Septembre 2017 - Février 2018

M2 Informatique ACDI
Rapport de management
Environnement de développement en ligne
L.I.D.E.

Chefs de projet :

BAZANTE Alice
CHEVALLIER Sullivan
MAINFROID Pierre-Olivier

Référent :

GARCIA Laurent

Développeurs :

IBRIR Yassine
MARTINS MOSCA Jérôme
RAHIER Valentine
VIOLLETTE Paulin

Table des matières

Remerciements

Nous tenons dans un premier temps à remercier monsieur GARCIA, responsable de notre projet, monsieur GENEST pour l'intérêt qu'il a porté au projet durant son déroulement ainsi que messieurs BARICHARD, RICHER et mesdames COUDRAY et LEFEVRE. De plus, nous remercions monsieur CHANTREIN pour son aide concernant Docker. Nous voudrions aussi remercier les étudiants de première année de Master, affectés au projet : Yassine IBRIR , Jerôme MARTIN MOSCA, Valentine RAHIER et Paulin VIOLETTE pour leur dévouement pour ce projet.

Introduction

Dans le cadre des enseignements de première année proposés par l'Université d'Angers concernant le parcours MPCIE, les étudiants vont recevoir une introduction aux sciences et notamment à l'informatique. Lors des cours de première année de licence d'introduction à l'algorithmique, ils sont amenés à développer leurs premiers programmes dans un langage simple. Malheureusement, les outils nécessaires pour programmer (au minimum un compilateur et un éditeur de texte) sont difficilement installables et configurables pour des personnes non expérimentées. De plus, sur certains postes, il est même impossible de les installer, car il faut les droits d'administrateurs ce qui n'est pas le cas pour les postes de la bibliothèque par exemple.

Certains sites internet offrent la possibilité, à tous et partout, de développer des programmes en ligne sans installation. Il suffit simplement d'avoir un navigateur web. Cependant, le code écrit doit rester simple et le nombre de demandes d'exécutions par utilisateur est limité, par exemple, une toutes les 5 minutes. Pour la faculté des sciences d'Angers, une centaine de personnes doit être en mesure de pouvoir y accéder en même temps ce qui pose problème, car la faculté est considérée par ces sites comme une seule personne connectée.

De ce fait, l'Université d'Angers souhaite proposer à ses étudiants un environnement de développement en ligne qui leur permettrait d'écrire et de compiler du code simplement et sans installation préalable. Aussi, cette application pourrait être utilisée lors des séances de travaux pratiques, ce qui permettrait aux étudiants de retrouver l'environnement qu'ils utilisent chez eux.

Chapitre 1 : Expression fonctionnelle du besoin

1.1 Sujet

Fonctionnellement, les professeurs de première année de Licence MPCIE ont besoin d'une application web permettant d'écrire et tester des programmes avec une interface très simple. L'application devra permettre d'écrire et de compiler du code sans avoir à installer de compilateur sur le poste client (la compilation s'effectuant sur le serveur). Cette application pourra être utilisée, dans le cadre des enseignements informatiques à la faculté des sciences d'Angers, dans plusieurs unités d'enseignement de première année de licence (où rares sont les étudiants qui disposent d'un environnement de développement sur leur poste personnel) jusqu'à la troisième année voire jusqu'au master si certaines fonctionnalités indispensables à des utilisateurs avancés sont disponibles.

Notre objectif était de réaliser une première version d'un environnement de développement en ligne avec une interface épurée au maximum pour permettre aux étudiants débutants de ne pas se perdre. L'interface devait prendre en compte différents langages pour correspondre aux différentes unités d'enseignements (C, C++). L'application doit pouvoir supporter une centaine d'utilisateurs en parallèle notamment pour les évaluations.

Certaines fonctionnalités étaient obligatoires :

- La coloration syntaxique du langage
- Pouvoir compiler et exécuter un programme
    - avec l'accès aux messages d'erreur de la compilation
    - avec l'accès à l'affichage de sortie de l'exécution
- Pouvoir interagir avec le programme
- Prendre en compte plusieurs langages

D'autres étaient envisageables :

- Intégration d'un débogueur
- Gestion plus poussée de l'ensemble de fichiers
- Auto-complétion du code
- Intégration d'outils d'analyses

1.2 Problématique soulevée

Après une réflexion poussée sur le sujet, nous avons constaté que l'enjeu principal de ce projet était la sécurité. En effet, l'application va compiler et exécuter du code inconnu. Il ne faut pas qu'un utilisateur puisse obtenir des accès qu'il ne devrait pas obtenir (accès aux fichiers serveurs, fichiers d'autres étudiants, …), ou faire ralentir le serveur (programme trop gourmand) et influer sur les autres utilisateurs. Il était donc nécessaire d'élaborer une architecture solide et orientée sécurité.

Chapitre 2 : Management première partie : Début du projet

2.1 Etude du sujet

Pour faire l'étude du sujet, nous avons simplifié au maximum l'idée d'une application. Nous avons divisé les fonctionnalités obligatoires et facultatives en plusieurs versions. Cela permet une gestion agile du projet. Chaque version est fonctionnelle. Les versions que nous avons créées en premier lieu pour ce projet sont au nombre de 5. La première, 0, étant la maquette (base créée par les étudiants pour s'entraîner aux technologies) et les versions opérationnelles suivantes découlent ensuite des besoins prioritaires.

La première version, mise en place de l'interface :

- Compiler un fichier
- Transfert entre Front et Back
- Exécution sécurisée du code (conteneur)
- Connexion restreinte (authentification par mail spécifique)

La deuxième version de l'application consistait à prendre en compte le c++ :

- Gestion du C++ : le premier langage utilisé par les étudiants en L1
- Compiler un programme complet depuis l'interface
    - Mise en place du multi-fichier
    - Gestion en onglets des fichiers
- Coloration Syntaxique C++

La troisième phase permet la gestion des fichiers annexes :

- L'ajout de nouveaux fichiers qui seront utilisés dans le code

Ensuite une version orientée vers l'expérience utilisateur :

- Gestions des exports (sauvegarde du code)
- Gestion des imports de code

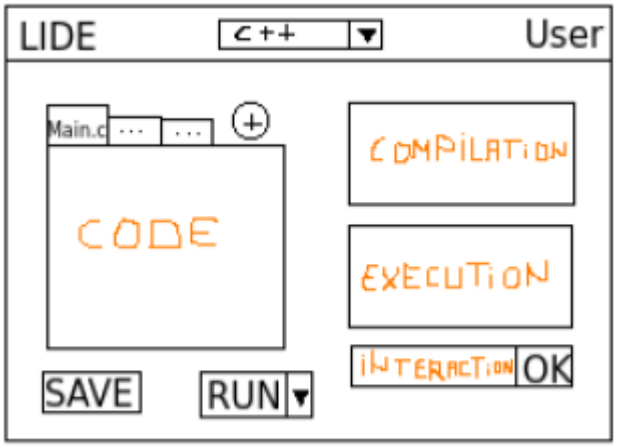
La version finale de l'application permet :

- Le changement de langage de programmation
- La modification des paramètres de compilation

La dernière version permet la gestion de plusieurs langages. Nous avons ajouté une administration à la première version pour rendre autonomes les enseignants. Cela leur permet d'ajouter des langages de manière dynamique.

Nous avons fait plusieurs retours sur les versions à créer pour les faire confirmer aux clients, afin qu'elles coïncident au maximum avec leurs attentes. Nous avons avancé des fonctionnalités dans la première version comme la coloration syntaxique par exemple.

Lors des rencontres avec les clients, nous étions à l'écoute de leurs besoins tout en orientant selon ce que nous pouvions produire dans le temps imparti.



Première maquette

Maquette validée par le client à la couleur près

2.2 Réflexion base de données

Pour permettre aux enseignants d'être autonomes, nous avons décidé de stocker en base de données certaines informations. En effet, les langages seront en base et pourront être ajoutés dynamiquement. Ainsi que leurs extensions.

De plus, pour éviter la surcharge du serveur, nous avons pensé stocker les serveurs disponibles pour les exécutions pour permettre d'ajouter dynamiquement un nouveau serveur en cas de surcharge.

Nous avons pensé à trois tables :

- Langage : qui contient les différents langages pris en compte dans l'application
- Detail Langage : qui contient les extensions des différents langages avec leur modèle
- Serveur : qui contient les serveurs disponibles pour les exécutions

2.3 Séparation du travail

Afin de travailler efficacement, nous avons séparé le projet en trois parties plus ou moins autonomes. Etant trois managers, chacun à pris le management d'une partie notamment pour se spécialiser et pouvoir mieux guider les développeurs. Nous avons essayé de respecter les goûts de nos développeurs pour les placer dans la partie qu'ils préféraient.

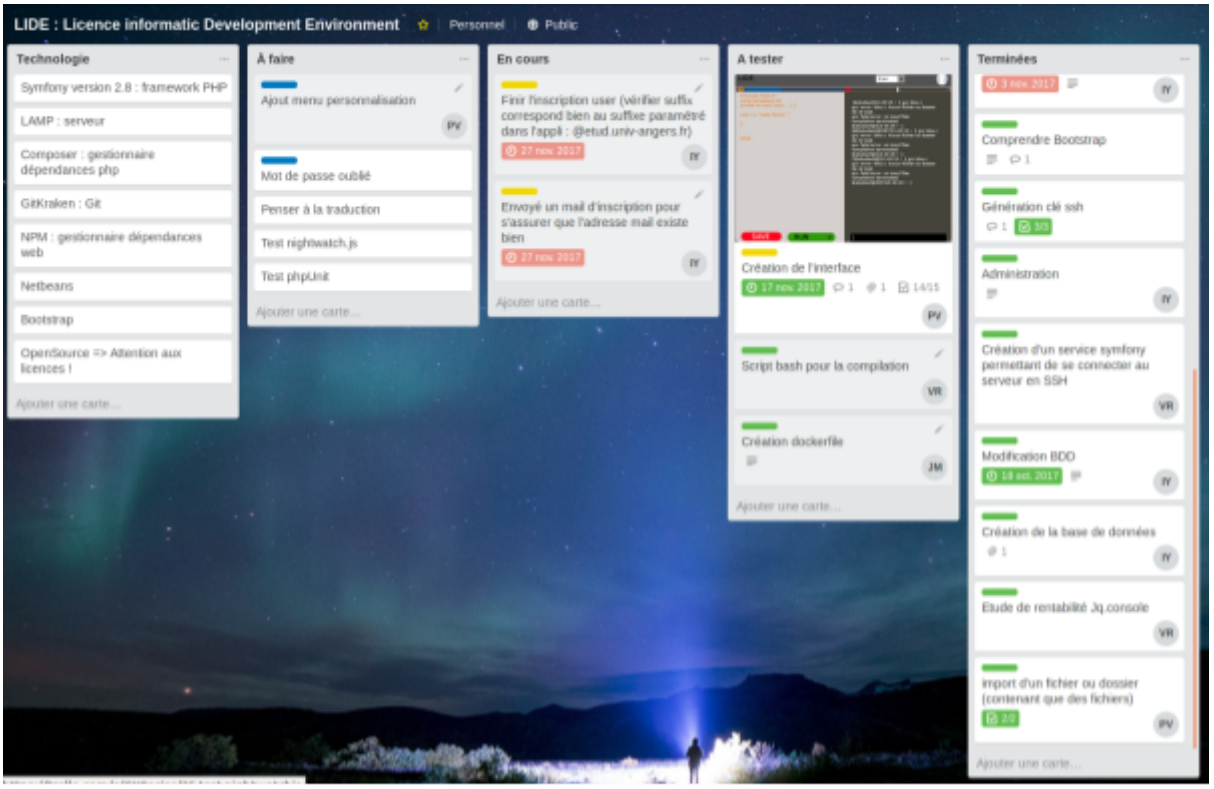
Le groupe application (interface, base de données et administration), composé d'Alice, Paulin et Yassin. Pour la sécurisation, le groupe Sullivan et Jérôme. Pierre-Olivier et Valentine pour la communication entre le serveur d'exécution, l'application et client en direct.

Grâce à cette répartition, chaque manager a pu se concentrer sur une partie et ainsi comprendre plus en détails les problèmes de son équipe. C'est un management de proximité. Cela permet d'aplanir la hiérarchie.

2.4 Outils de communication

Pour communiquer tous ensemble, nous utilisons différents outils pour différentes utilisations, tout en privilégiant le contact réel. Pour la gestion du code, nous avons mis en place Git. Git est un système de versionning de code. Il favorise le travail en équipe et permet de suivre l'avancé du projet. De plus, il permet de gérer les versions et de faire des retours en arrière. Notre repository est https://github.com/bazanta/LIDE/tree/develop.

Pour la répartition des tâches et l'avancée de chaque version, un Trello a été mis en place. Trello est une application de gestion de post-it sur un tableau virtuel.



Notre tableau Trello

Notre Trello était composé de cinq colonnes :

- Technologies : rappelle les technologies du projet et la licence libre
- À faire : où sont placées les tâches à faire
- En cours : les tâches qui sont en cours de développement avec la personne affectée
- À tester : les tâches finies de développer et qui sont à tester
- Terminée : les tâches complètement finies

De plus, nous avons mis en place des étiquettes de différentes couleurs pour avoir une meilleure vision du projet :

- Bleu : Objectif
- Orange : En attente
- Jaune : En cours
- Rouge : Problème
- Violet : Annulé
- Vert : Terminé

Lorsqu'une tâche était prise, nous fixions le prochain objectif de cette personne pour toujours avoir une tâche en avance à faire. Chaque manager gérait les tâches de son équipe. De plus, certaines tâches vitales impactant les autres avaient une date limite à respecter. Si la date était impossible à respecter, nous étudions avec la personne concernée pourquoi et si la justification était pertinente (tâche prioritaire passé devant,...), nous ajoutions un délais.

Ensuite, Slack était notre outil de communication non-officiel. Slack est une application de discussion instantanée. Cet outil nous permettait d'être disponibles pour aider les développeurs en cas de problème, de répondre à leurs questions et d'échanger tout au long du projet. Par ce moyen, les développeurs nous tenaient au courant de l'avancée de leur tâche nous permettant d'avoir une meilleure vision de l'avancée du projet. Après, pour les communications officielles, nous utilisions les mails. Un mail laisse une trace en cas de problème future et est plus structuré. Communiquer à l'écrit est plus compliqué qu'en personne c'est pourquoi nous préférions les face-à-face. En effet, un mail ne laisse pas passer les émotions contrairement à un face-à-face.

2.5 Fréquence des rencontres

Pour avoir une bonne gestion du projet et de l'avancée des développeurs, nous avons décidé de nous rencontrer en réunion toutes les deux semaines environ avec un point oral toutes les semaines. En effet, voulant agir en agile, nous voulions faire des sortes de sprint. De plus, nous étions ouverts aux idées de nos développeurs et en cas d'idée pertinente, il fallait être dynamique et étudier la faisabilité, voir si nous pouvions intégrer la modification et améliorer le planning. Ce sont les rencontres fréquentes qui ont permis cette modularité.

Chapitre 3 : Outils et technologies

Pour le choix des outils, nous avons choisi seulement des logiciels open source et libres de droits. En effet, nous avons décidé dès le début de partir sur de l'open source avec la Licence GPL (GNU General Public License). En effet, créant une application pour l'université d'Angers, nous restons dans le même état d'esprit du logiciel libre.

Avant de commencer le projet, nous avons créé un mini TP aux développeurs pour apprendre les technologies choisies. Nous leur avons laissé une semaine et demie pour se former aux outils et s'approprier le sujet.

3.1 Technologie principale

Le projet consistait à faire une application en ligne, c'est pourquoi la technologie principale devait correspondre au web. Nous voulions obligatoirement utiliser un framework pour avoir un cadre de travail et ne pas avoir à réinventer la roue. De plus, les grands frameworks ont souvent une grosse communauté derrière eux et une structure qui a été testée et qui est fiable. Nous avons choisi d'utiliser le langage PHP avec le framework Symfony. En effet, nous avions déjà des compétences dans ce langage et avec ce framework en particulier ce qui facilite la conception et permet un meilleur soutien aux développeurs. Nous aurions pu partir sur du Java mais aucun de nous ne connaissait de framework sur ce langage et partir de zéro (sans architecture prédéfinie) aurait été trop long. C'est pour cela que nous sommes parti sur PHP et sur Symfony.

L'un des avantages de ce framework est le concept de Bundle. Un bundle regroupe une ou plusieurs fonctionnalités autonomes pouvant être intégrées facilement avec un simple paramétrage. Par exemple FosUserBundle qui permet la gestion des utilisateurs (connexion, inscription). Grâce à ce concept, il n'y a plus besoin de réinventer les briques de base d'une application.

De plus, un moteur de template est couplé au framework : Twig. Cela permet d'avoir une hiérarchie dans nos templates. Nous pouvons créer un template de base que tous les autres hériteront pour factoriser le code et faciliter la maintenance.

Schéma de l'imbrication des templates twig

Symfony intègre un ORM (Object Relationnel Mapping), Doctrine, qui permet d'être indépendant du type de base de données (MySQL, DB2, Oracle, …). Avoir un ORM permet d'abstraire la complexité de la base de données ce qui simplifie les interactions.

3.2 Base de données

Au niveau base de données, du fait de l'ORM de Symfony, nous étions libre. Le seul pré-requis était que la base de données soit supportée par Doctrine. Etant donnée que nous sommes pour l'open source, MariaDB, une version open source de MySql a été une évidence. En effet, MariaDB est un SGBD (Système de Gestion de Base de Données) complet et robuste open source qui a une bonne communauté ce qui assure une maintenance sur le long terme.

3.3 Technologies pour l'interface

Pour la partie Front du projet, soit l'interface, nous avons utilisé les langages de base, soit le HTML pour l'architecture des pages, le CSS (feuille de style) pour le design et le JavaScript pour les interactions.

Pour la mise en forme, le framework Bootstrap (framework css/js) était inévitable. Ce framework permet de gérer le responsive des applications assez facilement. De plus, il est simple d'utilisation et permet d'accélérer le développement au niveau du style. En ajoutant des classes aux éléments HTML, un style particulier leur est attribué. Par exemple pour les boutons, la classe "success" permet d'avoir le fond en vert. Côté javascript, il permet de créer facilement des modals, pop-ins, alertes, dropdowns… De plus, il gère le responsive du menu qui sera par exemple transformé en trois barres sur petit écran.

Au niveau du JavaScript, nous avons utilisé plusieurs plugins différents :

- Jquery : framework JavaScript le plus simple à apprendre et riche en fonctionnalités
- Ace : éditeur de texte pour le web avec coloration syntaxique (environ 110 langages). Ce plugin possède de nombreuses fonctionnalités utiles pour notre projet (indentation, numéro de ligne, Chercher/remplacer, coloration, …)
- JQconsole : simulateur de la console
- FileSaver.js : permet de sauvegarder des fichiers
- SweetAlert2 : affichage des alertes plus esthétiques que les alertes standards

3.4 Technologies serveur

Les étudiants exécuteront leur code sur le serveur, c'est pourquoi nous avons réfléchi au meilleur moyen de protéger le serveur et de limiter les exécutions pour éviter les boucles infinies. Il fallait trouver un moyen d'isoler les différentes exécutions.

La conteneurisation, encore jeune mais prometteuse, s'est imposée d'elle-même. Elle permet de créer un espace propre indépendant et sans incidence sur l'environnement qui héberge le service de chaque exécution. Le noyau de linux a la possibilité de créer des conteneurs : LXC (LinuX Containers) mais avec une gestion bas niveau.

Des solutions basées sur LXC existent et notamment Docker qui est la solution la plus utilisée (leader du marché). Ces solutions ajoutent des capacités de niveau supérieur par exemple, faciliter l'installation d'application dans des conteneurs ou encore l'installation de mise à jour. La création des conteneurs reste faite par le noyau linux.

Ayant Docker au programme, nous avons choisi de partir sur cette technologie qui est plus légère que la virtualisation et qui permet de travailler toujours sur le même environnement (la même image est utilisée autant de fois que nécessaire) ce qui correspond parfaitement au besoin du projet. Chaque exécution aura le même environnement et sera isolée des autres sur une image.

3.5 Technologie de communication

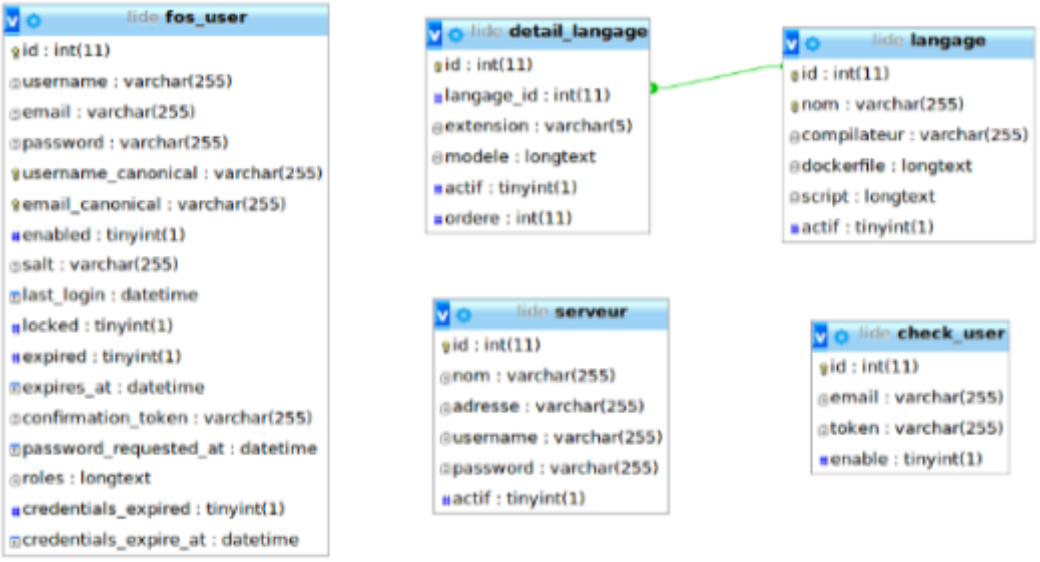
Pour faire communiquer le serveur contenant les exécutions avec l'interface, nous avons décidé d'utiliser SSH (Secure Shell) qui permet de se connecter de façon sécurisée au serveur contenant les dockers.

3.6 Déploiement

Pour déployer en automatique, nous avons utilisé Capistrano. Capistrano est un outil de déploiement automatique d'application web. Il permet de gérer facilement les versions déployées et permet des déploiements à chaud c'est à dire sans que l'utilisateur ne s'en rende compte. Si le déploiement rate, il fait un retour en arrière comme si rien ne s'était passé.

Chapitre 4 : Application version M1

4.1 Base de données



Modèle conceptuel des données de la base de données de l'application

Ci-dessus, vous pouvez retrouver le MCD (Modèle Conceptuel des Données) de l'application. Il permet de représenter la base de données de façon compréhensible.

On remarque des tables supplémentaires par rapport à la base de données réfléchies au début du projet (voir point 2.2). Ce sont les tables qui correspondent aux utilisateurs :

- Fos_user : qui contient les utilisateurs
- Check_user : qui contient les e-mails pour l'inscription des utilisateurs

Les autres tables :

- Serveur : qui contient les serveurs d'exécution (adresse, utilisateur, mot de passe)
  - Permet de connaître les serveurs disponibles
- Langage : qui contient les langages avec
  - le dockerfile (fichier qui permet de créer l'image docker contenant le nécessaire pour compiler/exécuter le langage)
  - le script qui permet d'automatiser la compilation, exécution
- Detail Langage : qui contient le détail des langages (extension du langage, modèle par défaut correspondant à l'extension (par exemple le Hello World), un ordre qui permet de gérer l'extension principale du langage)

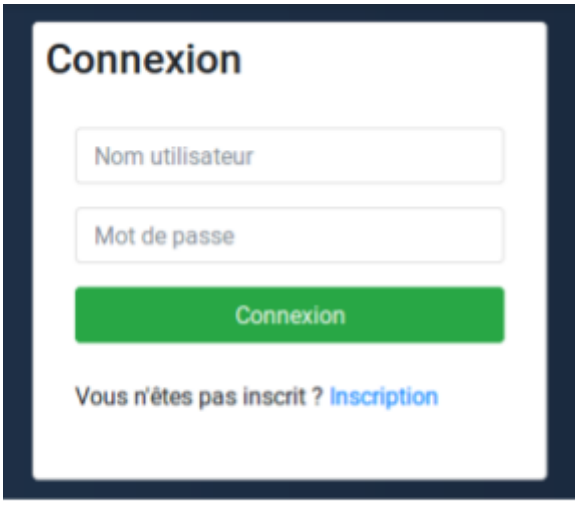
Chacune des trois tables à un champ "actif" qui permet de désactiver ou activer le tuple. Par exemple, si un langage n'est plus au programme, nous pouvons le désactiver. Si un serveur est non disponible, en maintenance ou autre, nous pouvons le désactiver.

4.2 Connexion

Pour gérer la sécurité de l'application, nous avons mis en place un système d'authentification. Le bundle FosUserBundle s'occupe de cette partie. Il a juste fallu faire le paramétrage et redéfinir les interfaces. Pour gérer les accès, nous avons défini deux rôles :

- ROLE_ADMIN : rôle pour les administrateurs permettant l'accès à l'administration qui permet de modifier les langages, les serveurs, la gestion des utilisateurs…
- ROLE_USER : rôle pour les utilisateurs qui auront seulement accès à la page principale de l'application qui permet le développement de programmes.

Pour se connecter à l'interface utilisateur ou administrateur, il faut rentrer son nom d'utilisateur et son mot de passe.



Page de connexion à l'application

4.3 Administration

L'administration est générée grâce au bundle EasyAdmin. Dans la configuration du bundle, il suffit de renseigner les différentes tables de la base de données et le bundle s'occupe du reste. L'administrateur peut paramétrer la base de données (ajout, suppression, modification) :

- Les langages
- Les détails des langages
- Les serveurs pour l'application
- Les utilisateurs

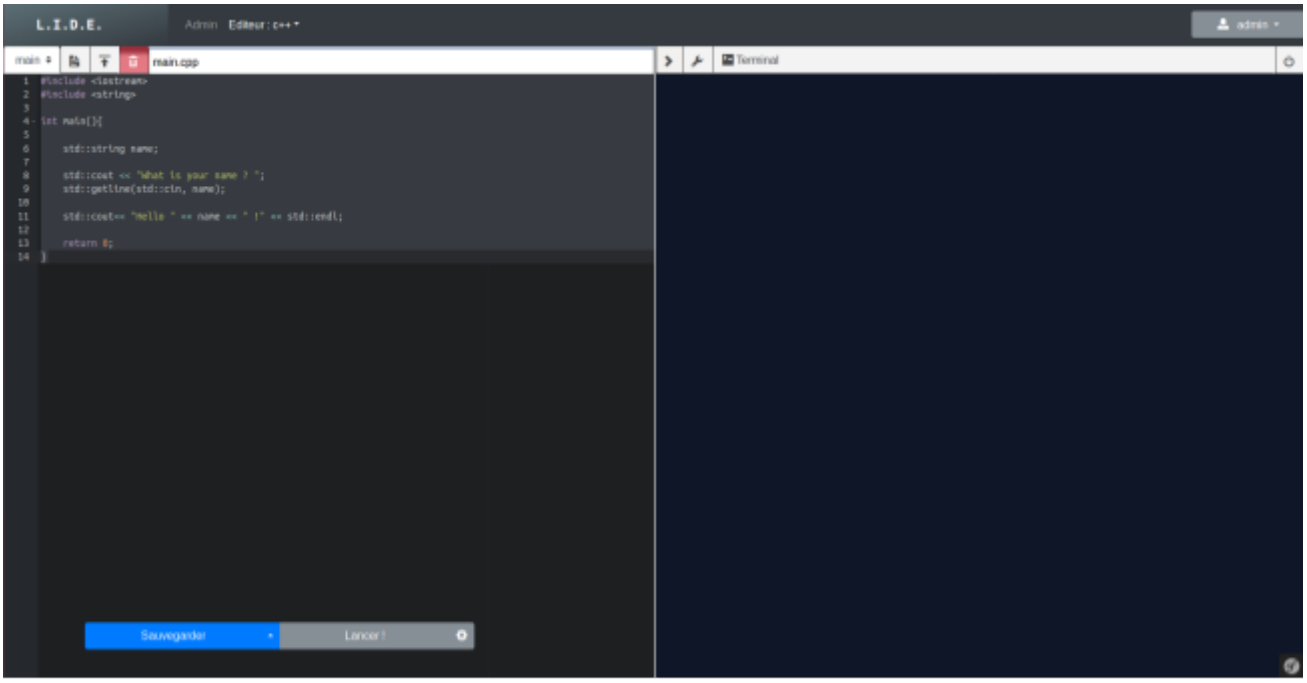
Interface de l'administration de l'application

4.4 Interface Utilisateur

4.4.1 Design

Une fois l'utilisateur connecté, il est redirigé vers la page principale de l'application : l'interface de développement (un éditeur de texte et une console).
L'interface est divisée en quatre parties :

- Le menu, contenant les liens vers les autres parties du site ainsi qu'une liste pour changer de langage
    - Le lien vers l'administration est visible uniquement pour les administrateurs
- La barre d'outils, qui contient les fonctionnalités propres au developpement
    - Ajout/import de fichiers
    - Suppression de fichiers
    - Onglets pour changer de fichiers
    - Contrôle de la console
- L'éditeur, implémenté par le plugin Ace
    - Avec en bas deux boutons :
        - Sauvegarde du travail
        - Lancement (compilation, compilation et exécution, …)
- La console, implémentée avec le plugin JQconsole



Interface principale permettant le développement d'un programme.

4.4.2 Fonctionnalités

a- Gestion des fichiers

Pour répondre à la deuxième version, l'application devait pouvoir gérer plusieurs fichiers. Il a été décidé de les représenter sous forme d'onglets pour faciliter le changement et la vision globale des fichiers présents. Pour une meilleure expérience utilisateur, cette gestion a été faite avec le JavaScript sur le navigateur (côté client).

Les fichiers peuvent être créés de deux façons :

- Depuis son ordinateur grâce au bouton *Importer*
- En créant un nouveau fichier vierge grâce au bouton *Nouveau*
    - Possibilité de choisir l'extension du langage voulu (suivant les extensions paramétrées par les administrateurs)
    - Possibilité de choisir d'utiliser le modèle par défaut
    - Possibilité de créer un fichier sans extension pour les fichiers non compilés qui serviront dans le code (fichier de données par exemple)

b- Sauvegarde des fichiers

Pour récupérer le code écrit dans l'éditeur de texte, une gestion d'export a été implémentée. Ainsi, nous avons trois possibilités différentes pour sauvegarder le travail :

- Exporter seulement le fichier en cours
- Exporter tous les fichiers un par un
- Exporter tous les fichiers en archive

Cette fonctionnalité a été codée grâce au plugin JavaScript FileSaver.js.

c- Compilation/exécution des fichiers

Par défaut, nous compilons puis exécutons le code. Si un utilisateur veut avoir la main sur sa compilation et son exécution, un formulaire de paramétrage est accessible.
Le formulaire comporte sept champs :

- Les paramètres de compilations (arguments passés au compilateur)
- Les paramètres de lancement (arguments donnés au programme)
- Les fichiers additionnels
    - l'utilisateur peut choisir des fichiers depuis son ordinateur qui seront joints aux fichiers présents dans l'IDE
- Une option "Compilation Uniquement" qui permet de compiler sans exécuter
- Le choix de gestion du flux d'entrée :
    - Aucune : aucune gestion des entrées n'est effectuée
    - Interactive : entrées interactives, l'utilisateur écrit dans la console au fur et à mesure de l'exécution du programme
    - Texte : les entrées sont définies à l'avance, le programme utilise un fichier comme flux d'entrée.
- Les entrées à donner au programme : seulement si le mode *Texte* de la gestion des flux d'entrée est sélectionné.

Lorsque l'utilisateur valide la compilation/exécution, le formulaire est traité ainsi que les fichiers de l'éditeur. Si une exécution est en cours, elle sera stoppée pour lancer la prochaine.

Si un utilisateur se rend compte d'une erreur dans son code, d'une boucle infinie,… et qu'une exécution est en cours, il peut la stopper grâce à un bouton.

d- Gestion des langages

L'une des demandes obligatoires était la coloration syntaxique du langage. Le plugin Ace, utilisé pour l'éditeur de texte, possède cette fonctionnalité. C'est l'une des raisons qui nous a poussé à choisir ce plugin plutôt qu'un autre.

La gestion de plusieurs langages était importante pour nous c'est pourquoi elle a été pensée dès le début pour avoir une architecture directement bien pensée. Les langages sont ajoutés en base de données par les enseignants (administrateur sur l'application) avec les extensions voulues et un modèle par défaut pour chacune des extensions. Pour l'utilisateur, une liste déroulante dans le menu permet de choisir le langage voulu. Une fois le langage choisi, les informations le concernant sont récupérées pour avoir les extensions correspondantes, les modèles, le nom du langage pour gérer la coloration syntaxique et le nom du compilateur pour gérer l'affichage de la commande de compilation. En dynamique, grâce au JavaScript nous allons mettre à jour les parties de l'interface impactées.

e- Gestion de la personnalisation de l'interface

Toute personne ayant déjà travaillé en groupe sur un projet informatique a pu remarquer que chacun a ses préférences de thème : certains préfèrent un fond sombre et d'autres un fond clair. L'éditeur Ace dispose par défaut de 24 thèmes et permet de modifier la police d'écriture. Pour la console, il a fallu créer des thèmes, c'est pourquoi pour l'instant, seuls trois styles sont implémentés. Tous ces changements sont pour l'instant uniquement gérés en local.

4.5 Communication

La communication entre le serveur de l'application et les conteneurs docker contenant les exécutions repose sur une connexion SSH. L'application se connecte au serveur contenant les dockers afin de communiquer les messages et d'exécuter les commandes qui lui sont données. Le processus de communication repose sur un système de questions/réponses. Le client demande des informations et le serveur lui répond.

Communication entre le client et le conteneur Docker

Chapitre 5 : Application version M2

5.1 Corrections

5.1.1 Connexion/Inscription

L'inscription des utilisateurs n'était pas bien implémentée. Nous avons donc remis en marche l'inscription en corrigeant les problèmes et en rendant paramétrable les préfixes autorisés. De plus, nous avons ajouté la gestion du mot de passe oublié et sa modification.

Pour faire ces modifications et ajouts nous avons paramétré le bundle FosUserBundle mis en place par les développeurs et nous avons surchargé les vues manquantes (modification de mot de passe).

5.1.2 Compilation/Exécution

Suivant la version du noyau linux, nous ne pouvons pas utiliser toutes les options misent en place concernant docker. En effet, la gestion du temps du docker avant de le détruire pour éviter les boucles infinies, ne fonctionnait pas tout le temps. Nous avons donc choisi de ne pas utiliser l'option de docker mais directement une commande linux.

5.1.3 Gestion des exécutions non interactives

L'exécution sans interaction ne fonctionnait pas bien. En effet, au début après la compilation, nous devons appuyer sur entrer pour lancer l'exécution et donc interagir. Hors ce mode indique "aucune interaction" ce qui bloquait le programme. Nous avons donc autorisé une seule interaction pour éviter les problèmes.

5.2 Améliorations

5.2.1 Ajout loader

Pour prévenir l'utilisateur que la compilation et l'exécution sont en cours, nous avons décidé d'ajouter un "loader", c'est-à-dire un élément visuel qui tourne pour que l'utilisateur puisse savoir que son action a bien été prise en compte. En effet, avec notre implémentation, l'utilisateur doit attendre au moins quatre secondes.

5.2.2 Ctrl+s

Au final, l'export des fichiers seul n'était pas suffisant après manipulation. En effet, si nous voulons sauvegarder notre code après chaque grosse modification, nous nous retrouvons vite avec pleins d'exemplaires du même fichier sur l'ordinateur.

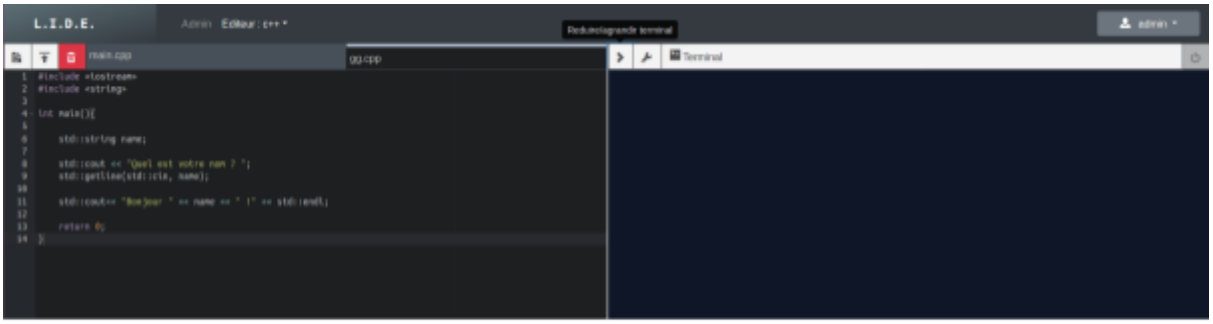
Nous avons donc ajouté la gestion du “ctrl+s” pour sauvegarder le code en session. Cela permet à l’utilisateur d’actualiser la page sans perdre son code, en cas de coupure de courant, de récupérer son code, … Le code est gardé sept jours en session.

5.2.3 Options de personnalisation

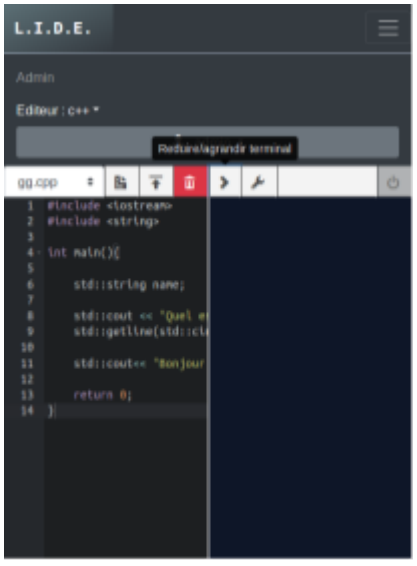
Pour compléter le travail des développeurs, nous avons relié les options de personnalisation en base à l’utilisateur. Cela évite à chaque reconnexion de devoir recommencer le paramétrage de l’interface (couleurs, tailles de la police, …).

5.2.4 Responsive

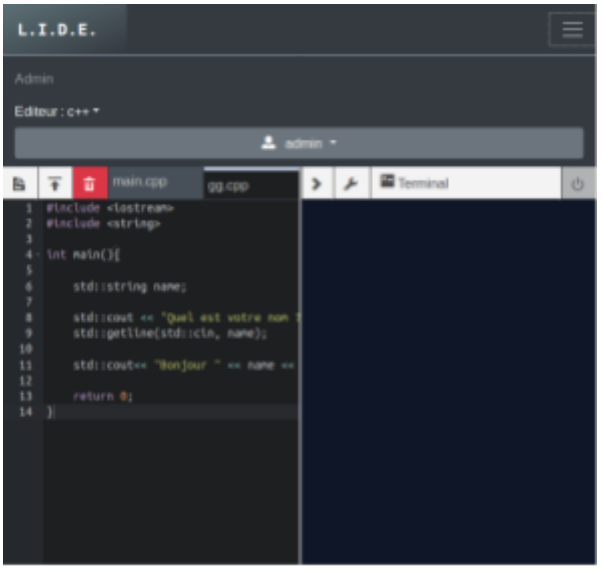
L’application n’était pas totalement responsive. Nous avons fait en sorte que l’application fonctionne sous tous les navigateurs et sur tous les types d’appareils (ordinateur, tablette, mobile). Elle s’affiche bien et est fonctionnelle sur téléphone portable mais n’est pas très ergonomique (bouton pas assez gros,...). Les ordinateurs sont la cible de l’application, c’est pourquoi nous n’avons pas fait d’étude particulière pour être ergonomique sur téléphone portable. En effet, il est compliqué de coder sur son téléphone (trop petit, pas pratique sans clavier physique, …). Cependant, il est tout à fait possible de l’utiliser sur téléphone.



Application sur Ordinateur



Application sur Mobile



Application sur Tablette

Sur les trois captures ci-dessus, la console est ouverte, il y a la possibilité de la réduire. On remarque bien que l'application est responsive. Sur téléphone, les fichiers sont accessibles grâce à une liste déroulante ce qui permet de gagner de la place par rapport à l'affichage en onglets sur ordinateur et tablette. De plus, le menu s'affiche différemment sur tablette et téléphone que sur ordinateur.

5.3 Déploiement

Nous avons ajouté au projet le déploiement automatique grâce à capistrano. Un fichier de paramétrage est nécessaire précisant les tâches à effectuer. De plus, il faut un fichier de configuration par serveur sur lequel on veut déployer (test, pré-prod, prod).

Ayant déployé sur un serveur, nous avons pu mettre en place une procédure pour expliquer comment déployer l'application (installation des technologies nécessaires, déploiement et configuration de l'application). De plus, nous avons constaté les problèmes pouvant être rencontrés et comment les résoudre.

5.4 Tests, montée en charge

Une fois l'application déployée sur un serveur, nous avons pu mettre en place les tests plus poussés. En effet, nous avons testé notre application en compilant, exécutant un code avec trois utilisateurs différents en même temps sans détecter de différence. Si l'un de nous trois fait une boucle infinie avec un affichage, son ordinateur ralentit (l'affichage est trop gourmand) mais les deux autres utilisateurs ne sont pas impactés. Nous avons ensuite testé à six machines sans aucun souci.

La deuxième étape consistait à tester avec un vrai TP mais elle n'a pas pu être mise en place. En effet, il aurait fallu déployer sur le serveur de l'université mais pour cela, il faut d'abord prouver la robustesse de l'application et sa sécurité. Cette étape sera réalisée par la suite par l'université.

Chapitre 6 : Avenir du projet

Le projet peut désormais être déployé sur n'importe quelle machine en suivant la procédure présente dans le README.txt se trouvant sur le git du projet (https://github.com/bazanta/LIDE/blob/develop/README.md).

Nous avons réfléchi à une nouvelle architecture logicielle pour notre application. Le client a exprimé le besoin d'utiliser plusieurs serveurs répartis sur tout le territoire français pour les exécutions des programmes. Ces serveurs seront dotés d'un système de load-balancing pour répartir les charges. La nouvelle architecture a été pensée afin de répondre à cette nécessité, assurer une haute disponibilité et respecter la bonne utilisation des principes de Docker. Dans sa version finale, l'application n'aura plus besoin d'un serveur d'hébergement avec les technologies nécessaires à notre application, seulement docker. En effet, l'application entière sera conteneurisée sous forme de services : un service global pour l'exécution des conteneurs, un pour la partie web et un autre pour la base de données.

Au niveau de la communication, il y a aussi des choses à revoir. L'inconvénient de notre implémentation est que l'application doit récupérer la réponse du docker juste après lui avoir envoyé le message. Pour se faire, le service attend 2 secondes avant de retourner une valeur. Des problèmes apparaissent dès que l'exécution met trop de temps à répondre notamment en mode non interactif. Il faut donc creuser un peu plus pour trouver une meilleure façon de faire.

Il reste aussi beaucoup de fonctions à implémenter, qui nous semblent moins prioritaire ou qui n'étaient pas intégrées dans notre plan de charge. Nous n'avons pas pu tout intégrer car nous étions tenus à un planning très serré.

- L'ajout d'un débogueur serait fortement apprécié par notre client. C'est une tâche conséquente et qui doit avoir sa propre étude.
- Pouvoir cliquer sur la ligne de compilation en erreur pour accéder à la ligne de code associée.
- Il faut ajouter l'AutoComplétion soit par création d'un module d'autoComplétion qui demande une autre étude, soit par intégration d'un module déjà fait.
- C'est aussi la mode du Co-Working, il faut donc ajouter la possibilité de travailler à plusieurs sur le même fichier. (à la façon Google Drive)
- Pour finir avec les exemples d'améliorations, nous pouvons ajouter des raccourcis claviers comme dans un IDE hors ligne classique (par exemple : commenter/décommenter une ligne)

Chapitre 7 : Management deuxième partie : retour fin de projet

7.1 Répartition des tâches

7.1.1 GANTT

Ci-dessous, vous trouverez le diagramme de Gantt du projet. Le diagramme a été conçu dans les premières semaines du projet, c'est pourquoi il ne respecte pas exactement les tâches effectuées ainsi que le temps pris pour la tâche. Cependant, dans l'ensemble, il est cohérent et permet d'avoir une vision d'ensemble du projet.

Planning (En rouge : tâches réalisées par les M2,en bleu celles des M1 et en orange liens avec les enseignants).

7.1.2 Développeurs

Yassine s'est occupé de la base de données, de la connexion avec les différents rôles de l'application et de l'administration. Il a codé la base de données grâce à l'ORM Doctrine de Symfony et mis en place les fixtures (données par défaut pour avoir le même jeu de test). Il a créé l'administration de cette base de données avec le bundle EasyAdmin grâce à une configuration. Pour finir, il a utilisé le bundle FosUserBundle pour gérer l'authentification et les rôles de l'application.

Jérôme a imaginé l'architecture et la sécurisation des exécutions avec Docker. Il a d'abord mis en place une architecture simple mais efficace en conditionnant chaque exécution sur un docker lancé à la volée par l'application. Il a créé une image Docker qui s'exécute avec des paramètres de sécurisation : Limite CPU , RAM et temps d'exécution. Il a ensuite conçu une toute nouvelle architecture beaucoup plus robuste mais nous n'avions pas assez de temps et d'expérience pour la mettre en place.

Valentine s'est occupée de la communication entre l'application web et l'architecture de Jérôme. Elle a tout d'abord, dans le cadre de nos recherches de technologies fait une étude d'intégration du plugin JQconsole. L'étude faite et la rentabilité prouvée, elle a aidé Paulin à mettre en place le plugin dans l'interface. Elle a ensuite créé un service SSH en PHP pour faire la connexion entre le plugin et le docker. Elle a mis en place un protocole d'accès de l'application vers le Docker et inversement puis s'est occupée des scripts de compilation, exécution.

Paulin a construit l'interface de la page principale et créé la base de l'application (menu et forme générale du site) pour que toutes les pages en héritent. Pour l'interface, il a dû implémenter Ace pour l'éditeur et JQConsole (étudié par Valentine) pour la console. Il a mis en place la compilation, exécution en coopération avec Valentine. Il devait récupérer les langages en base pour que l'utilisateur puisse choisir son langage. Ensuite, il était libre de choisir les plugins qu'il voulait pour faire les autres fonctionnalités (gestion de plusieurs fichiers, import, sauvegarde, …). Pour finir, il a ajouté la personnalisation de l'interface (couleur, taille, ...).

7.1.3 Managers

Pour notre part, nous nous sommes occupés de faire toutes les tâches extérieures aux développements. Nous avons tout d'abord créé une fiche récapitulative du sujet et de notre interprétation avec les technologies que nous envisagions. Après avoir consulté nos développeurs, nous nous sommes rendu compte qu'ils ne maîtrisaient pas toutes les technologies que nous avions choisies. Nous avons créé une formation personnalisée pour que chacun s'approprie les technologies. Suite à cette formation, chaque développeur s'est vu affecter une tâche. Certains étaient au développement pur, d'autres étaient encore dans la conception (architecture docker). Nous préférions totalement finir la conception avant de faire trop de gâchis de code et de temps.

L'interface et le modèle conceptuel des données (base de données) ont été créés par nos soins et présentés au client. Pour l'interface, nous l'avons guidé dans ses idées, pour rester dans ce qui était possible de faire dans le temps imparti. L'interface fût validée et envoyée au développement.

Nous avons ensuite créé le cahier des charges pour expliquer l'application que nous allions créer (conception, technologies utilisées, architecture,...) et les contraintes (chiffrage, licence, …).

La première idée de la base de données n'intégrait pas la totalité de ce qui est présent aujourd'hui. Notre structure d'application a permis ces modifications en direct assez simplement grâce à l'ORM de symfony.

Suite au rendu de nos développeurs, nous avons créé un manuel utilisateur de l'application, utile aux administrateurs ainsi qu'aux utilisateurs finaux. Le manuel administrateur contient un protocole de déploiement à appliquer au lancement de l'application.

Pour finir, nous avons participé au développement pour corriger les bugs et ajouter des améliorations. De plus, nous avons mis en place le déploiement automatique.

Tout au long du projet, nous étions des supports pour nos développeurs afin de les aider à résoudre les bugs et problèmes qu'ils rencontraient.

7.2 Évaluation des M1

L'évaluation du travail de nos développeurs s'est faite en deux parties. Étant donné que nous adaptions les tâches en fonction de l'accomplissement de la précédente et de la capacité du développeur, il n'aurait pas été juste de noter seulement les tâches faites. En effet, certains ont eu des tâches beaucoup plus compliquées et moins de temps pour les terminer.

Nous avons choisi d'effectuer une évaluation la plus équitable possible. Ainsi, nous avons décidé d'évaluer sur 10 les tâches effectuées et "l'implication" sur 10 aussi.
"L'implication" est évaluée comme suit:

- autonomie / pro-actif
- efficacité
- investissement
- commentaire du code
- difficulté des tâches
- comportement

Pour nous, le comportement est aussi important que le travail en lui-même. En effet, quelqu'un de doué naturellement qui maîtrise déjà les technologies peut faire le strict minimum mais bien, alors qu'un autre peut travailler très dur avec un gros investissement, mais ne pas bien réussir les tâches demandées (trop dur par rapport au niveau, pas assez de temps, …).

7.3 Post Mortem

Notre technique de management de proximité fût la technique à employer. Il n'y a pas de différence avec nos développeurs, donc une méthode de management plat est plus appréciée par les deux partis. Nous étions à leur écoute et ouverts à leurs propositions. Nos points forts en tant que managers étaient la disponibilité à tous moments. La prise de nouvelle fréquente et le suivi constant sur Git et Slack nous ont permis de nous tenir au courant alors que nos développeurs n'étaient pas habitués à donner des nouvelles. C'est cette démarche qui a été bénéfique pour le projet. En M1, nous ne pensions pas à l'importance de la communication au sein d'un projet. C'est en manageant qu'on voit que si la communication ne remonte pas, le projet n'avance pas. Nos erreurs viennent de cette idée, au départ, nous ne pensions pas assez souvent à prendre des nouvelles, à leur demander leurs avancements. Aujourd'hui, la communication dans un projet est essentielle, plus la communication intervient tôt, plus nous pouvons réagir en tant que manager.

Contrairement à cela, nous n'aurions pas dû appliquer le même management aux quatre développeurs. La majorité était bien autonome mais pas tous. Nous aurions dû adapter notre façon de travailler avec certain. Les moins autonomes auraient dû être suivis plus strictement et plus formellement par mail. C'est la seule façon de les faire travailler et de voir si le travail est fait en temps et en heure.

Nous avons repoussé les échéances de certaines tâches car elles ont mal été chiffrées. Nous aurions dû chiffrer chaque tâche selon un chiffrage de développeur débutant et non selon notre expérience. Il faut constamment prévoir une marge de sécurité, du temps en plus qu'il faut chiffrer quitte à faire payer légèrement plus le client. Il vaut mieux pour nous que le client paye un peu plus, dans la limite du raisonnable, plutôt que de devoir repousser la date de livraison. Heureusement, nous pouvons livrer en temps et en heure, mais quelques jours de plus auraient été utiles et auraient permis d'être plus sereins.

D'un point de vue global, nous avons bien su nous adapter à la situation. Nous avons adapté nos techniques de management au fur et à mesure de l'avancée du projet et de la connaissance de notre équipe. Nous ne pouvions pas imaginer les difficultés à apprendre à manager "sur le tas". La technique primordiale pour un bon manager, c'est le contact avec son équipe et l'écoute de ses développeurs : c'est la partie humaine.

Conclusion

Le projet s'est bien déroulé. Nos développeurs étaient investis et disciplinés (ils nous écoutaient bien et respectaient nos choix). Nous avons travaillé dans de bonnes conditions avec une bonne entente dans l'équipe. Grâce à ce projet, nous avons compris ce que c'est d'être un manager et les enjeux que cela entraîne, surtout au niveau de la communication. Nous avons grandi et appris beaucoup de choses sur le travail en équipe. En tant que chef de projet, nous avons appris à déléguer et accepter les erreurs de nos développeurs.

Notre application est fonctionnelle, mais n'est pas terminée. En effet, il faudrait retravailler l'architecture notamment sur les dockers et la communication. Nous tenons à remercier nos développeurs pour leur implication et le travail qu'ils ont fournis tout au long de cette unité.

Bibliographie

Rapport des M1

Sitographie

Symfony et les bundles utilisés :
https://symfony.com/
https://twig.symfony.com/doc/2.x/
https://fr.wikipedia.org/wiki/Doctrine_(ORM)
https://openclassrooms.com/courses/developpez-votre-site-web-avec-le-framework-symfony
https://github.com/javiereguiluz/EasyAdminBundle
https://github.com/FriendsOfSymfony/FOSUserBundle

Les technologies utilisées :
https://doc.ubuntu-fr.org/lamp
https://www.docker.com/
https://doc.ubuntu-fr.org/docker
https://fr.wikipedia.org/wiki/LXC
https://www.grafikart.fr/formations/serveur-linux/capistrano
https://github.com/
https://doc.ubuntu-fr.org/git
https://trello.com/
https://slack.com/intl/fr-fr
https://fr.wikipedia.org/wiki/Licence_publique_g%C3%A9n%C3%A9rale_GNU

Les plugins JavaScript et css :
https://github.com/replit/jq-console
https://ace.c9.io/
https://github.com/eligrey/FileSaver.js/
https://stuk.github.io/jszip/
https://jquery.com/
https://sweetalert2.github.io/
https://getbootstrap.com/