

M1 Informatique



Projet de concrétisation disciplinaire

# UA - LIDE

**Référent :** *GENEST David*

**Auteurs :**

*VIRET Thibaud*

*POIRON Guillaume*

*LARDEUX Aloïs*

*MALASHCHYTSKI Timotei*

*NOUNE Ibrahim*

# Table des mati res

<b>I - Pr�sentation du projet</b>	<b>4</b>
Pr�sentation du projet	4
Objectifs initiaux	5
Pr�sentation technique	6
<b>II - Migration</b>	<b>7</b>
1. Migration de Symfony 3 vers Symfony 4	7
1.1 Le framework Symfony	7
1.2 Sch�ma classique d'une mise � jour majeure de Symfony	8
1.3 Particularit� de la migration de Symfony 3 vers Symfony 4	9
2. Migration du projet lide-pma	10
2.1 Contexte	10
2.2 Mise � jour de symfony	10
2.3 Migration vers symfony/flex	12
3. Migration du projet lide-web	15
3.1 Les indispensables de Symfony 4	15
3.3 L'int�gration de Vue JS	15
3.4 - Probl�matique	16
1. Architecture de l'application	17
2. Les conteneurs Docker	18
3. L'installation	19
<b>IV - Authentification LDAP</b>	<b>21</b>
Pr�sentation	21
1.1 - Technologie Ldap	21
1.2 - Objectif	22
Mise en place	22
2.1 - Serveur LDAP	22
2.2 - Authentification LDAP en php	23
<b>Int�gration � LIDE</b>	<b>23</b>
3.1 - Int�gration de LDAP dans FOSUser	23
<b>3.2 - Utilisation d'un controller externe</b>	<b>24</b>
<b>3.3 - Conclusion LDAP</b>	<b>24</b>
<b>V - Conclusion</b>	<b>25</b>
<b>Annexe</b>	<b>26</b>

# I - Présentation du projet

## 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 transparente pour l'utilisateur.

Cette application est destinée principalement aux étudiants de première année de licence de L'Université d'Angers à la faculté des Sciences. Le développement informatique fait parti de leur formation et doivent pouvoir continuer leur travail en dehors de l'Université sans avoir la nécessité d'installer d'environnement de développement sur leur poste personnel.

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

Voici un récapitulatif des fonctionnalités de l'application :

### **Les fonctionnalités de base qui fonctionnent :**

- La connexion à l'application.
- La création / modification / suppression d'un projet pour l'environnement C++.
- La création / duplication / suppression des fichiers (uniquement avec l'extension *.cpp*)
- L'ajout des différents modèles.
- La compilation et l'exécution en C++.
- L'affichage des résultats et des erreurs du programme compilé.

### **Les fonctionnalités de base qui ne fonctionnent pas :**

- La création d'un projet pour l'environnement *Java*
- La création des fichiers sous l'extension *.java*
- La compilation et l'exécution en *Java*
- La partie administrateur
- La persistance des données

### **Les fonctionnalités à ajouter :**

- L'authentification LDAP
- L'auto-complétion
- Le debugger

## 2. Objectifs initiaux

Au début de notre projet, nous avons eue une réunion avec le client. Cette réunion a permis de faire un état des lieux sur le projet. Suite à cette réunion trois objectifs ont été évoqués.

### **Rendre l'application facilement déployable**

Le projet que nous avons récupéré n'était pas déployable, plusieurs tests avaient été effectués en amont afin de vérifier que l'application puisse s'installer et démarrer mais cela fut sans succès. Le client nous a alors demandé de faire de cet objectif notre mission principale : l'application devait être déployable et son installation devait être simplifiée.

### **La mise en place d'un système d'auto-complétion du code**

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.

### **L'ajout d'un debugger**

Le debugger est une fonctionnalité importante qui permettra aux étudiants d'analyser les bugs de leurs programmes. Le debugger permettra d'exécuter le programme de l'étudiant pas-à-pas, c'est à dire ligne par ligne. Ils pourront afficher la valeur des variables à tout moment, de mettre en place des points d'arrêt sur des conditions ou sur des lignes de programme.

## 3. Présentation technique

L'application LIDE est composée de deux serveurs Symfony : l'un étant le serveur web, l'autre étant le "project manager". Pour plus de clarté, nous les nommerons lide-web et lide-pma.

Le projet lide-web est un serveur web à part entière. C'est sur celui-ci que se connectent les clients, et c'est ce serveur qui fournit une interface graphique à notre application. Lorsque l'utilisateur crée un projet, construit ses fichiers, les sauvegarde, etc. ses actions sont traduites en requêtes et sont émises vers le serveur pma qui les traitera. La réponse reçue est elle aussi traitée en conséquence. Bien que le serveur soit Symfony, toute la partie front est intégrée en VueJS, qui rend l'aspect graphique très moderne, mais qui

complexifie néanmoins le code, le modèle MVC n'étant pas respecté dans l'architecture de ce serveur.

Le projet lide-pma quant à lui comporte la partie gestion des projets et la partie exécution des conteneurs. Ce serveur web tourne en permanence et répond aux requêtes générées par le projet lide-web. Ce serveur est appelé pour toutes les étapes du cycle de vie d'un fichier source, allant de la création, à la suppression en passant par la modification et la compilation.

La gestion des projets se compose d'une base de données enregistrant les chemins d'accès aux fichiers sources d'un projet créé par l'utilisateur depuis l'ihm. Ces chemins d'accès sont des liens ou sont enregistrés en dur les fichiers sources à partir du répertoire racine configurable appelé "lide-storage/". Par exemple: "lide-storage/2/3/helloworld.cpp" correspond au chemin d'accès du fichier helloworld.cpp contenu dans le projet ayant pour id "3" de l'utilisateur "2". En plus des chemins d'accès, on peut trouver dans la base de données des informations concernant le possesseur du fichier, le langage utilisé ou la date de création.

La partie exécution des conteneurs de compilation, comme son nom l'indique, à pour but d'effectuer des commandes bash de compilation dans un conteneur au travers d'un WebSocket. Cette partie est utilisée lorsque qu'un utilisateur demande via l'ihm à effectuer une compilation d'un fichier. Le serveur va alors exécuter la commande et transmettre le résultat de la compilation au serveur lide-web qui pourra l'afficher dans l'ihm.

## II - Migration

### 1. Migration de Symfony 3 vers Symfony 4

Suite à la répartition des tâches, il a été décidé de migrer le projet de la version 3 vers la version 4 de Symfony afin d'avoir une plateforme plus récente pour le développement du projet ainsi que de rendre les sources du projets plus claires.

Il existe heureusement un protocole à suivre pour effectuer une migration majeure qu'est la migration de Symfony 3 vers Symfony 4, nous allons donc le détailler dans la suite de cette partie.

#### 1.1 Le framework Symfony

Symfony est un des framework php les plus populaires à l'heure actuelle et a pour philosophie de permettre le développement ainsi que le déploiement d'une application web PHP le plus rapidement possible.

Pour ce faire, le framework utilise un grand nombre d'abstractions sur les tâches les plus fastidieuses ou redondantes.

Ainsi ce framework dispose, sans s'y limiter de caractéristiques suivantes:

- Utilisation d'un gestionnaire de dépendances appelé *composer*
- Intégration de commandes au sein du framework via *php bin/console*
- Abstraction du code par annotations
- Utilisation d'un générateur d'entités et de bases de données
- Serveur web intégré au projet
- Automatisation du wiring des contrôleurs
- Automatisation du routing
- Injection de dépendances
- Intégration d'un mode debug (dev) et d'un mode production (prod)
- Grand nombre de bundles étant parfois de réels "sous-framework"
- Gestion et accessibilité des paramètres

Ces caractéristiques impliquent donc qu'une migration de version majeure peut amener à la modification des sources afin de les rendre compatible avec la version supérieure et n'est pas simplement une mise à jour en un clic.

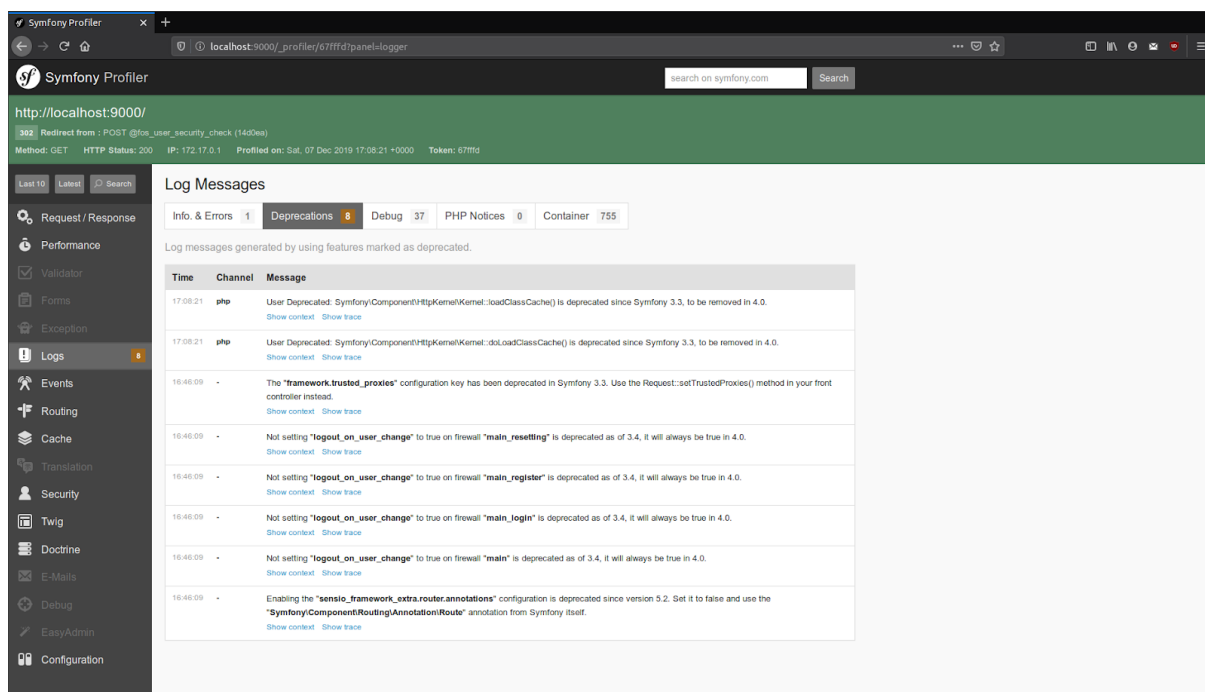
Pour pallier à cette difficulté, l'avantage d'un framework tel que Symfony est la présence d'une très bonne documentation disponible en anglais, et si celle-ci n'est pas suffisante, la communauté autour du framework peut aussi jouer un rôle dans la résolution des problèmes bloquants.

## 1.2 Schéma classique d'une mise à jour majeure de Symfony

Dans le cas du framework Symfony, trois étapes principales sont nécessaires pour toute montée de version majeure:

1. Enlever les "deprecations" du code présentes avant la mise à jour
2. Mettre à jour la version de symfony à l'aide de composer
3. Mettre à jour le code pour qu'il fonctionne avec la nouvelle version

- 1) La première étape consiste à lancer le projet en mode *dev* et à utiliser le mode *Profiler* intégré à symfony. Ce mode permet d'accéder depuis n'importe quelle page du site web, à une page de debug contenant toutes sortes d'informations allant de simples logs, en passant par les erreurs, l'analyse des route, de la base de données, et, des *deprecations*. Il suffit ensuite de lire le contenu de la *deprecation* pour la résoudre, il n'y a ici pas de recette "magique" permettant d'automatiser cette tâche même si quelques fois la montée de version de symfony ou des dépendances permet de corriger l'erreur.



Un exemple de page Profiler

- 2)  tape la plus simple de la mont e de version, il suffit de mettre   jour la d pendance *symfony/symfony* au sein du fichier *composer.json* contenant l'ensemble des d pendances du projet. Ainsi dans notre cas, on souhaite passer la version de Symfony vers la version "4.3". On modifie donc la ligne associ e par la valeur "4.3.\*", l' toile indiquant que l'on souhaite n'importe quelle version mineure de la "4.3". Cela permet d'obtenir la version la derni re version de symfony 4 maintenue et comportant les mise   jour de s curit . Pour effectuer la mise   jour, il faut ensuite effectuer la commande: *composer update symfony/symfony*.
- 3) Enfin derni re  tape, la plus fastidieuse, la mise   jour du code pour permettre le fonctionnement avec la version 4 cette  tape d pend enti rement des nouveaut s apport es   la version et sont correctibles   l'aide du fichier *UPGRADE-4.0.md*.

### 1.3 Particularit  de la migration de Symfony 3 vers Symfony 4

La version 4 au del  d' tre une version majeure,  tait en r alit  une refonte partielle du framework.

En effet, la version 4 utilise maintenant une toute autre hi rarchie de fichier ainsi qu'une refonte du syst me de fichiers de configurations. Cela se traduit au sein du framework par l'utilisation forc e du bundle *symfony/flex* offrant une gestion des sources plus claire et coh rente.

La version 3 du framework incitait en effet   fragmenter son code en "bundles" for ant ainsi   fragmenter tout le projet en divers blocs fonctionnels contenant chacun leurs fichiers sources (controller, entit s, fixtures ou autres sources) de mani re ind pendante.

Cependant la nouvelle philosophie apport e par *symfony/flex* dicte que les bundles ne doivent d s lors  tre utilis s uniquement pour du code import  depuis un autre projet. En

d'autres termes, la séparation en bundles ne doit être utilisée que si l'on souhaite importer du code transposable ou ses propres bibliothèques, cela n'a donc plus de sens dans un

#### Structure de Symfony 3:

```
your-project/
├── app/
│   ├── config/
│   │   ├── parameters.yml
│   │   ├── services.yml
│   │   └── routing.yml
│   ├── AppCache.php
│   └── AppKernel.php
├── bin/
│   ├── console
│   └── symfony_requirements
├── src/
│   ├── ...
│   ├── bundle1
│   │   └── ...
│   ├── bundle2
│   │   └── ...
│   └── ...
├── tests/
├── var/
└── vendor/
```

#### Structure Symfony 4 (utilisant symfony/flex):

```
your-project/
├── assets/
├── bin/
│   └── console
├── config/
│   ├── bundles.php
│   ├── packages/
│   ├── routes.yaml
│   └── services.yaml
├── public/
│   └── index.php
├── src/
│   ├── ...
│   └── Kernel.php
├── templates/
├── tests/
├── translations/
├── var/
└── vendor/
```

projet partant de zéro.

La migration vers *symfony/flex* est la partie la plus chronophage de cette migration puisqu'elle implique la réécriture de la totalité des fichiers de configurations. Notamment au travers de l'apparition des fichiers de configurations spécifiques aux bundles ainsi que du fichier *.env*, remplaçant du fichier *parameters.yml* contenant toutes les variables globales du projet.

Mais surtout, il faut modifier la structure des fichiers et par conséquent tous les imports ou autres références à l'ancienne structure qui ne sont plus correctes.

## 2. Migration du projet lide-pma

### 2.1 Contexte

A la suite des premières semaines nous ayant permis de mieux comprendre l'architecture du projet ainsi que de nous familiariser avec le framework symfony, il a été décidé par l'équipe de gestion du projet d'effectuer une migration du projet lide-pma de Symfony 3 vers Symfony 4.



Cette mise à jour avait plusieurs objectifs:

1. Mettre à jour le framework avec une version maintenue le plus longtemps possible, version qui au moment de la prise de cette décision était la version 4.3.X
2. Utiliser la nouvelle hiérarchie des sources ainsi que le nouveau système gestion des fichiers de configuration pour apporter une certaine clarté aux sources et permettre une maintenance du code plus simple.
3. Profiter de cette lecture complète du code pour identifier et corriger les erreurs pouvant exister ainsi que supprimer le code mort ou désuet.

Nous avons, dans le cas du projet lide-pma, choisi de procéder à une mise à jour telle que la documentation technique de symfony le recommande, mise à jour décrite dans la partie précédente.

## 2.2 Mise à jour de symfony

Pour la migration du serveur lide-pma, serveur qui pour rappel ne contient que la partie gestion de projet et le web socket permettant le lancement des conteneurs de compilation, nous avons tenté d'utiliser la méthode classique de migration évoquée dans la partie précédente.

Malgré une décision initiale de partir sur un projet vierge utilisant la version 4 de symfony afin de régénérer la totalité des sources (Controllers, Routes, Entities), nous nous sommes aperçu qu'une telle migration n'avait pas vraiment de sens.

En effet, tout le code qui avait été généré à la création du projet Symfony 3 est en soit compatible avec Symfony 4 modulo l'import des dépendances ainsi que les namespaces qui doivent refléter la nouvelle structure de fichiers qu'apporte le bundle *symfony/flex*.

De plus, la plupart des réelles problématiques qu'implique la migration de symfony 3 vers symfony 4 sont des problématiques que nous aurions eu de toute façon en partant d'un projet "propre", notamment la gestion des routes ou des services.

Les sources récupérées à partir du travail des années précédentes utilisaient pour la plupart du code générique régulièrement mis à jour, de ce fait, très peu de *deprecations* sont à recenser. Nous n'avons donc pas perdu particulièrement de temps sur ce point puisque les quelques *deprecations* apparaissant dans le page *Profiler* de Symfony étaient liées au changement dans la structure même de symfony qu'implique le passage au bundle *symfony/flex* gérant la structure des fichiers du projet.

Il reste cependant toujours quelques *deprecations* extérieures à la migration mais nous avons fait le parti-pris de ne pas perdre de temps sur cette partie de la migration puisque ce n'est en soit pas un élément bloquant et apporte dans l'immédiat peu de résultats.

La mise à jour de symfony quand à elle a posé plus de problèmes, le système de mise à jour des dépendances utilisé par symfony utilisant une notation à la fois simple est puissante mais qui lors d'une mise à jour forcée des dépendances, peut se trouver en situation bloquante.

En effet, composer utilise un fichier json permettant "d'exiger" certaines versions des dépendances et en interdire d'autres.

La syntaxe classique permettant d'exiger une dépendance au projet symfony est la suivante:

```
"require": {  
    "symfony/symfony": "3.4.1",  
    "php": "7.2.1"  
}
```

Cette syntaxe est modulable et permet l'utilisation d'annotations (\* : utilise n'importe quelle version compatible) (^ : utilise au moins équivalente à celle donnée).

Par exemple:

```
"require": {  
    "symfony/symfony": "^3.4.*",  
    "php": "^7.2.1"  
}
```

Ici, on force la mise à jour de php si une version supérieure à 7.2 et on force la mise à jour supérieure de symfony 3 tout en la limitant à une mise à jour mineure de la version 3.4.

L'intérêt étant notamment que l'on peut exiger n'importe quelle version d'une dépendance satisfaisant les autres dépendances ou bien exiger une version minimum sans interdire les versions mineures supérieures qui pourraient apporter des correctifs de sécurité.

Dans le cas de la migration de symfony 3 vers 4, un certain nombre de dépendances ont dû être supprimées ou mises à jour ce qui a provoqué des dépendances cycliques lors des mises à jours de certaines dépendances.

En effet, en forçant la mise à jour d'une dépendance A, il est possible qu'une dépendance B se retrouve dans une situation où elle nécessite une version antérieure à la nouvelle version de A. Pour résoudre ce problème il suffit de demander une nouvelle version de B qui fonctionne avec la nouvelle version de A. C'est évidemment possible via composer et c'est même encouragé par Symfony qui donne un descriptif des erreurs de dépendance quand elles apparaissent.

Cependant, en passant de Symfony 3 à Symfony 4, nombreuses dépendances sont tombées dans un problème de "dépendances cycliques": Ces dépendances dépendaient entre elles de versions antérieures ou supérieures incompatibles et la mise à jour ne pouvait s'effectuer. Nous avons donc dû réinstaller chacune des dépendances en trouvant les versions compatibles entre elles les plus récentes, cependant certaines concessions ont été faites et par conséquent toutes les dépendances ne sont pas nécessairement sur leur version la plus élevée.

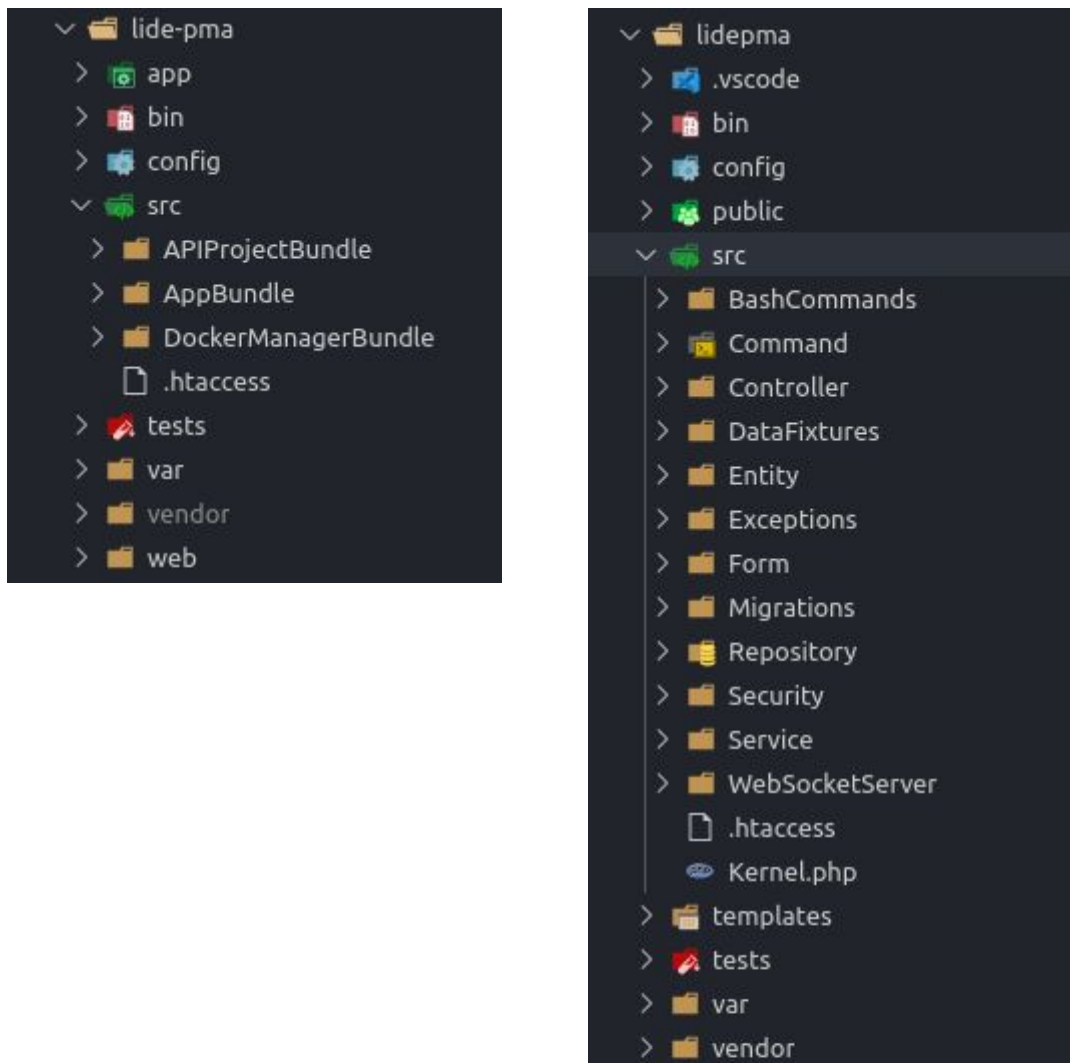
## 2.3 Migration vers symfony/flex

La migration vers le bundle *symfony/flex* nécessite une réorganisation complète des sources du projet:

- Tous les Bundles doivent être supprimés

- Les sources doivent  tre d plac es et ordonn es
- Tous les namespace doivent  tre modifi s en accordance
- Les imports sont eux aussi   changer
- Les fichiers de configurations sont   r  crire

Nous avons donc d plac  le contenu des trois Bundles pr sents dans les sources de Symfony 3 vers des r pertoires repr sentant la fonction technique du fichier source plut t que dans un r pertoire repr sentant sa fonction "m tier":



Nous avons ensuite chang  tous les namespace ainsi que les imports, t che fastidieuse compte tenu du nombre de fichiers   v rifier.

Exemple de modification des imports:

**namespace APIProjectBundle\Controller;** devient: **namespace App\Controller;**  
**use APIProjectBundle\Entity\Fichier;** devient: **use App\Entity\Fichier;**  
**use APIProjectBundle\Form\FichierType;** devient: **use App\Form\FichierType;**

Nous nous sommes ensuite attelés à la mise à jour des fichiers de configuration en commençant par la création du fichier `.env` regroupant la totalité des variables globales anciennement présentes dans le fichier `parameters.yml`. Notamment ce qui touche à la configuration de la base de données ou de nom du répertoire utilisé pour la sauvegarde des projet et des fichiers créés depuis l'ihm.

Après avoir fait cette première modification, nous avons modifié le fichier `doctrine.yml` responsable de la configuration et la base de donnée afin d'effectuer un test. Résultat, la base de donnée se crée correctement en utilisant les fichiers *Entity* (fichiers php patrons de BDD). De plus, la base de données se remplit correctement lorsque l'on utilise les *Fixtures* (fichiers php permettant de générer des objets en BDD).

Nous avons par la suite modifié les fichiers de configuration des services afin de faire fonctionner les controllers, en autorisant le "wiring" automatique des services en s'inspirant des fichiers de configuration de Symfony 3.

Toutes les routes fonctionnent et sont bien reconnues lors des appels via *Postman* ou via le serveur front.

Cependant, en testant les fonctionnalités du serveur nous nous sommes aperçus qu'une partie des méthodes écrites dans les classes ne fonctionnent pas, et ce, due à une erreur d'injection des dépendances apparue lors de la migration que nous n'arrivons pas à résoudre.

De plus la modification des fichiers de configuration concernant la sécurité et l'authentification ont été apportées mais malgré nos recherches, nous n'avons jamais pu rétablir la configuration de sécurité initiales, le temps nous ayant pris de court et nos compétences dans ce domaine étant trop limitées.

Par conséquent toutes les requêtes associées à l'utilisateur identifié, notamment la création ou la suppression d'un projet, n'est pas fonctionnelle puisqu'elles requièrent à l'utilisateur d'être identifié à l'aide des clés *jwt* sur le serveur front.

Cette partie est donc un échec, nous n'avons pas réussi à aller au bout de la tâche, ce pour deux raisons principales:

- une entrée sur le projet tardive dû à des problèmes de configuration hérité des années passées
- notre connaissance limitée du framework symfony

Peut-être qu'avec une meilleure connaissance framework ou tout simplement plus de temps, nous aurions pu aller au bout de cette migration.

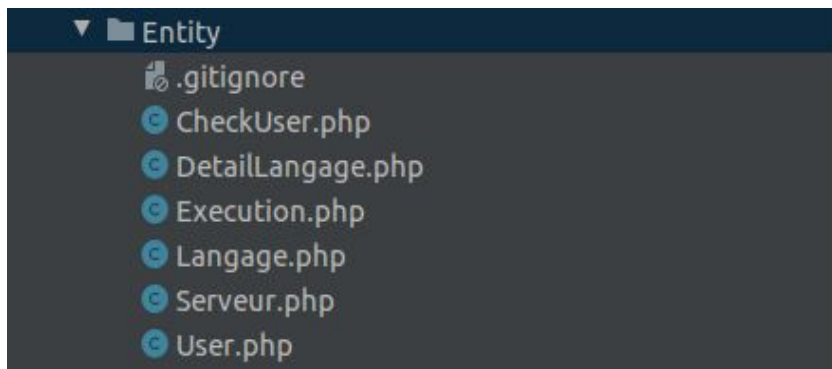
Mais nous trouvant face à une impasse, nous avons choisi d'abandonner la migration et de laisser en l'état les sources accessibles sur le dépôt afin de nous concentrer pour le temps restant sur la mise en route stable du projet utilisant Symfony 3.

### 3. Migration du projet lide-web

Pour la migration du projet lide-web, nous sommes partis sur une autre méthode. Nous avons réalisé la migration à partir d'un projet vierge de Symfony 4 pour ajouter nos fonctionnalités une par une et avec directement la bonne architecture d'un projet Symfony 4 pour avoir un projet propre sur de bonne base.

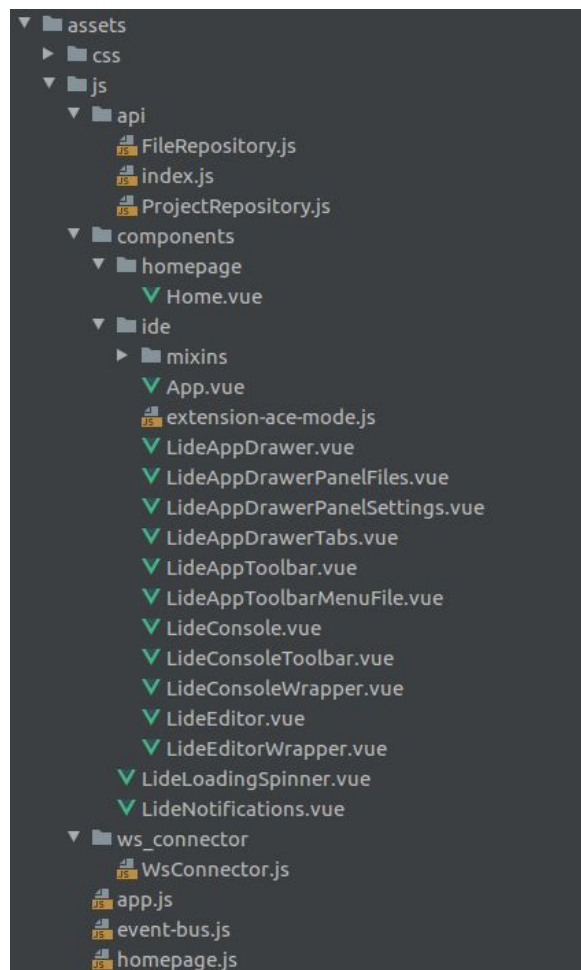
### 3.1 Les indispensables de Symfony 4

Le framework symfony dispose de tous les outils utiles pour la conception d'une application (base de données, les controllers, les différentes configs). Il a fallu dans un premier temps créer notre base de données avec la même configuration que le projet symfony 3. Nous avons repris à l'identique le contenu de la base de données de l'ancien projet, pour ce faire, nous avons utilisé le bundle make, pour construire les tables (voir figure ci-dessous) avec les mêmes enregistrements et les types utilisés. Ce bundle nous a permis également de créer nos controllers, AppController.php qui va gérer la partie principale de l'application c'est à dire l'interface de la partie compilation et HomeController.php pour l'interface création de projet.



### 3.3 L'intégration de Vue JS

La grosse difficulté dans la migration de LIDE-WEB a été d'intégrer VueJS, qui est un framework JavaScript open-source utilisé pour la conception des interfaces utilisateur. Pour intégrer au mieux VueJS, nous avons procédé étape par étape. La première étape a été d'installer Webpack Encore qui va permettre d'inclure tous les fichiers VueJS facilement. Ensuite, nous avons installé vue, le vue-loader, le vue template et vuetify avec le gestionnaire de paquets **npm** de Node.js. Il était important de bien respecter l'arborescence (voir figure ci-dessous) de l'ancien projet, pour éviter les erreurs d'inclusion.



### 3.4 - Problématique

Le projet lide-web dépend principalement du framework VueJS. Lors de la migration, nous avons eu beaucoup de problèmes avec ce framework, beaucoup d'erreur de configuration. Le principal problème a été l'intégration de Vuetify qui est une bibliothèque d'interface utilisateur permettant de simplifier la création des différents composants utiles. Pour manque de temps et de compétence, nous avons préféré ne pas continuer la migration du projet lide-web par la complexité de l'intégration de VueJS. Nous avons préféré utiliser une application qui marche pour se pencher sur des problèmes bien plus importants.

## III - Déploiement de l'application

### 1. Architecture de l'application

L'objectif principal de notre contribution au projet LIDE étant de rendre son installation et son déploiement plus simple qu'il ne l'était, ainsi nous avons pensé à utiliser la technologie de conteneurisation Docker pour déployer les différents serveurs de l'application.

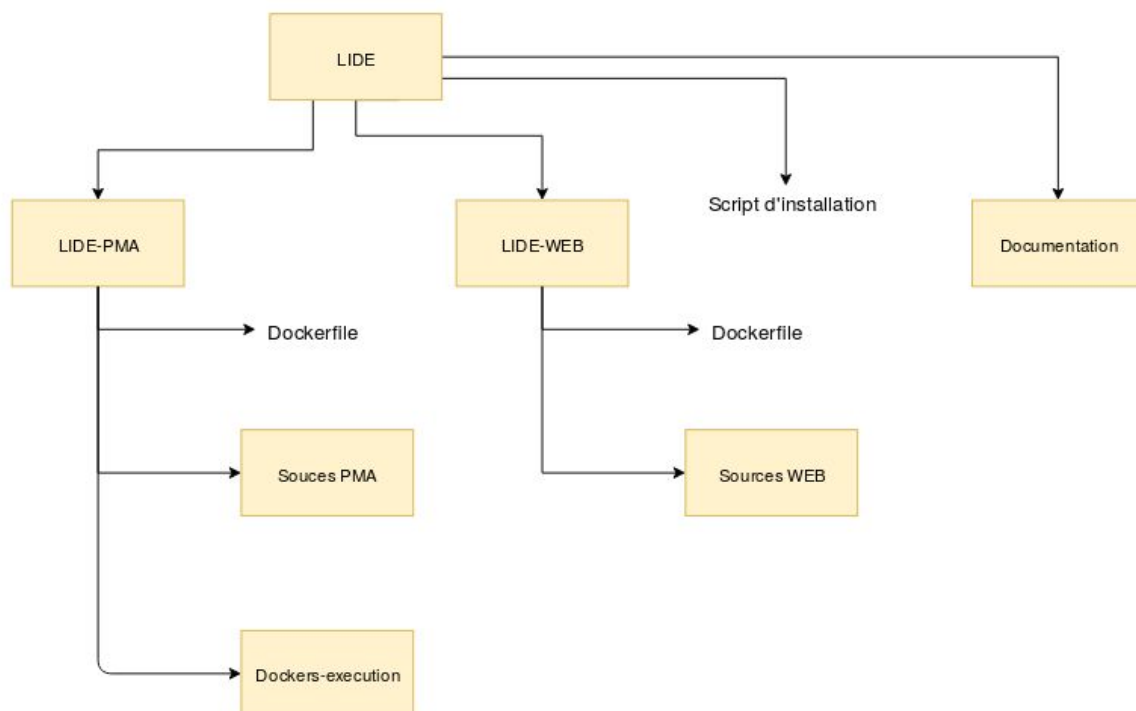
Il s'agissait donc de remplacer l'ancienne technologie de déploiement utilisée : Vagrant, afin

de remplacer le déploiement de machines virtuelles par celui de conteneurs, et ce de manière beaucoup plus simplifiée et automatisée. En effet, lorsque nous avons récupéré les sources de l'application, il était relativement difficile de la déployer ; les documentations d'installation n'étaient pas forcément à jour, le nombre de commandes et de scripts à jouer dans un ordre précis rendait la tâche fastidieuse et ces derniers avaient été conçus pour une configuration précise.

La première étape de ce travail a été de récupérer indépendamment les sources des deux serveurs, afin de réaliser des scripts Docker (Dockerfile) permettant leur déploiement dans des conteneurs différents.

Pour cela, nous avons dû repenser l'architecture générale de l'application :

- Un dossier pour chacun des deux serveurs
- Chaque dossier contient les sources indépendantes de son serveur et un Dockerfile permettant de construire une image Docker contenant le serveur déployé.
- Ces deux dossiers doivent être présents au même endroit, permettant l'utilisation d'un script d'installation unique.



*Schéma de l'architecture générale de l'application*

On observe dans le dossier LIDE-PMA la présence d'un dossier "Dockers-execution". Il s'agit du dossier contenant les images dockers qui seront utilisées par le serveur PMA pour la compilation et l'exécution de code.

## 2. Les conteneurs Docker

Après avoir décidé de la nouvelle architecture de l'application, nous avons dû rédiger les deux fichiers Dockerfile, web et pma.

Les deux serveurs étant des serveurs Symfony, le déploiement de ceux-ci sont relativement similaires, à quelques exceptions près. Les deux images *lide-pma* et *lide-web* sont basées sur la version 18.04 d'Ubuntu et permettent de déployer automatiquement les serveurs. Voici ce que les Dockerfile font, étape par étape :

- Téléchargement des paquets et dépendances nécessaires pour le déploiement du serveur
- Téléchargement des sources de Composer
- Téléchargement des sources de Symfony
- Copie des sources du serveur à l'intérieur du conteneur
- Installation des dépendances projet de Symfony ( via Composer )
- Création et initialisation d'une base de donnée
- Création d'un script de lancement du service Mysql et du serveur, qui sera joué au démarrage du conteneur.
- L'image *lide-pma* lance également le build des images des conteneurs de compilation, et lance le serveur websocket (qui organise la gestion de ces derniers).

Pour que les serveurs puissent être contactés, nous avons choisi arbitrairement leurs port d'écoute. Il s'agit du port 9000 pour le serveur web, et des ports 7000 et 7001 pour le serveur pma et son websocket. Ces ports pourront être changé lors de la mise en production.



### 3. L'installation

Pour rendre l'application facilement déployable et installable, nous avons choisi de réduire au maximum l'intervention humaine nécessaire à cela. Nous avons donc décidé qu'un seul script devrait permettre de construire les images dockers, et de lancer les conteneurs avec la bonne configuration, ce qui ferait grandement contraste avec l'ancien déploiement via Vagrant.

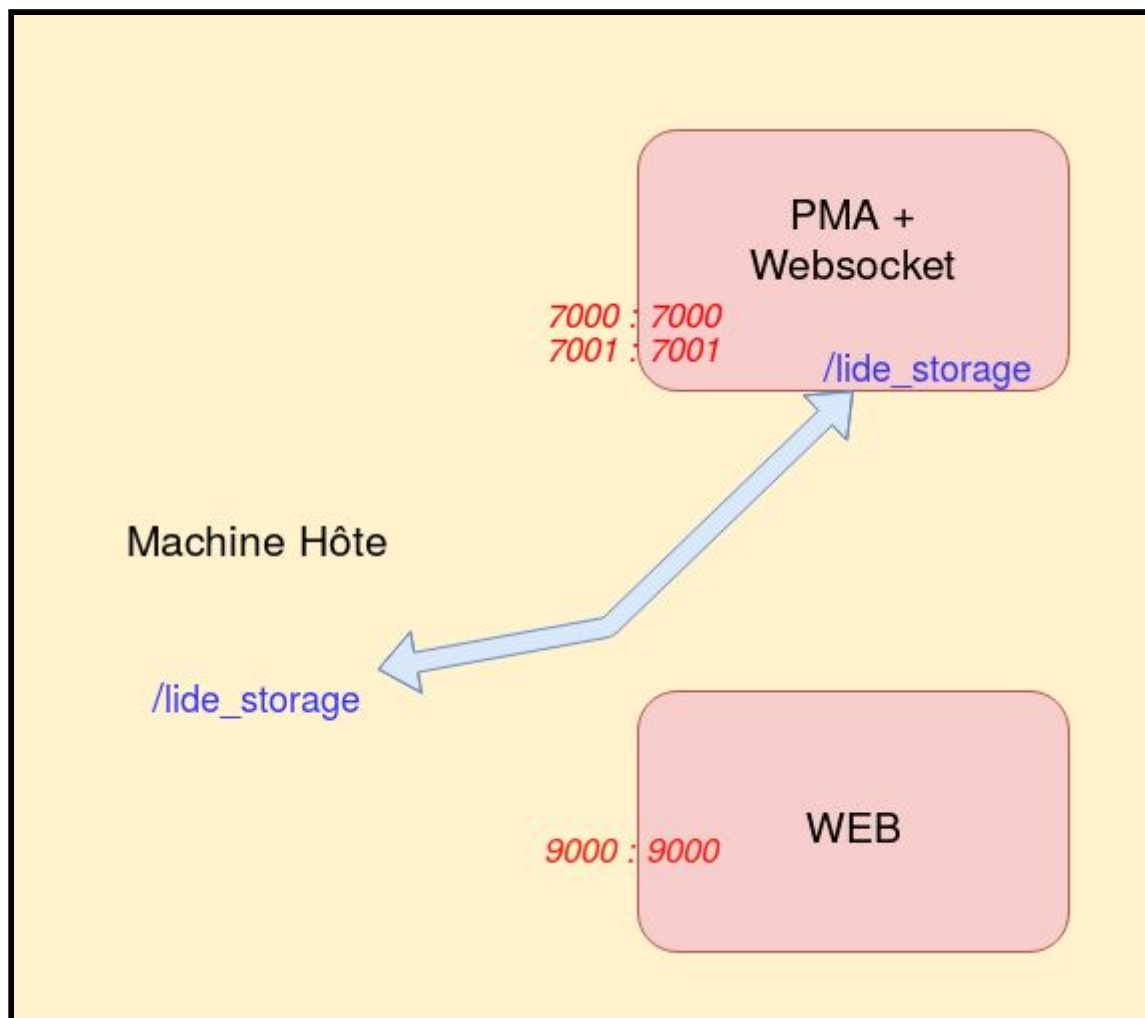
De plus, l'utilisation d'un tel script nous a été nécessaire durant notre phase de développement pour nous permettre de mettre à jour les sources de manière simple et rapide avant de pouvoir tester nos changements. En réalité nous avons créé deux scripts bash : l'un est destiné à n'être utilisé que lors de la première installation, l'autre sert seulement à relancer les conteneurs si ceux-ci venaient à tomber, ou bien après un redémarrage de la machine hôte.

Pour le premier script, il s'agit de ***installer.sh*** qui va construire les images voulues et les lancer dans un conteneur une fois construites. Il s'utilise de la façon suivante :

`./installer.sh` et prend en argument `pma`, `web`, ou bien les deux, et agira sur tel ou tel serveur.

Lors de la toute première installation de l'application : on utilisera `./installer.sh pma web`, en revanche, lors d'une phase de développement, nous pouvons avoir seulement besoin de régénérer l'image d'un des deux serveurs pour prendre en compte des mises à jours, sans avoir à arrêter l'autre conteneur qui tourne toujours (indépendance des deux serveurs). Dans ce cas on utilisera selon le besoin `./installer.sh pma` ou `./installer.sh web`, ce qui permet un gain de temps précieux.

Le deuxième script est ***lancer-conteneurs.sh***, qui, comme son nom l'indique, permet seulement de lancer les serveurs dans des conteneurs à partir de la dernière image construite.



*Représentation du fonctionnement de l'application*

Le schéma ci-dessus représente comment est déployé l'application avec notre solution. Nous avons en rouge nos deux conteneurs dans lesquels sont déployés les serveurs WEB et PMA. Les port d'écoutes des serveurs sont redirigés vers les mêmes ports sur la machine hôte.

Il existe un "bind mount" entre le conteneur PMA et la machine hôte (en bleu), ce qui permet l'accessibilité projets et fichiers des étudiants sur cette dernière, pour pouvoir en faire des sauvegardes. Il existe également un deuxième "bind mount" plus implicite entre ces deux entités, il s'agit du noyau docker "*docker.sock*" de la machine hôte. Ce noyau est utilisé par le Websocket du conteneur PMA afin de créer les conteneurs de compilation. En effet, il est difficile voire quasi-impossible de faire des conteneurs Docker imbriqués, et cette solution nous a permis de pallier le problème. Cela implique seulement que les conteneurs de compilation soient créés dans la machine hôte, ce qui ne change en rien le comportement ou la sécurité de l'application.

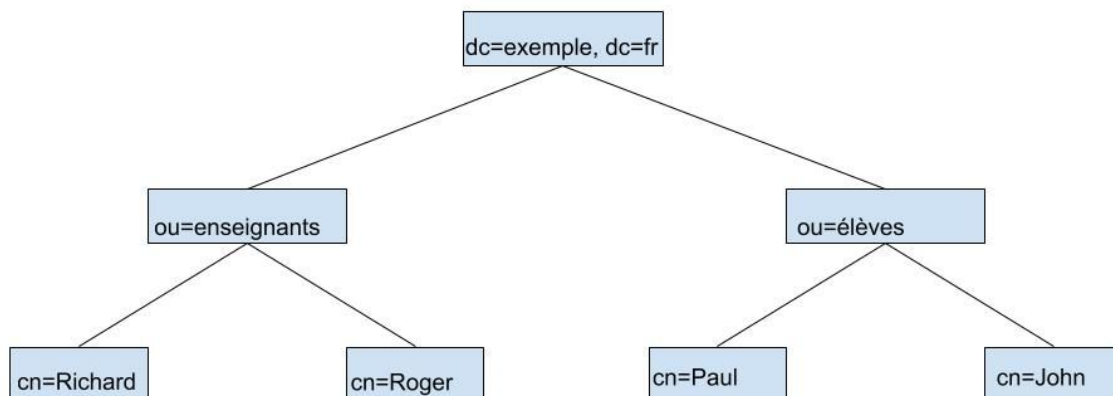
## IV - Authentification LDAP

### 1. Présentation

#### 1.1 - Technologie Ldap

Lightweight Directory Access Protocol (LDAP) est un protocole permettant la gestion d'un annuaire. La création de l'annuaire doit suivre une norme LDAP concernant le système de données, les règles de nommage, le modèle fonctionnel et le modèle de sécurité. Le protocole permet ensuite de faire des requêtes et des modifications sur l'annuaire.

L'annuaire est organisé sous forme arborescente, la racine représente souvent le nommage DNS utilisé pour le serveur LDAP, ensuite les noeuds ont des attributs auxquels on attribue des valeurs, le plus souvent chaque noeud va représenter un groupe dans lequel on va pouvoir ranger d'autres noeuds ou des feuilles, les noeuds proches de la racine vont représenter des groupes assez généraux et en descendant dans l'arbre on devient plus spécifique. Ainsi les feuilles représentent des utilisateurs avec un id (uid) unique ou un common name (cn). En assemblant tout le chemin de la racine vers l'utilisateur on obtient un distinguished name (dn) représentant un utilisateur ainsi que les groupes auxquels il appartient.



Par exemple : dn: cn=Paul, ou=élèves, dc=exemple,dc=fr

## 1.2 - Objectif

Dans l'université d'angers, chaque utilisateur est enregistré dans un annuaire LDAP. On utilise cet annuaire pour l'identification des utilisateurs sur la plupart des applications réservés à l'université. Notre objectif est ici de mettre en place cette identification commune à notre application LIDE. Pour ceci on suivra plusieurs étapes, étant donné qu'il faut du temps pour avoir accès au serveur LDAP de l'université, on élabore d'abord un serveur test pour pouvoir faire nos essais dessus. Ensuite il faudra se familiariser avec la technologie en faisant une page d'authentification ldap indépendante de notre projet. Enfin il faudra intégrer l'authentification ldap à notre projet et surtout la combiner au système d'utilisateurs et de sécurité déjà présent.

L'accès au serveur LDAP de l'université peut être géré de plusieurs manières. Soit on nous communique l'adresse du serveur LDAP afin qu'on l'utilise lors de l'authentification, soit l'authentification se fait avec un CAS de l'Université d'angers, qui fait l'authentification LDAP pour nous et nous renvoie l'utilisateur authentifié, on connectera alors cet utilisateur à LIDE avec le système déjà en place.

## 2. Mise en place

### 2.1 - Serveur LDAP

Plusieurs technologies sont disponibles, la plus populaire est open LDAP. C'est une implémentation libre de LDAP qui est facilement déployable sur la plupart des systèmes d'exploitation. Une fois installé on peut modifier notre annuaire en ajoutant de nouvelles entrées dans un fichier puis en ajoutant le contenu du fichier dans la base de donnée LDAP avec ldapadd ou ldapmodify si on veut modifier des entrées existantes. La commande ldapsearch est aussi très utilisée elle permet d'afficher les entrées correspondantes aux filtres qu'on renseigne.

On peut simplement installer openldap sur un serveur tel quel, cependant il peut être intéressant d'installer php ldap admin pour faciliter la gestion de l'annuaire, on pourra alors avoir une interface graphique ce qui simplifie grandement les choses par rapport aux commandes de bases de openLDAP.

Une alternative intéressante pourrait être le logiciel Apache Directory, qui permet le déploiement, la consultation et la gestion d'un annuaire LDAP de manière très facile.

Sinon pour les tests on peut simplement utiliser un serveur LDAP publique en ligne.

## 2.2 - Authentification LDAP en php

J'ai créé un formulaire html permettant de rentrer un identifiant et un mot de passe, je récupères alors ces valeurs et je les utilise pour faire une requête ldap. Pour faire cela il faut d'abord se connecter au serveur avec la fonction php `ldap_connect` avec l'adresse du serveur, on obtient alors un objet qu'on va utiliser plus tard pour faire des requêtes à partir de nos identifiants. On fait la requête avec la fonction `ldap_search`, si l'utilisateur est existant, la fonction retournera les informations stockés sur le serveur lui correspondant. Le résultat sera parsé pour ne garder que l'id de l'utilisateur que l'on pourra utiliser plus tard pour s'identifier sur LIDE.

# 3. Intégration à LIDE

L'intégration est rendue compliqué par la manière dont le projet a été fait. En effet FOSUserBundle permet la gestion de tout l'aspect utilisateur or FOSUser est un bundle qui permet une marge de manoeuvre assez mince concernant des fonctionnalités de base comme l'authentification et la gestion des utilisateurs. J'ai tout de même exploré quelques pistes:

## 3.1 - Intégration de LDAP dans FOSUser

Pour commencer on intègre LDAP et Guard dans le composer du docker web de l'application. Guard est un bundle permettant une gestion avancée de l'authentification et de l'utilisateur, et contrairement à FOSUserBundle on peut lui apporter de nombreuses modifications, en principe il faudrait utiliser Guard seul, sans FOSUserBundle mais ici le remplacement n'était pas envisageable étant donné qu'une grande majorité du code était basé sur FOSUserBundle.

Ensuite il faut paramétrer les fichiers config, (`security.yml`, `config.yml`, `config.yml`) de l'application, on fait en sorte que l'authentification soit gérée par ldap, on précise l'adresse du serveur LDAP et on peut potentiellement assigner des rôles à l'utilisateur en fonction de son groupe LDAP.

Cependant l'installation pose problème, la documentation sur symfony et LDAP est en grande majorité faite pour symfony 2, et malgré l'essai de différents paquets la plupart des options précisées dans les fichiers `.yml` ne sont pas reconnus par symfony.

Cette méthode n'a donc pas abouti.

### 3.2 - Utilisation d'un controller externe

Cette méthode a le mérite d'être utilisable avec le CAS, pour l'utiliser avec LDAP il faudrait connecter une page d'authentification LDAP au controller.

On met en place un controller, en envoyant une requête contenant l'ID de l'utilisateur sur l'adresse de ce controller on doit faire en sorte de connecter cet utilisateur à LIDE, ou si il n'existe pas, créer un nouvel utilisateur.

Ici l'authentification LDAP est fait avant que l'on arrive au controller on peut donc créer de nouveaux utilisateurs sans autres vérifications et sans craindre de doublons.

Il faudra simplement s'assurer que le controller ne peut recevoir de requêtes que de source vérifiés: le CAS de l'UA ou notre page d'authentification LDAP.

On peut même se permettre d'avoir le même mot de passe pour tous les utilisateurs, on peut ainsi le mettre en dur dans le code et simplement se soucier de l'utilisateur.

Pour faire la transition entre l'ID envoyé par la requête et la connexion d'un utilisateur dans LIDE, on fait une recherche de l'ID dans la base de données des utilisateurs avec un `user_manager`, si la recherche échoue on crée un nouvel utilisateur sinon on récupère l'utilisateur existant.

Ensuite on utilise l'utilisateur et le mot de passe pour générer un nouveau token, une nouvelle session et un nouvel event. On peut alors envoyé l'utilisateur sur la page approprié: vers la route "easyadmin" si l'utilisateur a la rôle admin et vers "main\_homepage" si l'utilisateur a un autre rôle.

Cependant, la configuration du controller pose problème, une fois configuré dans le fichier `routing.yml` approprié, on ne peut accéder à ce controller qu'une fois qu'on est parvenu à se connecter sur la page `/login` de LIDE, hors c'est ce qu'on veut précisément éviter. Même en surchargeant le controller `/login` de base avec notre controller, la redirection reste inchangée.

### 3.3 - Conclusion LDAP

Il sera difficile d'intégrer une authentification LDAP à notre projet avec la manière dont le projet a été fait. L'authentification de base devrait être basé sur LDAP, ce qui n'est pas réalisable avec `FOSUserBundle` et notre version de symfony, surtout si on change de version, on est forcé de contourner le problème ce qui nous donne un projet mal organisé avec de potentielles failles de sécurité ou malfunctions.

Serait-il possible d'enlever `FOSUserBundle` pour avoir une authentification "maison" permettant d'intégrer LDAP, Guard par exemple est une très bonne alternative. Le code est basé sur `FOSUserBundle` mais il est peut être possible de conserver une majorité du code existant en créant des classes semblables à ce que `FOSUser` propose.

## V - Conclusion

La réalisation de ce projet à été une expérience inédite pour nous, habitués à écrire majoritairement du code, nous avons rapidement dû nous adapter à ce projet qui demandait un autre type de compétences. En effet, une fois familiarisé avec les framework Symfony et VueJS, nous avons majoritairement travaillé sur de la configuration de projet et du débogage.

Nous avons pu découvrir de nouvelles technologies comme LDAP ou Docker et surtout nous avons beaucoup appris sur le travail en groupe et la gestion des tâches .

L'application est désormais fonctionnelle et facilement déployable, elle n'est cependant pas terminée:

Premièrement, la migration vers la version supérieure de Symfony ne fut pas concluante et est perçue pour nous comme un échec.

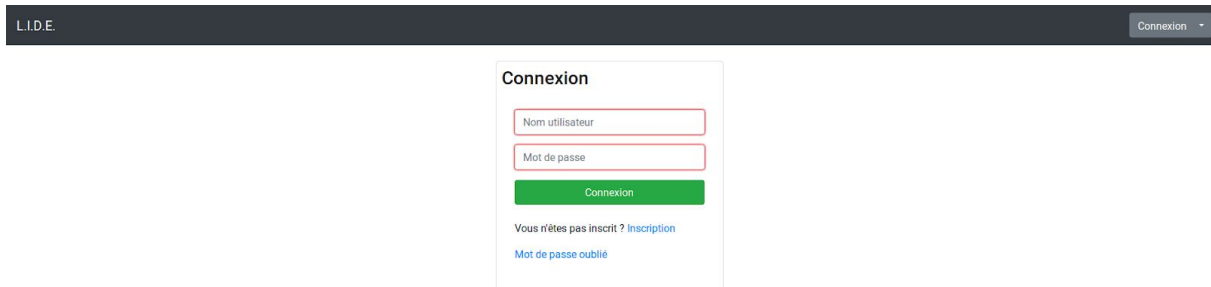
Deuxièmement, l'authentification reste à terminer une fois que le CAS de l'université sera disponible.

Enfin des fonctionnalités comme l'auto-complétion et le débogueur seront à ajouter pour disposer d'un IDE viable.

Nous tenons à remercier nos chefs de projets pour leur encadrement mais également notre professeur référent.

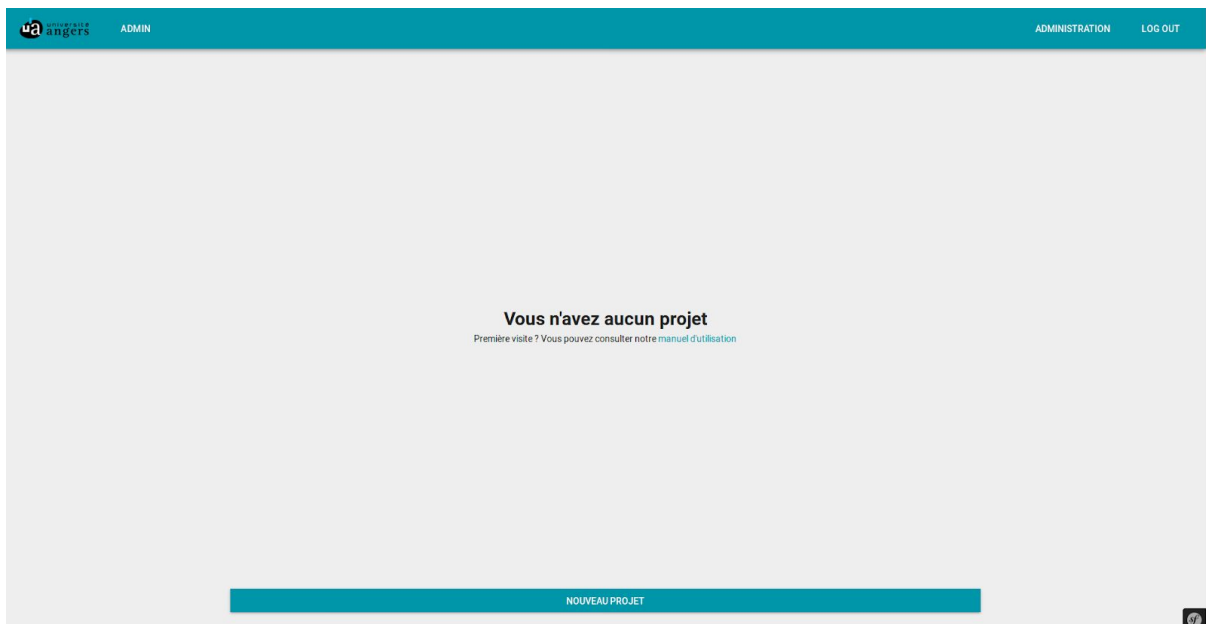
# Annexe

Page de connexion :



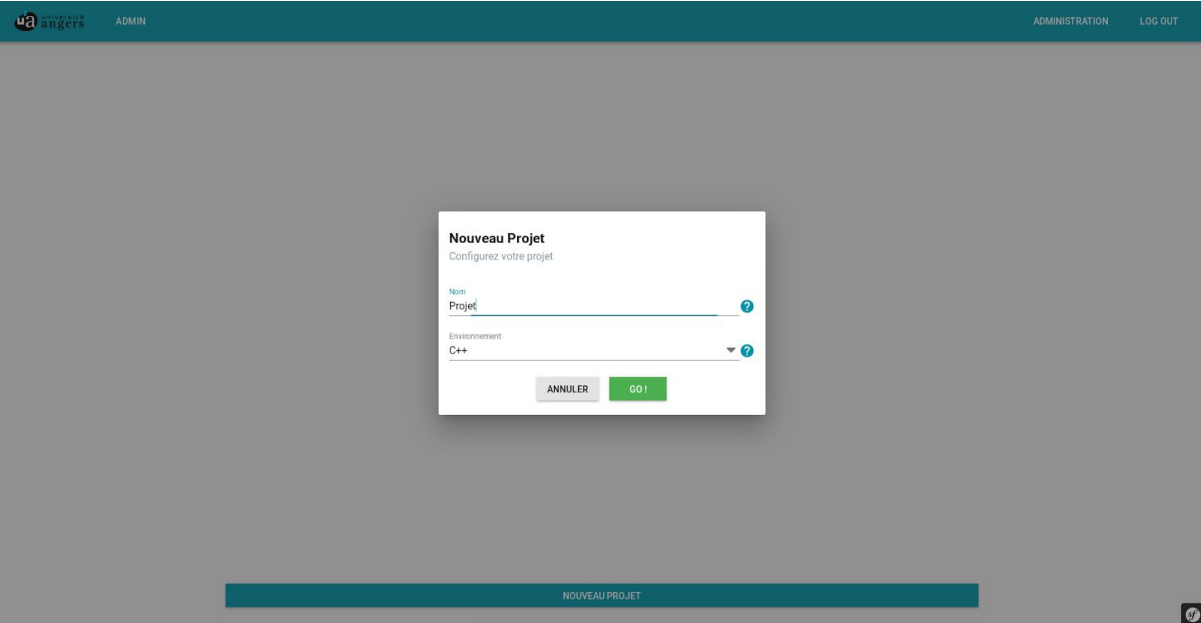
The screenshot shows the login page of the L.I.D.E. system. At the top, there is a dark header bar with 'L.I.D.E.' on the left and a 'Connexion' button on the right. The main content area is white and contains a central login box titled 'Connexion'. Inside this box, there are two input fields: 'Nom utilisateur' and 'Mot de passe'. Below these fields is a green 'Connexion' button. Under the button, there are two links: 'Vous n'êtes pas inscrit ? [Inscription](#)' and '[Mot de passe oublié](#)'.

Page création de projet :

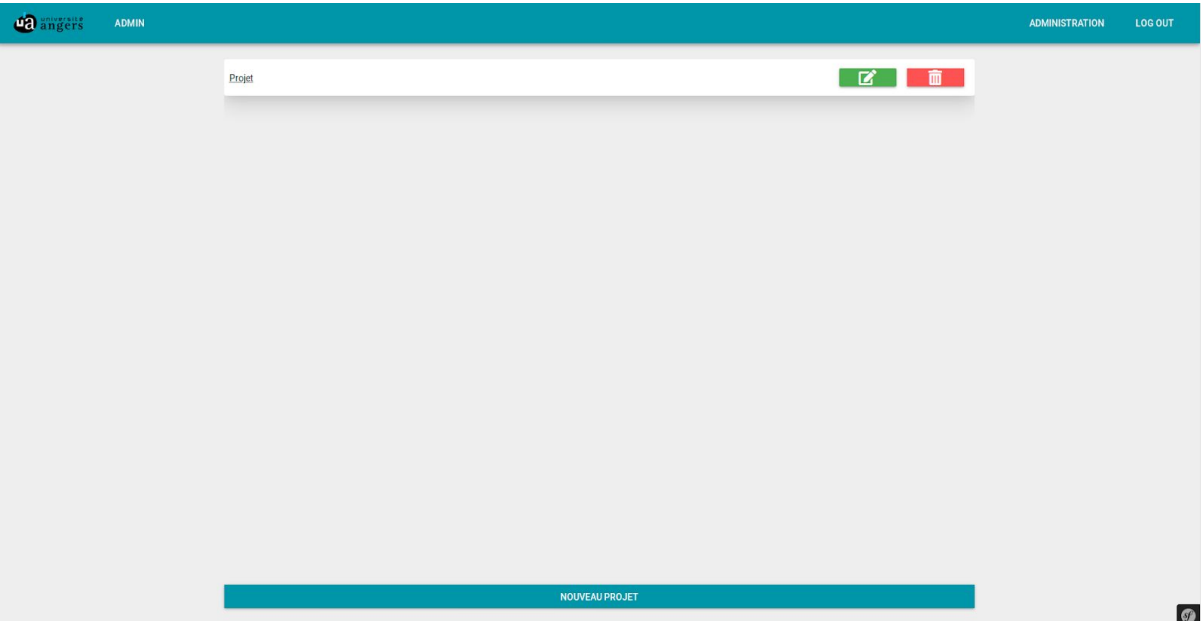


The screenshot shows the project creation page. At the top, there is a teal header bar with the 'ua université angers' logo on the left, 'ADMIN' in the center, and 'ADMINISTRATION' and 'LOG OUT' on the right. The main content area is light gray and contains the text 'Vous n'avez aucun projet' in bold. Below this text, there is a smaller line of text: 'Première visite ? Vous pouvez consulter notre [manuel d'utilisation](#)'. At the bottom of the page, there is a teal button labeled 'NOUVEAU PROJET'.



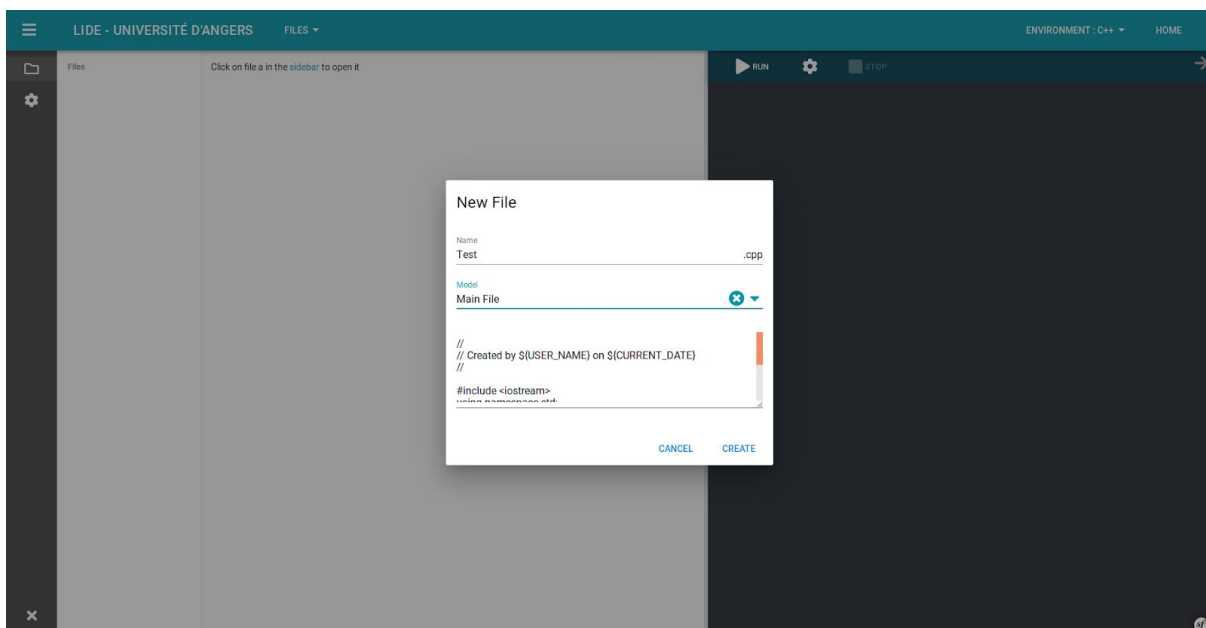
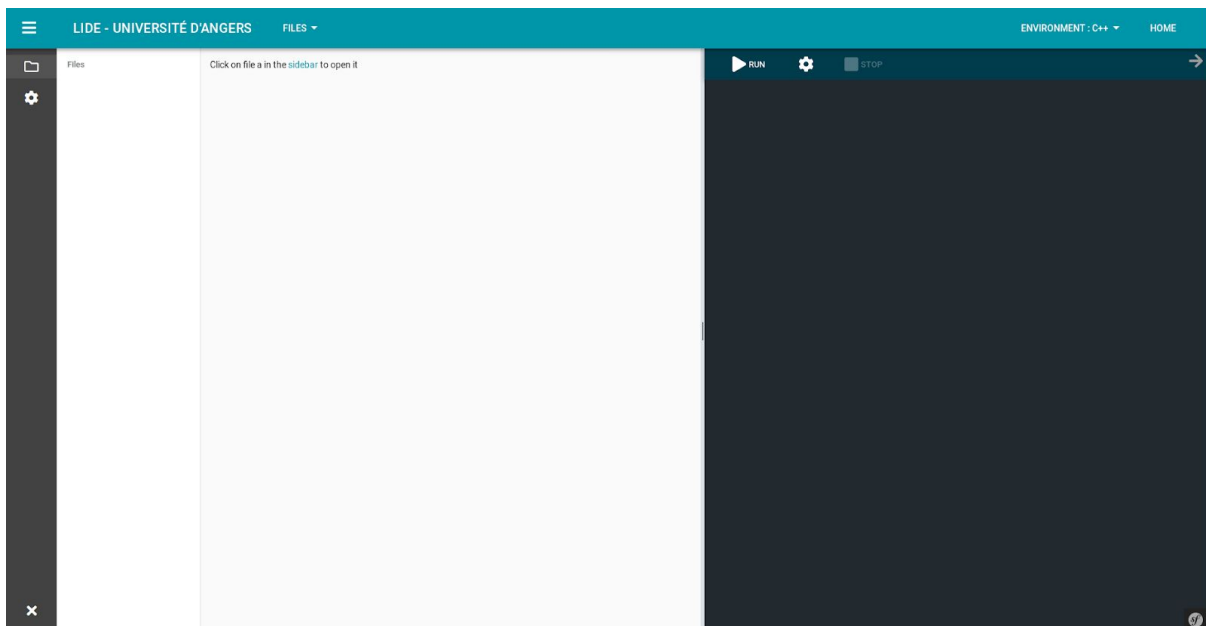


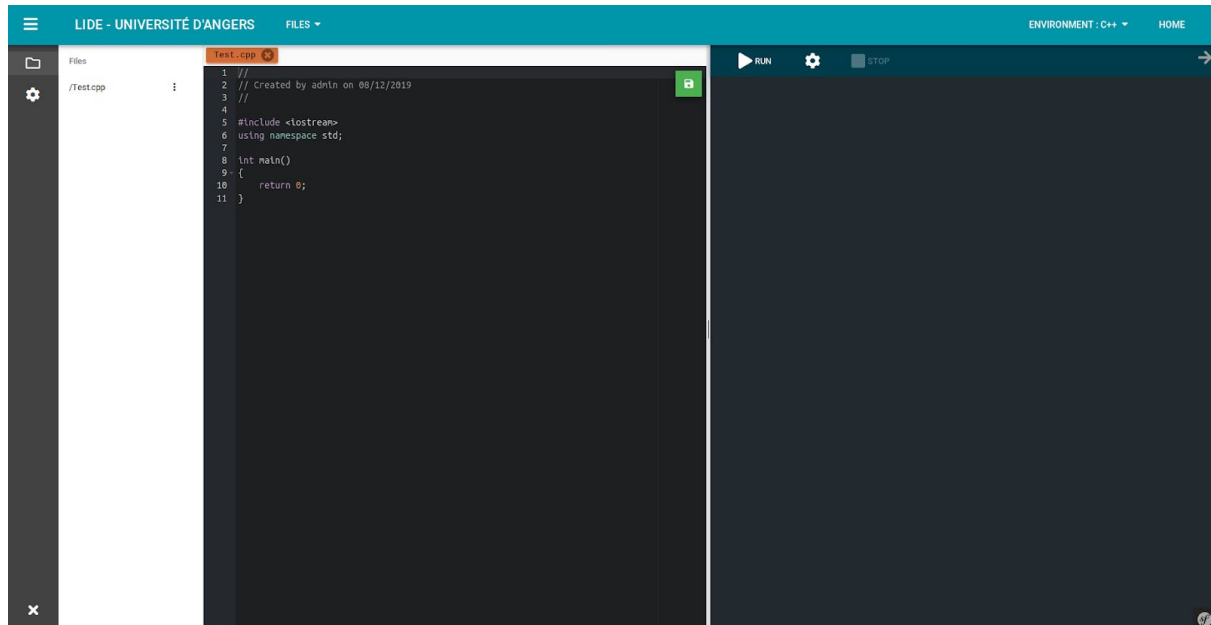
The screenshot shows the 'Nouveau Projet' (New Project) form in the UA-LIDE admin interface. The form is centered on a grey background. It has a title 'Nouveau Projet' and a subtitle 'Configurez votre projet'. There are two input fields: 'Nom' (Name) with the value 'Projet' and 'Environnement' (Environment) with the value 'C++'. Both fields have a question mark icon to the right. At the bottom of the form are two buttons: 'ANNULER' (Cancel) and 'GO !' (Go). The top navigation bar is teal and contains the UA logo, 'ADMIN', 'ADMINISTRATION', and 'LOG OUT'. A teal button labeled 'NOUVEAU PROJET' is at the bottom center of the page.



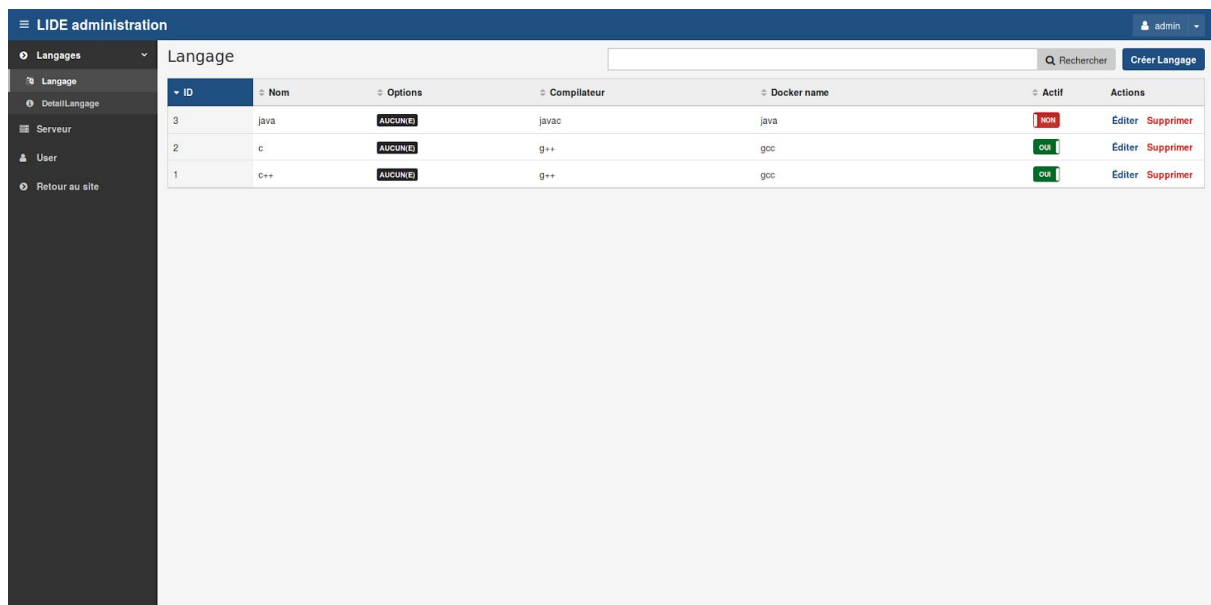
The screenshot shows the project list in the UA-LIDE admin interface. The top navigation bar is teal and contains the UA logo, 'ADMIN', 'ADMINISTRATION', and 'LOG OUT'. Below the navigation bar is a list of projects. The first project is 'Projet', which has a green edit icon and a red delete icon to its right. The background is light grey. A teal button labeled 'NOUVEAU PROJET' is at the bottom center of the page.

Page pour la compilation :





Page d'administration :



The screenshot shows the LIDE administration page. The top bar is dark blue with the text "LIDE administration" and a user dropdown menu showing "admin". The left sidebar has a menu with "Langages", "Langage", "DetailLangage", "Serveur", "User", and "Retour au site". The main area is titled "Langage" and contains a table with the following data:

ID	Nom	Options	Compilateur	Docker name	Actif	Actions
3	java	AUCUNE	javac	java	NON	Editer Supprimer
2	c	AUCUNE	g++	gcc	OUI	Editer Supprimer
1	C++	AUCUNE	g++	gcc	OUI	Editer Supprimer