université angers

# Projet d'Environnement de développement en ligne

M2 ACDI

Management de projet


**Référents**

*RICHER Jean-Michel*

*GARCIA Laurent*

**Étudiants**

*POIRON Guillaume*

*LARDEUX Aloïs*

*VIRET Thibaud*

*MALASHCHYSKI Timotei*

# Table des matières

# Remerciements

Tout d’abord, nous tenons à remercier **M.Richer Jean-Michel** et **M.Garcia Laurent**, professeurs référents du projet, ayant également le rôle de client, pour nous avoir fait part de leurs attentes et d’avoir avancé avec nous dans la réalisation de ce projet.

Nous remercions également **Mme Coudray Rachel** qui a pris le temps de répondre à nos questions concernant le management du projet, et nous a guidées pour celui-ci.

Nous tenons à témoigner toute notre reconnaissance à **M.Chantrein Jean-Mathieu**, pour son temps et son investissement sur la prise en charge et la mise en fonctionnement de la Machine Virtuelle. Celle-ci nous ayant permis de déployer l’application dans de bonnes conditions, dans une infrastructure déjà opérationnelle.

Nous remercions **M. Plard Jean-Marc** pour son aide concernant la mise en place de notre service dans l'écosystème CAS de l’Université.

Enfin nous remercions les étudiants de première année de Master informatique d’avoir participé aux développements de l’application et ayant travaillé en collaboration avec nous.

# Introduction

Dans le cadre de l'unité Management de projet, nous avions pour objectif de réaliser un travail en situation d'entreprise. C'est-à-dire, d'être confronté à la mise en œuvre technique et à la gestion d'une équipe sur la réalisation d'une application.

Plus précisément, il s'agissait pour nous d'encadrer et de conduire une équipe de cinq étudiants de Master 1 Informatique, d'élaborer un cahier des charges, de participer et d'animer des réunions hebdomadaires et de veiller au bon déroulement de ce projet.

Les compétences visées de l'unité Management de projet sont :

- Savoir rédiger un cahier des charges.
- Savoir faire une veille technologique (état de l'art).
- Être capable de suivre et coordonner une équipe.
- Être capable d'évaluer le travail d'une équipe.

Dans ce rapport, vous retrouverez une présentation du projet, le besoin et les objectifs définis avec le client. Nous parlons de l'organisation du projet, l'encadrement et la répartition du travail pour les M1, puis du choix des outils et des technologies.

Vous trouverez également une partie un peu plus technique, où nous expliquerons comment nous avons défini l'architecture de l'application pour son bon fonctionnement.

Nous évoquerons les fonctionnalités améliorées et ajoutées sur le travail réalisé par les M1. Puis, nous finirons par un bilan du management dans ce projet et une conclusion générale sur ce que l'unité nous a apporté, tant sur l'aspect management que technique.

# I - Présentation du projet

## 1. Contexte

Ce projet a pour but de développer un environnement de développement en ligne devant être accessible depuis un navigateur web classique sans avoir à installer d'outils sur le poste des utilisateurs. L'application doit pouvoir au moins fonctionner sur les navigateurs Firefox et Chrome.

Cet environnement doit permettre l'édition, la compilation et l'exécution de code dans différents langages informatiques directement depuis le navigateur sans avoir à installer un quelconque outil sur les postes des utilisateurs. Toute la partie concernant la compilation et l'exécution du code est donc effectuée sur un serveur et le résultat rendu à l'utilisateur dans son navigateur web. Tout ceci doit être transparent pour l'utilisateur.

Les cibles de l'application sont principalement les étudiants de L1 et de L2 de l'Université d'Angers. Dans le cadre des enseignements, ils sont amenés à faire du développement informatique. Dans le mesure où ces étudiants peuvent ne pas disposer d'un environnement de développement sur leur poste personnel, ils doivent pouvoir continuer leur projet en dehors des temps à l'université. La cible peut être poussée jusqu'aux étudiants de L3 voire de Master si les fonctionnalités proposées par l'application sont suffisamment avancées pour justifier une utilisation par ces étudiants.

Pour avoir un aperçu et se familiariser avec l'application dans son ensemble, vous pouvez aller consulter l'annexe de ce rapport, vous y trouverez des images des différentes interfaces de l'application.

## 2. Objectifs initiaux

L'objectif principal de ce projet était la mise en place d'une application fonctionnelle, permettant la compilation et l'exécution de programmes via une interface web. En effet, le projet a été lancé en 2017, et malgré les travaux de plusieurs promotions d'étudiants, n'a jamais pu atteindre le stade de mise en production. C'est pourquoi, d'un commun accord avec les professeurs, nous avons choisi de re-concevoir l'architecture du projet, et de reprendre des développements sur une base vierge, en choisissant des technologies et concepts étudiés et pertinents pour répondre à cet objectif.

Cet objectif de mise en fonctionnement peut être découpé en d'autres objectifs, correspondant à des volontés imposées par le client :

- La compilation et l'exécution de programmes doivent être effectuées sur les serveurs de l'Université sans avoir à installer quoique ce soit côté utilisateur. Ceci implique que ces traitements soient vérifiés et isolés pour garantir l'intégrité de l'application et la sécurité des données.

- Réaliser une interface web permettant à un utilisateur d'effectuer ces actions de compilation et d'exécution, accessible en ligne. Cette interface se doit d'être simple, ergonomique, et répondant aux besoins d'un profil d'étudiant non-initié en informatique, tout en convenant à ceux ayant une déjà une maîtrise du développement informatique

- L'accès à l'application doit être restreint aux étudiants de l'université d'Angers. L'identification et le contrôle des actions sur l'application doivent être vérifiés via le protocole CAS, que l'université utilise pour contrôler l'accès à ses services.

- L'application doit proposer un gestionnaire de projets et de fichiers, permettant aux étudiants d'avoir accès à distance à leurs données. Il faut rendre possible la création, la modification, la sauvegarde, ou la suppression de ces données.

- L'application doit être déployée de manière simple et suffisamment documentée pour permettre une reprise potentielle du projet.

C'est donc ces différents objectifs que nous avons essayé de mener à bien tout au long du projet. Ayant choisi de relancer le projet de zéro, nous devions être au clair dès le départ sur tous ces objectifs afin d'avoir une vision longue sur le déroulement du projet, ses étapes et ses difficultés.

# II - Management de projet

## 1. Organisation du projet

### 1.1. Définition des spécifications fonctionnelles

Après notre mise au point avec les clients sur les besoins et les objectifs du projet, nous avons en toute logique rédigé les spécifications fonctionnelles pour décrire précisément l'ensemble des fonctionnalités attendues de l'application.

La rédaction des spécifications fonctionnelles a été un facteur primordial dans le déroulement du projet. En effet, cela nous a permis de garder une ligne directrice dans les phases de conception du projet, et d'avoir des choix fixés, pour permettre au mieux de guider les M1.

Finalement, les spécifications fonctionnelles nous ont permis de garder une cohérence entre le besoin du client et le résultat final.

### 1.2. Première rencontre avec l'équipe de développement

Lors des premières phases du projet, nous avons pu rencontrer et découvrir l'équipe de développeurs que nous aurions à encadrer pour la réalisation de l'application. Cette première rencontre s'est faite sous la forme d'une réunion dans laquelle nous avons fait la connaissance de chacun.

Chacun des membres de l'équipe, développeurs et managers, a ainsi pu présenter aux autres son parcours et son niveau d'expérience dans le développement informatique. Ceci nous a permis de déterminer au plus tôt les préférences, difficultés et forces de chacun afin d'optimiser au mieux la répartition des tâches.

Nous avons présenté aux M1, la nature du projet, ses objectifs et fait une présentation macro de l'architecture que nous avions choisi de mettre en place, ainsi que les technologies utilisées.

Enfin, dans le but de préparer au mieux les développeurs aux futurs travaux, nous avons choisi de rédiger un TP de prise en mains des technologies. Le sujet consistait en la réalisation d'une application de *to-do-list* suivant la même architecture que celle du projet, à savoir une communication serveur-web ⇔ API dans le langage javascript. Les M1 ont ainsi pu se familiariser avec différentes technologies et paradigmes qu'ils ne connaissent pas ou peu.
Nous leur avons, après quelques temps, fourni une correction de cet exercice, afin qu'ils aient une idée de ce que l'on attendait de leur travail sur le projet, en la prenant comme un exemple ou un modèle.

### 1.3. Mise en place de l'environnement de travail

Pendant que les M1 prenaient en main les technologies grâce à la réalisation du TP, nous avons commencé à mettre en place l'environnement de travail, c'est-à-dire d'initialiser les sources du projet. Nous avons créé les différents serveurs identifiés, à savoir : le serveur web (IHM de l'application), le serveur back (API) et la base de données. Nous avons ensuite conteneurisé le tout afin de permettre un déploiement stable, facile, et identique pour chacun des acteurs du projet.

Nous avons fait ce choix dans le but qu'une fois le TP terminé, les développeurs ne perdent pas de temps sur cette partie et commencent à travailler sur une base commune prédéfinie, dans de bonnes conditions. Ceci dans l'idée que les M1 puissent directement se concentrer sur le développement fonctionnel de l'application.

### 1.4. Répartition des tâches

A l'aide du logiciel Jira, nous avons pu répartir les tâches de développement de manière claire et rapide.

Nous avons opté pour une séparation des tâches en deux parties. La première était consacrée au développement FRONT et la seconde au développement API. Au total, nous avons réalisé trois itérations.

L'itération 0 comportait toutes les tâches essentielles dites de "socle technique":

- Création du squelette de l'IHM
- Création du schéma BDD

L'itération 1 comportait :

- Intégration des composants complexe (Codemirror & Xtermjs)
- Intégration des services API côté front.
- Intégration des routes de l'interface de programmation.
- Création des images docker pour la compilation.

L'itération 2 comportait :

- L'intégration de la compilation et de l'exécution
- L'intégration de l'authentification CAS
- Des fonctionnalités UX (Barre d'onglet, Popup création projet et fichier)

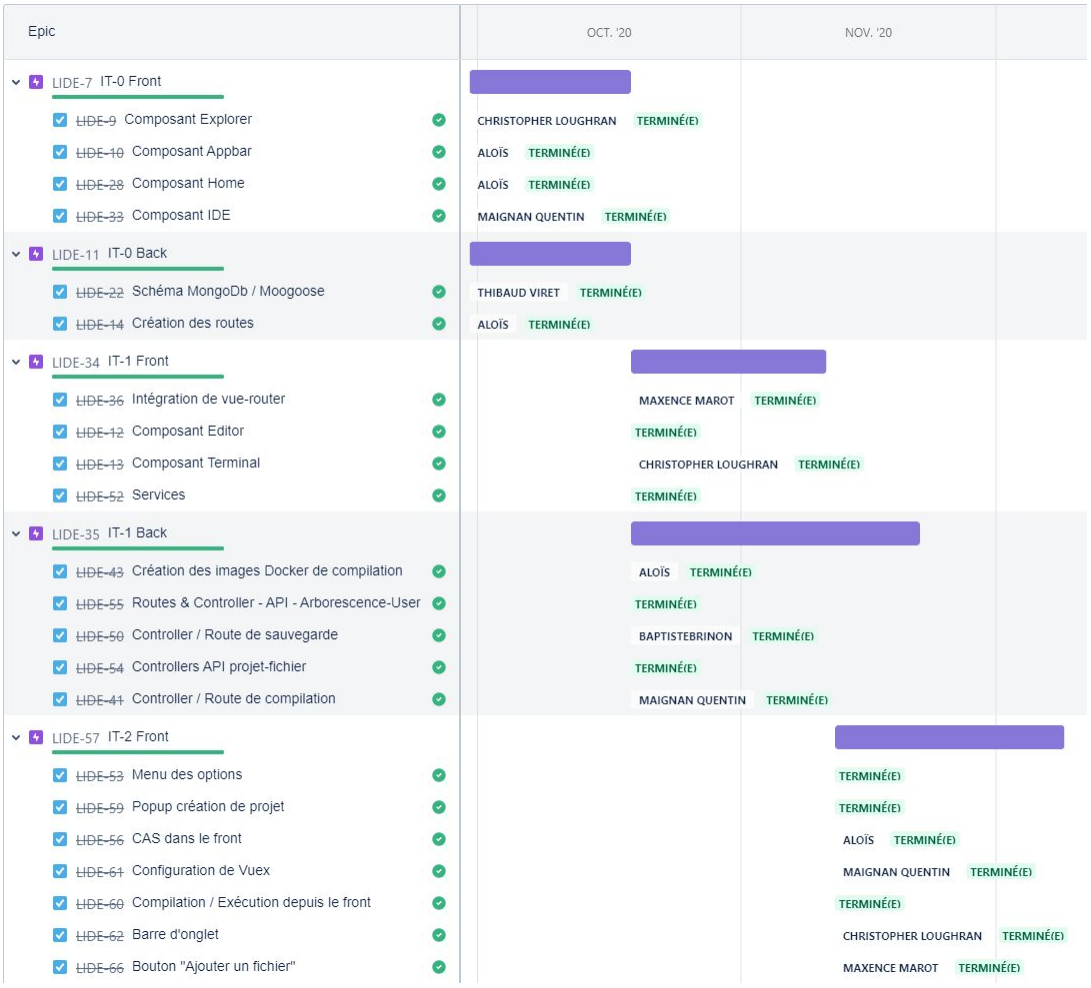



*Figure 0* : Capture d'écran des différentes tâches mères dans le planning*

Nous avons fait en sorte de répartir les tâches de manière à ce que les M1 puissent s'y retrouver, en fonction de leurs compétences de développement ou de leurs expériences à concevoir une application.

## 1.5. Échange et communication

Étant donné que la communication est importante dans la réalisation d'un projet, nous avons mis en place des solutions pour garder une communication fluide entre tous les acteurs du projet.
Premièrement, nous avons décidé d'organiser des réunions hebdomadaires afin de pouvoir avoir un temps d'échange et de mise au point avec tous. Ne pouvant plus se voir en physique à cause de ce *satané* covid, nous avons dû mettre en place un serveur Discord afin de réaliser les réunions à distance mais également pour avoir des canaux de discussion le reste du temps.

Durant ces réunions, nous avons demandé aux M1 de faire un tour de table pour :

- Présenter la réalisation de la tâche effectuée ou en cours.
- Donner un feed back sur les différents problèmes rencontrés.
- Apporter des points de précision sur les tâches.
- Demander de l'aide ou poser des questions.

# 2. Outils et technologies

## 2.1. Les outils de management

Afin d'organiser le bon déroulement du projet et de respecter nos objectifs, nous avons mis en place trois outils de suivi de projet.

Le premier est le logiciel ***Jira*** dédié à la gestion de projet de développement d'applications. Avec Jira, il est possible de :

- Suivre un projet selon la méthode Scrum.
- Travailler sur des tableaux Kanban.
- Schématiser une vue d'ensemble à travers une feuille de route.
- Consulter des rapports en temps réel  et prêts à l'emploi.
- Faire le lien avec un dépôt git pour une plus grande visibilité des développements.

Avec le contexte sanitaire, nous avons dû mettre en place un outil de communication à distance pour organiser nos réunions hebdomadaires.
Nous avons opté pour ***Discord*** qui est un logiciel de discussion instantanée. Cet outil nous a permis d'être disponibles à tout moment pour aider les M1 en cas de difficulté, de répondre à leurs questions et d'échanger tout au long du projet.

Pour la gestion du code nous avons utilisé le logiciel phare du développement, qui est bien évidemment ***Git***, sur la plateforme Github. Git est un système de versioning de code, il favorise le travail en équipe et permet de suivre l'avancée du projet.

## 2.2. Front-end

Pour le front-end nous avons choisis d'utiliser le serveur web NodeJS et d'y intégrer le framework VueJS. Nous avons fait ces choix pour :

- Simplifier la gestion d'installation des différents modules externes (Vuetify, Axios, CodeMirror, Xterm.js, etc)
- Créer une interface rapide et au goût du jour, pour que les utilisateurs puissent facilement prendre en main l'application. À l'aide de Vuetify, qui est une librairie de composants dédiée pour VueJS.
- Et pour sa simplicité d'utilisation.

De plus, nous utilisons VueJS sous son mode **Single App Application**, qui est différent d'un serveur web classique. Ce concept, très pratique pour construire des applications web, consiste à charger l'entièreté de l'application lors de la première requête vers le serveur. Ainsi tous les traitements sont effectués sur le navigateur, ce qui permet un rendu fluide et dynamique, efficace pour une application sur le web.

Nous avons intégré deux bibliothèques JavaScript :

- *CodeMirror* : permet l'intégration d'un éditeur de code supportant nativement la coloration syntaxique, l'auto-complétion, l'auto-indentation et d'autres fonctionnalités et ce, pour la plupart des langages. De plus, cette bibliothèque est personnalisable et supporte nativement le theming. Cette bibliothèque est notamment utilisée par les éditeurs intégrés à Firefox, Chrome et Safari.

- *Xterm.js* : permet l'intégration d'un terminal, ce terminal peut ensuite être connecté à un conteneur via un web-socket ssh. Cette bibliothèque est notamment utilisée par Portainer et Visual Studio Code et est donc en développement actif.

## 2.3. Back-end

Pour la réalisation d'un petit web service (API), ne comprenant que quelques routes, nous avons choisi d'utiliser un serveur NodeJS et son framework ExpressJS.

ExpressJS est un framework permettant de traiter des requêtes HTTP. Il est très léger et apporte peu de surcouches afin de garder des performances optimales. De plus, il est extrêmement simple à déployer.

Pour simplifier la relation entre le serveur backend et le serveur de base de données nous avons fait le choix d'utiliser Mongoose.
Mongoose est une librairie Javascript qui est couramment utilisée dans une application Node.js avec une base de données MongoDB. L'avantage d'utiliser Mongoose, est qu'il permet de définir des objets avec un schéma fortement typé, mappé sur un document MongoDB.

### 2.4. Base de données

Pour la base de données nous avons décidé d'utiliser MongoDB, pour rappel MongoDB est un serveur de gestion de base de données utilisant le paradigme no-sql, permettant une gestion plus fluide et plus libre des données. Toutes les données seront stockées sous la forme de collections JSON.

Nous avons voulu garder la même fluidité caractéristique du javascript dans notre gestion de la base de données. Ainsi, nous avons pu facilement faire la transition entre les objets javascript et notre base de donnée (via dé-sérialisation json).

Cela a permis une vraie continuité entre le code et la base de données. L'objectif était de conserver une cohérence entre les objets métier, la base de données et le code. De plus, l'utilisation de MongoDB nous permet de nous passer d'un système de fichiers puisque l'enregistrement des fichiers utilisateur sera fait directement au sein des collections. MongoDB permet en effet de remplacer le système de fichier en utilisant, si nécessaire, la fonctionnalité GridFS prévue à cet effet.

En résumé, l'utilisation de MongoDB s'inscrit dans une optique de simplification de l'architecture de l'application, tout en étant cohérente avec les technologies mises en place.

### 2.5. Déploiement & Compilation

Pour la gestion du déploiement et des compilations/exécutions nous avons opté pour la technologie de conteneurisation Docker. Nous avons choisi cette technologie car elle permet un déploiement et une configuration simple et rapide des différents serveurs (web, api, socket et bdd). Grâce à Docker Compose, l'installation et le déploiement de ces services se font en une seule ligne de commande. Chaque serveur étant conteneurisé sur la machine hôte et considéré comme un service indépendant. Une autre raison de l'utilisation de conteneurs Docker pour déployer nos serveurs, est d'avoir une architecture identique pour tous. Grâce à cela, l'ensemble de l'application est déployable facilement sur une architecture Unix (Linux et Mac), avec un portage Windows possible. Cela permet notamment de développer sur l'application sans avoir à installer quoique ce soit d'autre que Docker lui-même.

Pour la compilation et l'exécution du code des étudiants, Docker présente beaucoup d'avantages. Effectuer ces actions dans des conteneurs créés à la volée permet de retirer la notion de sécurité dans la compilation et l'exécution de programmes. En effet, chaque conteneur étant indépendant et isolé, il est impossible via le code exécuté de porter atteinte à l'intégrité de l'application. De plus, la prise en compte de nouveaux langages peut se faire facilement en ajoutant une nouvelle image Docker comportant le compilateur du langage visé.

Enfin, Docker permet une supervision facile des serveurs à distance. Grâce à portainer.io, qui est un outil de gestion de conteneurs Docker. Tous les services étant conteneurisés, il nous est possible de les gérer depuis cette interface. Ainsi on peut lancer ou arrêter un service, obtenir les logs ou accéder directement au conteneur via un terminal pour y effectuer des actions. Les conteneurs d'exécutions des étudiants sont également concernés.

# III - Architecture technique du projet

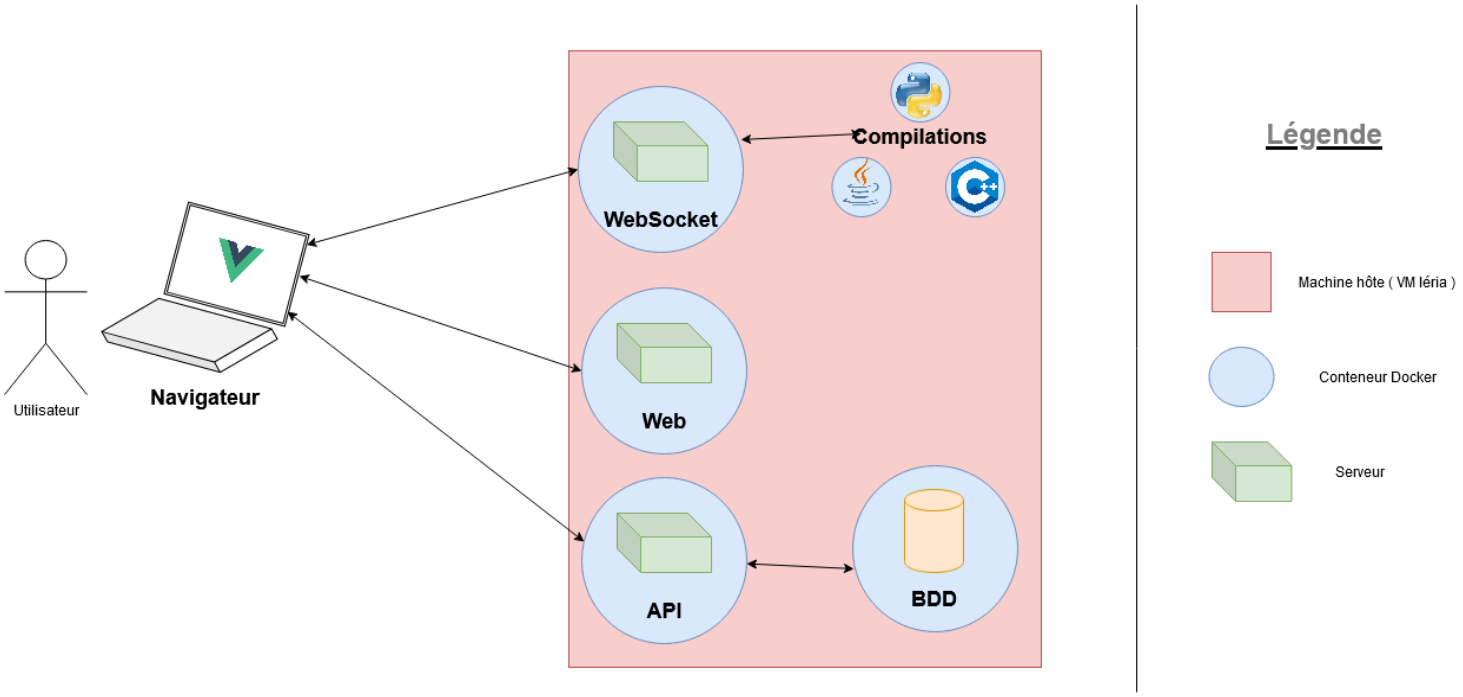



*Figure 1 : Schéma de l'architecture globale de l'application*

Le projet s'articule autour de 4 serveurs :

- Un serveur web qui va transmettre la **Single App Application** au navigateur.
- Un serveur de base de données pour **stocker les données**.
- Un serveur API pour les **traitements back-end**, et le **lien entre navigateur et BDD**.
- Un serveur de WebSocket pour faire **le lien entre le terminal de l'utilisateur et son programme** dans le conteneur d'exécution.

Nous expliquerons avec plus de détails leurs différents fonctionnement ci-après.

# 1. Interface utilisateur (front-end)

## 1.1. Squelette

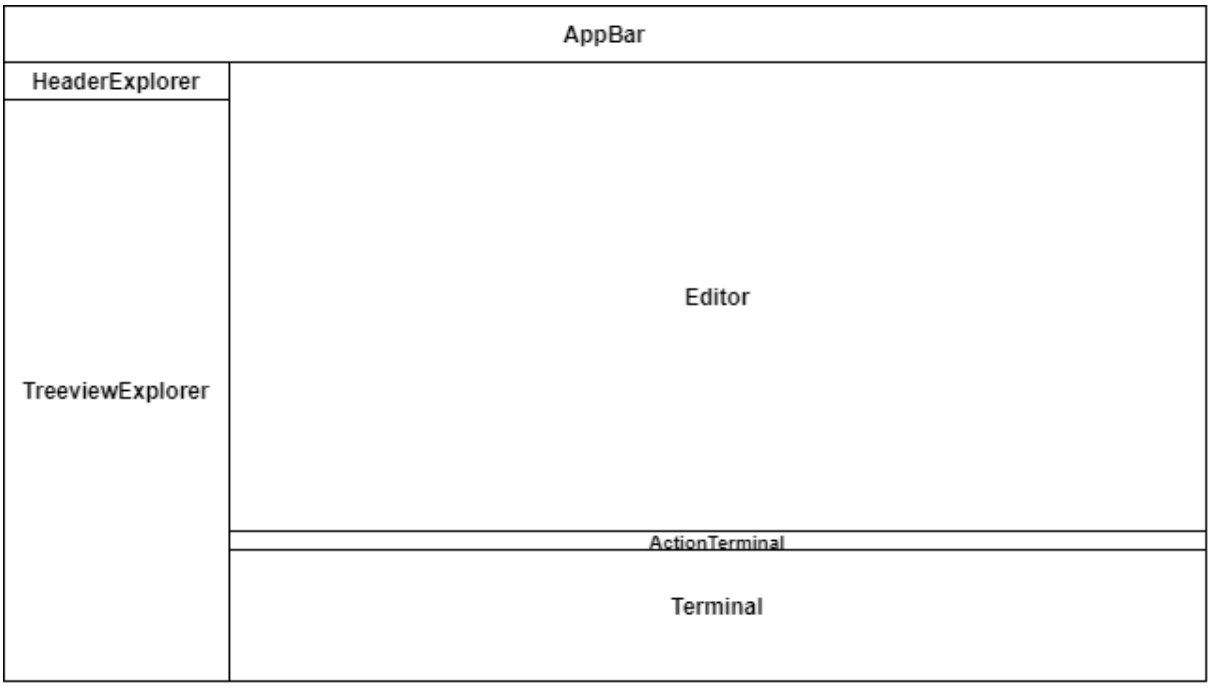



*Figure 2 : Squelette de l'architecture IHM de l'application*

Comme vous pouvez le voir sur la figure ci-dessus, nous avons répartis l'application en 6 composants principaux :

- ***AppBar :*** Barre d'outils de l'application qui offre l'accès à certaines fonctionnalités (aide, paramètre, déconnexion, etc).
- ***Explorer :*** Permet la gestion (ajout, renommage, suppression, exportation) des projets et fichiers de l'application.
- ***HeaderExplorer*** : Contient le bouton de création du projet ainsi que le bouton d'export.
- ***Editor*** *:* Fournit un éditeur de code avec coloration syntaxique. Contient les boutons d'exécution et de sauvegarde.
- ***Terminal*** *:* Fournit un terminal gérant les entrées/sorties de l'exécution.
- ***ActionTerminal*** : Contient le bouton de nettoyage du terminal ainsi que le bouton d'arrêt de l'exécution.

Ces composants sont des briques à part entière de l'application et fonctionnent indépendamment les uns des autres. Cela permet notamment de découpler les différentes fonctionnalités de l'application et ainsi d'en faciliter la maintenance.

## 1.2. Le store

Afin de faciliter la communication entre les composants, nous avons mis en place un gestionnaire d'état. Pour cela, nous avons utilisé la bibliothèque Vuex.

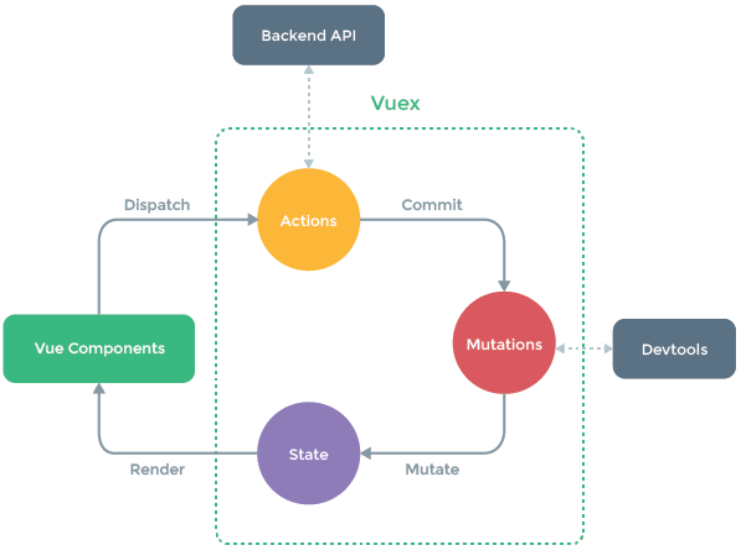



*Figure 3 : Processus du gestionnaire d'état de VueX*

VueX sert de zone de stockage de données centralisée pour tous les composants. Il permet de créer des actions qui vont pouvoir changer l'état des composants en réaction aux actions utilisateur.

VueX a été primordiale dans la réalisation de l'application et notamment sur la gestion de l'ouverture des fichiers et des onglets.

# 2. Interface logique (back-end & API)

## 2.1. Vue d'ensemble

L'API a trois fonctions principales:

- Permettre d'interagir avec la base de données
- Permettre l'exécution des fichiers
- Gérer le token de session
- Permettre l'export des fichiers au format ZIP

Pour chacunes des fonctions de l'API, nous avons fait le choix d'externaliser le code dans des services ayant chacun leur rôle propre, ainsi nous retrouvons :

- UserService : gestion des utilisateurs en BDD
- ProjectService : gestion des projets en BDD
- FileService : gestion des fichiers en BDD
- SessionService : génération et validation des token JWT de session
- ExecutionService : gestion de la compilation/exécution des fichiers
- ExportService : gestion de l'export des projets

A l'émission d'une requête HTTP vers l'API, le routeur va intercepter la requête, et rediriger vers le contrôleur puis le service associé à la requête. La réponse sera ensuite émise par le contrôleur.
Le service de session est un peu particulier puisqu'il est appelé avant chaque contrôleur afin d'autoriser la requête.

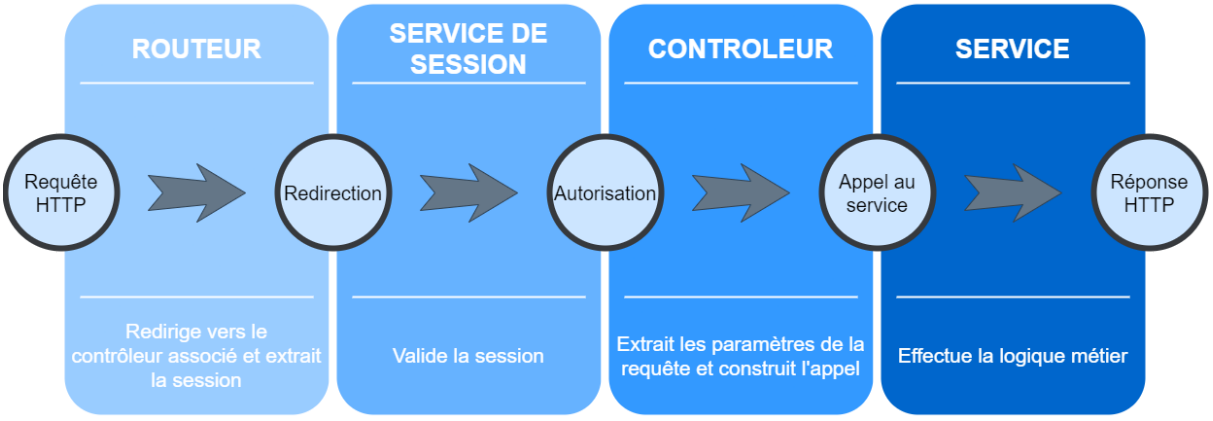



*Figure 4 : Fonctionnement général de l'API*

Par exemple, dans le cas d'un renommage de fichier, le cheminement des données du clic à la notification de réussite sera le suivant:

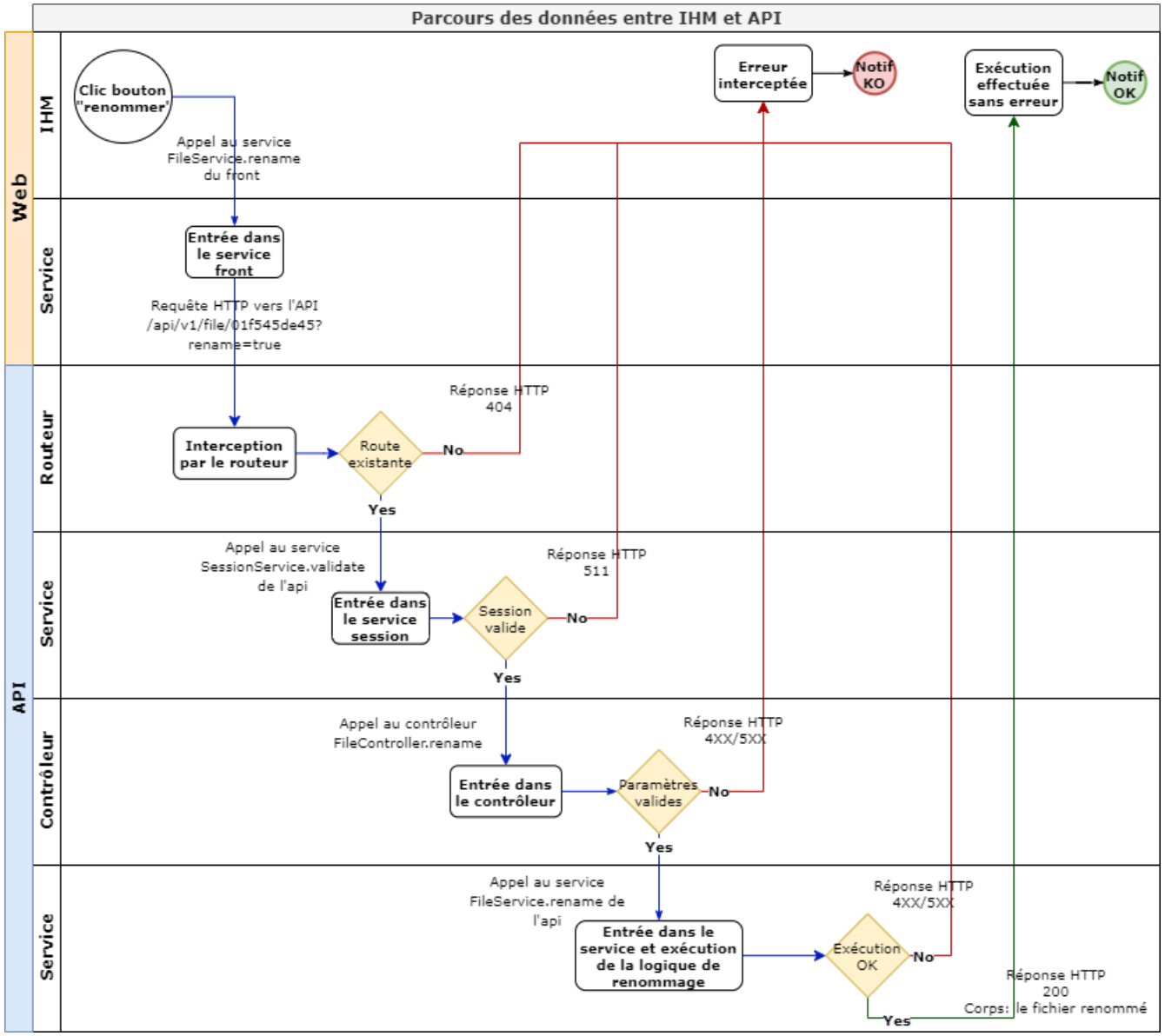



*Figure 5 : Exemple de parcours des données entre IHM et API*

## 2.2. Interaction avec la base de données

Un des rôles principal de l'API n'est autre que l'accès en lecture/écriture de la base de données. Pour ce faire, nous avons fait le choix d'utiliser le framework Mongoose puisque celui-ci est le framework de référence pour le couple **MongoDB/Node.js**. Il permet d'implémenter très facilement le paradigme d'accès à la base de données CRUD. Pour rappel, CRUD est un acronyme signifiant: **C**reate, **R**ead, **U**pdate, **D**elete (Créer, Lire, Mettre à jour, Supprimer).

Ainsi, le service d'accès à la BDD est composé de 3 sous services:
*Le symbole [X] indique l'opération associée à la méthode du service.*

UserService :

- **[C] create** → Créer un utilisateur à partir de son identifiant puis le retourne
- **[R] get** → Retourne un utilisateur selon son identifiant si celui-ci est existant, sinon fait appel à UserService.create
- **[U] rename** → Renomme un utilisateur à partir de son identifiant puis le retourne *(non utilisé)*
- **[D] delete** → Supprime un utilisateur *(non utilisé)*

ProjectService :

- **[C] create** → Créer un projet puis le retourne
- **[R] get** → Retourne le projet
- **[U] rename** → Renomme le projet puis le retourne
- **[D] delete** → Supprime le projet

FileService :

- **[C] create** → Créer un fichier puis le retourne
- **[R] get** → Retourne le fichier
- **[U] rename** → Renomme le fichier puis le retourne
- **[U] save** → Sauvegarde le contenu du fichier puis le retourne
- **[D] delete** → Supprime le fichier

### 2.3. Compilation et exécution des programmes

Pour bien comprendre le fonctionnement de la compilation et de l'exécution de programmes à distance, il est nécessaire de comprendre ce qu'est l'API Docker. En effet, ce service est parfois méconnu et nécessite une activation manuelle dans la configuration de Docker.

Ce service permet de gérer la totalité des conteneurs Docker de la machine, via l'envoi de requêtes HTTP. Dans notre cas, il va permettre de se connecter à distance au conteneur d'exécution d'un programme pour que l'utilisateur puisse interagir avec depuis son IHM. Pour cela, nous utilisons une des fonctionnalités disponibles de l'API : l'attachement par WebSocket.
( cf. https://docs.docker.com/engine/api/v1.41/#operation/ContainerAttachWebsocket )

Cependant, le navigateur de l'utilisateur n'étant pas sur le même réseau que celui des conteneurs Docker, il est impossible d'ouvrir une connexion socket entre les deux sans ouvrir l'API Docker vers l'extérieur. Cette solution n'est évidemment pas réalisable dans le sens où cela serait une porte ouverte vers la gestion de la totalité des conteneurs (serveurs y compris) et serait sans aucun doute une énorme faille de sécurité. C'est pourquoi nous avons décidé de placer un serveur de WebSocket entre les deux, afin de gérer l'utilisation de cette API et d'en contrôler l'accès.

Pour détailler le fonctionnement d'une compilation/exécution depuis l'application, il faut se rendre compte du scénario déclenché par l'action du clic sur le bouton d'exécution :

1. Un appel depuis le navigateur est effectué vers l'API à l'adresse **/execute/{fileid}** qui déclenche le contrôleur d'exécution.
2. L'API fait appel à son **FileService**, pour récupérer en base le fichier ciblé.
3. L'API génère ce fichier dans le filesystem de la machine hôte.
4. L'API démarre un **nouveau conteneur Docker** basé sur l'image du langage correspondant au fichier. Il récupère ensuite le fichier précédemment créé grâce à un volume sur ce dernier.
5. L'API répond au navigateur avec l'**ID du conteneur créé**.
6. Le conteneur créé exécute un script, qui le place en attente d'un input (grâce à la fonction read en shell).
7. Le navigateur ouvre **une connexion socket** vers le serveur de WebSocket, et lui transmet en premier lieu son **token de session**, ainsi que l'**ID du conteneur** visé.
8. Si les informations sont vérifiées, alors le serveur de WebSocket ouvre **une autre connexion socket vers l'API Docker**, et connecte la sortie d'un des sockets vers l'entrée de l'autre, et inversement.
9. Un fois le lien fait, le serveur de Websocket envoie un caractère "\n" pour signaler au conteneur que le lien avec le navigateur de l'utilisateur est effectué, **débloquant ainsi la compilation et l'exécution du fichier**.
10. Les sorties générées par le conteneur traversent le premier socket, puis le deuxième. Les entrées envoyées par l'utilisateur effectuent le chemin inverse.
    Lorsque le programme est terminé (correctement ou non), le conteneur est **automatiquement détruit** et les connexions sockets sont closes.

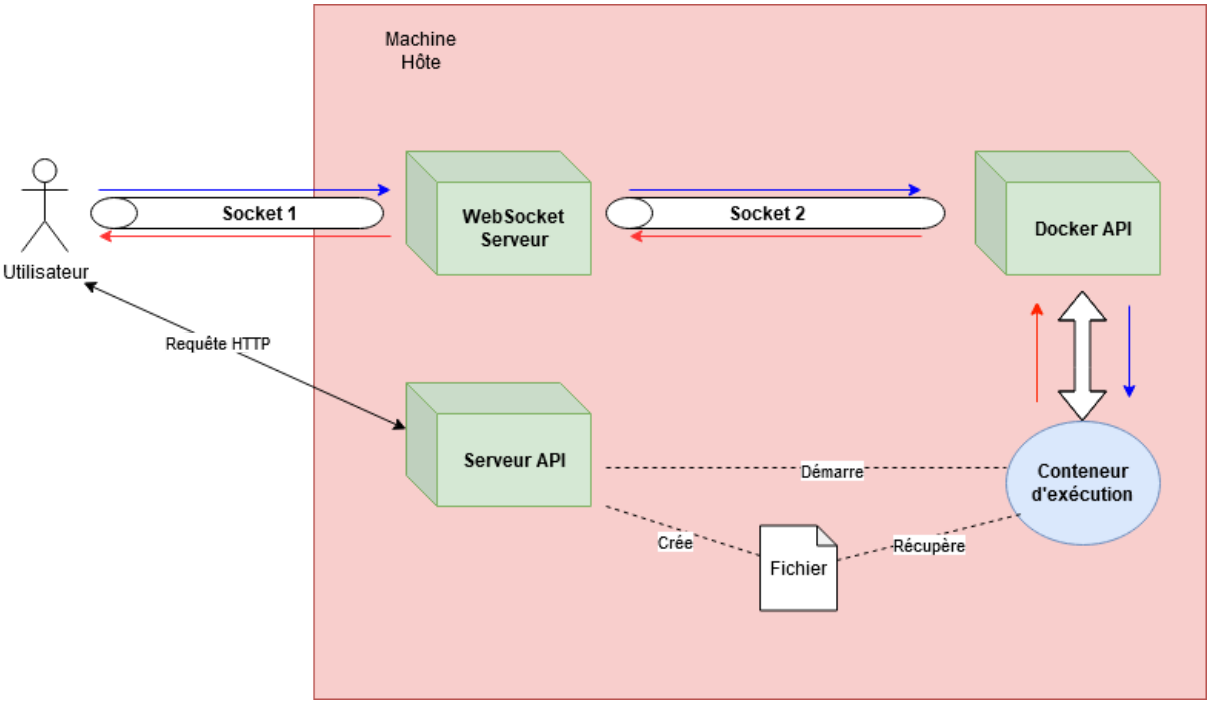



*Figure 6 : Schéma représentatif de l'exécution de programme à distance*

## 2.4. Génération et validation des sessions

L'application étant une Single App Application, c'est-à-dire chargée entièrement sur le navigateur, notre API a besoin d'être accessible depuis le web. C'est pourquoi il était nécessaire de sécuriser cette dernière pour en contrôler l'accès. Nous avons choisi pour cela d'utiliser la technologie **JSON Web Token,** qui nous permet de créer facilement un système de session grâce à la génération de token sécurisé.

Lorsqu'un utilisateur se connecte sur l'application, une requête est émise vers l'API pour lui générer un token de session. Ainsi, nous générons le token seulement et seulement si le ticket CAS obtenu est valide. La génération de ce token se fait simplement, à partir d'une clef privée définissable par un administrateur. Le token est envoyé dans la réponse vers le navigateur de l'utilisateur, qui sera stocké dans sa mémoire.

Ce système permet ainsi d'identifier chacune de ses requêtes vers l'API en fournissant le token pour chaque traitement demandé. Chacune des routes de l'API est protégée (excepté celle de la création du token) par une vérification sur ce token de session. Le contrôleur de session est toujours exécuté avant un autre, il vérifie la validité du token en le décryptant. Si le token est valide, alors l'utilisateur est identifié, et les traitements prévus peuvent se poursuivre, sinon une erreur HTTP 511 est retournée et l'utilisateur sera automatiquement redirigé vers la page d'authentification du CAS.

## 2.5. Inventaire des routes

L'API contient un certain nombre de routes utilisables par l'IHM, nous détaillons ci-dessous leur utilisation basique.

La présence en header d'un token de session valide est requise afin d'autoriser la requête et d'identifier l'utilisateur. Si l'utilisateur identifié tente d'accéder à une ressource qui ne lui appartient pas, la requête sera refusée.
Seule la route de génération de la session est accessible publiquement (surlignée en vert).

Il est à noter que pour toutes les routes, si la requête n'est pas valide d'un point de vue technique (ex: mauvais paramètre) ou métier (ex: création d'un fichier déjà présent), l'API retourne des erreurs que nous ne détaillerons pas ici.

Toutes les routes sont à préfixer par : **/api/v1**

| Route | Méthode | Paramètres | Corps | Réponse |
|---|---|---|---|---|
| /session | GET | **ticket**: ticket retourné suite à l'identification CAS | N/A | Token de session JWT |
| /validateSession | GET | N/A | N/A | Statut de validité de la session passée en en-tête |
| /user | GET | N/A | N/A | L'utilisateur associé à la session |
| /user/projects | GET | N/A | N/A | Liste des projets et fichiers de l'utilisateur |

| | | | | |
|---|---|---|---|---|
| /project/ | POST | **projectname**: nom du projet à créer | N/A | Le projet créé |
| /project/{projectid} | GET | N/A | N/A | Le projet |
| /project/{projectid} | DELETE | N/A | N/A | OK : suppression du projet |
| /project/{projectid} | PUT | **rename**: true/false | **newprojectname**: nouveau nom du projet | Le projet renommé |
| /file/ | POST | **filename**: nom du fichier à créer | N/A | Le fichier créé |
| /file/{fileid} | GET | N/A | N/A | Le fichier |
| /file/{fileid} | DELETE | N/A | N/A | OK : suppression du fichier |
| /file/{fileid} | PUT | **rename**: true/false<br>**save** : true/false | **newfilename**: nouveau nom du fichier<br>**content**: contenu à sauvegarder | Le fichier mis à jour en fonction des paramètres (renommage ou sauvegarde) |
| /export | GET | N/A | N/A | Lien de téléchargement des projets/fichiers au format ZIP |
| /execute/{fileid} | POST | N/A | N/A | OK : lancement de l'exécution |
| /killexec | POST | N/A | N/A | OK : terminaison de l'exécution |

# 3. Base de données

## 3.1. Contexte

La technologie MongoDB étant conçue autour du paradigme NoSQL ayant diverses implémentations, il convient de rappeler quelques termes autour de MongoDB:

- **Document** : Un document représente un objet métier contenant divers attributs.
- **Collection** : Une collection est un ensemble de documents d'un même type.
- **JSON** : Le format utilisé par mongodb pour représenter les documents. Un objet JSON est composé d'attributs sous forme de clés-valeurs et est basé sur les objets javascript.

Bien que les technologies SQL et NoSQL n'ont que peu de points communs et ce notamment du fait que la technologie NoSQL n'impose pas de schéma structuré, nous pouvons tout de même simplifier en disant qu'une collection représente une table et qu'un document représente une entrée de table. Les colonnes quant à elles sont non structurées et sont représentées par les attributs d'un document.

Une des raisons principales nous ayant poussé à utiliser cette technologie réside dans son format d'enregistrement des données, le JSON. En effet, l'utilisation du JSON permet une sérialisation/désérialisation des données en objets Javascript de manière transparente, ce qui facilite grandement l'interfaçage entre la base de données et les objets métier.

De plus, le format orienté document de cette technologie se montre très efficace pour cette application où les seules données à enregistrer sont des utilisateurs, des projets ainsi que des fichiers. Cela nous permet par exemple d'enregistrer une liste de fichiers au sein des projets, car contrairement à une base de données relationnelle, il est possible d'introduire des tableaux de données au sein même d'un document.

Enfin, dans l'éventualité où le nombre d'utilisateurs de l'application devait augmenter, il est possible de nativement mettre en place un système de distribution de la base de données. Cela permettra deux choses: réduire les temps de réponse de la base de données et permettre la mise en place d'un système de réplication des données, limitant ainsi les risques de pertes de données ou de "down-time".

### 3.1. Schéma

Le schéma de base de données s'articule autour de 3 collections:

- **Users** : Contient les documents représentant les utilisateurs
- **Projects** : Contient les documents représentant les projets utilisateurs
- **Files** : Contient les documents représentant les fichiers contenus dans les projets

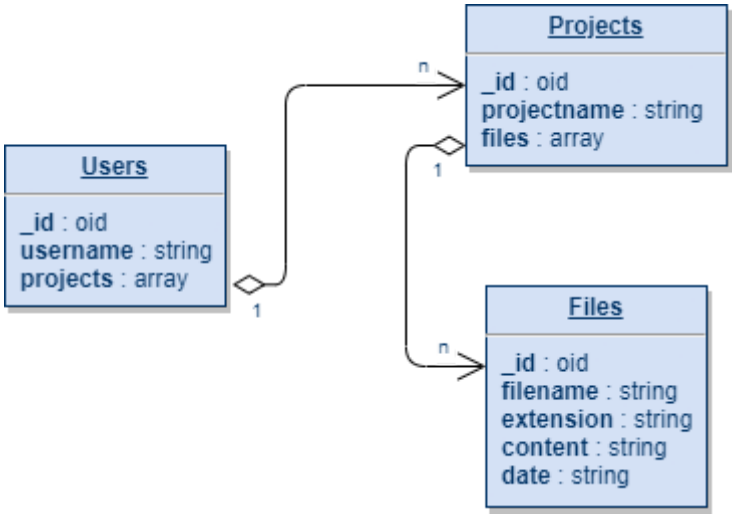



*Figure 7 : Schéma de la base de données*

Les documents de type User sont constitués de :

- **_id** : Id unique autogénéré par MongoDB permettant d'identifier l'utilisateur
- **username** : Identifiant CAS de l'utilisateur
- **projects** : Un tableau d'id de projets, initialisé à vide

Les documents de type Projects sont constitués de :

- **_id** : Id unique autogénéré par MongoDB permettant d'identifier le projet
- **projectname** : Nom du projet
- **files**: Un tableau d'id de fichiers, initialisé à vide

Les documents de type Projects sont constitués de :

- **_id :** Id unique autogénéré par MongoDB permettant d'identifier le fichier

- **filename** : Nom du fichier
- **extension** : Extension du fichier permettant de définir son langage de programmation
- **content** : Contenu du fichier, initialisé à vide (Permet de s'abstraire un système de fichiers)
- **date** : Date de création du fichier, généré à la création (non utilisé)

# 4. Authentification (CAS)

L'élément central de l'authentification du projet est le CAS, il permet de lier notre application au système d'authentification global de l'Université d'Angers, nous permettant ainsi de d'empêcher l'accès à toute personne extérieure à l'Université.
Le CAS est couplé à une gestion de session par token interne à l'application, expliqué plus haut.

## 4.1. Fonctionnement général de l'authentification

Tout d'abord certaines routes ne sont pas protégées pas le CAS :
- La page Home
- La route menant au login
- La route menant au logout

Toutes les autres routes, c'est-à-dire l'accès à la page principale de l'application ainsi que les appels à l'API sont protégées.
Un utilisateur a donc un accès à la page Home, puis il doit se connecter pour utiliser l'application.

La connexion et la gestion du CAS sont regroupées dans une fonction surchargeant le routeur de base de l'application, ainsi cette fonction est appelée à chaque fois que l'on contacte une URL.

Cette fonction gère plusieurs types d'appels:

1. Si l'URL contient **/login** alors on est redirigé vers l'URL de login du CAS, avec l'URL du projet en argument qui permettra au CAS de nous rediriger.
2. Si l'URL contient **/logout** on fait appel à l'URL de déconnexion du CAS en fournissant également notre URL en argument.
3. Si l'URL est protégée, soit on redirige sur **/login** dans le cas où il n'y a pas de token de session présent dans le navigateur de l'utilisateur. S'il y en a un, alors on vérifie sa validité et on récupère son identifiant.
4. Si une URL n'est pas protégée, la fonction ne fait rien.
5. L'appel le plus complexe est celui effectué lors de la réponse de la connexion par le CAS, la gestion de cet appel peut être divisé en plusieurs étapes :

    a. Connexion : on effectue un appel depuis le site vers le CAS en indiquant l'URL de l'application dans la requête
    b. Une fois l'identification validée, le CAS redirige sur l'URL du site passé en paramètre en ajoutant un ticket

c. L'application front récupère le ticket et l'envoi à l'API
d. L'API effectue une validation de ce ticket en faisant un appel au service de validation du CAS avec le ticket et l'URL de l'application en argument.
e. Si le ticket est validé alors le CAS va envoyer l'identifiant unique de l'utilisateur que l'application utilisera comme identifiant.
f. Une fois l'identifiant récupéré, l'API crée la session JWT de l'utilisateur pour une validité d'une heure.
g. L'API renvoi la session et l'identifiant qui seront gardés en mémoire dans l'application front

## 4.2. Schéma récapitulatif de l'authentification

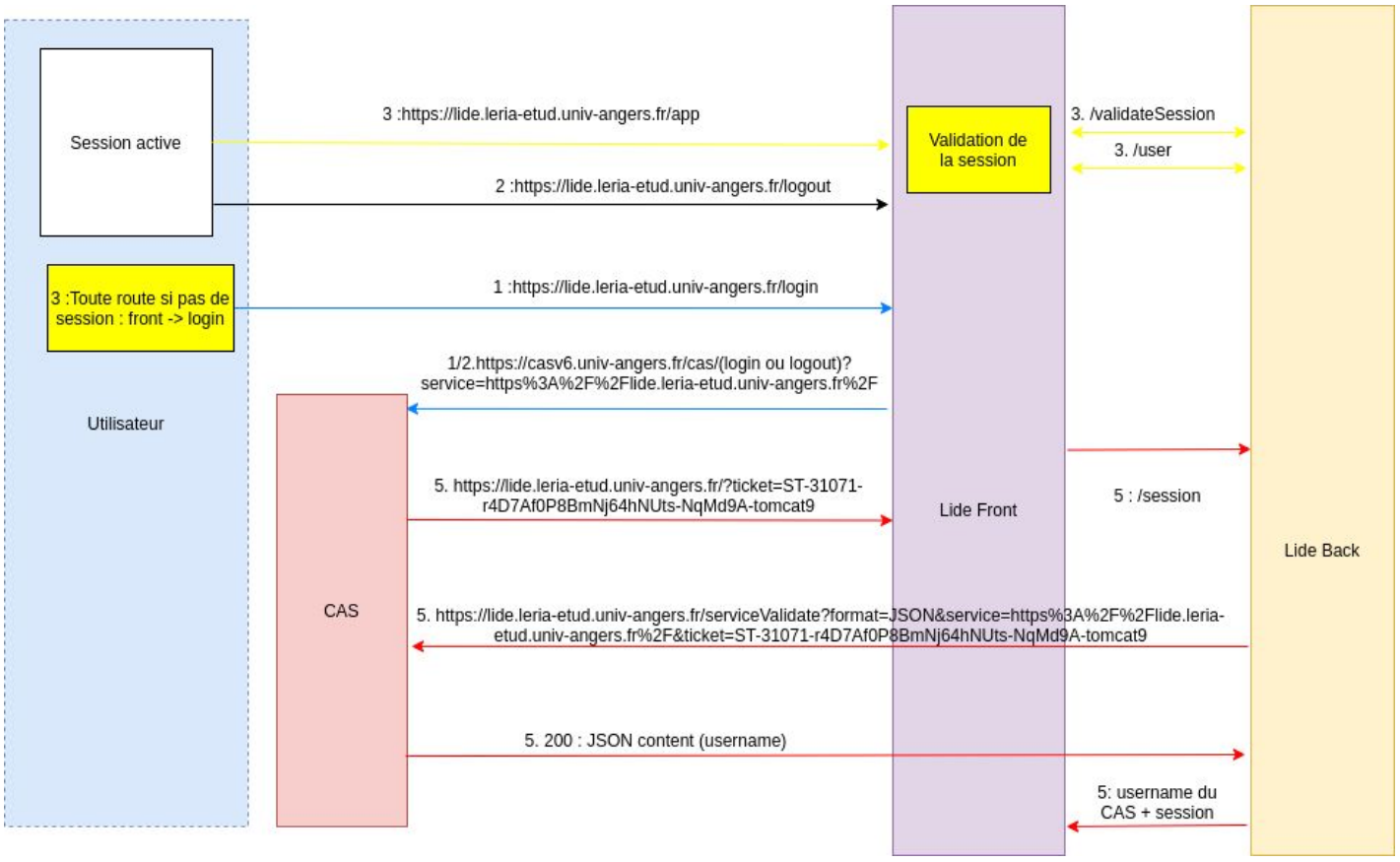



*Figure 8 : Schéma des interactions avec le CAS*

# IV - Les fonctionnalités

La première phase du projet fut réalisée en collaboration avec les M1, la deuxième quant à elle fut menée exclusivement par les M2. Nous avons notamment pu reprendre certains développements des M1, ainsi qu'ajouter de nouvelles fonctionnalités.

## 1. Fonctionnalités améliorées

La première et grande action que nous avons décidé de mener lors de notre reprise du projet, a été d'harmoniser et d'optimiser la qualité du code.

Pour cela, nous avons d'abord repensé la base de données, afin de mieux utiliser le paradigme nosql et de correspondre au mieux à l'esprit de MongoDB. L'ensemble des contrôleurs de l'API a dû être revu en conséquence, permettant un code plus "propre" et une meilleure gestion des erreurs. Nous avons de plus, introduit certaines règles telles que l'interdiction de la création ou du renommage de fichiers lorsque le nom de ce dernier est déjà existant au sein de son projet.

Côté front, nous avons fait un gros travail afin de rendre l'application responsive. En effet l'ancienne version de l'application ne gérait pas différentes tailles d'écrans. Désormais, l'application s'adapte sans trop de soucis à toute taille d'écrans, et également au format mobile (même si cet aspect pourrait encore être amélioré).
Nous avons aussi revu le design de certains éléments tel que l'explorateur de fichiers.
Nous avons également amélioré la gestion des onglets pour l'ouverture et la fermeture des fichiers. Tous les cas alternatifs autre que le cas nominal ont été pris en compte et les bugs identifiés ont été corrigés.

Enfin, la gestion de l'identification (CAS/session) a été fiabilisée, son code simplifié. Nous avons notamment ajouté une étape de validation de la session avant d'accéder à la page utilisateur.

## 2. Fonctionnalités ajoutées

Le premier ajout a été de sécuriser les appels vers l'API, ce qui n'était pas le cas avant, laissant ouvert une faille disponible pour des potentiels utilisateurs malveillants. Il s'agit d'avoir ajouté la gestion du token JWT expliqué plus tôt, afin de s'assurer que chacune des actions demandées à l'API soit légitime et que son émetteur soit vérifié.
La connexion/déconnexion de l'application se gère mieux grâce à ce token de session, et la reconnexion dynamique est gérée : un utilisateur dont le token ne serait plus valide se verra au mieux reconnecté automatiquement (si sa connexion CAS est toujours effective), au pire redirigé vers la page de connexion du CAS avant de retrouver son travail là ou il en était.

Concernant l'exécution de code nous avons ajouté une valeur de timeout paramétrable par un administrateur rendant impossible l'exécution de programmes infinis. Les performances des conteneurs étudiants ont été partagées ce qui permet qu'un conteneur exécutant une boucle infinie par exemple, ne puisse plus ralentir voir bloquer l'ensemble des autres services de l'application, la rendant de fait inaccessible.

Un bouton d'arrêt a été ajouté pour permettre à l'utilisateur de stopper prématurément une exécution en cours.

Nous avons ajouté dans l'IHM la gestion des raccourcis claviers les plus utilisés, à savoir la sauvegarde automatique, la sélection totale et le copier-coller.
Toujours au sein de l'IHM, nous avons ajouté des notifications permettant d'informer l'utilisateur de la bonne exécution des actions, ou au contraire, des erreurs rencontrées.
De plus, nous avons ajouté la possibilité d'exporter la totalité des projets d'un utilisateur et ses fichiers au format ZIP en un clic.

Comme dernière fonctionnalité ajoutée, nous avons mis en place une première itération de la compilation multi-fichiers en C++. Cette fonctionnalité permet pour le moment de compiler tous les fichiers C++ (.cpp) d'un même projet et d'ensuite effectuer la liaison entre les fichiers générés (.o).
Nous avons de plus ajouté la possibilité de créer des fichiers d'en-tête (.h) qui seront automatiquement importés lors de la compilation.
A noter qu'il existe une limitation à cette fonctionnalité: il n'est pas possible d'avoir deux fichiers C++ comportant une fonction main dans un même projet, l'application ne permettant pas en l'état de définir quel fichier "main" à prendre en compte. Il est cependant facile de contourner cette limitation en créant un projet pour chaque exécutable C++.

# V - Avenir du projet

## 1. Améliorations graphiques

Pour l'aspect graphique, on peut identifier quelques améliorations possibles à apporter. Tout d'abord il serait intéressant de faire un menu de paramétrage afin de pouvoir personnaliser l'aspect graphique du code, ou encore de passer l'éditeur en mode sombre. Tout ceci pourrait se paramétrer dynamiquement grâce à CodeMirror.

Il serait intéressant d'ajouter une page de tutoriel lors de la première connexion d'une personne ou alors créer des infobulles pour chaque fonctionnalité de l'application. afin de guider au mieux les nouveaux utilisateurs.

## 2. Améliorations fonctionnelles

Hormis, la fermeture automatique des parenthèses et des délimiteurs de bloc, Il n'y a actuellement pas d'autocomplétion, il serait intéressant de l'implémenter.
A noter que la communauté CodeMirror est actuellement en train de développer une version 6 de CodeMirror, cette version est une réécriture complète de la bibliothèque et comportera à terme l'autocomplétion sur les langages les plus courants. Il pourrait donc être intéressant de mettre à jour CodeMirror lorsque la version 6 sera stable.

Une autre amélioration majeure serait l'ajout d'un débogueur sur l'application, ce qui semble difficilement réalisable étant donné la diversité des langages gérés ainsi que la communication limitée par l'API docker entre l'utilisateur et le conteneur d'exécution. Il est

cependant techniquement envisageable de le mettre en place, cela mériterait un sujet à part entière.

Une amélioration plus facilement réalisable mais qui présente quand même ses défis serait d'ajouter la possibilité d'importer dynamiquement des librairies externes au langage de base par exemple en se basant sur un fichier requirements pour python.

Une autre amélioration majeure serait l'ajout de profils sur l'application: étudiant, professeur, administrateur. Le professeur pourrait par exemple avoir accès au code des étudiants directement depuis son interface, pour les aider à déboguer leur code à distance.

# VI - Bilan management

## 1. Gestion de projet

Le management de projet a été une expérience extrêmement enrichissante. Nous nous sommes rendu compte que porter un projet n'était pas chose aisée, et que la responsabilité d'une équipe pouvait parfois être compliquée. Néanmoins nous tirons toutes les leçons et enseignements que ce projet a pu nous apporter.

Nous avons pu réaliser tout le parcours nécessaire à la réalisation d'un projet. Ce parcours est universel et donc applicable en entreprise. Nous nous sommes confrontés à l'étude de faisabilité, les spécifications fonctionnelles, spécifications techniques et même l'écriture d'un devis, le tout en simulant l'interaction avec un client.
Nous avons pu mettre en application les méthodes utilisées en entreprise comme la méthode agile SCRUM que nous avons choisi d'utiliser pour notre équipe. Nous avons constaté aussi l'importance des rapports de réunion ainsi que des échanges écrits.

## 2. Gestion d'équipe

Nous avons également eu la chance d'avoir une équipe d'étudiants de M1 à gérer, ce qui nous a offert une expérience unique de chef de projet/manager dans notre parcours. Nous savons enfin ce qu'est la responsabilité d'un projet, de déléguer que d'encadrer un développement.

Notre relation avec les M1 a bien débuté, malgré quelques difficultés de maîtrise des technologies choisies, aspect plutôt normal pour une équipe débutante. Nous avons pu les encadrer et leur expliquer la marche à suivre afin qu'ils puissent avancer et bien assimiler le projet.

Les développements ont avancé avec une motivation et une efficacité très variable en fonction des personnes. Parfois le développement de notre équipe ne correspondait pas à nos consignes, parfois le développement d'une tâche n'avançait pas de semaine en semaine.

Grâce à nos réunions hebdomadaires, nous pouvions discuter avec les personnes concernées, ce qui a permis de régler les problèmes sans les laisser en suspens. Ce fut une bonne expérience en termes de communication dans une équipe et pour mieux sonder nos responsabilités en tant que chef de projet.

Dans les dernières semaines, le rythme a fortement augmenté, les M1 progressaient à une grande vitesse et le cahier des charges a pu être rempli. Nous avons pu finaliser notre encadrement en les accompagnant dans leur présentation. Finalement nous avons assisté à cette dernière avec les clients, le résultat a été très concluant et les clients satisfaits.

## 3. Nos limites

Nous avons tout de même fait quelques erreurs lors de notre management: nous avons mal anticipé les difficultés et la timidité de certaines personnes, nous aurions pu être plus présent et prendre les devants au lieu de leur dire de venir nous voir s'il y avait un problème. Nous aurions également dû plus insister sur les exercices que nous leur avons fournis pour s'entraîner.
Nous aurions également pu mieux peser les forces et les faiblesses de chacun afin de leur donner un travail adapté. Par exemple, nous avons constaté qu'une personne peu motivée pouvait avoir un regain de motivation en travaillant en binôme.

## 4. Ressenti

Nous pouvons dire que la partie management s'est bien passée, cela a été une expérience agréable et innovante pour nous.
La gestion d'une équipe sera probablement une situation que nous rencontrerons dans le monde professionnel et ce genre d'exercice permet de se préparer au management d'équipe.

# VII - Conclusion

## 1. Contexte et notre rapport avec le projet

Lors de notre première année de Master, nous avons travaillé sur une ancienne version de ce projet en tant que développeurs. Nous avons gardé une mauvaise expérience car nous n'avons pas pu apporter la plus-value que l'on espérait.

Nous avions récupéré un projet qui avait déjà une certaine ancienneté, nous étions la troisième année de développement de ce projet. Sur le papier nous devions faire un travail de finalisation du projet, le rendre facilement déployable, intégrer une authentification et potentiellement intégrer quelques fonctionnalités supplémentaires.

Malheureusement le projet fut très difficile à prendre en main, nous étions seulement parvenu à le rendre plus stable et déployable. Ce manque de plus-value de notre part a entraîné une grande frustration, nous avons donc décidé de reprendre ce projet cette année dans l'objectif de produire un résultat supérieur à celui des années précédentes.

Arrivés cette année, nous savions que le projet présentait beaucoup trop de défauts de structure difficiles à régler. Nous avons donc décidé de nous lancer un défi : repartir de zéro.

## 2. Notre bilan, ce qu'on a appris

Pour finir nous pouvons dire que ce projet nous a apporté énormément de choses. Premièrement nous avons acquis des connaissances techniques : le développement web avec Vue et Vuetify, la conception de serveurs, MongoDB, l'interaction avec des conteneurs Docker. La liste est très longue et ce rapport témoigne de toutes les technologies que nous avons découvertes, apprises et appliquées tout au long de ce projet.
Nous avons choisi d'utiliser des technologies modernes et populaires et nous ne doutons pas du fait que nous utiliserons encore ces technologies dans un avenir proche.

Au-delà des technologies, nous avons pu concevoir des fonctionnalités que nous n'avions jamais ou peu vu auparavant. Des concepts comme l'authentification mélée à une sécurité fiable, l'API docker, l'architecture et l'organisation d'un grand projet, le store pour une application web et bien d'autres.

Au fur et à mesure de notre conception nous nous sommes rendu compte de certains besoins et nécessités. Nous avons donc dû chercher des solutions qui nous ont permis de découvrir des technologies et fonctionnalités qui nous étaient jusqu'alors inconnues. Ce chemin parcouru nous a été très formateur.

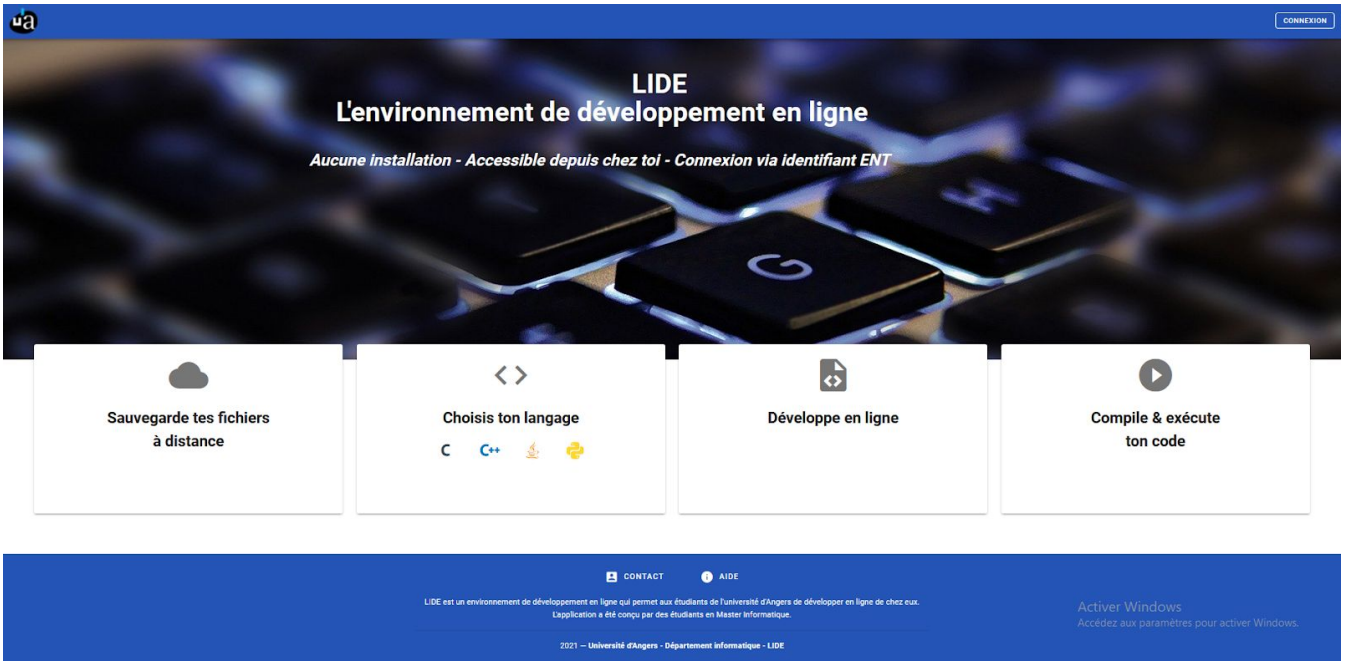
Nous avons également beaucoup appris sur le management et la gestion de projet, compétences utiles pour notre avenir professionnel. Toutes les étapes que nous devons suivre pour porter un projet du stade de l'idée jusqu'à la production, la gestion d'une équipe

de développeurs, la méthode agile SCRUM, l'interaction avec des clients sont des expériences précieuses qui nous ont enrichis.

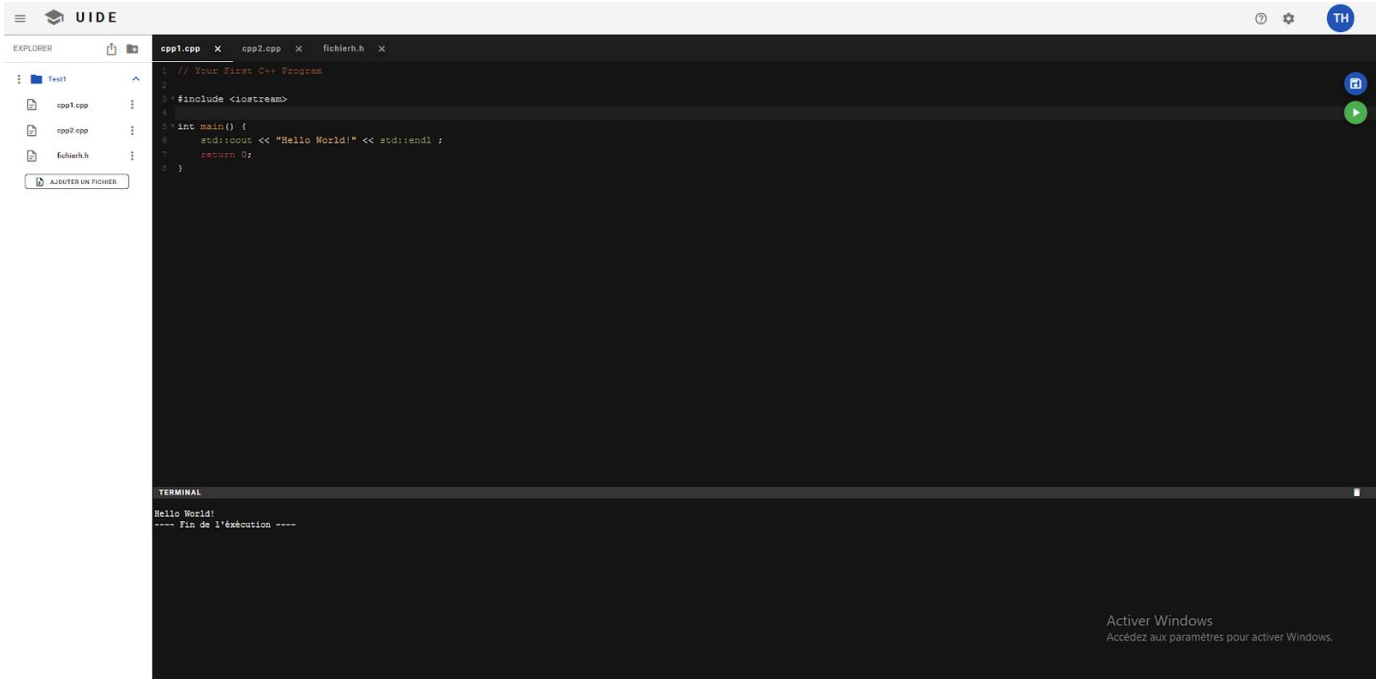
Ce projet fut pour nous le fil conducteur de cette année, nous nous sommes investi dans celui-ci, sa gestion, son développement et nous espérons qu'il aura un bel avenir à l'université d'Angers.

Nous sommes heureux de pouvoir fournir le fruit de notre travail aux étudiants débutants en informatique et nous espérons surtout qu'il simplifiera et rendra plus agréable leur travail au sein de l'université.

# ANNEXE 1 : Page d'accueil de l'application




# ANNEXE 2 : Page principale de l'application




*** Un tableau commence à l'indice 0**