



M1 INFORMATIQUE

CONCRÉTISATION DISCIPLINAIRE

---

# L.I.D.E. Application Web de Développement

---

*Auteurs*

Yassine IBRIR  
Jerôme MARTINS MOSCA  
Valentine RAHIER  
Paulin VIOLETTE

*Chefs de projet*

Alice BAZANTÉ  
Sullivan CHEVALLIER  
Pierre-Olivier MAINFROID

*Référent :*

Laurent GARCIA

Octobre - Décembre 2017



# Table des matières



## Contexte

Dans le cadre des enseignements d'algorithmie proposés par l'Université d'Angers pour le parcours MPCIE, les étudiants vont apprendre à programmer avec différents langages. Un problème peut apparaître lorsque ces étudiants souhaitent retravailler leurs exercices sur d'autres postes que ceux du département informatique ; les outils nécessaires pour programmer (a minima un compilateur et un éditeur de texte) ne seront pas systématiquement installés sur les ordinateurs et leurs installations peuvent se montrer compliquées voire impossibles si les droits d'administrateurs ne sont pas attribués à l'utilisateur (c'est le cas sur les postes de la bibliothèque universitaire, par exemple).

De ce fait, l'Université d'Angers a souhaité 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

## Présentation du projet

### 1.1 Sujet

Notre objectif était de réaliser la première version d'un environnement de développement simplifié accessible depuis un navigateur web. Il devait permettre d'écrire et de compiler du code rapidement et simplement sans avoir à installer de compilateur sur le poste client (la compilation s'effectuant sur le serveur). Cet outil pourrait être utilisé, dans le cadre des enseignements à l'Université d'Angers, dans plusieurs unités dès la L1 jusqu'à la L3 (voire jusqu'au master si certaines fonctionnalités adaptées sont implémentées).

Certaines caractéristiques nous étaient demandées :

- l'édition de code dans un éditeur proposant la coloration syntaxique
- la possibilité de compiler le code grâce à des compilateurs installés sur le serveur. Le logiciel devait prendre en compte différents langages afin d'être utilisable dans différentes UE
- l'accès aux messages d'erreur de la compilation
- l'exécution de l'application compilée sur le serveur avec affichage de la sortie
- des fonctionnalités avancées devaient être développées telles que l'intégration du débogueur, une gestion plus poussée de l'ensemble de fichiers composant un projet, l'auto-complétion du code, l'intégration d'outils d'analyse

### 1.2 Problématique soulevée

L'enjeu principal de ce projet était la sécurité du serveur. En effet, l'application va compiler et exécuter du code inconnu. Il faut, d'une part, empêcher qu'un utilisateur puisse obtenir des accès qu'il ne devrait pas obtenir, et aussi, gérer les inévitables programmes trop gourmands. Lorsqu'un étudiant tente d'exécuter un programme qui contient des erreurs ou qui demande trop de mémoire CPU, il ne faut pas que le serveur qui gère l'exécution ni que les exécutions d'autres étudiants soient ralenties ou bloquées. Il est donc nécessaire d'élaborer une architecture visant à répondre à ces problématiques. Nous avons décidé d'utiliser une technologie de conteneurisation pour répondre à cette problématique.

Le projet se devait également d'être modulable, afin de pouvoir facilement ajouter le support de nouveaux langages. À cette fin, nous avons décidé de conserver les informations liées à chaque langage supporté dans une base de données relationnelle.

Nous avons aussi décidé de réduire l'accès à l'application, en demandant aux utilisateurs de s'inscrire via une adresse e-mail (inscription pouvant être ouverte seulement à certaines personnes, par exemple sur la base du nom de domaine d'une adresse e-mail : cela permet d'empêcher les personnes extérieures à l'Université d'utiliser l'application, réduisant ainsi la charge sur les serveurs). Afin de pouvoir répondre plus facilement à cette problématique et à la précédente, nous avons choisi d'utiliser un framework déjà existant qui nous permettrait de faciliter la gestion de ces fonctionnalités.

### 1.3 Choix des principaux outils et technologies

Lors de la première séance de concrétisation disciplinaire, nos chefs de projet nous ont proposé d'utiliser le framework Symfony<sup>1</sup> pour la réalisation de notre application. Ce framework force à organiser le code et permet une gestion simple de la base de données qui ne dépend pas du type de celle-ci. En effet, Symfony intègre la

---

1. Voir <https://symfony.com/>

bibliothèque Doctrine<sup>2</sup>, un ORM<sup>3</sup> facilitant la manipulation de bases de données et permettant le mapping d'une base de données relationnelle avec des objets PHP. De plus, Symfony permet une génération simple des pages grâce à ses contrôleurs et au moteur de templates twig. L'autre avantage qui nous a décidé à choisir Symfony est la possibilité d'utiliser des bundles (par exemple, FOSUserBundle permet la gestion des utilisateurs) qui simplifient et accélèrent véritablement la réalisation des projets.

Nous avons choisi d'utiliser MariaDB comme système de base de données. MariaDB est l'un des systèmes de gestion de base de données les plus importants et a l'avantage d'être distribué sous licence libre. De plus, cet outil est soutenu par une large communauté, ce qui assure une maintenance sur le long terme. Néanmoins, ce SGBD<sup>4</sup> peut facilement être remplacé par un autre au besoin, puisque toutes les manipulations de base de données se font via l'ORM Doctrine. Le seul pré-requis est que le SGBD soit supporté par Doctrine.

Bootstrap<sup>5</sup> est un framework HTML/CSS/JS simple d'utilisation. Il utilise un layout sous la forme d'une grille qui s'adapte en fonction de la taille de l'écran. Il inclut aussi un style de base pour tous les éléments HTML, ce qui permet d'obtenir un style indépendant du navigateur ou de l'OS utilisé par l'utilisateur. Ce style peut également être modifié via des constantes SASS<sup>6</sup> si on souhaite aller plus loin dans la personnalisation de l'application.

Enfin, Bootstrap possède de nombreux composants faciles à manipuler via des attributs HTML (ou via le Javascript) tels que les dropdowns, modals, alert...

La conteneurisation, et plus particulièrement Docker<sup>7</sup>, s'est vite imposée dans l'architecture du serveur. Docker est une méthode de conteneurisation légère qui permet de travailler toujours sur le même environnement (la même image est réutilisée autant de fois que nécessaire) et qui nous a permis d'isoler les compilations et exécutions des programmes.

## 1.4 Répartition des tâches

Afin de travailler efficacement, nous avons séparé le projet en 4 parties plus ou moins autonomes : Paulin s'est occupé de l'interface graphique, Yassine de l'administration, Jérôme de l'architecture du serveur et Valentine de la communication entre les serveurs.

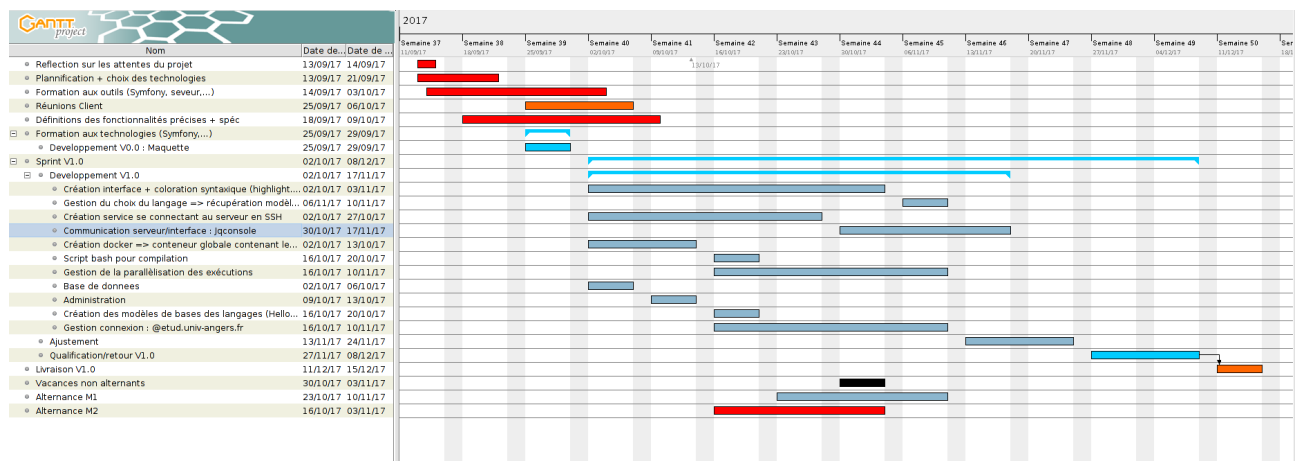


FIGURE 1.1 – Planning (En rouge : tâches réalisées par les M2, en bleu tâches réalisé par les M1, en orange avec les enseignants)

2. Plus de détails sur <http://www.doctrine-project.org/>

3. Object-Relationnal Mapping

4. Système de Gestion de Base de Données

5. Voir <https://getbootstrap.com/>

6. Syntactically Awesome Stylesheets

7. <https://www.docker.com/>

## Chapitre 2

# Base de données et administration

## 2.1 Conception

### 2.1.1 Les démarches de conception

Les Acteurs : un acteur représente l'abstraction d'un rôle joué par des entités externes. Dans notre application on distingue principalement deux acteurs qui sont les suivant :

- L'utilisateur qui peut écrire et compiler son code
- L'administrateur qui possède les même droits que l'utilisateur mais possède aussi un accès à la page d'administration qui permet de modifier les tables Langages, Détails\_langages et Serveur (voir ??)

### 2.1.2 Modèle conceptuel des données (MCD)

Le modèle conceptuel des données a pour but de définir de façon formelle la structure de la base de données qui sera utilisée par le système d'information. Il s'agit donc d'une représentation du modèle de données, facilement compréhensible, permettant de décrire le système d'informations à l'aide d'entités. La figure suivante présente le modèle conceptuel des données.

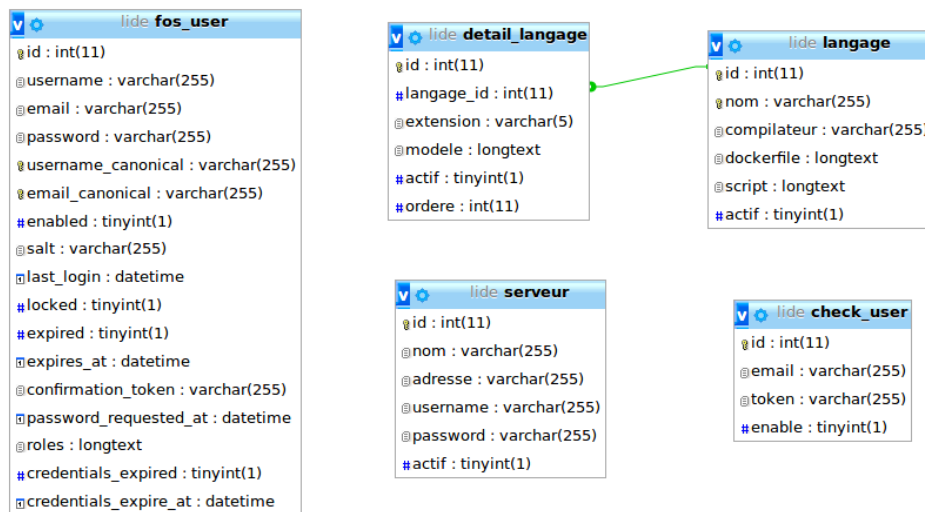


FIGURE 2.1 – Modèle conceptuel des données

## 2.2 Base de données

### 2.2.1 Partie applicative

Les trois tables suivantes contiennent un champ actif qui permet d'activer ou de désactiver le tuple. Par exemple si un serveur est en maintenance ou ne doit pas être utilisé, il pourra être désactivé et réactivé au besoin par l'administrateur. C'est la même chose pour la table langage et details\_langage.



### 2.2.1.1 Serveur

La table Serveur permettra de connaître les serveurs disponibles. S'il y a une surcharge, l'administrateur aura juste à ajouter un nouveau serveur et l'application pourra directement l'utiliser si besoin.

### 2.2.1.2 Langage

Le nom d'un langage est UNIQUE. La table Langage contient le Dockerfile permettant d'installer le nécessaire pour pouvoir compiler/exécuter le langage. Il y a aussi le script qui permet de lancer la compilation avec les fichiers de l'utilisateur ainsi que le compilateur. L'avantage de stocker en base de données les langages est de rendre autonome les enseignants. S'ils veulent rajouter un nouveau langage, il leur suffit de l'ajouter avec son Dockerfile et son script.

### 2.2.1.3 Détails langage

La table details\_langage contient les extensions du langage correspondant ainsi qu'un modèle de base (par exemple le Hello World). Cela permet de proposer à l'utilisateur l'extension qu'il veut écrire.

## 2.2.2 Partie gestion des utilisateurs

### 2.2.2.1 Fos user

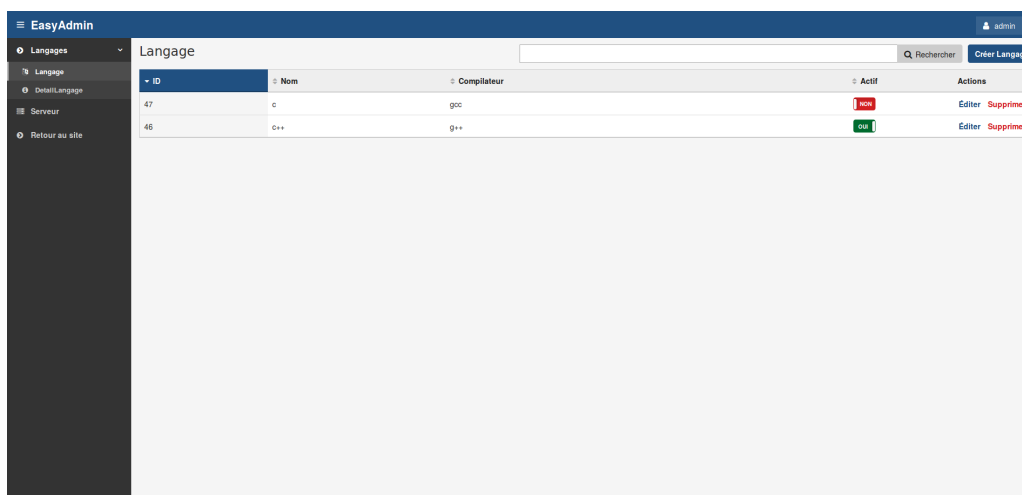
Dans cette table, on sauvegarde tous les informations des utilisateurs.

### 2.2.2.2 Check\_user

Cette table a pour but de sauvegarder les e-mails des utilisateurs qui n'ont pas encore confirmé leur inscription via le lien de validation reçu dans leur boîte mail.

## 2.3 Administration

Pour mettre en place l'interface administrateur, nous avons utilisé différents outils proposé par Symfony et notamment les bundles FOSUserBundle et EasyAdminBundle.



ID	Nom	Compilateur	Actif	Actions
47	c	gcc	non	Editer Supprimer
48	C++	g++	oui	Editer Supprimer

FIGURE 2.2 – Page d'administration

### 2.3.1 FOSUserBundle

FOSUserBundle est un bundle qui fournit un système de gestion des utilisateurs complet.

Ce système comprend notamment :

- Un formulaire d'inscription avec e-mail de confirmation pour vérifier l'authenticité de l'adresse fournie par l'utilisateur
- Un système de récupération de mot de passe (de type «mot de passe oublié?»)
- Une compatibilité avec la librairie Doctrine pour le stockage en Base de données

Ces différentes fonctionnalités nous ont permis de sécuriser l'accès à l'application et aussi de paramétrer les droits d'accès à certaines ressources en attribuant des rôles.

### 2.3.2 EasyAdminBundle

Ce bundle nous a permis de créer facilement l'interface administrateur en générant automatiquement les différents formulaires associés aux entités, telles que User et Langage. Il fournit aussi la possibilité d'appliquer les opérations CRUD<sup>1</sup> sur les différentes entités.

Doctrine permet de créer des objets (entités) pour gérer les flux entrants et sortants vers la base de données. Ces entités gèrent également le côté relationnel des données. L'ORM Doctrine possède diverses commandes permettant la génération automatique des entités. Néanmoins, cette génération reste très basique et il est nécessaire de les modifier afin d'ajouter les relations inter-entités (clé étrangère, relation multidirectionnelle ...)

Les différentes requêtes SQL sont définies sous forme de fonctions dans les fichiers entités, ce qui permet de sécuriser l'appel à ces dernières depuis les contrôleurs.

---

1. Create, Read, Update, Delete

# Chapitre 3

## Serveur

Afin de garantir la bonne compréhension des différents termes techniques qui vont suivre, un lexique donnant toutes les définitions nécessaires est disponible en annexe.

### 3.1 Architecture

L'application web vise à être utilisée par des étudiants, certaines conditions de sécurité doivent donc être respectées.

Les différents programmes doivent pouvoir être lancés sur un environnement vierge indépendant et, bien sûr, isolés du serveur d'exécution afin de prévenir toute corruption de données ou problème de sécurité. Pour répondre à cette problématique, nous nous sommes tournés vers une technique encore jeune et prometteuse : la conteneurisation. Cette technique peut être utilisée via différentes technologies, celle que nous avons choisi est l'une des leaders du marché : Docker.

#### 3.1.1 Choix effectués

##### 3.1.1.1 Choix de la conteneurisation

La conteneurisation répond à nos critères d'environnement puisqu'elle permet de créer un espace propre indépendant et sans incidence sur l'environnement qui héberge le service de chaque exécution. Grâce à cette technique, nous avons pu gérer les problèmes de sécurité et d'allocation de ressources. Nous avons choisi la conteneurisation plutôt que les machines virtuelles car elle a l'avantage d'être plus légère mais aussi plus malléable.

##### 3.1.1.2 Choix de Docker

Docker est une technologie en renouvellement permanent, nous l'avons choisie pour plusieurs raisons.

D'abord, la conteneurisation avec Docker est une option enseignée par l'Université d'Angers. Aussi, Docker est accessible et documenté. La communauté de ce dernier est très importante. Cela nous donne l'avantage de compléter nos cours en trouvant des solutions à des problèmes réguliers sur des forums, tutoriels et autres outils communautaires.

#### 3.1.2 Détails de l'architecture

L'architecture du site se fait en plusieurs étapes. L'architecture actuelle est une première version qui doit être améliorée par une étude de produit plus poussée pour obtenir une architecture définitive.

### 3.1.2.1 Architecture actuelle du site

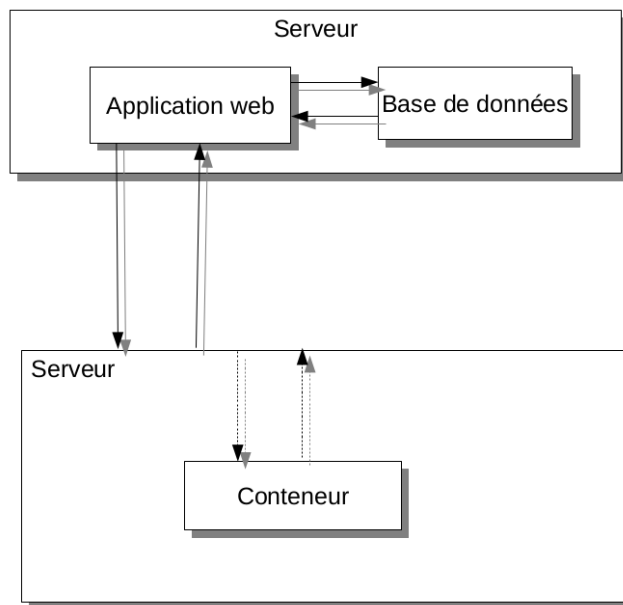


FIGURE 3.1 – Architecture actuelle

La version actuelle du site utilise deux serveurs : un serveur pour l'hébergement du site et un autre pour l'exécution des différents programmes créés par les étudiants. Docker permet de créer des conteneurs éphémères sans persistance de données à partir d'une image officielle (présente dans le catalogue de docker) ou d'une image personnelle générée à partir d'un Dockerfile. Quotidiennement, les Dockerfiles sont envoyés au serveur et les images générées afin de s'assurer de la bonne version de ces dernières. Chaque langage utilisé par les étudiants possède une image unique qui lui est propre, de cette façon les images sont plus légères, plus rapides à la génération et ne possèdent que les paquets et applications nécessaires au bon fonctionnement du langage utilisé.

Pour la première version du site, la communication entre l'application web et le serveur d'exécution s'effectue par SSH. Cette connexion permet l'envoi des instructions d'exécution du conteneur ainsi que les directives pour récupérer le fichier généré par l'application web qui sera à compiler. De cette manière, la gestion et la création des conteneurs est dynamique.

### 3.1.2.2 Architecture envisagée

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. 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.

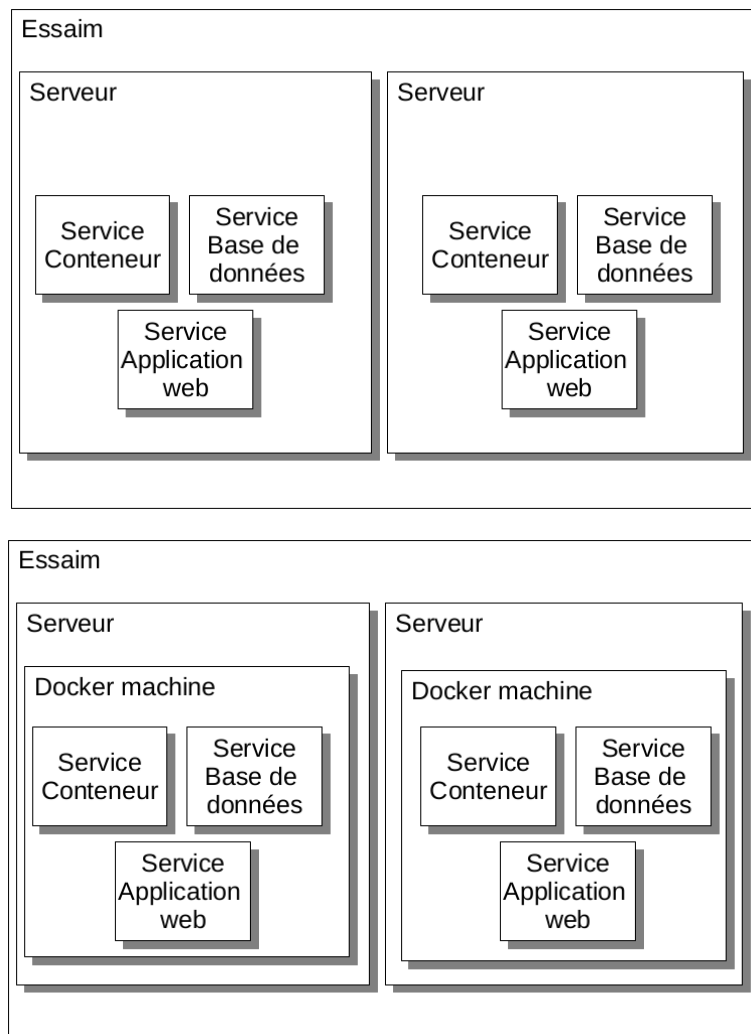


FIGURE 3.2 – Architectures envisagées pour la prochaine version

**3.1.2.2.1 Avantages de l'architecture envisagée** L'application sera accessible en haute disponibilité. Par conséquent, les différents services pourront posséder un certain nombre de répliques et pourront gérer le load-balancing. Docker s'occupera de la synchronisation des différentes répliques de façon automatique. De plus, si l'une des répliques n'est plus active, une autre est automatiquement créée empêchant ainsi l'application d'être impactée si un serveur venait à tomber.

La transformation de l'application sous forme de service permet également de créer un Composefile. Ce fichier permet de lancer les différents services de l'application simultanément mais également de les paramétrer. Ainsi, nous avons la certitude que tous les services sont lancés et ont la bonne configuration et, également, de garder une trace des versions.

De plus, un essaim Docker va être mis en place. Tous les serveurs utilisés par l'application feront partie de l'essaim, cela permettra de lier les serveurs entre eux afin de mettre en place le load-balancing.

La mise en place d'une telle structure soulève plusieurs questions.

**3.1.2.2.2 Quelle est la configuration des différents serveurs ?** Docker est une technologie dite portable mais comme évoqué précédemment cette technologie est encore jeune et possède nombre de limites pour le moment. Pour assurer un fonctionnement identique et sans problèmes des images entre plusieurs serveurs, il faut que ces dernières aient la même configuration (même version de docker, même kernel, et même configuration système que docker pourrait exploiter, comme la prise en charge de la mémoire swap).

**3.1.2.2.3 Comment gérer la communication des conteneurs entre les différents serveurs sur des réseaux différents ?** Docker gère les répliques de services, le reverse proxy ainsi que les différents services SSH. Cependant, si les serveurs ne sont pas sur le même réseau, la communication peut vite devenir problématique. Comment assurer la sécurité ? Comment prioriser le serveur le plus proche physiquement afin de limiter la communication et avoir une vitesse maximale ? Faut-il mettre en place un serveur de communication entre les

services au travers des serveurs de l'essaim ? Le principe de communication vers un conteneur à travers un service sur un essaim avec Docker reste encore assez flou à l'heure actuelle.

**3.1.2.2.4 Quel service utiliser pour mettre en place l'essaim et le load-balancing ?** Actuellement, deux alternatives dominant le marché : Kubernetes et Docker swarm. Nous avons tenté de mettre en place Docker swarm mais plusieurs problèmes sont apparus.

Deux façons de faire sont possibles ; soit ajouter les serveurs directement à l'essaim soit créer des Docker machines sur les serveurs que l'on ajoute ensuite. Les machines Docker permettent de créer un sous-réseau par hôte et d'améliorer la sécurité. Une fois les services définis et l'essaim en place, nous nous sommes rendus compte que la communication entre les services n'était pas aussi aisée que ce que l'on pouvait penser.

Après avoir parcouru la documentation de Docker au sujet des essaims, nous nous sommes aperçus que la communication inter-services via l'essaim était peu répandue et assez complexe. La mise en place d'un réseau de communication via Consul a été testée mais le temps restant ne nous a pas permis d'aller au bout de cet essai ni de l'étudier en profondeur. Ces différents problèmes ont remis en question l'utilisation de Docker swarm. Une étude des capacités de Docker swarm et de Kubernetes est nécessaire afin de déterminer laquelle est la plus à même de répondre à nos besoins.

**3.1.2.2.5 Les avantages de cette architecture** L'architecture envisagée permet de définir notre application en micro-services avec une charge répartie sur plusieurs serveurs. Cela permet de rajouter des ressources au besoin.

Elle permet aussi d'avoir une application en haute disponibilité : la gestion des micro-services permet, en cas de problème sur l'un d'eux, de minimiser l'ampleur de la panne mais également de répliquer sur les serveurs pour ne pas avoir de coupure si l'un des serveurs n'est plus disponible.

Les conteneurs permettent d'avoir une sécurité supplémentaire à l'aide de sous-réseaux et d'environnements sans impact sur le serveur d'exécution. L'utilisation de micro-services avec un orchestrateur de conteneurs tel que Kubernetes ou Docker swarm permet la gestion d'une grande partie de la communication et de la sécurité.

### 3.1.3 Problèmes rencontrés

Afin de correspondre aux besoins du client, mais aussi dans un but d'optimisation de la sécurité et de l'accessibilité, l'architecture initiale du site a dû être divisée en deux étapes : une architecture basique avec la mise en place d'un système de conteneurs et une communication temporaire. L'architecture temporaire a été pensée afin de minimiser les changements lors du passage à l'architecture envisagée.

La communication a également posé des soucis, notamment lors de l'ouverture du socket SHH qui vient créer le conteneur et ne permet pas à l'étudiant d'accéder au serveur mais uniquement au conteneur.

Les différentes configurations de sécurité du réseau de l'Université d'Angers ont été très dérangeantes notamment lors de la génération des images grâce au Dockerfile. L'utilisation des services SSH étant bloquée, il a également été compliqué de mettre en place la connexion SSH de la première version de l'application.

## Chapitre 4

# Interface Utilisateur

### 4.1 Présentation

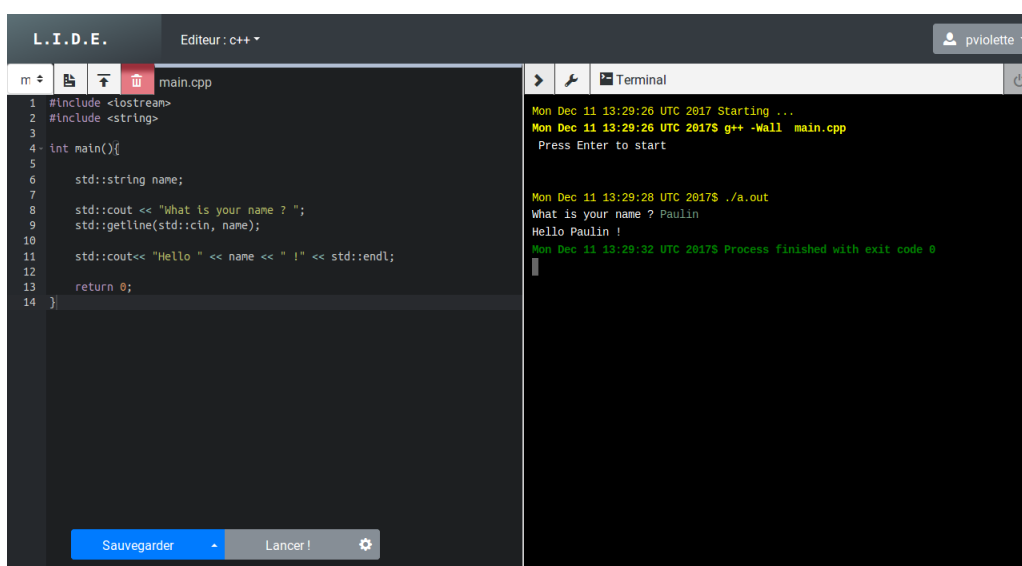


FIGURE 4.1 – Interface utilisateur en utilisation

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

- La barre de navigation, contenant les liens vers les autres parties du site (gestion de compte...), ainsi qu'un menu dropdown permettant de changer de langage.
- La barre d'outils, qui contient des contrôles spécifiques à l'applications (création ou importation de fichiers, contrôle de la console).
- L'éditeur, implémenté par le plugin Ace
- La console, implémentée par le plugin jqconsole

### 4.2 Outils utilisés

En plus des outils déjà décrits dans la section ??, l'interface utilisateur utilise plusieurs plugins Javascript.

L'éditeur de texte est Ace<sup>1</sup>, un éditeur de texte pour le web supportant la coloration syntaxique de près de 110 langages, mis à disposition sous licence BSD<sup>2</sup> et maintenu comme le principal éditeur pour l'IDE AWS Cloud9. Cet éditeur a pour avantage de supporter de nombreuses fonctionnalités, parmi lesquelles :

- L'indentation automatique
- Chercher/Remplacer avec des expressions régulières
- Changement entre tabulation avec des espaces ("soft tab") ou avec une réelle tabulation (caractère t, "hard tab").

---

1. Voir <https://ace.c9.io/>

2. Berkeley Software Distribution License, une license libre. Plus de détail sur [https://fr.wikipedia.org/wiki/Licence\\_BSD](https://fr.wikipedia.org/wiki/Licence_BSD)

- Numérotage des lignes
- Et bien d'autres...

Toutes ces raisons nous ont poussés à utiliser ce plugin.

Afin de gérer la console, nous avons choisi d'utiliser le plugin `jqconsole` (Plus de détails en ??).

Les icônes proviennent de la bibliothèque d'icônes `openiconic`<sup>3</sup>.

La sauvegarde des fichiers est gérée par les plugins `FileSaver.js`<sup>4</sup>, qui implémente la méthode `HTML5 saveAs()`, et `jszip`<sup>5</sup>, qui permet de créer des archives zip pouvant être ensuite sauvegardé via `FileSaver`.

Dernièrement, pour afficher les alertes relatives à l'interface, nous utilisons le plugin `SweetAlert2`<sup>6</sup> qui nous permet d'avoir des alertes bien plus esthétiques que les alertes standards, et personnalisable à la fois au niveau esthétique que du contenu (champs, texte des boutons, etc...).

## 4.3 Organisation des templates TWIG

Le template twig de l'application, `index.html.twig`, hérite du template `layout.html.twig`, qui définit une base pour l'application, et qui contient la barre de navigation. Nous avons également créé différents templates pour les modales d'options (personnalisation, lancement, importation et création de fichier).

Les templates du bundle FOS (gérant la partie utilisateur du site) sont également redéfinis pour s'intégrer avec le reste de l'application.

## 4.4 Environnement de Développement

### 4.4.1 Gestion des langages

Une des contraintes données pour le projet était d'avoir un éditeur disposant d'une coloration syntaxique, et pouvant supporter plusieurs langages, et pouvoir ensuite compiler (si besoin) et exécuter le code écrit dans ces langages. Il a donc fallu penser l'interface pour répondre à cette problématique.

Une liste des langages configurés est stockée dans la base de données, comme vu dans le chapitre ??. La page est donc générée à partir de ces informations : un menu déroulant est inséré dans la barre de navigation, avec un choix pour chacun des langages marqué comme actif. Le langage sélectionné est marqué via la classe CSS `.choix-langage-selected`, qui cache le lien dans le menu déroulant. Lors du chargement de la page, ou à la sélection d'un autre langage, une requête AJAX est effectuée afin de récupérer les informations sur le langage sélectionné :

- le mode de l'éditeur (on appelle ensuite la méthode `setMode(mode)` sur l'éditeur Ace).
- la liste des modèles configurés pour le langage (avec l'extension et le contenu du modèle).
- le nom du compilateur (utilisé dans le formulaire d'exécution pour afficher la commande de compilation qui va être exécutée).
- le nom du langage

Chaque lien dans le menu déroulant contient un attribut `data-id` qui permet d'identifier le langage (correspond à l'id du langage dans la base de données).

Ces informations sont ensuite utilisées pour mettre à jour l'éditeur.

### 4.4.2 Personnalisation

Toute personne ayant déjà travaillé en groupe sur un projet informatique a pu remarquer que chacun a ses préférences de thème pour un éditeur : certains préfèrent un fond sombre, d'autres un fond clair, etc... L'éditeur Ace est facilement personnalisable et dispose, par défaut, de 24 thèmes. Il était donc assez rapide d'implémenter un formulaire permettant à l'utilisateur de choisir le thème qui lui convient le mieux, lui permettant ainsi de facilement s'approprier son outil de travail.

La police est également personnalisable, permettant à chacun d'utiliser une taille de police qui lui convient.

La console ayant un style implémenté par CSS, il fut de même aisé de créer des thèmes qui s'appliquent grâce à une classe attribuée à l'élément `div` contenant la console. Pour l'instant, seuls trois styles de console sont implémentés, mais il serait aisé d'en ajouter d'autres dans des versions futures. Chaque style a une classe maîtresse `.console-nom_style`. On redéfinit ensuite les classes `jqconsole` grâce aux sélecteurs CSS (voir le fichier `console.css`).

Tous ces changements sont pour l'instant uniquement gérés en local (voir fichier `options.js`).

---

3. <https://useiconic.com/open>

4. <https://github.com/eligrey/FileSaver.js/>

5. <https://stuk.github.io/jszip/>

6. Voir <https://limonte.github.io/sweetalert2/>



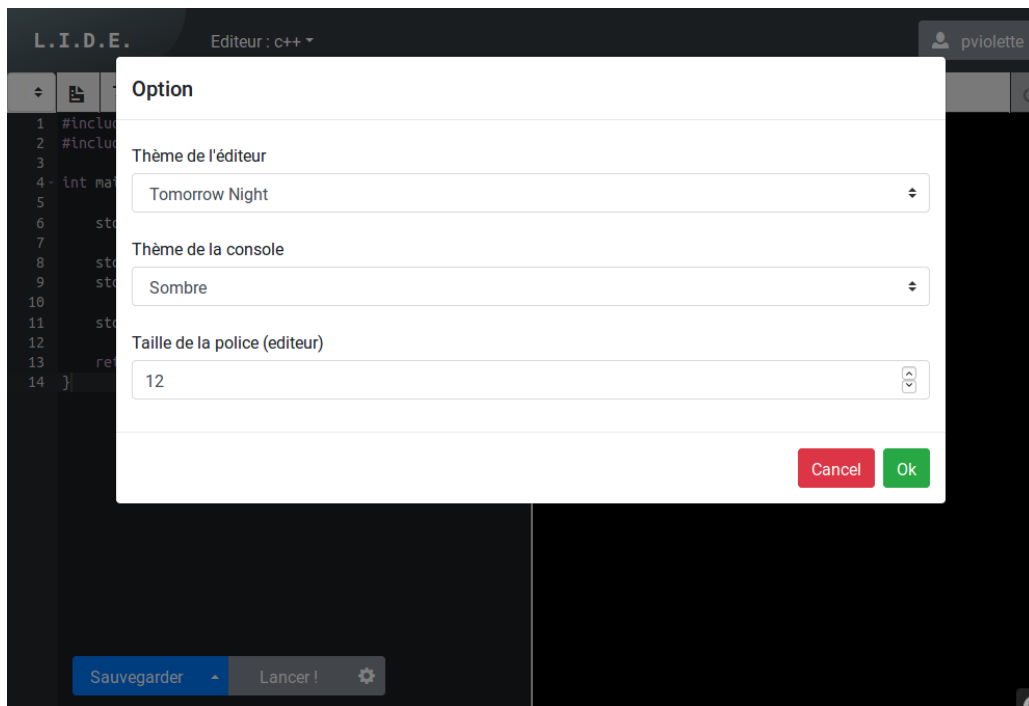


FIGURE 4.2 – Formulaire permettant la personnalisation de l'interface

### 4.4.3 Gestions des fichiers

Tel un véritable EDI, notre application permet la gestion de multiples fichiers. Cette gestion est effectuée sur le navigateur par du javascript.

Les fichiers sont enregistrés dans un prototype contenant deux champs : *name*, le nom du fichier, et *content* son contenu. Ces prototypes sont ensuite stockés dans la variable globale *files*, un tableau de fichier. À chaque changement de fichier en cours d'édition, le contenu de l'éditeur est sauvegardé et est ensuite remplacé par le contenu du nouveau fichier à éditer.

Les fichiers peuvent être créés de deux façons : soit en important un fichier depuis son ordinateur (bouton importation), soit par création à partir de modèles définis par l'administrateur pour le langage. On laisse également la possibilité de créer des fichiers vides (par exemple pour créer un fichier de données utilisé lors de l'exécution du programme).

## 4.5 Compilation et exécution

### 4.5.1 Formulaire

Le formulaire pour compiler et exécuter est généré par Symfony à partir de l'entity **Execution** et de la classe **ExecutionType**. Le formulaire comporte 7 champs :

- Les paramètres de compilations : arguments passés aux compilateurs
- Les paramètres de lancement : arguments donnés au programme
- Fichiers additionnels : l'utilisateur peut choisir des fichiers depuis son ordinateur qui seront joints aux fichiers présents dans l'IDE
- Une option Compilation Uniquement : si activée, seule la compilation sera effectuée ; le programme ne sera pas lancé
- Un choix de gestion du flux d'entrée :
  - Aucun : aucune gestion des entrées n'est effectuée
  - Interactive : entrée interactive, l'utilisateur écrit sur le flux d'entrée 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 de gestion des entrées Texte est sélectionné.
- Un champ caché alimenté par le javascript, contenant le JSON correspondant à la liste des fichiers.

## 4.5.2 Lancement de la compilation et de l'exécution

Le lancement de la compilation et de l'exécution du code écrit est lancé par l'appui sur le bouton "Lancer" qui appelle une fonction JS envoyant une requête AJAX vers la méthode `ConsoleController::execAction` (fichier `ConsoleController.php`). Cette requête contient le formulaire qui est ensuite traité : les fichiers vont être écrits dans le système du fichier dans un dossier temporaire, le script de lancement et d'exécution correspondant au langage est récupéré dans la base de données et placé dans ce même répertoire. Une commande, qui va permettre de lancer le docker, est ensuite construite. Cette commande va :

- Stopper le container de l'utilisateur s'il existe.
- Lancer un container basé sur l'image correspondante au langage, de nom `id_[id_user]/A`, paramétré pour être supprimé à la fin de l'exécution de la commande de lancement. Cette commande de lancement va :
  - Récupérer le script de lancement sur le serveur de l'application via un wget
  - Attribuer les droits d'exécution sur ce script
  - Une commande sed remplaçant les caractères indésirables (dus à la base de données).
  - Lancer le script avec les bons paramètres

Les paramètres du script à lancer sont :

- `-o 'options_de_compilation'`, pour les paramètres à passer au compilateur
- `-f 'fichier1 fichier2...'`, la liste des fichiers de l'utilisateur (récupérée par un wget)
- `-i fichier`, le nom du fichier contenant les inputs s'il est nécessaire
- `-n`, pour un mode non interactif.
- `-a 'agr0 arg1 ...'`, les arguments à donner au programme
- `-c`, pour uniquement effectuer la compilation
- `-w 'wget_adr'` l'adresse où effectuer le wget.

## 4.5.3 Terminal

La mise en place de la console proposait deux options : soit la création d'une vue ad-hoc soit l'intégration d'une vue déjà existante.

Notre choix s'est vite tourné vers l'intégration d'une vue déjà existante. La création d'une vue nous aurait certes donné une modularité de la console en ce qui concerne les modifications mais, la console une fois implémentée n'a pas nécessairement besoin de modifications.

Après étude de rentabilité, nous avons décidé d'implémenter la vue JQConsole<sup>7</sup> (utilisée dans repl.it<sup>8</sup>, un projet similaire au nôtre) qui correspondait exactement à nos besoins. En plus d'être esthétique, son code source était placé sous licence libre et toutes les fonctions qui nous étaient nécessaires étaient déjà implémentées. L'inconvénient principal est que la modification du code source peut se montrer compliqué, celui-ci étant rédigé en CoffeeScript et étant assez compliqué.

Nous n'avions donc plus qu'à intégrer la vue JQConsole à notre interface et faire appel aux bonnes fonctions (notamment JQConsole.Write qui permet d'afficher du texte dans la console et JQConsole.Prompt qui permet de lire du texte) pour obtenir notre console.

## 4.6 Problème rencontré et amélioration possible

### 4.6.1 Rendre l'interface Responsive

Un des problèmes de l'interface actuelle est son manque d'adaptabilité sur les plus petits écrans, le design actuel n'ayant pas été conçu dans le but de répondre à cette problématique. Cela est principalement dû au manque d'expérience en web qui nous a mené vers quelque chose de fonctionnel sur ordinateur qui n'est l'est pas sur les plateformes mobiles aux écrans plus petits. C'est une amélioration qu'il serait souhaitable de réaliser dans une version future.

### 4.6.2 Persistance des options de personnalisation et de compilation

Actuellement, la persistance des options de compilation ou de personnalisation n'est pas assurée. Cette fonctionnalité n'a pas été implémentée principalement par manque de temps mais pourrait rapidement être implémentée en ajoutant les champs nécessaires dans la base de données, afin de lier les préférences à un utilisateur, et en implémentant une méthode renvoyant les choix effectués dans le formulaire de personnalisation au serveur via une requête AJAX, afin que ces choix soit sauvegardés en base de données.

---

7. <https://github.com/replit/jq-console>

8. <https://repl.it/>

### 4.6.3 Revoir la gestions des fichiers : sauvegarde de sessions plutot que simplement du contenu

Actuellement, lors d'un changement de fichier, seul le contenu du fichier est sauvegardé : l'état de la session (position du curseur...) n'est pas sauvegardé. Il pourrait être intéressant de s'assurer que lorsque l'on retourne sur un fichier précédement ouvert, on se retrouve exactement dans les même conditions que lorsqu'on en a quitté l'édition. Cette fonctionnalités peut être implémenté grâce aux fonctionnalités de sessions du plugin Ace, dont nous n'avons pris connaissance que trop tard.

## Chapitre 5

# Communication Serveur/GUI

### 5.1 Présentation

La communication entre le serveur de l'application et les conteneurs 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 donnés.

L'utilisation de Symfony a permis l'utilisation de la librairie libssh2 de PHP. Cette librairie permet de gérer facilement une connexion SSH en PHP.

### 5.2 Processus de communication

Le processus de communication repose sur un système de questions/réponses. Le client demande des informations et le serveur lui répond.

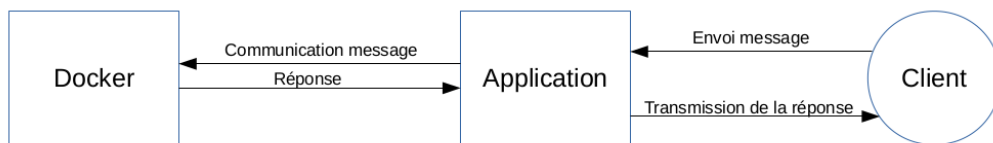


FIGURE 5.1 – Communication entre le client et le conteneur Docker

#### 5.2.1 Première requête envoyée (clic sur le bouton Lancer)

L'application reçoit la requête du client. Elle va créer la commande qui permet de démarrer un conteneur docker et de lancer le script qui se charge de la compilation et de l'exécution du programme. La commande est ensuite passée au service GestionSSH qui se charge de se connecter en SSH au serveur qui contient les conteneurs et d'exécuter la commande grâce à un shell.

L'application récupère ensuite la sortie standard du programme, toujours grâce au service GestionSSH. Elle répond ensuite au client qui attend toujours la réponse du serveur. Le client affiche ensuite la réponse dans la vue JQConsole.

#### 5.2.2 Envoi des requêtes suivantes

Dans certains programmes, l'utilisateur a besoin de répondre au programme pour que celui-ci se termine (fonction d'inputs). Dans ce cas, l'application teste si le docker est encore ouvert et le redémarre si oui. Le message est ensuite passé au service de GestionSSH qui se charge de le transmettre au conteneur grâce à SSH.

L'application récupère de nouveau la sortie standard du programme pour l'envoyer au client. Cette opération se répète jusqu'à ce que l'exécution du programme soit terminée.

### 5.3 Problème de la communication

Le principal problème apparu lors du développement de la communication est dû à la non persistance des objets en PHP. En effet, à la fin de chaque requête, PHP supprime tous les objets qui ont été créés pendant la requête. Il nous était donc impossible de garder une connexion SSH ouverte pendant toute l'exécution du

programme. Après de nombreuses recherches, nous avons décidé de créer une nouvelle connexion SSH à chaque nouvelle requête. Le conteneur docker ne se supprimant qu'une fois l'exécution terminée, il était alors possible de communiquer avec lui sans pertes d'informations.

L'inconvénient de cette 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.

Cette implémentation est très critiquable et sera totalement modifiée dans la prochaine version de l'application.

## Chapitre 6

# Conclusion

La réalisation de ce projet a été pour nous tous un exercice très intéressant. Nous nous sommes beaucoup investis et avons tenté de faire au mieux. Le rendu n'est bien sûr pas parfait, mais nous avons appris beaucoup sur le travail d'équipe, la gestion des tâches et le respect du temps. De plus, nous avons pu découvrir de nouveaux outils tels que Symfony, Bootstrap, Docker.

Notre application est fonctionnelle mais est loin d'être terminée. Il faudrait notamment faire évoluer l'architecture du serveur et de la communication. De plus, il serait intéressant d'ajouter des fonctionnalités comme l'auto-complétion du code, l'intégration du débogueur et l'intégration d'outils d'analyse.

Nous tenons à remercier nos chefs de projets qui ont su nous expliquer le projet et nous guider tout au long de sa conception et de sa réalisation.

## Annexe A

# Renseignements sur le repository git

Lien du repository git : <https://github.com/bazanta/LIDE>

Pseudonymes :

- Paulin Violette : pviolette-fr
- Valentine Rahier : vrahier
- Jerome Martins Mosca : ibriryassine
- Yassine Ibrir : jejedcll