Cahier des charges
Environnement de développement en ligne

Chefs de projet :

MAINFROID Pierre-Olivier
CHEVALLIER Sullivan
BAZANTE Alice

Développeurs :

VIOLLETTE Paulin
RAHIER Valentine
IBRIR Yassine
MARTINS MOSCA Jérôme

Table des matières

I/ Introduction....3
II/ Expression fonctionnelle du besoin....3
A- Fonctionnalités de base....3
B- Fonctionnalités avancées....3
C- Futures fonctionnalités envisagées....4
III/ Contraintes du projet....4
IV/ Charte graphique....5
V/ Management....5
VI/ Solutions proposées....5
A- Technologies....5
B- Base de données....6
C- Interfaces....7
1) Page principale....7
2) Page de connexion et inscription....8
3) Administration....8
D- Fonctionnement général....8
E- Licence....9
F- Hébergement....9
VII/ Planning....9

I/ Introduction

Lors de leur première année, les étudiants en Licence de la faculté des sciences d'Angers accèdent au portail MPCIE. Ce portail permet d'introduire tout genre de science aux étudiants intéressés par la science. Cela permet à tous les jeunes élèves d'ouvrir leur esprit à la logique de l'informatique, souvent pré-introduite au lycée. Lors des cours de L1 d'introduction à l'algorithmique, ils sont amenés à développer leurs premiers programmes dans un langage simple. Malheureusement, les IDE (Integrated Development Environment), outils pour réunir et simplifier l'utilisation d'éditeur de texte, de compilateurs etc.… sont difficilement installables et configurables pour des personnes non expérimentées.

Certains sites offrent la possibilité, à tous et partout, de développer des programmes en ligne sans installation. Cependant, les codes doivent rester simples et le nombre de demandes par utilisateur est limité. Par exemple, une toutes les 5 minutes. Pour la faculté, une centaine de personnes doivent pouvoir y accéder en même temps ce qui pose problème car la faculté est connecté comme une seul personne.

II/ Expression fonctionnelle du besoin

Fonctionnellement, les professeurs de première année de Licence ont besoin d'une application web permettant d'écrire et tester des codes simples avec une interface très simple. Cette interface devra être épurée au maximum pour permettre aux étudiants débutants de ne pas se perdre. Il faudra prendre en compte différents langages. L'application devra pouvoir supporter une centaine d'utilisateurs en parallèle (pour les évaluations).

A- Fonctionnalités de base

- Choix du langage (prise en compte de plusieurs langages)
  - C++
  - C
- Coloration syntaxique du code
- Possibilité de compiler et exécuter le code
  - Accès aux messages d'erreur de la compilation
  - Accès à l'affichage de l'exécution
- Possibilité d'interagir avec le programme
  - Répondre à des questions,
  - Renseigner des nombres, ...
- Possibilité d'importer du code
- Possibilité d'exporter le code écrit
- Possibilité d'ajouter des arguments

B- Fonctionnalités avancées

- Gestion de plusieurs fichiers de code suivant les extensions
  - Choix de l'extension suivant le langage
- Possibilité de modifier les options de compilation
  - O2, Wall, …

- Possibilité d’ajouter des fichiers externes au code
- Mettre en place une administration
  - Gestion des serveurs
  - Gestion des langages et de leurs extensions
- Authentification bloquée suivant une adresse mail configurable
  - Par exemple : @etud.univ-angers.fr

C- Futures fonctionnalités envisagées

- Auto-complétion du code
- Débogueur
- Intégration d’outils d’analyse

III/ Contraintes du projet

Le coût du projet a été fait à partir d’un macro chiffrage :

Développeur	Chef de projet	Consultant
400,00 €	600,00 €	600,00 €

Item	Nombre de jours de développement	TJM	Total HT €
Interface	**19**	**400**	**7600 €**
Page de connexion	1		
Page d’inscription	1		
Page principale	15		
Administration	2		
Base de donnée	**3**	**400**	**1200 €**
Serveur d’exécution distinct	**16**	**400**	**6400 €**
Création Docker	2		
Connexion Serveur/Interface	4		
Connexion SSH	4		
Script d’exécution	6		
Design	**5**	**400**	**2000 €**
Mise en responsive	3		
Adaptation charte graphique	2		
Internationalisation des écrans	**1**	**400**	**400 €**
Étude fonctionnelle	**15**	**600**	**9000 €**
Recette et mise en production	3	600	1800 €
Hébergement de l’application 1 an		FORFAIT	300 €
Tests		INCLUS	INCLUS
Gestion projet		INCLUS	INCLUS
Documentation		INCLUS	INCLUS
Correction des anomalies pendant 1 an		INCLUS	INCLUS
Prix par application	102		26600 €
Taux TVA			20 %
Prix TTC			31920 €

Le délai de livraison a été fixé par les enseignants des étudiants de première année de Master à la semaine du 11 décembre 2017.

IV/ Charte graphique

Le client n'a pas de charte graphique. Il n'a pas non plus d'exigences particulières. Nous avons donc le champ libre pour laisser cours à notre imagination. Cependant, nous devons rester sobres. Nous allons utiliser Bootstrap, un framework CSS permettant de faciliter le développement de l'interface. Le logo sera LIDE : Licence Informatic Development Environment.

V/ Management

Nous gérons le projet en mode agile. Pour répartir les tâches, nous utilisons Trello. (https://trello.com/b/PCq4IDlC/lide-licence-informatic-development-environment). De plus, pour faciliter la communication entre l'équipe, nous utilisons Slack. Chaque semaine, nous faisons un point pour voir l'avancé.

VI/ Solutions proposées

A- Technologies

Nous avons choisi d'utiliser le framework Symfony2 (PHP) pour développer l'application. En effet, utiliser ce framework va nous permettre de respecter une structure MVC et d'avoir accès à une communauté open-source. Pour gérer les dépendances PHP, nous utiliserons Composer et pour les dépendances web, NPM. Grâce à NPM, les librairies Javascript et CSS seront téléchargées en local. Pour l'automatisation de tâches, nous avons mis en place Gulp. Ensuite, pour déployer automatiquement nous utiliserons Capistrano. Pour la coloration syntaxique du code, nous intégrerons la librairie Javascript highlight.js.

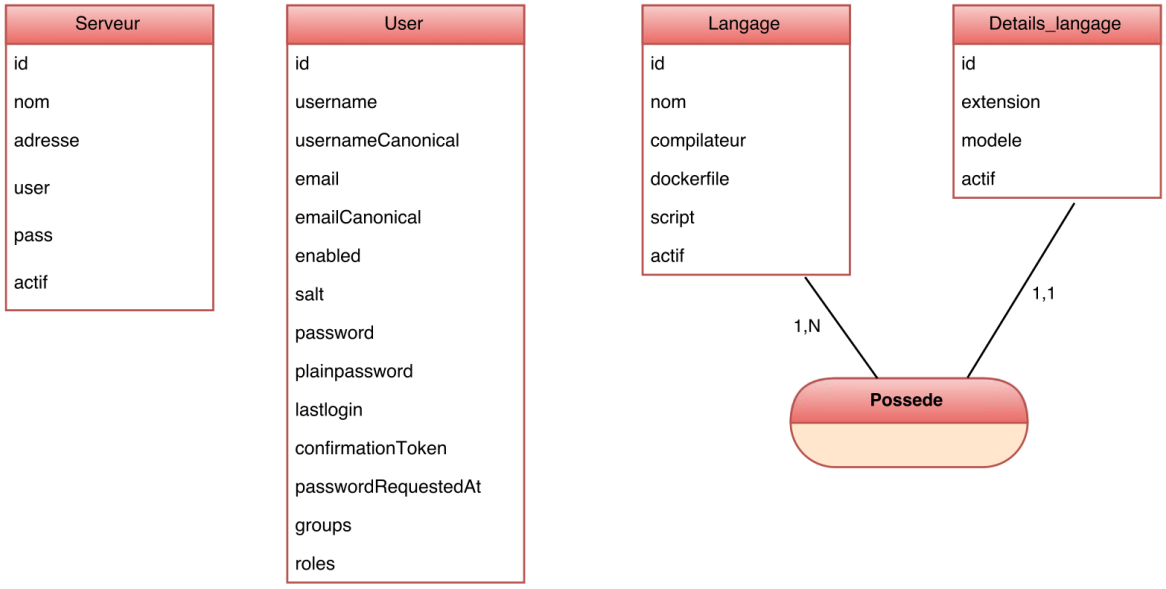
Au niveau du serveur web, nous allons prendre Apache avec une base de données mariaDB qui est un dérivé de MySQL mais en licence libre. Un serveur Linux sera créé pour accueillir l'application et pouvoir faire les démonstrations aux clients.

Pour versionner l'application et pour partager l'avancement de notre code, Git sera notre dépôt accompagné de l'interface GitKraken en local pour simplifier son utilisation. (https://github.com/bazanta/LIDE/tree/develop)

Pour gérer plus facilement les serveurs qui exécuteront le code, nous avons décidé d'utiliser Docker. En effet, il sera plus facile d'avoir la même base (compilateurs,...) sur chaque serveur à notre disposition. La seule contrainte est d'avoir docker d'installé sur les nouveaux serveurs.

Pour la communication entre les serveurs de compilation et d'exécution et l'interface, nous utiliserons la technologie SSH, Websocket et peut-être la librairie Javascript console.js. Ces technologies ne sont pas encore sûres car la partie communication est délicate et n'est pas encore bien définie. Nous adapterons suivant la mise en place de Docker.

B- Base de données



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.

LANGAGE :

Le nom d'un langage est unique. La table langage contient le dockerfile, qui est un fichier 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. Pour finir, il y a 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.

DETAILS_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.

USER :

Un utilisateur est unique. La table user possède l'id unique pour différencier chaque utilisateurs, ainsi qu'un e-mail unique pour le contacter, la date de l'inscription, une valeur pour crypter, le mot de passe, la date de la dernière connexion, le groupe de l'utilisateur, un token de confirmation pour validé les e-mails, le rôle des utilisateurs (Administrateur, Utilisateur, Super Administrateur), le nom de l'utilisateur et son prénom.

Toutes les tables 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.

C- Interfaces

1) Page principale

Illustration 1: Structure de la page principale, aux couleurs près

Le logo sera affiché en haut à gauche. Une liste déroulante contenant les différents langages pris en charge sera dans la barre de menu qui sera fixe. Lorsque le langage sera changé, une « modale » s'ouvrira en demandant une confirmation à l'utilisateur, car le changement de langage entraînera la perte du code en cours. En haut à droite, le profil de l'utilisateur connecté avec la possibilité de se déconnecter.

En bas à gauche de l'écran, deux boutons fixes seront présents. Un pour exporter le code en cours dans un ou des fichier(s) sur l'ordinateur suivant le nombre d'onglets ouvert. L'autre pour lancer la compilation et l'exécution. Une petite flèche à coté du deuxième bouton permet de changer les paramètres de la compilation et de l'exécution. Ainsi, il y aura le choix :

- compiler seulement ou compiler et exécuter
- ajouter des fichiers externes à l'exécution
- définir les options de compilation (-Wall, -O2, …)
- ajouter des arguments à l'exécution

A gauche de l'écran, il y aura la zone du code avec en en-tête de la zone un bouton « + » pour ajouter un fichier (importer ou créer un nouveau) avec le choix de l'extension suivant le langage choisi. Ainsi, il y a une gestion sous forme d'onglets des différents fichiers qui composent le code.

A droite de l'écran, il y aura la console avec deux onglets, résultat de compilation et résultat de l'exécution. Dans l'onglet de l'exécution, une zone de texte sera présente pour interagir avec le programme. Une petite flèche permet de réduire la zone console (compilation et exécution) pour laisser toute la place au code.

2) Page de connexion et inscription

Pour accéder à la page principale de l'application, il faut être connecté. Pour cela il existera une page de connexion. Si l'utilisateur se rend sur une URL sans être connecté, il sera redirigé vers la page de connexion. Sur cette page, il y aura la possibilité de s'identifier, de s'inscrire et une explication de l'application. Pour l'inscription, une adresse e-mail par défaut permet de limiter les inscriptions. Par exemple @etud.univ-angers.fr. Cette adresse e-mail sera paramétrée dans un fichier de configuration.

3) Administration

Une administration sera aussi mise en place pour permettre aux enseignants de gérer l'application par eux-même. Ils pourront ajouter un langage, supprimer un langage ou bien le modifier à leur convenance. Ils pourront gérer les extensions liées aux langages ainsi que le modèle par défaut. Les administrateurs auront également la possibilité de gérer les serveurs (Ajouter, supprimer et modifier les serveurs de compilation et d'exécution). Seul les utilisateurs ayant le rôle d'administrateur pourront y accéder.

L'administration sera mise en place à l'aide du bundle symfony EasyAdminBundle.

D- Fonctionnement général

Nous avons divisé l'application en 3 sections distinctes : la partie web, la partie serveur et la communication entre les deux parties. La communication est assurée grâce au protocole SSH associé à des web-sockets.

La partie serveur est la plus importante car il faut bien sécuriser les exécutions pour ne pas corrompre le serveur. En effet, nous allons exécuter le code écrit par les étudiants mais ne pouvons pas vérifier leur code en détails (boucle infinie, accès non autorisé, copie de fichiers,…). Il faudra mettre en place une gestion des droits, limiter la mémoire autorisée par utilisateur, tuer les exécutions trop longues, … Pour pouvoir exécuter le code écrit par les utilisateurs sans mettre en danger le serveur principal de l'application, nous avons décidé d'utiliser d'autres serveurs. Ainsi, si une surcharge survient, il y aura la possibilité d'ajouter un nouveau serveur avec Docker de disponible. Ensuite, grâce à Docker, nous lancerons notre image contenant les différents compilateurs nécessaires.

L'application fonctionnera comme suit :

- L'interface et le cœur de l'application se trouveront sur un serveur principal ainsi que la base de données. Il y aura aussi les images Docker qui serviront à la compilation, le script qui permettra la compilation des images Docker et celui qui permet de les lancer.
- Les serveurs de compilation et d'exécution recevront par le biais d'un script les images compilées. Un deuxième script permettra le lancement des images. Ces serveurs posséderont eux-même un script principal qui exécutera le script de lancement des images, dès leur réception. Ce script principal sera relancé à intervalles réguliers à l'aide d'une tâche Cron.
- Pour manager les serveurs à disposition et éviter les surcharges, nous utiliserons un reverse proxy qui basculera entre les serveurs de compilation.

E- Licence

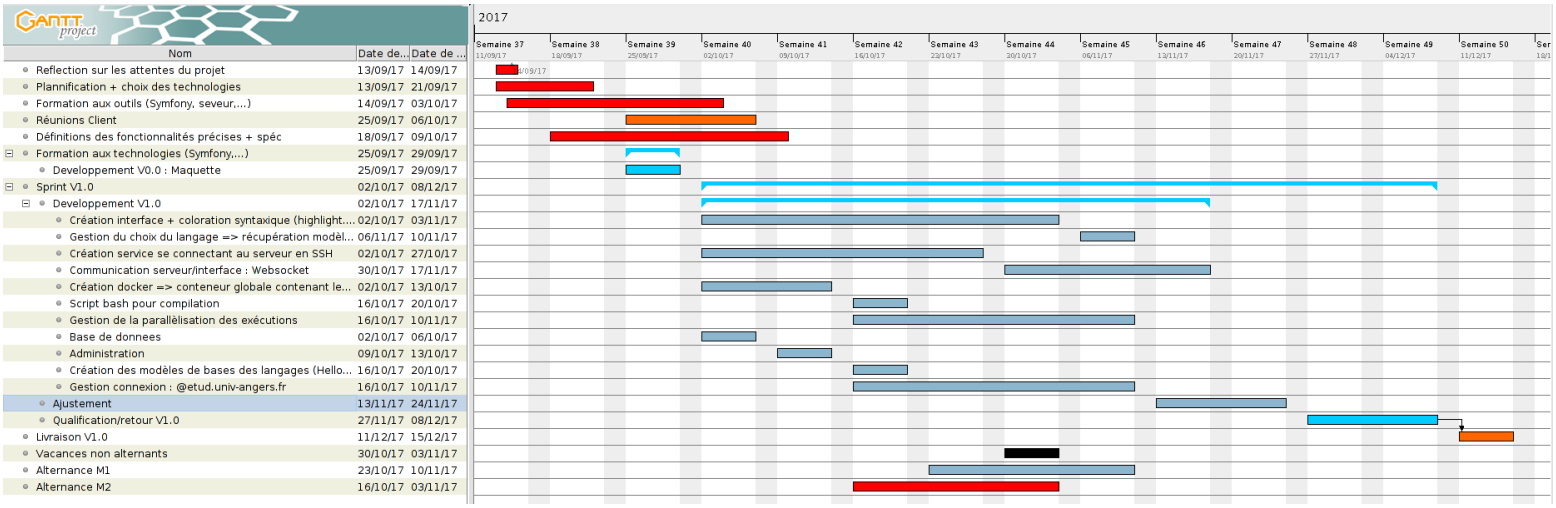
La licence GPL a été choisie pour que le code soit open-source. Cela permet à tout le monde d'accéder au code et de pouvoir le perfectionner. Cependant, personne ne pourra utiliser notre code à des fins commerciales.

F- Hébergement

Un serveur personnel est loué durant le temps de développement pour centraliser, vérifier et tester le code. Nous acheterons le serveur sur OVH.

VII/ Planning

Un rétro planning a été fait à partir de la date de livraison du 11 décembre.



Nom	Date de...	Date de ...
Reflection sur les attentes du projet	13/09/17	14/09/17
Plannification + choix des technologies	13/09/17	21/09/17
Formation aux outils (Symfony, seveur,...)	14/09/17	03/10/17
Réunions Client	25/09/17	06/10/17
Définitions des fonctionnalités précises + spéc	18/09/17	09/10/17
Formation aux technologies (Symfony,...)	25/09/17	29/09/17
Developpement V0.0 : Maquette	25/09/17	29/09/17
Sprint V1.0	02/10/17	08/12/17
Developpement V1.0	02/10/17	17/11/17
Création interface + coloration syntaxique (highlight....	02/10/17	03/11/17
Gestion du choix du langage => récupération modèl...	06/11/17	10/11/17
Création service se connectant au serveur en SSH	02/10/17	27/10/17
Communication serveur/interface : Websocket	30/10/17	17/11/17
Création docker => conteneur globale contenant le...	02/10/17	13/10/17
Script bash pour compilation	16/10/17	20/10/17
Gestion de la parallélisation des exécutions	16/10/17	10/11/17
Base de donnees	02/10/17	06/10/17
Administration	09/10/17	13/10/17
Création des modèles de bases des langages (Hello...	16/10/17	20/10/17
Gestion connexion : @etud.univ-angers.fr	16/10/17	10/11/17
Ajustement	13/11/17	24/11/17
Qualification/retour V1.0	27/11/17	08/12/17
Livraison V1.0	11/12/17	15/12/17
Vacances non alternants	30/10/17	03/11/17
Alternance M1	23/10/17	10/11/17
Alternance M2	16/10/17	03/11/17