Projet # 2
Développement de tests d'acceptation automatiques avec un outil de BDD

MGL7460
Automne 2016

1 Objectif

L'objectif de ce deuxième projet est de vous familiariser avec le développement de tests d'acceptation automatiques de style BDD, donc vous familiariser avec l'utilisation d'un outil de BDD, qui permet de décrire des récits utilisateurs, de les formaliser — avec des *features* et scénarios — puis de les exécuter pour les vérifier.

2 Ce qu'il faut faire

Pour ce travail, vous devez développer **des tests d'acceptation** exécutés de façon automatique par un outil de BDD pour une «petite application», **dans le langage de votre choix** (ou presque : voir plus bas).

Comme dans le Projet #1, cette application (et ses tests) devra être développée et livrée (cf. section 5, «Code source») à l'aide d'un logiciel de contrôle de code source et la vérification du bon fonctionnement à l'aide des tests d'acceptation devra pouvoir être faite de façon automatique, à l'aide d'une simple commande à un outil d'assemblage approprié.

Vous êtes «relativement libres» de choisir...

- Le langage de programmation (voir plus bas) ;
- Le cadre de tests d'acceptation ;
- L'outil d'assemblage.

Cette application n'a pas à être complexe — ce sont les tests d'acceptation à la BDD qui sont importants. Notamment, une application en «ligne de commandes» serait acceptable — et certainement plus simple qu'une application Web — mais c'est à vous de décider. Comme dans le Projet #1, votre application devra aussi comporter une forme de «données persistantes» — y compris, comme dans le projet #1, sous forme de données textuelles.

Vous devrez me soumettre, **pour approbation**, votre choix d'application, de langage, de cadre de tests, etc. au plus tard jeudi le 10 novembre.

Contraintes sur l'application et le langage

> *In most projects, the first system built is barely usable... Hence* **plan to throw one away; you will, anyhow.**
>
> *Fred Brooks, «The Mythical Man-Month»*

Voici quelques alternatives possibles quant au choix de l'application et du langage :

a. Vous concevez et mettez en oeuvre une nouvelle application, de votre choix, dans le langage de votre choix.

b. Vous choisissez la même application (semblable ou simplifiée) que pour le Projet #1, mais vous la mettez en oeuvre **en utilisant un autre langage que celui utilisé pour le Projet #1**.

c. Vous développez des tests d'acceptation `cucumber` pour l'application Ruby `Biblio`, application qui permet de gérer des emprunts de documents en ligne de commandes. Dans ce cas, les scénarios devront être décrits avec le *gem* `aruba` pour `cucumber`.

   Note : Il s'agit de l'application vue (en partie) dans le cadre du labo #4 et dans des diapositives des chapitres «Tests d'acceptation et BDD» et «Le patron "*Repository*"».[1]

   Pour cette option, comme l'application est déjà développée, vous serez aussi évaluée **sur la couverture de vos tests**, telle que mesurée par l'outil `simplecov`.[2] L'objectif n'est pas d'obtenir une couverture de 100 %. L'objectif est plutôt que l'on puisse voir, au fur et à mesure de l'avancement du projet, **la progression du taux de couverture**, et ce jusqu'à atteindre *un niveau raisonnable*.

Voici quelques **suggestions** (non limitatives) de combinaisons possibles de langages et d'outils de BDD :

Ruby + `cucumber`

Python + `behave!`

Java + `JBehave` Java + `Cucumber-JVM` Java + `Fit`

C# + `SpecFlow`

JavaScript + `behat`

L'accent est sur **les tests d'acceptation**. Il est évidemment souhaitable de développer un certain nombre de tests unitaires, mais contrairement au Projet #1, **les tests unitaires ne seront pas évalués.** Par contre, pour les alternatives a. et b. mentionnées plus haut, l'organisation et l'architecture de l'application seront aussi évaluées.

[1] Si votre application pour le Projet #1 ne fonctionnait pas ou était mal structurée/organisée, cette option **serait un bon choix**!

[2] Ce *gem* est indiqué dans le fichier `.gemspec` qui vous est fourni. Voir l'URL suivant pour sa description : `https://github.com/colszowka/simplecov`

3 Présentation orale

- Chaque «équipe» devra faire une présentation orale, d'environ ≈ 15 minutes, comportant les éléments suivants :
  - Brève description du langage et du cadre de tests utilisés, de l'environnement de développement, etc.
  - Description des principales fonctionnalités mises en oeuvre par l'application.
  - Description de l'architecture de l'application et des tests.
  - Brève description de ce que vous avez effectivement testé dans les tests d'acceptation.
  - Quelques exemples visant à l'illustrer le cadre de tests choisi — spécification des récits utilisateurs, des *steps*, etc.
  - Analyse de votre expérience : difficultés rencontrées, choses intéressantes apprises, etc.

  Si vous avez choisi l'alterative c., votre présentation devra aussi montrer la progression de la couverture des tests au fur et à mesure de l'avancement du projet.

- **Date des présentations : Jeudi** 1^{er} **décembre, 18h00.**

- La remise des diapositives doit être faite **avant** la présentation, **par courriel**.

  Date limite pour me transmettre vos diapositives :
  Jeudi 1^{er} **décembre, 16h00.**

  Si les diapositives ne sont pas remises à l'heure limite, l'équipe ne pourra pas faire sa présentation et se verra attribuer la note «0» pour cette partie du travail.

- Ordre des présentations : Aléatoire (annoncé au début du cours).

4 Rapport écrit

Chaque «équipe» devra remettre un (1) rapport écrit (≈ 10–12 pages, **fichier PDF**), rapport contenant les mêmes éléments que ceux de la présentation orale, **mais un peu plus détaillé**. Vous devrez aussi expliquer brièvement comment je peux utiliser votre logiciel, notamment, comment lancer les tests.

La remise du rapport se fait par courriel.

La date limite pour la remise du rapport (**fichier PDF**) : **Lundi 5 décembre, 10:00**.

5 Code source

Comme dans le Projet #1, je dois pouvoir obtenir une copie de votre code source — initalement en clonant votre dépôt, puis en le mettant à jour (`fetch`, `pull`, etc.) — **et ce même avant la remise finale** : Utilisez `GitHub`[3] ou `bitbucket`[4].

La date limite pour la remise du code source est la même que pour le rapport écrit.

6 Rapports de participation

- Chaque personne devra compléter et remettre un «Rapport de participation» :

  `http://www.labunix.uqam.ca/~tremblay/MGL7460/Projets/rapport-participation-2.docx`

- Les rapports de participation devront être remis **sous forme papier**.
- Date de remise : Jeudi 8 décembre, 18h00 (au début du cours).

[3] `tremblay-guy`
[4] `tremblay_g`