

M1 Informatique

Concrétisation disciplinaire

Projet LIDE : Environnement de développement en ligne pour les étudiants de licence de l'Université d'Angers

Auteurs

Gabriel MOUGET
Romain GRELIER
Romane POIRIER
Etienne TIGNON

Referent

David GENEST

Chefs de projet

Julien FONTAINE
Louis MARCHAND
Paulin VIOLETTE

Octobre - Décembre 2018

Table des matières

1	Introduction	3
1.1	Présentation du projet	3
1.2	Architecture	4
1.3	Déroulement du travail	4
1.4	Choix des principaux outils et technologies	5
2	Interface utilisateur	5
2.1	Outils Utilisateur	5
2.2	Accès aux projets	6
2.3	Problèmes rencontrés	8
3	API gestion de projet	8
3.1	Présentation	8
3.2	Outils utilisés	8
3.3	Fonctionnalités implémentées	9
3.4	Problèmes rencontrés	9
4	L'autocomplétion	10
4.1	Recherche de solution	10
4.1.1	Solution 1 : Fonctionnalité native d'un éditeur de texte	10
4.1.2	Solution 2 : Cookie local	10
4.1.3	Solution 3 : Fichier index côté serveur	11
4.1.4	Solution 4 : Language Server Protocol	11
4.2	Solution retenue : Langage Server Protocol	11
4.2.1	Initialiser la communication	11
4.2.2	Exemple de discussion	12
4.3	Difficultés rencontrées	12
5	Déploiement automatisé des environnements	13
5.1	API de gestion des environnements	13
5.1.1	API environnement	13
5.1.2	Implémentation	13
5.2	Intégration continue (CI)	13
6	Debugger	14
6.1	Problématique	14
6.2	Solutions	14
7	Conclusion	16
8	Annexe A	18
9	Annexe B : renseignements sur le dépôt Git	19

1 Introduction

1.1 Présentation du projet

Ce projet a pour but de développer un environnement de développement en ligne qui doit être accessible depuis un navigateur web classique sans avoir à installer d'outils sur le poste des utilisateurs. Les principales fonctionnalités de cet environnement sont l'édition, la compilation et l'exécution de code simple. La compilation et l'exécution sont effectuées sur un serveur en étant transparentes pour l'utilisateur.

Cette application est destinée principalement aux étudiants de première année de licence de l'Université d'Angers. Ils font du développement dans le cadre de leur formation et doivent pouvoir continuer leur travail en dehors de l'Université sans installer d'environnement de développement sur leur poste personnel. Si besoin et si les fonctionnalités de l'application le permettent, elle pourra être utilisée tout au long de la licence et même en première année de master.

L'application a été commencée l'année dernière dans le cadre de la concrétisation disciplinaire.

Elle possédait les fonctionnalités de base suivantes :

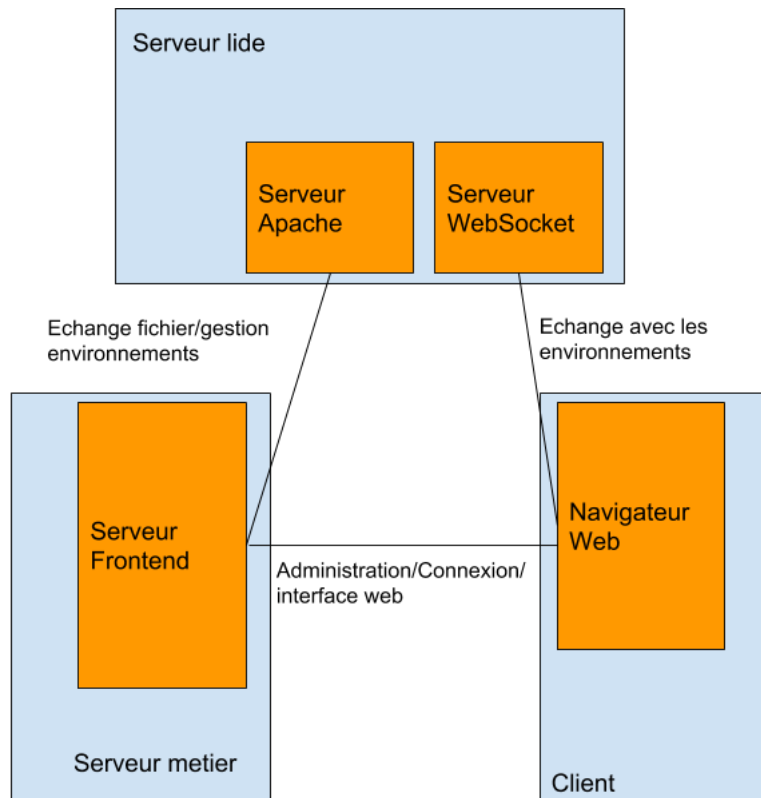
- Plusieurs langages de programmation peuvent être utilisés. Les administrateurs peuvent ajouter des environnements d'exécution.
- La coloration syntaxique qui s'adapte au langage.
- La compilation et l'exécution s'effectuent sur le serveur et la sortie est affichée dans l'application.
- L'utilisateur a accès aux messages d'erreur.

Les fonctionnalités suivantes doivent être ajoutées dans le cadre de ce projet :

- L'auto-complétion du code.
- Afin de supprimer des pertes de données pour l'utilisateur et de fluidifier l'utilisation de l'application en réduisant les délais entre les actions de l'utilisateur pendant l'exécution, l'implémentation technique de la partie compilation/exécution doit être repensée.
- La mise à disposition d'un espace de stockage personnel pour chaque utilisateur avec une sauvegarde transparente des fichiers.
- La gestion de projets qui peuvent contenir plusieurs fichiers. Il doit être possible de partager les projets entre utilisateurs en les clonant.
- Un débogueur simple sur l'interface graphique avec ajout de breakpoint dans le code, possibilité de faire du pas à pas lors de l'exécution et d'observer l'état de la mémoire.
- Les professeurs doivent pouvoir utiliser un outil d'analyse du code des étudiants afin de pouvoir vérifier que la compilation du code fonctionne, de tester la sortie du programme, etc.

1.2 Architecture

L'architecture générale de l'application est composée de deux serveurs, un dédié à un site web servant d'interface pour les utilisateurs et un serveur métier pour l'exécution des différents environnements de développement avec lesquels les utilisateurs communiquent (voir cahier des charges).



1.3 Déroulement du travail

Nous avons décidé ensemble de la répartition des tâches suivante :

- API Gestion de projet : Romane encadrée par Paulin.
- Déploiement automatisé des environnements : Romain encadré par Julien.
- Interfaces EDI et liste des projets : Gabriel encadré par Paulin
- Auto-complétion : Etienne encadré par Louis.

Nous avons établi un planning selon cette répartition (voir le diagramme de Gantt en annexe A).

1.4 Choix des principaux outils et technologies

L'application existante était développée en PHP 7.2 (dernière version LTS¹) à l'aide du framework Symfony qui permet une bonne organisation du code et une gestion simple des bases de données par un ORM², grâce à la bibliothèque Doctrine. Il inclut de nombreuses briques logicielles facilitant le développement du site, telles que des outils de validations des requêtes, un routeur ou encore l'injection de dépendances. Nous avons donc continué d'utiliser ce framework.

Le système de gestion de base de données choisi est MariaDB mais l'utilisation de l'ORM Doctrine permet d'être compatible avec d'autres systèmes de gestion de base de données si besoin (MySQL, PostgreSQL, etc).

2 Interface utilisateur

2.1 Outils Utilisateur

Les langages choisis sont Javascript, HTML5 et CSS3.

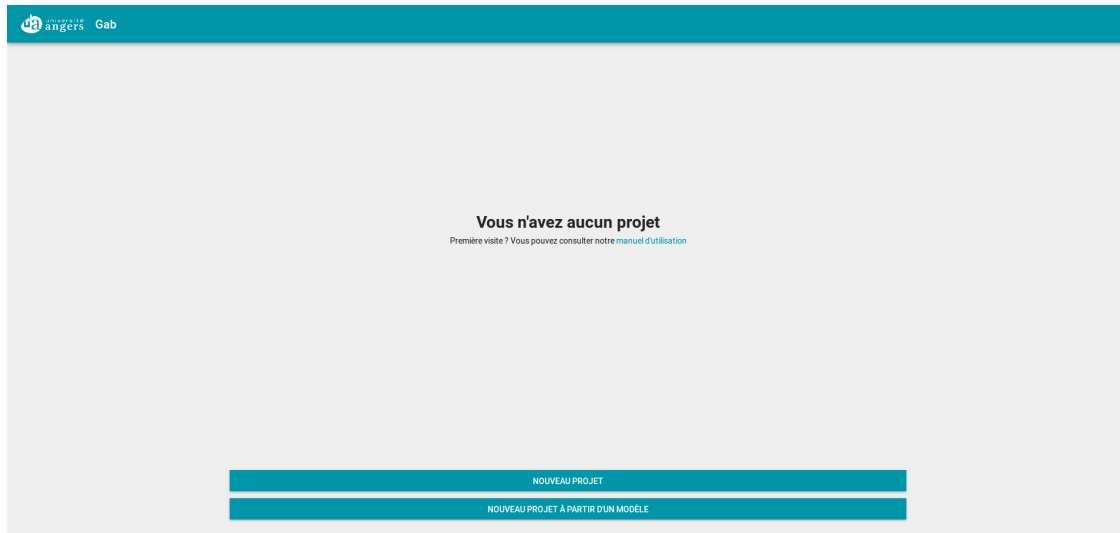
VueJS est très performant à l'exécution et ne met à jour que les composants qu'il faut. Le code est facilement testable et maintenable. Ce framework permet de découper les pages web en composants réutilisables, ayant chacun sa propre partie. En plus de ces caractéristiques VueJS est très réactif et si une donnée est changée, elle est mise à jour automatiquement sur tout le site. Axios fonctionne avec un système de promesses utilisé pour réaliser des traitements asynchrones. C'est un proxy vers une valeur qui n'est pas nécessairement connue au moment de sa création. VueJS récupère les promesses de l'API.

Vuetify est un framework VueJs fournissant tous les composants (boutons, formulaires, ...) qui sont basés sur les normes Material Design de Google. Vuetify est également facile à prendre en main, ce qui est avantageux pour réaliser des projets en peu de temps.


1. Long-Term Support
2. Object-Relational Mapping

2.2 Accès aux projets

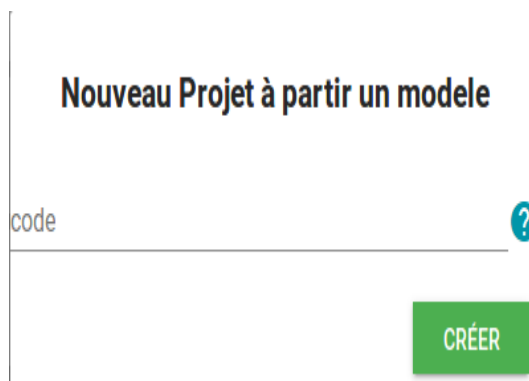
Lorsque l'utilisateur se connecte sur la page d'accueil, son nom s'affiche en haut à gauche.



Les 2 boutons ouvrent une boîte de dialogue dans laquelle on peut créer le nouveau projet. Si l'utilisateur veut créer un nouveau projet, il entre un nom au projet ainsi que le langage avec lequel il souhaite coder. Sinon, il peut toujours dupliquer un projet existant en entrant le code du projet.

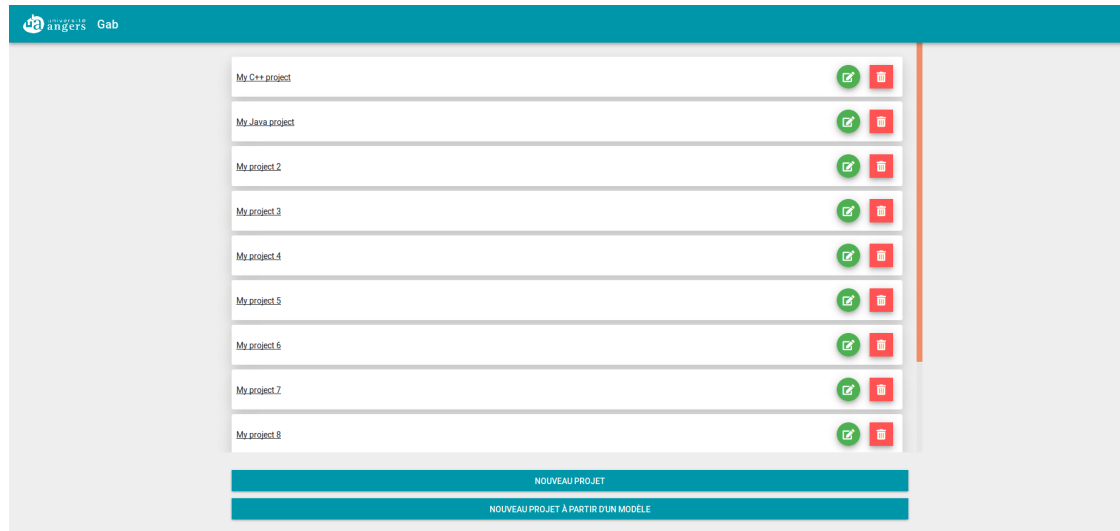
This form is titled 'Nouveau Projet' with the subtitle 'Configurez votre projet'. It contains two input fields: 'Nom' and 'Environnement', each with a help icon. At the bottom, there are two buttons: 'ANNULER' and 'GO!'.

(a) Boîte de dialogue pour créer un nouveau projet

This form is titled 'Nouveau Projet à partir un modele'. It features a 'code' input field with a help icon. A 'CRÉER' button is located at the bottom right.

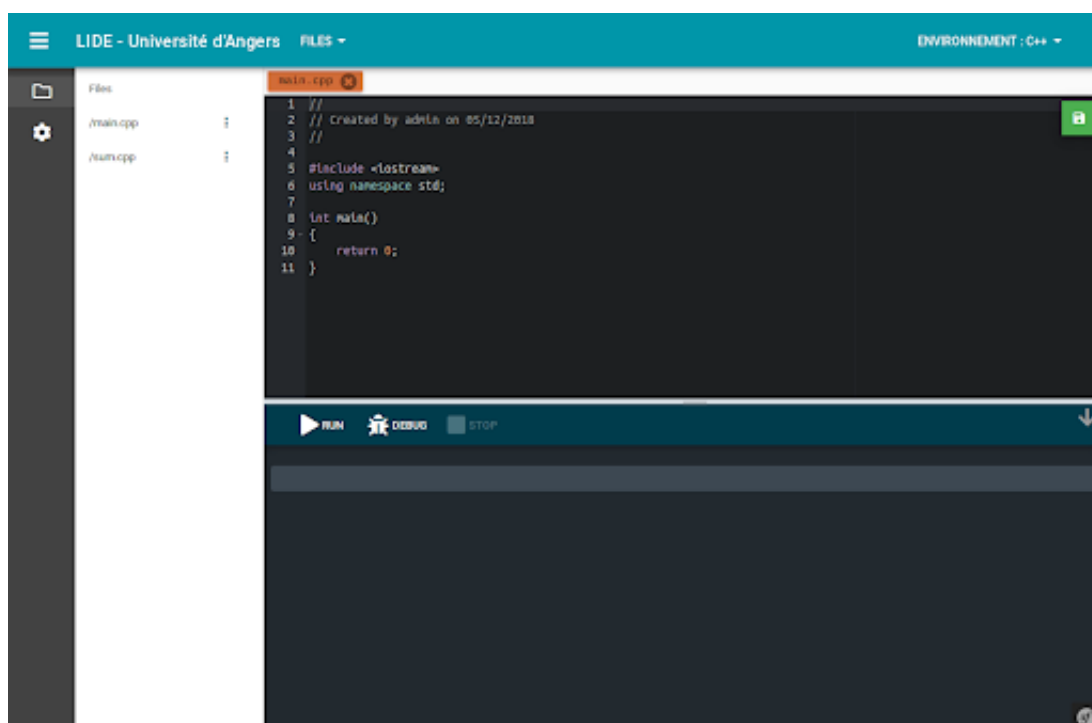
(b) Pour dupliquer le projet

Dans le cas où l'utilisateur possède des projets, la page d'accueil affiche une liste de projets et il est possible de supprimer un des projets.



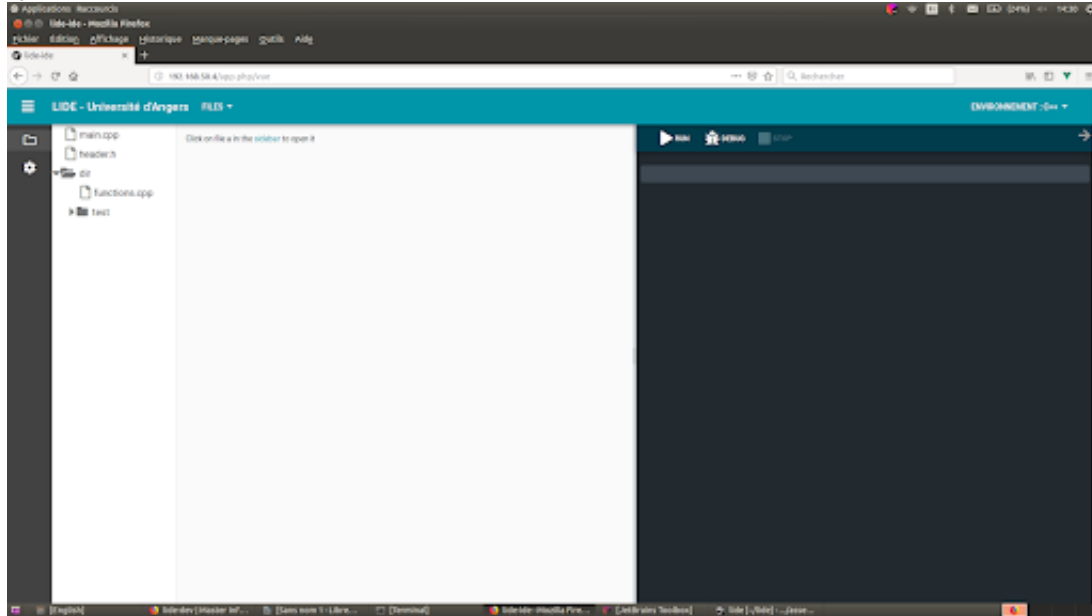
S'il clique sur le bouton vert ou sur le nom, il sera redirigé vers l'interface de l'application divisée en 4 parties :

- 1 - la possibilité de basculer entre le projet et la configuration de l'éditeur de texte.
- 2 - Le projet sous forme de liste ou le panneau de configuration du projet.
- 3 - Un éditeur de texte
- 4 - Une console



2.3 Problèmes rencontrés

La gestion des fichiers d'un projet était faite sous la forme d'un arbre mais le TreeView étant un composant apparu récemment, il n'était pas assez complet pour pouvoir ouvrir un fichier.



La prise en main de VueJs a ralenti le développement de l'interface graphique du fait de sa syntaxe.

3 API gestion de projet

3.1 Présentation

Cette API permet de fournir aux utilisateurs la possibilité de travailler avec des projets pouvant contenir plusieurs fichiers. L'interface utilisateur de l'application doit communiquer avec elle pour faire des requêtes HTTP CRUD³ pour gérer les projets.

3.2 Outils utilisés

Il a été proposé par les M2 d'utiliser le bundle Symfony FOSRestBundle qui fournit des outils pour créer une API Rest. Ce bundle facilite la désérialisation/sérialisation des données (passage du format json à un objet et inversement) et fournit un certain nombre d'annotations pour les contrôleurs.

Le composant Symfony FileSystem fournit des méthodes utiles pour interagir avec un système de fichiers. Il a été choisi d'utiliser ce composant pour stocker les fichiers. Les tests fonctionnels ont été réalisés avec Simple-PHPUnit.

3. Create, Read, Update, Delete

3.3 Fonctionnalités implémentées

- Projet
 - Récupérer les informations de tous les projets
 - Récupérer les informations d'un projet spécifique
 - Créer un nouveau projet
 - Supprimer un projet
 - Modifier les informations d'un projet
- Fichier
 - Récupérer les informations des tous les fichiers d'un projet
 - Récupérer les informations d'un fichier spécifique
 - Ajouter un fichier à un projet
 - Supprimer un fichier
 - Modifier les informations d'un fichier

Les informations des projets et fichiers sont enregistrées en base de données et le contenu des fichiers est stocké dans un répertoire sur le serveur métier “lide project manager app”. Il a fallu gérer le fait que les données sont stockées en base de données et dans un système de fichier afin éviter la désynchronisation des données. Pour cela, des classes de service ont été créées pour s'occuper de la partie système de fichier et les modifications sur les projets et fichiers sont réalisées à l'intérieur de transactions qui sont annulées si un problème quelconque est rencontré.

Les tests fonctionnels réalisés testent les routes afin de vérifier que les réponses sont correctes selon les requêtes. Ces tests n'opèrent que sur le code http retourné, les tests sur le contenu retourné n'ont pas été ajoutés par manque de temps. De même, des tests unitaires auraient pu être ajoutés mais ne l'ont pas été par manque de temps, les tests fonctionnels ayant été privilégiés afin de vérifier que l'application fonctionne comme voulu.

3.4 Problèmes rencontrés

Le principal problème rencontré était le manque de connaissance sur php et sur le framework Symfony. Il a donc fallu passer par un temps de documentation nécessaire pour la suite du projet.

Le manque de temps, en partie dû à la période en alternance, a aussi été un obstacle important qui a entraîné un retard dans l'implémentation de cette API. C'est pourquoi la possibilité de cloner un projet n'a pas été implémentée et les tests ne sont pas complets. Le retard de l'implémentation de l'authentification, réalisée par les M2, a aussi été un frein pour terminer rapidement les développements de cette partie, notamment pour la gestion du système de fichiers.

4 L'autocomplétion

L'autocomplétion est l'une des fonctionnalités les plus importantes parmi celles encore manquantes dans le projet. Les utilisateurs ciblés étant les étudiants en première année de licence novices en informatique, les fonctionnalités facilitant le développement sont essentielles. L'autocomplétion est une de ces fonctionnalités, permettant de limiter fortement les erreurs de saisie. Il est nécessaire de pouvoir fournir à tout moment à l'utilisateur une liste de propositions de complétion pertinente.

4.1 Recherche de solution

Avant l'implémentation, une longue phase de recherche a été menée afin de collecter et comparer différentes solutions d'implémentation de l'autocomplétion permettant de répondre à toutes les exigences demandées, notamment de fournir des propositions tirées de fichiers contenus dans le projet de l'utilisateur.

Le point commun entre toutes les solutions est le remplacement de l'éditeur de texte présent (Ace). Plusieurs alternatives existent, comme CodeMirror. Finalement, Monaco a été choisi pour des raisons que je décrirai plus loin.

4.1.1 Solution 1 : Fonctionnalité native d'un éditeur de texte

Une des premières pistes que j'ai explorées était d'intégrer directement dans le projet un éditeur de texte qui supporte nativement l'autocomplétion. Malheureusement, si beaucoup d'éditeurs de texte permettent l'autocomplétion, je n'en ai pas trouvé un capable de le réaliser par lui-même au niveau demandé par le projet. La principale limitation était que la liste de propositions se limitait aux mots clés du langage et aux données du fichier, là où obtenir des propositions provenant d'autres fichiers rédigés par l'utilisateur étaient nécessaires.

4.1.2 Solution 2 : Cookie local

Une autre possibilité était de réaliser un index qui serait stocké côté client. Un fichier contenant toutes les propositions de complétion du projet (par exemple, les noms de variables ou de fonctions créées par l'utilisateur) servirait de référence pour l'éditeur. Deux grandes difficultés se présentent alors.

Premièrement, devoir passer par des cookies a des limitations. Leur taille et leur nombre sont limités, ce qui pourrait forcer à séparer l'index en plusieurs fichiers. De plus, les cookies ont une durée de vie limitée dont il faut tenir compte.

Deuxièmement, réaliser cet index pour ensuite l'envoyer au client soulève en soit quelques soucis majeurs, le principal étant de devoir réaliser cet index et donc de parser les fichiers de l'utilisateur. De plus, l'index doit s'adapter en temps réel aux modifications de l'utilisateur, ce qui oblige à télécharger en plus de l'index un parseur adapté au langage utilisé. Cette surcharge inutile pour le client peut facilement être évitée en optant pour une solution plus adaptée.

4.1.3 Solution 3 : Fichier index côté serveur

Cette solution est équivalente à la précédente. L'éditeur de texte se réfère toujours à un index de propositions de complétion. Néanmoins, cet index serait situé dans le back-end, avec les autres fichiers sauvegardés par l'utilisateur. A chaque fois que l'éditeur cherche à obtenir la liste des propositions de complétion pertinentes, il envoie une requête au back-end qui, à l'aide de l'index, renvoie en réponse une liste de propositions de complétion.

Cette solution nécessite une communication fréquente entre le client et le back-end, mais cela ne pose pas de réels soucis. Par contre, la réalisation de l'index et donc le parseur reste nécessaire, même s'il n'a plus à être téléchargé par l'utilisateur. Il faut néanmoins, afin que l'index soit constamment à jour, que le client communique également au serveur chaque modification effectuée sur le fichier en cours de modification en temps réel.

La principale difficulté reste le parseur en lui-même. Il semble difficile de considérer devoir réaliser un parseur pour chaque langage proposé par l'IDE. Trouver des parseurs déjà réalisés et implémentables s'est aussi montré plus difficile que prévu.

Néanmoins, cette solution, mis à part les quelques difficultés qu'elle présente pour son implémentation, répond à toutes les demandes concernant la fonctionnalité demandée.

4.1.4 Solution 4 : Language Server Protocol

Cette dernière solution est, dans son concept, proche de la précédente. A chaque fois que l'éditeur de texte a besoin d'une liste de propositions de complétion, il envoie sa requête côté serveur. La différence ici est que cette communication se déroule entre l'éditeur de texte et le serveur langage, en suivant le protocole LSP. Ce protocole définit la communication entre un éditeur et un serveur langage via des requêtes JSON empaquetées dans des web-sockets.

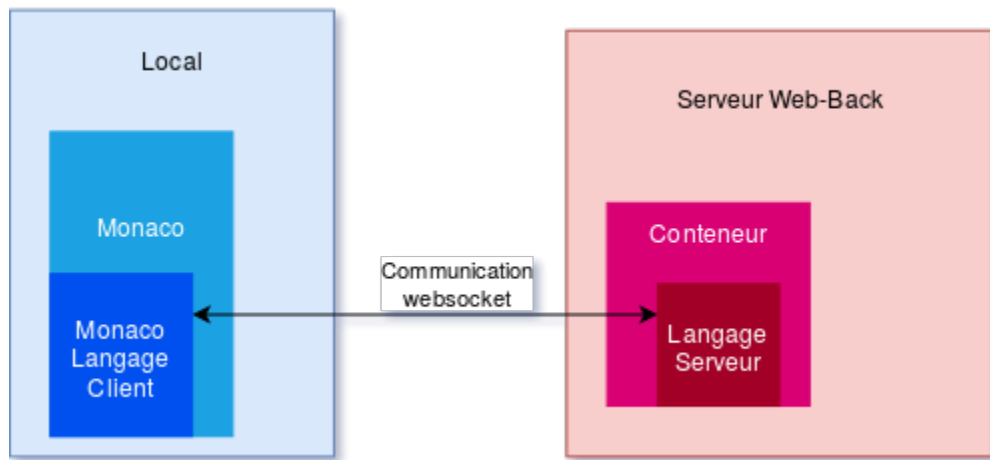
Adopter ce protocole présente plusieurs avantages intéressants. Premièrement, plusieurs éditeurs de texte et plusieurs serveurs langage disposent déjà d'implémentations existantes pour le LSP, ce qui économise une grande charge de travail. Par exemple, l'éditeur de texte Monaco dispose d'un module dédié à la communication LSP. Deuxièmement, ce protocole nous épargne la réalisation d'un parseur ou d'un index. La plupart des serveurs langage effectuent déjà ces tâches et peuvent directement discuter avec l'éditeur sur ces sujets via LSP. Enfin, ce protocole standardise la plupart des communications nécessaires entre un éditeur de texte et un serveur langage ce qui permettra à l'avenir de pouvoir facilement intégrer des fonctionnalités supplémentaires (telles que le formatage, le saut vers la définition ...)

4.2 Solution retenue : Language Server Protocol

4.2.1 Initialiser la communication

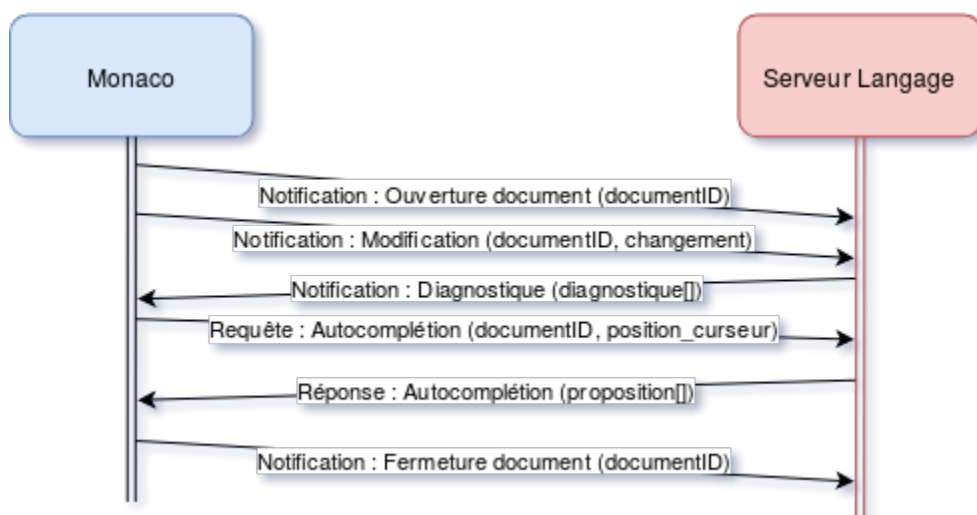
Côté serveur, un conteneur déploie un langage serveur possédant une implémentation LSP (il est possible de développer une implémentation de LSP adaptée au langage serveur utilisé, mais il est plus simple d'implémenter directement un langage serveur possédant déjà une implémentation LSP ; il est possible d'en obtenir facilement pour la plupart des langages courants). Il suffit d'y ouvrir une connexion en websocket.

Une fois l'éditeur de texte Monaco et son module Monaco Language Client incorporé au projet, il faut qu'avant l'exécution de Monaco une connexion en websocket soit configurée et initialisée avec le langage serveur situé dans le web-back. L'initialisation de cette connexion est facilitée par la bibliothèque contenue dans le module.



4.2.2 Exemple de discussion

Une fois la communication établie, lorsque l'utilisateur ouvre un fichier, le serveur langage est notifié de l'ouverture du document. A partir de cet instant, le fichier n'est plus pris en compte et les deux intermédiaires en possèdent une copie en mémoire. Chaque modification sur Monaco est notifiée au serveur langage, qui y répond par un rapport de diagnostic sur le texte prenant en compte ces modifications. Lorsque Monaco a besoin d'une liste de propositions à afficher (par défaut, à chaque fois qu'un caractère est tapé par l'utilisateur), une demande est effectuée. Le serveur langage, ayant accès à l'ensemble des fichiers de l'utilisateur, y répond avec une liste "tokenisée" de tous les éléments pertinents. La discussion se termine par la notification de fermeture du document.



4.3 Difficultés rencontrées

L'intégration de l'autocomplétion a été provisoirement mise de côté lors du projet. En effet, l'intégration de Monaco était une nécessité, mais cette intégration s'est révélée plus difficile que prévu. Elle est actuellement mise en attente en faveur d'autres tâches plus fondamentales.

5 Déploiement automatisé des environnements

5.1 API de gestion des environnements

initialement la création de nouveaux environnements de développement se faisait directement sur le serveur métier “lide project manager app” via la construction d’images docker créées par un enseignant.

Le but était de déplacer cette gestion en fournissant une api accessible par le serveur web depuis l’interface d’administration.

5.1.1 API environnement

C’est une API de type REST qui permet la communication. Le framework utilisé étant Symfony, il a directement été possible d’y intégrer ce type d’API sur le serveur. Docker possède sa propre API REST accessible soit par socket unix soit par HTTP. L’utilisation de l’api via HTTP pose un certain problème de sécurité car toutes les fonctionnalités seraient accessibles de l’extérieur du serveur, d’où l’utilisation de socket unix servant à diminuer le nombre de fonctions via l’API environnement.

5.1.2 Implémentation

L’API environnement utilise un bundle Symfony appelé Docker-php pour simplifier les requêtes avec Docker. Ce bundle utilise le standard psr-7 de php pour les interfaces des messages HTTP, ce qui permettra une communication uniforme même si une refonte du code est nécessaire.

L’API est composée de quatre fonctionnalités via des routes Symfony :

- lister les images
- supprimer une image
- construire une image
- récupérer les logs de construction

Lors de l’envoi d’une nouvelle image, une copie de celle-ci est faite sur le serveur ainsi que des logs générés par Docker lors de la construction. En effet, la construction d’une image pouvant échouer en cas d’erreur d’écriture du Dockerfile, les logs générés sont conservés jusqu’à la destruction de l’image via l’API.

5.2 Intégration continue (CI)

L’intégration continue est l’une des étapes du développement logiciel, elle permet la vérification du bon fonctionnement de celui-ci lors de modifications via des tests automatisés.

Pour la réalisation des tests, Symfony possède un bundle intégrant php-unit, il permet la création de tests sans avoir à utiliser une connexion http ce qui permet de détecter directement les problèmes liés au code et non juste à de mauvaises configurations. Les sources du développement de lide étant stockées sur Gitlab, cette plateforme permet aussi la mise en place de tests automatisés via l’intégration continue, c’est la raison principale de l’utilisation de Gitlab, les outils nécessaires sont déjà présents et intégrés à la plateforme, en effet les autres plateformes comme Github par exemple demanderaient une intégration avec un outil de test d’intégration, une autre pour le déploiement.

Elle utilisera donc les tests écrits à l'aide de php-unit mais cette fois, ils seront exécutés par Gitlab. Pour ce faire, nous devons préciser une image Docker que la plateforme de CI utilisera pour la réalisation des tests. Cette image peut être une image déjà existante mais aussi une image créée et stockée par l'utilisateur. Dans notre cas, pour le serveur métier, il nous suffit d'y installer la stack logicielle nécessaire comme Symfony et le serveur HTTP Apache puis d'héberger l'image sur Gitlab grâce à sa fonction de registre d'image.

Les tests étant liés à la création d'images Docker, la simple installation de Docker dans un conteneur ne fonctionnera pas sans une exécution privilégiée du conteneur de test ainsi qu'un appel à un service dind (Docker in Docker). Une exécution privilégiée peut être demandée via un fichier contenant les options pour que Gitlab le prenne en compte. Le service dind est lui mis à disposition par Gitlab, ici utilisé pour les tests appelant Docker. Les appels à Docker du conteneur seront redirigés vers la machine host.

La création d'une image pour les tests permettra de simplifier la création de futurs tests mais nous donne aussi la possibilité d'utiliser cette image comme environnement de développement à la place des machines virtuelles utilisées actuellement, voire pour une utilisation en production.

6 Debugger

Le debugger est une fonctionnalité importante qui permettra le suivi des variables utilisées ainsi que l'utilisation de breakpoints pour suivre l'exécution d'un programme.

6.1 Problématique

Chaque langage ayant son propre debugger, et chaque debugger sa propre API, il nous faudrait pouvoir utiliser un protocole permettant au moins la pose de breakpoint et la consultation des variables. De plus, les API mises à disposition par les debuggers sont utilisables sous forme de bibliothèque et non de serveur.

C'est pourquoi il est nécessaire pour chaque langage de créer un serveur permettant la communication avec le debugger correspondant, ce serveur aurait une API de plus haut niveau et commune aux langages.

6.2 Solutions

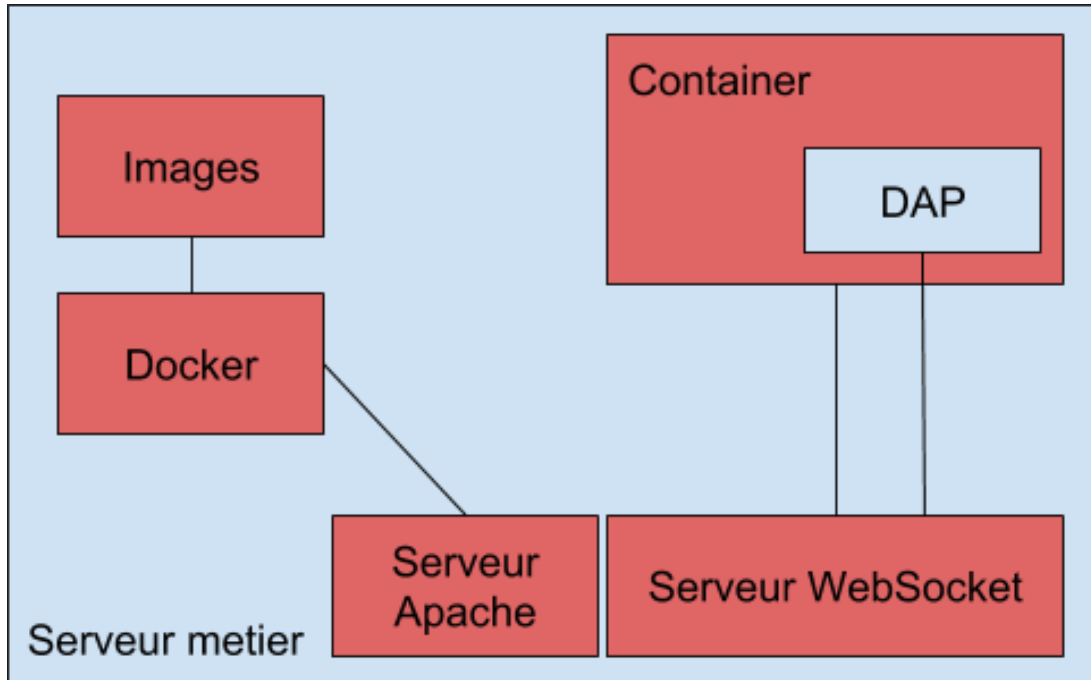
Afin de permettre aux développeurs de déboguer à distance dans le cadre de projets ayant des impératifs de configuration différente de la machine de développement par exemple, des serveurs de débogage peuvent exister comme gdbserver, mais demandent d'utiliser le client correspondant ou demandent certaines adaptations, ce qui en fait une solution possible mais limitant le nombre de langages utilisables.

Dans le cadre du développement de Visual Studio Code (VS Code), Microsoft a développé un protocole qui répond à ce besoin, appelé le Debug Adapter Protocol (DAP). Les extensions de VS Code mettant à disposition un debugger utilisent ce protocole pour communiquer avec l'éditeur.

Malheureusement, il n'existe pas d'implémentation du DAP qui ne soit pas couplée avec une extension de VS Code, c'est pourquoi il nous faudrait extraire l'implémentation sans la partie extension de VS Code et la mettre à disposition dans un conteneur. Lors du lancement

d'un programme, via le serveur websocket, l'utilisateur pourra directement interagir avec le DAP et ainsi utiliser le debugger.

Le minimum à développer serait une image Docker par langage avec leur DAP respectif, ces images serviront de base pour les environnements créés par les enseignants. Le serveur websocket communiquera avec le DAP du conteneur en relayant les requêtes émises par l'éditeur de texte du client.



Le principal problème est de réimplémenter des serveurs par langages implémentant le DAP ou l'extraction de cette partie depuis les extensions de VS Code. La seconde option semble être plus rapide car nous partons du déjà existant mais demande un travail en amont en ce qui concerne le framework de VS Code.

7 Conclusion

Ce projet a été très enrichissant pour nous tous. Il nous a permis d'acquérir de nouvelles compétences techniques, mais aussi d'apprendre à travailler en équipe avec des chefs de projets. Grâce à cette expérience, nous avons pu nous confronter à des problématiques de délais et de respect du cahier des charges qui seront importantes pour notre future vie professionnelle.

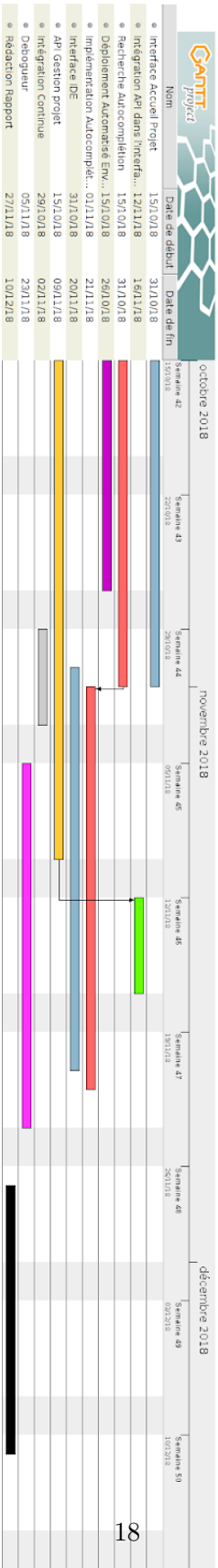
L'application n'est pas terminée car toutes les fonctionnalités prévues ne sont pas implémentées, il faudra donc l'améliorer afin qu'elle puisse être opérationnelle l'année prochaine pour les étudiants de première année de licence de l'Université. Il manque notamment l'auto-complétion et le debugger qui sont deux fonctionnalités nécessaires pour que l'application soit complète et confortable pour l'utilisateur.

Nous voudrions remercier nos chefs de projet qui nous ont accompagnés et guidés tout au long de ce projet.

Références

- [1] Monaco Editor
<https://microsoft.github.io/monaco-editor/>
- [2] Monaco langage client
<https://github.com/TypeFox/monaco-languageclient>
- [3] Explication du LSP
<https://docs.microsoft.com/fr-fr/visualstudio/extensibility/language-server-protocol?view=>
- [4] Site officiel du LSP
<https://microsoft.github.io/language-server-protocol/>
- [5] Aggregateur d'implémentation du LSP
<https://langserver.org/>
- [6] Debug Adapter Protocol
<https://microsoft.github.io/debug-adapter-protocol/>
- [7] Docker-php
<https://docker-php.readthedocs.io/en/latest/>
- [8] Symfony
<https://symfony.com/>
- [9] Doctrine
<https://www.doctrine-project.org/>
- [10] Simple-PHPUnit
https://symfony.com/doc/current/components/phpunit_bridge.html

8 Annexe A



9 Annexe B : renseignements sur le dépôt Git

Liens des dépôts Git : <https://gitlab.com/ua-lide>

Pseudonymes :

- Romain Grelier : romgrelier
- Gabriel Mouget : Gab961
- Romane Poirier : rompoirier
- Etienne Tignon : TeddyTi
- Paulin Violette : pviolette-fr / pviolette
- Julien Fontaine : fonfonfaya
- Louis Marchand : louismarchand