Serveur d'application PLM

---

1 15 janvier 2014

1.1 Données à conserver

- Énoncé de l'exercice ;
- code de l'élève ;
- nombre tentatives ;
- temps entre chaque tentatives ;
- changement d'exercice alors que l'actuel n'est pas résolu.

1.2 Structure stockage

- Un dépôt par utilisateur ;
- une branche par leçon ;
- une branche par exercice dans chaque branche de leçon.

- Message du commit : FAIL ou SUCCESS, l'ID de l'exercice, date, langage, indice utilisé, nombre de tests passés.
- Contenu du commit : code source, le message d'erreur s'il existe, le template et l'énoncé.

Fichiers avec le code source, le template et le résultat.
Commit lors de la compilation. Quand il est en ligne, il push.

1.3 Questions

- Serveurs ?
  - `http://www.loria.fr/~oster/plm-cloud/`
  - `http://jlmserver-chmod0.appspot.com`
- Utilisation de Play ? Utilité de Google App Engine (a priori non compatible avec Play 2.x) ?

2 24 janvier 2014

2.1 Git

Git pourrait faire l'espion et la session.

Il faut plutôt faire une branche par exercice.

Template, source, énoncé dans un seul commit. Chargeur qui récupère les données depuis le git. Faire le git en local et le push sur un serveur.

Première version pour monter les gits : (premier tiers de notre travail). Commit en local. Monter automatiquement sur le serveur.

2.2 Modification du code de la PLM

Regarder le code de la session de la PLM.

Besoins d'authentification des élèves.

ImportCloudSession.java SessionDB.java Exo, language, fichier : en mémoire ; ISessionKit gère l'écriture sur le disque. Exercice.java -> SourceFile.java : template et le body (ce que l'élève à tapé) actuel. Finir de mettre dans SessionDB le code de l'élève. SourceFile : ce qui vient du jar. SessionDB : ce qu'il faut écrire sur le disque. Qui est en charge de getCompilableContent ?

Ramener code de l'élève dans SessionDB.

Les tests doivent passer.

2.3 Mode professeur

Mode professeur à reprendre. Fonctionnalités attendues (serveur d'utilisation) :

- explorer graphiquement l'évolution de chaque élèves ;
- mise en forme des données : représentation ;
- alertes configurables (nombre essai, temps passé) ;
- détection d'exercices infaisable : personne n'y arrive ;
- traces utilisées en direct : savoir qui aider, et après coup, pour repérer les exercices qui marchent ou pas.

2.4 Autres

Google App Engine était une première tentative. Play plus simple et plus portable. Il faut éviter de dépendre d'une technique.

Système de badges : `openbadges.org`

Récupérer des leçons sur un serveur.

Notre rapport doit permettre de faire gagner du temps à la prochaine équipe. Rapport à rendre pour le 22 mai 2014. Soutenance : 28 mai 2014

Gestion de projet : ToDo liste, planning des grandes étapes.

Trois cibles : apprenants, auteurs de ressources : création de contenu, enseignants : faire leur séquences d'exo : éditeur d'exercice et de séquences. Moulinette d'extraction des données. Modélisation apprenante.

Prendre le contrôle à distance de la machine de l'élèves / chat vidéo

Nouveaux langages : scratch, js, C.

Forum collaboratif : commenter et proposer des exercices (les élèves par exemple). On propose aux autres élèves et les notes. (Recapcha)

Bonus : Poster sur GitHub pour le feedback des problèmes : dans les issues. PLM doit se loguer sur GitHub pour poster les messages d'erreurs. `http://developer.github.com/v3/issues/#create-`

2.5 Questions

- Différences entre SessionDB et ExecutionProgress ?
- Utilisation exclusive du git pour stocker le code en local ?
- Le serveur alerte le professeur d'un élève en difficultés à partir des données du git pushé en temps réel ? (Commit log en JSON -> données récupérables avec un fetch moins coûteux qu'un pull)

3 Mercredi 26 mars 2014

Le code est un sur un serveur git public :
- il n'y a aucune information personnelle ou identifiante ;
- il est librement accessible en lecture/écriture car git assure que rien n'est détruit (taille max des commits pour décourager les attaquants et éviter les pb de quota disque) ;
- les branches des élèves sont identifiées par le uuid stocké localement dans PLM.

Scénario :

1. les élèves peuvent pousser sur leur branche ;
2. un élève a plusieurs papareils ;
3. l'élève veut connecter/rattacher plusieurs UUID ;
4. le prof veut suivre ses élèves ;
5. l'élève peut retrouver sa session sur une machine fraiche ;
6. au démarrage :
   - nouvelle partie anonyme ;
   - continuer partie anonyme ;
   - se connecter à une identité ;
   - créer une nouvelle identité ;
7. créer un UUID au premier démarrage ;
8. une fois identifié, revendiquer une partie anonyme.

Il y a un mapping UUID(s) <-> identité optionnel et on peut le faire à postériori.
Scénario bonus :
- les élèves peuvent se créer une identité ;
- l'élève veut voir la version de son code d'il y a deux jours.

Il faut changer les ZipSessions et GitSessions :
- aucune écriture ;
- lecture sur le même layout que ZipSession ;
- attention : je change d'exo après avoir édité mais pas exécuté ;
- attention : savoir rapidement quels exos réussi en quel langage.

Tables :
- une table des identités : clé, nom complet, UUID(s), mail, ... ;
- une table authentifications : login, mot de passe, clé utilisateur ;
- une table participants : id individu, id classe ;
- une table classe : nom, descriptif, id classe, prof(s).

4 Mercredi 23 avril 2014

Pour gérer le multiutilisateur sur une machine, on remplace le dossier repository par un dossier dont le nom est le début du UUID.

Le nom de la branche associé à un UUID est son hashé.

Un utilisateur pour continuer sa session sur une nouvelle machine (ou pas nouvelle) en saisissant son UUID.

Ceci permet d'éviter de continuer une partie anonyme d'un autre (en récupérant son UUID à partir d'un git clone).

Continuer session avec JList des sessions locales Crée nouvelle session Ajouter une session à partir d'un UUID