}" --filter "name=$1")

## ------  OUTPUT DE L'EXÉCUTION DU FICHIER ------ ##
echo $(docker exec "$id" sh $2 $1)
```

Le fichier *correct.sh* est un script shell qui permet de lancer depuis le container du back-end un container pour corriger le fichier dans un environnement isolé et donc augmenter la sécurité.
Les étapes extraites consistent :

- à copier des fichiers nécessaires à la correction (le fichier de correction et le fichier à corriger);
- à la création de l'image avec ces fichiers;
- à la suppression de ces fichiers dans le serveur back-end;
- au lancement du container (possible car on utilise le Docker daemon de l'hôte);
- à la récupération de l'id du container (pour pouvoir le supprimer définitivement plus tard);
- à l'affichage du résultat de l'exécution de la correction du fichier (récupéré dans un fichier *"nom_fichier.res"*)

Le script se termine par la suppression des images de docker.

7.3.4 Système de points et d'accès au projet

Chaque projet est noté sur 100 et peut donc rapporter jusqu'à 100 points d'expérience qui correspond à un niveau. Par exemple lorsqu'un utilisateur valide à 45 % un projet il gagne seulement 45 points sur les 100 points disponibles et il pourra éventuellement retenter le projet pour obtenir les points qui lui manquent.

Ainsi l'attribut *user.exp* évolue au fil du temps et s'affiche sur l'interface de l'utilisateur qui le convertit en un niveau et une barre d'expérience sur 100. Cependant ce n'est pas l'expérience

globale qui permet à l'utilisateur d'accéder aux autres projets mais l'expérience sur le module du projet. Cela permet de toujours démarrer les modules par les premiers projets.
L'attribut *projet.exprequis* dans la base de données nous a permis de définir le pourcentage de complétion du module pour tolérer l'accès ou non dans le front-end.

idprojet	exprequis
1	0
2	0
3	20
4	20
5	40
6	40
7	60
8	60

En suivant le tableau précédent il n'y a pas besoin d'expérience pour avoir accès au projet 1 cependant pour avoir accès au projet 7 et 8 il faut déjà avoir validé au moins 60% du module c'est à dire les projets de 1 à 6 a 80%.

7.4 Authentification dans le back-end

Nous avons choisi de décrire dans cette sous-partie le fonctionnement de l'authentification de l'utilisateur.

7.4.1 L'étape de connexion

La connexion de l'utilisateur se fait lorsque l'application client fait une requête vers *http://localhost:3000/api/login*. Le "body" de la requête doit contenir le courriel et le mot de passe de l'utilisateur.

Voici un extrait du traitement de la requête lorsque l'utilisateur a saisi le bon mot de passe :

```
let payload = { subject: userdata.courriel };
let token = jwt.sign(payload, 'secretkey');
res.status(200).send({ token });
```

Dans ce cas nous allons créer un jeton qui contient le courriel de l'utilisateur et le signer avec la clé secrète puis l'envoyer au front-end qui se chargera de le stocker dans le local storage.

7.4.2 verifyToken()

La fonction *verifyToken* qui se trouve dans le fichier *./services/tokenverify.js* ligne 9 est la fonction qui permet d'authentifier si le token fourni par l'utilisateur correspond à celui qui est attribué lors de la connexion d'un utilisateur.

```
async function verifyToken(req, res, next)
{
    if (!req.headers.authorization)
    {
        return res.status(401).send("Unauthorized request");
    }
    let token = req.headers.authorization.split(' ')[1];
    if (token === 'null')
    {
        return res.status(401).send("Unauthorized request");
    }
    let payload = jwt.verify(token, 'secretkey');
    if (!payload)
    {
        return res.status(401).send("Unauthorized request");
    }
    req.courriel = payload.subject;
    req.username = await get_name(req.courriel);
    next();
}
```

Cette fonction est ajoutée comme fonction de callback aux points où il est nécessaire comme par exemple :

```
router.get('/user', tkn.verifyToken, async (req, res) => {...}
```

la fonction récupère le token se trouvant dans le header de la requête (vous trouverez en partie conception détaillée - côté client sous la partie authentification, plus d'informations à ce sujet) puis vérifie à l'aide de la fonction *verify* de la librairie *jsonwebtoken* si le token récupéré correspond bien à un token créé avec la clé 'secret key'. La fonction *jwt.verify* retourne le contenu du token lorsqu'il a été signé à l'étape de connexion, dans notre cas il s'agit du courriel de l'utilisateur.
Nous avons également choisi d'ajouter cette donnée a la requête avant de la transmettre aux autres instructions.

Finalement lorsqu'un appel est fait à l'API et qu'il implique que l'utilisateur soit authentifié nous vérifions si le jeton est contenu dans le header de la requête et nous ajoutons à la requête le courriel et le nom de l'utilisateur.

7.5 API

voici la table des endpoints de l'API:

Méthode	URL	Body	Description
Authentification			
post	/api/register	{ courriel: string, password: string }	Permet a un utilisateur de s'enregistrer dans la base de donnée
post	*/api/login*	{ courriel: string, password: string }	Permet a un utilisateur de se connecter et d'obtenir un token d'authentification
Données			
get	/api/user	null	Permet a un utilisateur d'obtenir ses données
get	*/api/photo/:id*	null	Retourne la photo associée a l'id
Projets			
get	*/api/projets*	null	Retourne les données de tous les projets
get	*/api/projets/:id*	null	Retourne les données du projet associé
Put	*/api/projets/sub/:id*	Null	Permet a l'utilisateur de s'inscrire au projet id
delete	*/api/projets/sub/:id*	Null	Permet a l'utilisateur de se désinscrire au projet id
post	*/projet/upload_projet/:id*	{'MyFile', File, Filename}	Permet d'enregistrer le fichier lie au projet id de l'utilisateur et de le faire corriger
get	*api/projets/stats/:id/:tentative*		Permet d'obtenir des donnes sur le projet Id en fonction du numero de la tentative

8 Conception détaillée de la base de données

8.1 Schéma relationnel

Pour modéliser notre base de données nous avons utilisé MySQL car c'est l'outil le plus utilisé et la documentation est très alimentée sur Internet.
Soit le schéma relationnel suivant décrivant notre base de données:

- DEPOTS(**iddepots,** projetid, courriel, filename, date, note, tentative) : Lorsque l'étudiant dépose un projet (qui porte le nom *filename*) pour le faire corriger, le serveur analyse sa demande et redirige la correction grâce à *iddepots* et *projetid*. Le *courriel* de l'utilisateur est son identifiant, le serveur lui attribue une *note* et incrémente le nombre de *tentatives* de l'utilisateur pour ce projet.

- ENCADRANT(**idencadrant**, nom) : Chaque projet possède un encadrant qui lui ai propre. Chaque encadrant est décrit de manière unique par un identifiant *idencadrant* et par son *nom*.
- MODULE(**idmodule,** nom, cursus) : Chaque module est associé a un module précis (SHELL, Java, PHP,...). Chaque module est décrit de manière unique par un identifiant *idmodule* et par son *nom*. Il est présent dans un *cursus* précis (1A, 2A,...).

- PROJETS(**idprojet**, encadrantid, moduleid, nom, Infos, détails, projetscol, tag1, tag2, tag3, tag4, fileNameCorrection fileNameOutput, idprojetparent, lvlrequis, ptsprojet) : Un projet est décrit de manière unique par un identifiant *idprojet* et par son *nom* propre. Il est associé à un module et à un encadrant. Il possède ses propres *Infos* et *détails*. Un projet possède 4 tags *tag1, tag2, tag3, tag4*, qui permettent une meilleure recherche dans la barre de recherche par chaque étudiant. Pour débloquer d'autres projets parents (identifié par idprojetparent), il faut que l'étudiant valide les projets au fur et à mesure. Le fait qu'un étudiant puisse tenter un projet où non est représenté par *lvlrequis*: il faut que le niveau de l'utilisateur soit supérieur ou égal à *lvlrequis*.

- REGISTEREDUSER(**userCourriel,** projetId, note, date) : Un utilisateur authentifié est décrit de manière unique par un identifiant *userCourriel.* Il peut être inscrit à plusieurs projets identifiés par leur *projetId,* il obtient une *note* à la *date* ou il a déposé son fichier.

- USER(**username,** password, courriel, exp, urlimg, date-registered) : Un utilisateur est décrit de manière unique par un identifiant *username* et son mot de passe *password.* Il possède de l'expérience *exp* qu'il peut augmenter au fur et à mesure qu'il valide des projets. Il peut personnaliser son interface utilisateur en ajoutant une image *urlimg.*

8.2 Diagramme UML

Voici la modélisation de notre base de données grâce à un diagramme UML:

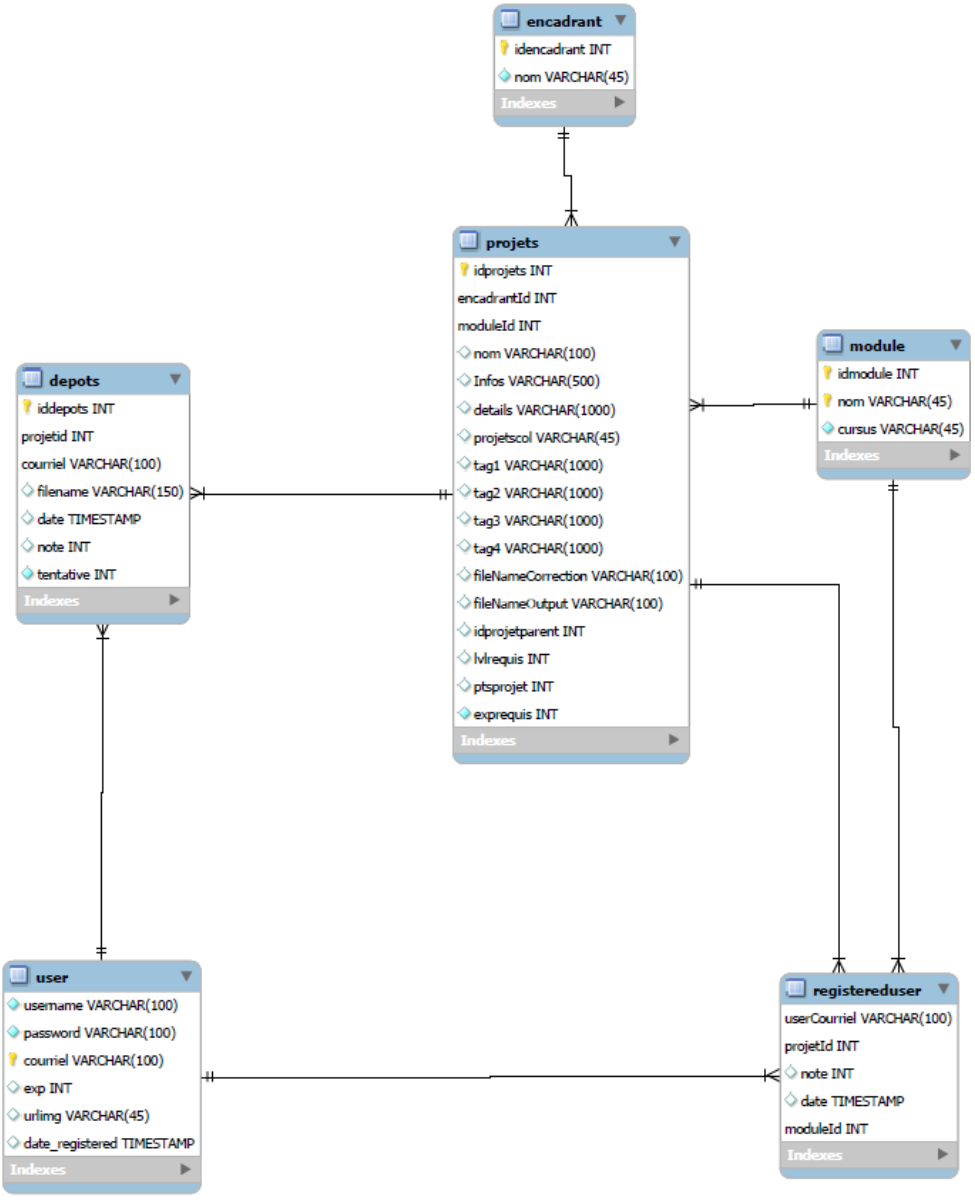


Illustration 1: Diagramme UML

9 Bilan gestion de projet

Le projet de développement informatique nous a permis de développer nos compétences en informatique et d'apprendre de nouveaux langages de programmations. Cependant, la partie gestion de projet s'est avérée être cruciale et très utile dans notre travail de groupe, plus que l'on ne l'avait imaginée.

La gestion de projet nous a permis de bien amorcer le projet, en posant les bases et en dressant un cahier des charges comportant des critères précis. Mais elle nous a été utile également tout au long du projet, grâce aux différents échelons et jalons. Ainsi, dans cette partie, nous nous proposons d'établir un bilan de la gestion de projet effectuée, en distinguant les trois étapes clés et en analysant leur impact dans le projet.

9.1 Utilisation des outils collaboratifs

Il a été nécessaire de se fixer des tâches précises pour poser les fondations d'un projet. Pour cela, plusieurs outils ont été utilisés:

- La mise en place d'un cahier des charges nous a permis de fixer les fondations de notre future application. Nous nous sommes tous réunis et nous nous sommes tous mis d'accord sur la façon dont notre application allait être orchestrée.

9.1.1 Grille de critères

La grille de critère a été créé le 09 Mars, recense les fonctionnalités principales attendues pour chaque composant (barre de navigation, settings) nous apportant une aide sur les étapes principales et essentielles à réaliser pour chaque composant. Il était modifiable à tout moment par tous les membres de l'équipe.

Grille de critère essentiels

Seq	Composant	Niveau	Type	Fonctionnalités recherchées	Etat
	Barre de navigation				
		1		Redirection vers les liens des composants	Fait
		1		responsive	Fait
	Projet/Correction				
		1		l'utilisateur peut sauvegarder son fichier	Fait
		1		Correction du projet	Modifié
		1		Description du projet/enoncé	Modifié
		1		Avoir accés a sa note finale	Fait
	Roadmap				
		1		Utilisateurs peut visionner des projets	Fait
		1		L'utilisateur voit ses notes sur chaque projet	Fait
		1		Afficher les tags	Fait
	Settings				
		1		Pouvoir modifier ses infos:MDP	Fait
				Pouvoir ajouter une photo de profil	Fait
	Data User				
		1		Experience acquise	Fait
		1		Derniers projets validés, dernier projet en cours	Annulé
		1		Grades en fonction du niveau	Fait
		1		Nbr de projet succeeed	Annulé
	Authentification				
		1	Sécurité	Se connecter/ s'enregistrer	Fait
		1	Sécurité	Hashage sur mdp	Fait

Figure 5: Grille des critères essentiels

Conseil de lecture de la grille :

- La colonne “Niveau” correspond au degré d’importance de la fonctionnalité désignée. Le niveau “1” étant le degré le plus élevé d’importance.
- La colonne “Type” correspond au type de la fonctionnalité recherchée.
- La colonne “État” correspond à l’avancement de la tâche, trois états sont disponibles: “Fait”, “Modifié”, “Annulé”.

9.1.2 Utilisation de l’outil GitLab

Notre utilisation de GitLabENS ne s’est pas seulement cantonnée au partage du code informatique, elle s’est élargie aussi à la réalisation d’une ToDo list pour chaque composants (front-end, back-end et Base de données). Mais aussi à la visualisation de l’historique des commits, la création des branch et les merge.

9.1.2.1 TodoList

Pour illustrer cette fonctionnalité de GitLab nous allons fournir un exemple sur le front-end ci-joint:

Dans cet exemple vous pouvez voir :

- le nom de la tâche à réaliser (en gras)
- La date de création de la tâche
- Une icône de couleur qui représente l’avatar du responsable de tâche présentée.
  - L’icône bleue correspond à Samson Mazeau
  - L’icône rouge correspond à Juliette Ould-Aklouche
  - L’icône violette correspond à Lorenzo Nadal-Santa

Cette Todo List a aussi été réalisé pour le back-end et la base de données.

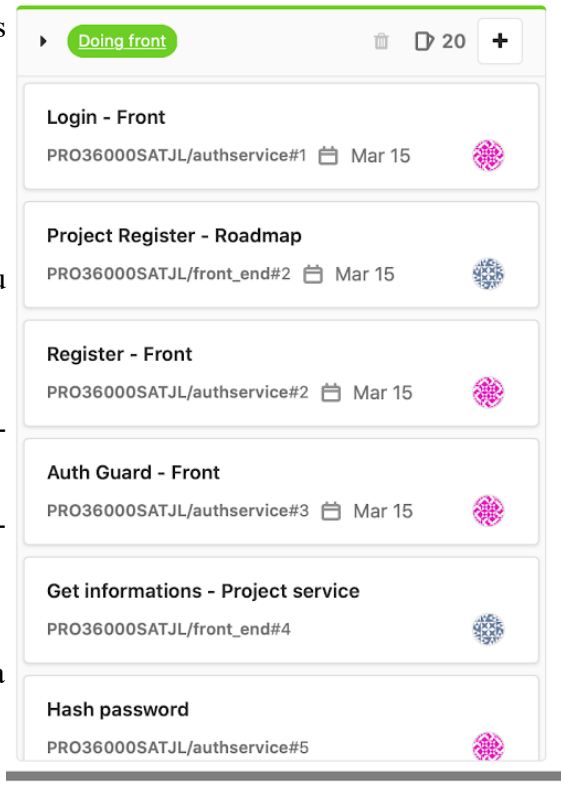


Figure 6: Vue d'un des tableaux du board de Gitlab

9.1.2.2 Historique des commits

Une fonctionnalité GitLab nous permet aussi de visionner les historiques des commits réalisés.

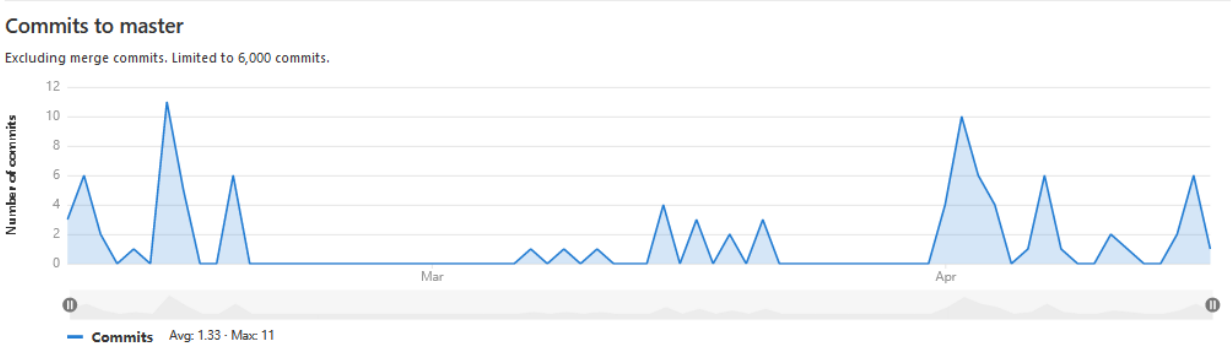


Figure 7: Historique des commits dans le front-end

Figure 8: historique des commits dans le back-end

Comme on peut le voir sur les graphiques précédents, on observe des pics de commits au même période, ce qu'on peut traduire par le développement simultané de fonctionnalité liées au front-end et également au back-end.

9.1.2.3 Historique des branches git

Nous avons voulu aussi insérer le graphique des merge et des branch réalisées pendant le projet :

Vous pouvez voir ici que les membres de l'équipe qui ont utilisé GitLab régulièrement (à savoir Lorenzo, Juliette et Samson) ont chacun créé leur propre branche en plus de la branch master. Ils ont ensuite merge leur branch avec la branch master pour fusionner leur code à l'application existante.

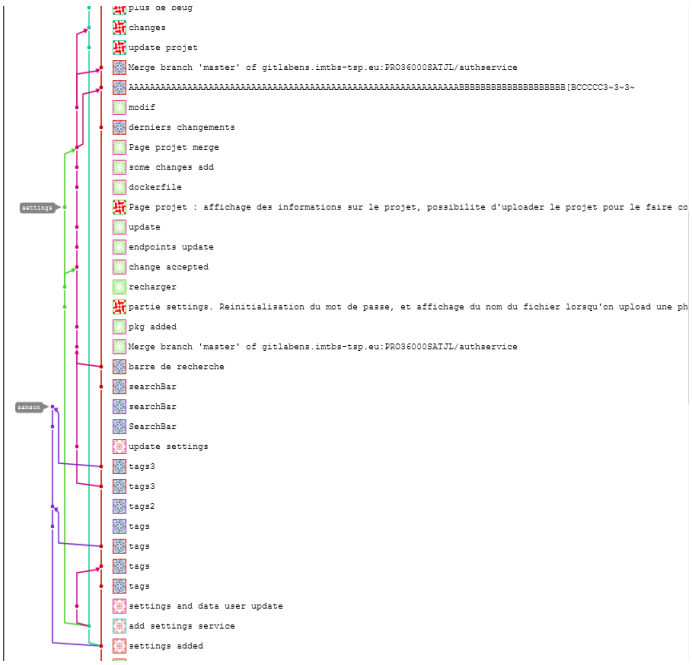


Figure 9: Historique des branches git

9.1.3 Task-tracker

Un task-tracker a aussi été réalisé. C'est un tableau qui recense le travail effectué et le nombre d'heures passées pour chacun des membres de l'équipe. Il a été créé pour remplir deux objectifs précis:

- Le premier objectif était de se rendre compte, pour les membres du groupe, de leur avancement au cours du projet et de prendre du recul sur notre avancement. Nous nous y référions souvent pour tirer des conclusions entre nous sur les avancements de chacun.
- Le second objectif était de laisser une trace sur la durée réellement dépensée pour réaliser des tâches précises afin que les données puissent illustrer notre façon d'avancer au cours du projet.

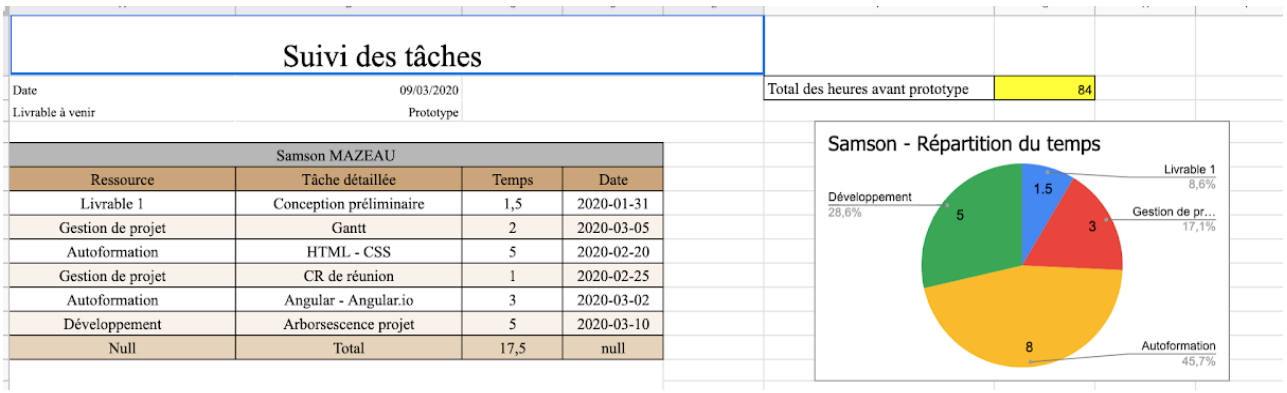
Le task-Tracker va être commenté en précision dans la partie suivante 9.2.

9.2 Analyse des données

9.2.1 Analyse du Task-tracker

Dans cette analyse, nous nous baserons sur l'exemple d'un des membres du projet: Samson. Voici le visuel global de ce Task-tracker. Nous l'avons divisé en deux partie:

- La première partie a été consacré à l'intervalle entre le début du projet et le rendu du prototype (Janvier - 15 mars) ;
- La seconde partie a été consacré à l'intervalle entre le rendu du prototype et le rendu du livrable 3 (16 mars - 18 mai).



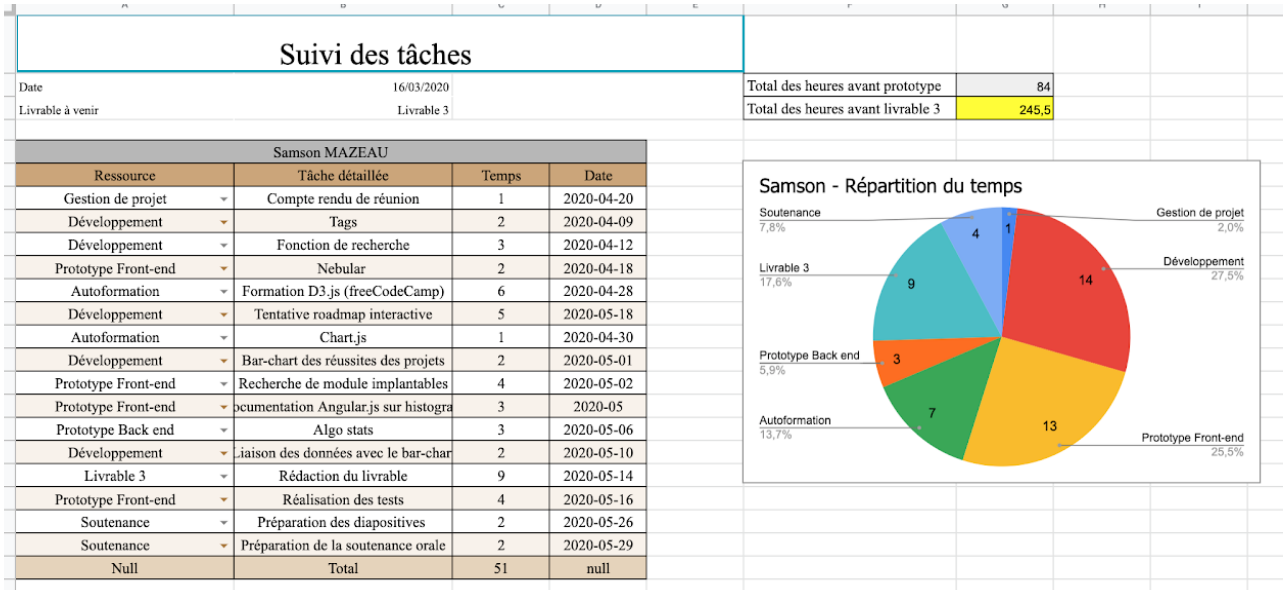
Suivi des tâches

Date 09/03/2020
Livrable à venir Prototype

Total des heures avant prototype 84

Samson MAZEAU			
Ressource	Tâche détaillée	Temps	Date
Livrable 1	Conception préliminaire	1,5	2020-01-31
Gestion de projet	Gantt	2	2020-03-05
Autoformation	HTML - CSS	5	2020-02-20
Gestion de projet	CR de réunion	1	2020-02-25
Autoformation	Angular - Angular.io	3	2020-03-02
Développement	Arborsescence projet	5	2020-03-10
Null	Total	17,5	null

Figure 10: Vue suivie des taches jusqu'au prototype(15 mars)



Suivi des tâches

Date	16/03/2020
Livrable à venir	Livrable 3

Total des heures avant prototype	84
Total des heures avant livrable 3	245,5

Samson MAZEAU			
Ressource	Tâche détaillée	Temps	Date
Gestion de projet	Compte rendu de réunion	1	2020-04-20
Développement	Tags	2	2020-04-09
Développement	Fonction de recherche	3	2020-04-12
Prototype Front-end	Nebular	2	2020-04-18
Autoformation	Formation D3.js (freeCodeCamp)	6	2020-04-28
Développement	Tentative roadmap interactive	5	2020-05-18
Autoformation	Chart.js	1	2020-04-30
Développement	Bar-chart des réussites des projets	2	2020-05-01
Prototype Front-end	Recherche de module implantables	4	2020-05-02
Prototype Front-end	ocumentation Angular.js sur histogra	3	2020-05
Prototype Back end	Algo stats	3	2020-05-06
Développement	Liaison des données avec le bar-char	2	2020-05-10
Livrable 3	Rédaction du livrable	9	2020-05-14
Prototype Front-end	Réalisation des tests	4	2020-05-16
Soutenance	Préparation des diapositives	2	2020-05-26
Soutenance	Préparation de la soutenance orale	2	2020-05-29
Null	Total	51	null

Figure 11: Du prototype au livrable final

9.2.1.1 Analyse détaillée

Jusqu'au 15 Mars, le travail principal était de **se former à de nouveaux langages informatiques** (HTML, CSS, JavaScript, etc). Ce qui représente pas moins de **46% de la charge de travail** pour Samson. De 8 heures dans la première partie, la durée d'autoformation passe à 7 heures dans la seconde, mais reste majoritairement en début de projet. En effet, la partie autoformation est importante en début de projet pour apprendre les nouveaux langages, puis, à mesure que l'on gagne en expérience, l'autoformation devient plus spécifique: Nebular pour le style, pour la création de roadmap interactive…

Finalement, ce n'est pas moins de **15 heures passées à l'autoformation**, soit plus d'un quart des 50 heures suggérées pour le projet développement informatique, sans compter les recherches sur internet pour résoudre des problèmes rencontrés durant le codage et pour trouver de nouvelles méthodes.

De plus, la charge de travail liée à la gestion de projet diminue au cours du temps, comme nous l'avions suggéré: une préparation solide en début de projet permet d'économiser de précieuses heures au cours de l'avancement du projet.

Par ailleurs, si des équipes front-end et Back-end ont été créées, il est parfois nécessaire de naviguer entre les deux: dans la partie 'Développement" de 14 heures, Samson codait à la fois en front-end et en back-end.

Finalement, pour une durée totale de **68,5 heures de travail** pour Samson, **33,5 heures** ont été consacrées à d**es tâches de préparation** et de complément au développement (formation, gestion de projet, rédaction des livrables…) pour **35 heures de code** pur, soit presque la moitié.

9.2.1.2 Analyse globale

On retrouve cela dans le bilan global des membres du groupe: il faut travailler presque autant en préparation de projet et en synthèse que pour le contenu propre.

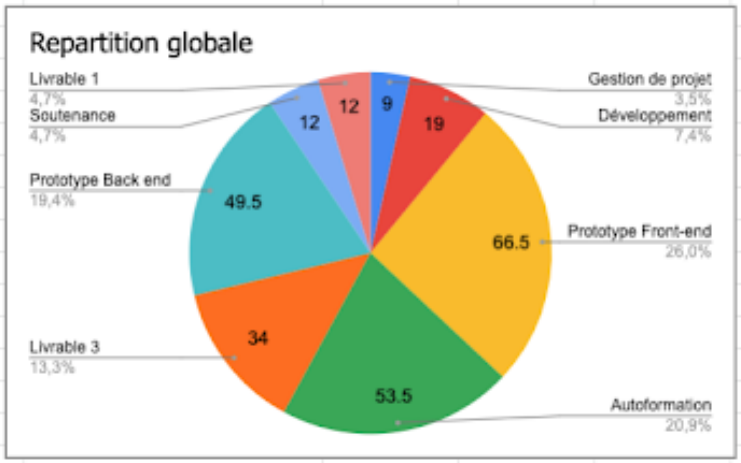


Figure 12: Statistiques globales

Nous voyons dans ce dernier graphique que **20% du temps** a été dédié à **l'autoformation**, **50%** temps a été dédié au **codage**. Nous nous attendions a une durée plus longue pour cette partie 10% du temps a été dédié à la gestion de projet, mais le rapport Importance/temps n'est pas respecté pour ce point car la gestion de projet a été essentielle pour la coordination de l'équipe. Enfin 20% du temps fut dédié à la rédaction des différents livrables que l'on a dû rendre. Nous avons donc considéré pertinent de réaliser ce travail-bilan pour illustrer le cheminement de notre travail afin d'avoir un certain recul appréciable.

9.2.2 Bilan du gantt

Le diagramme de Gantt, rédigé pour la réunion numéro 4 du 09/03, se trouve dans les détails dans le GitLab du projet: dans Livrable dans le dossier Gestion de projets, de même que le task-tracker et le tableau des critères.

Il est difficile d'appréhender le temps nécessaire à chaque tâche et de le séquencer, surtout en début de projet. Si le Gantt permet de se fixer des échelons, et de se faire un programme, c'est également un exercice peu précis sur la durée. Dans l'ensemble, les échelons n'ont été respectées: nous avions par exemple surestimé la partie debugging, qui se fait en fait dans la continuité du codage. Ainsi, nous pourrions penser à utiliser une méthode agile pour mettre à jour le planning au fur et à mesure que l'on connaît mieux les outils, et que l'on sait donc mieux les appréhender.

Diagramme de Gantt

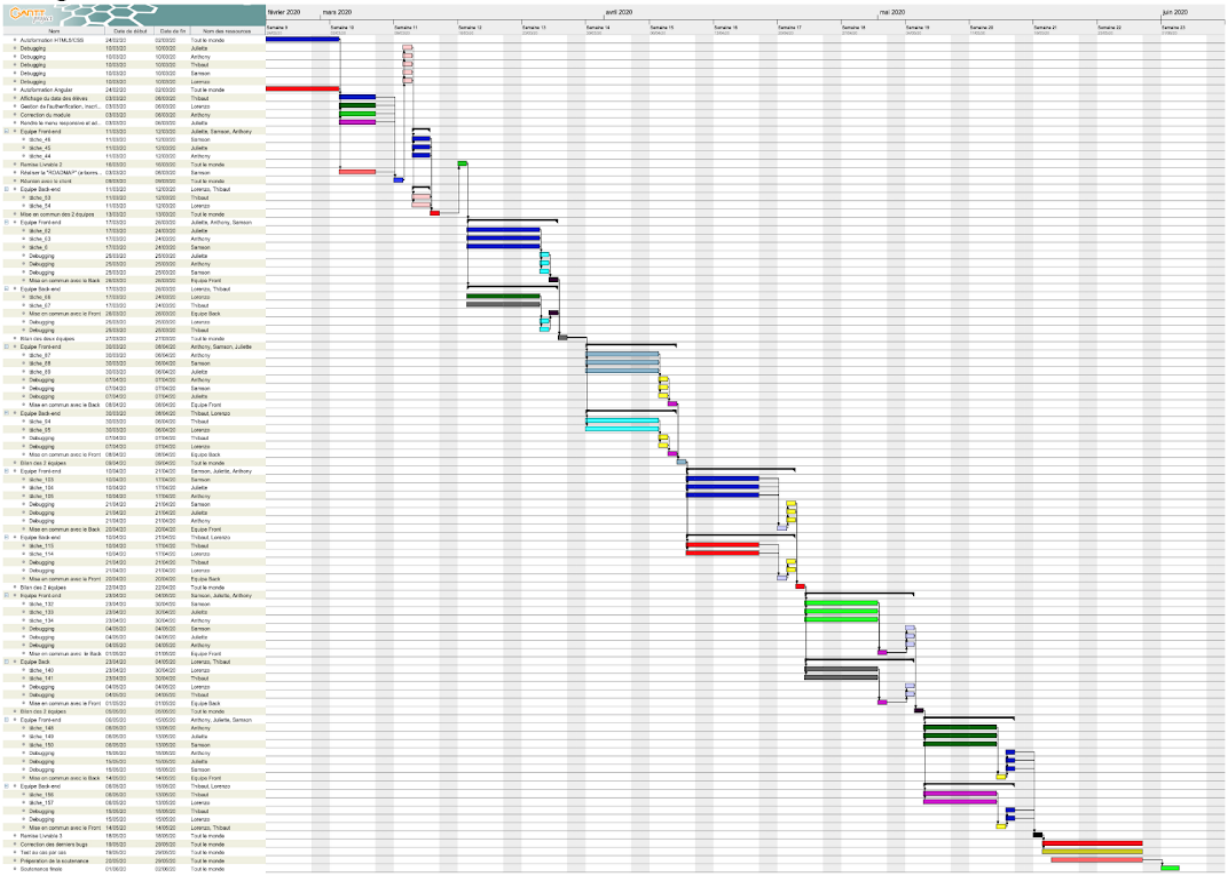


Figure 13: Diagramme de Gantt

9.3 Conclusion

Il nous a semblé pertinent de réaliser ce travail-bilan pour illustrer le cheminement de notre travail afin d'avoir un certain recul appréciable. Il permet aux professeurs qui nous évaluent d'avoir la vision de notre projet en globalité et en n'omettant aucun pan de celui-ci. La partie essentielle du projet reste la partie développement/codage mais la partie gestion de projet est aussi un point important.

L'outil de gestion collaboratif GitLab, possède de nombreux outils agréables qui permet de réaliser un retour-projet instructif (cf partie 9.1).

10 Bilan technique

10.1 Conformité par rapport au cahier des charges

Nous allons reprendre pas à pas et commenter chaque point du cahier des charges de la partie 2.2.1.

- L'objectif de "gamification" de l'enseignement est atteint car lorsqu'un étudiant valide un projet, la note obtenue s'ajoute à sa barre d'expérience. Lorsque sa barre d'expérience atteint 100%, il augmente son niveau et peut accéder à d'autres projets jusque là inaccessibles pour lui.
- La mise en place de la roadmap est réussie, néanmoins nous n'avons pas réussi à lui donner l'aspect final espéré. Elle reste tout de même fonctionnelle et facilement utilisable.
- La correction automatique se réalise comme attendue. De plus nous avons créé plusieurs ports afin que l'utilisateur lance plusieurs corrections en même temps.
- L'administrateur du serveur et de la base de données a accès à tous les projets et à leur correction, que ce soit au niveau du serveur comme au niveau de la base de données. Il a donc accès à toutes les données des utilisateurs. Mais ce droit reste réservé à l'administrateur du site et non pas à tous les professeurs.
- L'ajout/suppression des projets/correction par les enseignants ne peut se faire que par l'intermédiaire de l'administrateur. La création d'un profil propre aux enseignants n'a donc pas été réalisé. La condition est donc partiellement remplie.

Par conséquent, une grande partie du cahier des charges est respecté.

10.2 Conformité par rapport aux spécifications fonctionnelles

Nous allons énumérer et commenter chaque spécification fonctionnelle (cf partie 3).

La plateforme web est containerisée à travers trois éléments dépendants les uns des autres:

- La partie client
- la partie serveur
- L'API

10.2.1 client

- L'ergonomie de l'application est réalisé grâce à Angular et Nebular.

- L’application est compatible avec la plupart des moteurs de recherche (Mozilla Firefox, Google Chrome, Internet Explorer, Oracle et Yahoo). Nous n’avons cependant pas testé si l’application était compatible sur mobile.
- L’application est accessible au sens des règles W3C.

10.2.2 Utilisateurs

Pour le visiteur:

- Lorsqu’il n’est pas enregistré ni authentifié le visiteur est directement redirigé vers le service d’authentification et de connexion.

Pour l’élève:

- Il possède des droits de lecture sur toutes ses données générales et sur les projets qu’il réalisent. Les droits d’écriture sont réservés au dépôt du fichier pour la correction et au niveau de leur mot de passe.

L’ajout et la suppression des projet sont accordés à l’administrateur. Les droits ne sont pas proprement transmis aux professeurs.

10.2.3 Menu

Notre page contient bien les trois rubriques suivantes:

- Accueil ;
- Roadmap ;
- Settings (Informations personnelles) ;
- Nous avons rajouté une rubrique “Toggle” qui peut être affichée/cachée et qui contient des informations personnelles sur l’utilisateur.

10.2.4 Rubrique : Accueil/Toggle

- La Photo de l’élève est optionnelle. Nous avons choisi de lui laisser le choix sur la photo qu’il veut afficher (dans le Toggle) ;
- La barre de progression est présente dans le Toggle ;
- Ses projets en cours sont indiqués dans la rubrique Accueil de l’application ;
- Nous avons choisi d’afficher le dernier projet que l’élève a validé et non pas ses derniers projets pour éviter une surcharge.

10.2.5 Rubrique : Roadmap

- Pour que l'étudiant puisse voir s'il a validé le projet il faut qu'il clique sur le bouton "voir le projet" et qu'il consulte son historique de dépôt et sa note ;
- Pour l'aspect validation nous avons voulu le matérialiser grâce déblocage d'autres projets. Tant que l'étudiant n'a pas réussi assez de projet ou n'a pas réuni assez de points il ne peut pas accéder aux autres projets ;
- Les projets non accessibles sont grisés via le bouton "s'inscrire".

10.2.6 Rubrique : Settings / Toggle (Informations personnelles)

Les seules informations qui sont disponibles dans ces deux rubriques sont soulignées, les autres sont écrites en italique:

- l'adresse courriel ;
- le mot de passe (possibilité de modification) ;
- *une photo ;*
- *l'année de promotion ;*
- *la date de naissance ;*
- *des informations complémentaires.*

10.2.7 Élément "Projet"

Chaque projet comporte bien un nom, un fichier de correction mais ne comporte pas de fichier PDF descriptif des attentes du projet (par manque de temps).

Dans chaque projet, il existe une page associée qui fournit à l'élève la moyenne sa note obtenue. Un graphique est également présent pour illustrer la réussite global du projet par les autres étudiants.

10.3 Difficultés rencontrées

Comme tout projet, celui-ci a été jalonné d'échecs et de réussites. Heureusement plus de réussites que d'échecs. Notre échec principal est de ne pas avoir réussi à coder une roadmap "zoomable" à l'aide de *D3.js*. Côté front-end, nous avons eu aussi beaucoup de difficultés avec le flux de données entre composants. C'est à dire que chaque composants se lançaient simultanément et avaient du mal à récupérer des données depuis leurs composants parents. Le problème est connu et la solution est simple. Mais la résolution ne marchait pas toujours, ce qui nous a valu quelques heures de débogage...

Côté back-end, l'aspect "asynchrone" des fonctions était parfois difficile à gérer.

Le dimensionnement du projet était difficile à gérer. Dans le sens où le temps de formation est trop court par rapport aux réelles connaissances dont on a besoin pour se spécialiser. Le plus dur étant de faire le “puzzle” entre la partie codage et la partie formation.

10.4 Conclusion

En partant de zéro en programmation web, nous avons réussi a développer complètement une application depuis l’interface client en passant par le serveur jusqu’à la base de donnée. Dans un soucis d’optimisation mais également curieux d’en apprendre davantage et de mener notre projet au-delà de ce que nos compétences nous permettaient, nous avons pris en main des technologies actuelles comme Angular, Docker et Nodejs. Nous avons pu explorer les fondamentaux de ces technologies et nous avons réussi a les faire fonctionner ensembles dans le cadre de notre projet.

11 Manuel utilisateur

Nous allons dans cette partie décrire comment exploiter le logiciel final.

11.1 Pré-requis

Conformément aux explications dans les parties relatives à l'utilisation de Docker dans notre projet les pré-requis sont :

- Docker
- Docker CLI
- Docker-compose
- les ports 3000, 4200 3306 disponibles

11.2 Exploitation

Depuis un terminal, lancer la commande suivante :

```
git pull git@gitlabens.imtbs-tsp.eu:PRO36000SATJL/livrables.git
cd ./livrables/livrable3/logiciel
```

Vous vous trouverez dans le répertoire suivant :

```
.
├── front-end
├── back-end
├── database
└── docker-compose.yml
3 directories, 1 file
```

Dans le terminal lancer la commande suivante: docker-compose up ou sudo docker-compose up si vous avez besoin des droits administrateurs. Cela peut prendre un certain temps la première fois car les images et les différentes librairies ne sont pas installées. Une fois que tout semble s'être déroulé correctement, lancer un navigateur et aller faire un tour à http://localhost:4200 !

12 Tests unitaires

Dans le cadre de notre projet, nous avons choisi de réaliser un set de tests unitaires sur 5 fonctions. Nous avons choisi de tirer ces fonctions du back-end car la majorité des fonctions dans le front-end sont des fonctions qui récupèrent des données depuis le back-end. Nous n'avons pas pu réaliser un lot de tests complet pour l'ensemble du projet car le dimensionnement en temps du projet ne nous l'a pas permis.

Dans cette partie nous décrirons dans un premier temps l'exploitation des tests puis dans un second temps comment traiter les résultats.

12.1 Exploitation du set de tests

le set de tests se trouve dans le répertoire du back-end : *./back-end/tests/*.

Les pré-requis à son utilisation sont :

- les pré-requis à l'exploitation de l'application (voir manuel utilisateur);
- l'application en cours de fonctionnement (via Docker);

Lancer un terminal en parallèle de celui utilisé pour l'exploitation de l'application et rendez-vous dans le répertoire suivant:

```
.
├── front-end
├── back-end
├── database
├── tests.sh
└── docker-compose.yml
3 directories, 2 files
```

Exécuter le fichier sh comme suit : `sh tests.sh`.

Vous trouverez dans le même répertoire les résultats des tests dans les fichiers de format test_"nomdelafonction".js.res comme suit :

```
.
├── front-end
├── back-end
├── database
├── tests.sh
├── test_get_tentative.js.res
└── docker-compose.yml
```

12.1.1 Le set de tests

Voici l'ensemble des fichiers de test:

- test_setpts.js
- test_get_tentative.js
- test_subscribe.js
- test_processdata.js
- test_noteTotaleModule.js

Les fonctions testées sont:

- *set_pts(courriel, note old_note)* du service CorrectionService.js ligne 187
  - La fonction doit ajouter à la base de données les points gagnés à la suite d'une correction en fonction de la nouvelle note et de la plus haute note obtenue auparavant;
- *get_tentative(id, courriel)* du service ProjetService ligne 322
  - Cette fonction retourne le numéro de la tentative pour la correction qui vient d'être demandée par l'utilisateur.
- *subscribe(id, courriel)* du service ProjetService ligne 195
  - Cette fonction permet à l'utilisateur de s'inscrire à un projet
- *note_totale_module(rows, nbProjet)* du service ProjetService ligne 482
  - Cette fonction permet d'accéder à l'avancement total du module (en pourcentage) de l'utilisateur.
- *process_data(rows)* du service ProjetService ligne 379
  - Cette fonction permet de comptabiliser le nombre total de notes obtenues pour un projet donné.

De plus, afin de tester le service de correction depuis l'interface client nous avons mis à disposition dans le répertoire *./back-end/fichiers_a_corriger* des fichiers à soumettre pour la correction avec des notes différentes.

12.1.2 Exploitation des résultats - exemple test_setpts.js

Voici un extrait du résultat du *test_setpts.js* :

```
----AFFICHAGE DES DONNÉES DE L'UTILISATEUR AVANT TEST-----
Connected to database
RowDataPacket {
 username: '202',
 password:
  '$2b$10$E3ySY.aRP/aimhhdYdW7XeJ0ukyiUBHeuza26ZcQb9AxGJ1e7i0Qm',
 courriel: '2020@2020.com',
```

```
  exp: 220,
  urlimg: '202_1587412427721.PNG',
  date_registered: 2020-04-20T19:52:11.000Z }
---------- lancement du test 1 ----------

Test 1 avec note > old_note
L'expérience doit augmenter de 20 pts
RowDataPacket {
  username: '202',
  password:
   '$2b$10$E3ySY.aRP/aimhhdYdW7XeJ0ukyiUBHeuza26ZcQb9AxGJ1e7i0Qm',
  courriel: '2020@2020.com',
  exp: 240,
  urlimg: '202_1587412427721.PNG',
  date_registered: 2020-04-20T19:52:11.000Z }
```

Dans ce test nous affichons les données que la fonction set_pts() va être amenée à changer. Nous annonçons les valeurs passées en paramètre, le résultat attendu et puis nous affichons les valeurs dans la base de données en guise de résultat obtenu. Nous avons raisonné de la même manière pour les 5 fichiers tests.

13 Annexes

13.1 Liens utiles

- Lien vers le Task-traker ;
- Lien vers la grille de critères ;