Projet de POOCAv — eVALUATOR

version 1.0

Université Paris Diderot – Master 2

24 octobre 2017

1 Principe du projet

Le projet de POOCAv vise à vous faire progresser dans votre capacité *à concevoir un système objet extensible*. Il se déroule en *deux étapes*. Dans la première étape, un sujet vous est fourni. Ce sujet est *volontairement* décrit de façon informelle, sans vous donner d'indication pour le réaliser. Vous devez donc définir vous-même les *requis* (**délivrable 1**), l'*architecture* (**délivrable 2**), et une squelette exécutable de votre projet (**délivrable 3**). Une *implémentation* écrite en Scala de cette première version constitue le **délivrable 4** (squelette).

Vous recevrez ensuite un *autre sujet* qui est une extension du premier. Cette extension sera objet du **délivrable 5** (plus de détails seront fournis avec le deuxième sujet) et mettra à l'épreuve votre architecture initiale : un projet bien pensé *n'aura pas besoin de modifier* le code de la première version du projet pour traiter cette extension mais seulement de *rajouter de nouveaux composants*.

2 Sujet

Ce projet consiste en la réalisation d'un système d'évaluation automatique pour l'enseignement. Par exemple, un cas typique d'utilisation d'un tel système est l'évaluation et l'entraînement des étudiants d'un cours en ligne (MOOC). Pour ce projet, il est important de considérer les possibles extensions futures des fonctionnalités du logiciel. Par exemple, on peut considérer que dans une première version du projet, un seul utilisateur réponde à des questions à choix multiples. Une deuxième version du logiciel permettrait à l'usager de soumettre des rédactions, ou des images (ex. pour l'architecture). Enfin, une troisième version pourrait aussi gérer des questions de programmation attendant comme réponse du code source, dans un langage de programmation donné, évalué en exécutant des tests unitaires. [1]

Le système a au moins deux types d'usagers : les enseignants peuvent gérer les contenus des questionnaires (questions et réponses), accéder aux données de ses étudiants, donner des notes, etc. Les étudiants peuvent seulement accéder aux questionnaires des cours auxquels ils sont inscrits, répondre à des questions (d'entraînement ou d'examen), observer leur progression, etc. Ils ne peuvent pas par contre accéder aux réponses, ni aux données des autres usagers.

L'objectif final de ce projet est d'offrir l'application comme un service web à des universités ou des entreprises. Donc il est important, dès le debut, d'organiser l'application selon une conception client/serveur, où on fait comme si les usagers accédaient déjà au système par le web. Il est aussi conseillé de concevoir dès le début des interfaces claires, de type REST, entre le client et le serveur. [2]

Vous pouvez par exemple considérer, entre autres, les extensions ajoutant les fonctionnalités suivantes :

— permettre à deux étudiants de répondre à des questions posées par eux-mêmes, en mode compétition ;

— permettre à plusieurs étudiants de faire une compétition avec une limite de temps (à la TopCoder) ;

— permettre de discuter des solutions après l'examen/compétition dans des forums ;

1. Voir Web-Cat [5], TopCoder [4], CodeForces [2].
2. Il est conseillé de commencer à apprendre des présents les frameworks Play [3] et Akka[1] (ou équivalent) pour Scala.

— proposer des sessions d'entraînement où les usagers peuvent voir les réponses des enseignants en direct, etc.

Ces suggestions sont juste des idées d'extensions possibles pour vous inspirer. Vous devez imaginer d'autres extensions. Demandez au client (les enseignants) si elles sont pertinentes avant de les coder.

3 Travail demandé

Le projet est à traiter en groupes de minimum 3, maximum 5 personnes. Compte-tenu de la quantité importante de travail à effectuer, il est[?] essentiel de vous organiser en répartissant le travail entre les différents membres de l'équipe. Vous devez suivre une méthode de conduite de projet et avoir un planning rigoureux vous permettant de tenir les deadlines. Les retards seront sanctionnés dans la notation.

L'utilisation du Git est *obligatoire*. L'historique de ce dernier permettra de déterminer la contribution des membres de l'équipe et la gestion du temps dont vous avez fait preuve. Vous êtes libre de choisir l'hébergement Git que vous souhaitez—l'UFR offre un hébergement Git [3] mais vous pouvez aller ailleurs si vous le souhaitez. La seule contrainte et de donner *dès le début* des comptes Git aux enseignants pour pouvoir suivre votre travail.

3.1 Délivrables

Sauf si on précise le contraire, tous les documents délivrés seront au format PDF.

1. **Manuel d'utilisateur et cahier des charges (deadline : 31/10)**
   Dans un fichier `deliverables/requirements.pdf` sur votre dépôt, vous produirez une liste de propriétés (fonctionnelles et non fonctionnelles) que vous avez identifiées et que votre projet s'engage à respecter. Voir [6] et le cours de Génie logiciel avancé (M1).
2. **Architecture et modèle logique (deadline : 7/11)**
   — Dans un fichier `deliverables/architecture.pdf` sur votre dépôt, vous devez nous soumettre une architecture sous la forme de plusieurs diagrammes de classes UML et d'autres diagrammes de votre choix accompagnés d'explications justificatives. Ce document sera fourni au format PDF et pourra suivre le plan suivant :

   interprétation du sujet Vous expliquerez de façon informelle ce que vous avez compris du sujet et de ces enjeux. Quels sont les problèmes techniques et conceptuels que vous avez exhibés ?

   concepts Dans cette section, vous définirez les concepts utilisés pour modéliser le problème ainsi que les invariants essentiels du système.

   description de l'architecture Vous donnerez ici les diagrammes UML décrivant votre architecture et surtout sa justification. Attention à fournir différents points de vue à différents niveaux de détails sur votre architecture. Le but de cette partie est d'avoir une compréhension complète et global de votre application : cette explication doit passer en revue la totalité des fonctionnalités de votre logiciel sans rentrer dans les détails sans intérêt. Choisissez bien le niveau d'abstraction de vos explications !

   extensions envisagées Vous énumérez ici les généralisations et les extensions que vous avez imaginées et vous expliquerez pourquoi votre architecture permet de les traiter facilement.

   — **Attention :** Ce travail d'analyse et de rédaction est essentielle dans le projet. Vous ne devez pas le négliger et soigner la façon dont vous communiquez avec nous à travers les documents produits.

3. http://moule.informatique.univ-paris-diderot.fr:8080/users/sign_in

3. **Version 0 (deadline : 14/11)** (*walking skeleton*)
   Dans un répertoire `deliverables/skeleton/` vous devez nous soumettre une première ébauche de l'implémentation des votre projet. L'implémentation ne devra pas être nécessairement complète, loin de là, mais devra contenir toutes les abstractions prévues par votre architecture (même si beaucoup d'entre elles auront des méthodes non implémentées) et un jeu de tests exécutable pour le code déjà implémentés.
   Le squelette doit évidemment compiler à l'aide d'une commande `make` effectuée à la racine de ce répertoire. Un fichier `README` doit être également fourni et doit expliquer comment compiler et exécuter votre logiciel.
4. **Implémentation (deadline : 21/11)**
   Dans un répertoire `deliverables/version1/`, vous devez nous soumettre la première version complète de votre projet.
   Votre projet doit aussi être testé. La qualité du code—c'est-à-dire sa correction, sa robustesse et son élégance—sera prise en compte dans la notation.

Références

[1] Akka. https://akka.io/.

[2] Code forces – official website. http://codeforces.com/.

[3] Scala play. https://www.playframework.com/.

[4] Top coder – official website. https://www.topcoder.com/.

[5] Web cat – official website. http://web-cat.org/.

[6] Wikipedia - Software requirements specification. https://en.wikipedia.org/wiki/Software_Requirements_Specification. Retrieved October 2016.