ENSEIRB
Année 2012-2013

3ème année
IT306

TD n°5 - Exemple de test d'une application web

Résumé : Cette séance considère la question du test d'une application Web développée avec le framework Rails. Du fait qu'une telle application profite du *pattern* MVC (Modèle-Vue-Contrôleur), elle présente des particularités qui permettent d'étudier les problématiques du test dans un environnement différent, mélangeant les langages et les interactions avec l'application : test unitaire, fonctionnel, d'intégration...

Pour les besoins de l'exercice, il est nécessaire de disposer d'un environnement Ruby (version 1.8.7) fonctionnel, ainsi que d'un certain nombre de bibliothèques, dont Rails (version 2.3.5). Sur les machines de l'ENSEIRB, positionner les variables d'environnement suivantes :

- export PATH=/net/ens/renault/local/bin:$PATH
- export GEM_HOME=/net/ens/renault/local
- export RUBYLIB=/net/ens/renault/local/lib: /net/ens/renault/local/lib/ruby/1.8: /net/ens/renault/local/lib/ruby/1.8/i686-linux

Sujet

Sur la page du cours est disponible une archive d'une application Rails. En utilisant cette application comme une source d'exemples, écrire une synthèse des différentes pratiques du test logiciel vues lors de ce cours.

Il est demandé de présenter ces pratiques dans un ordre logique suivant une démarche de test aussi exhaustive que possible. Il *n'est pas demandé* de tester entièrement l'application. Pour chacune de ces pratiques, il est demandé de fournir un exemple représentatif mettant en valeur cette pratique. Comme d'habitude, un fichier de code ne constitue pas à lui tout seul une explication. De plus, la synthèse ne devra pas contenir plus de 10 pratiques.

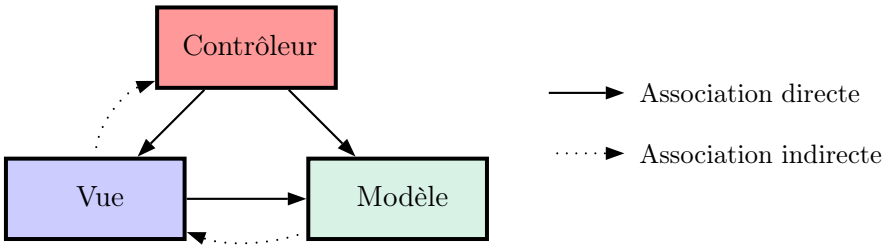
Liens

Vous pourrez vous aider des annexes fournies, ainsi que des ressources suivantes :

- A guide to testing Rails applications :
  `http://guides.rubyonrails.org/testing.html`
- Mocha, a library for mocking and stubbing in Rails :
  `http://mocha.rubyforge.org`
- RCov : code coverage for Ruby :
  `http://eigenclass.org/hiki/rcov`
- Selenium web application testing system :
  `http://seleniumhq.org`

Motif MVC (Modèle-Vue-Contrôleur)

Le motif de programmation MVC (pour *Model-View-Controller*), même s'il n'apparaît pas dans la liste de [EGV94], est considéré comme un motif classique de programmation. Il a été initialement utilisé en SmallTalk-80 [Bur92] pour la conception d'interfaces, et maintenant largement utilisé pour séparer les responsabilités des différentes couches de logiciels comme des serveurs d'applications web (J2EE, Rails, Pylons ...). Canoniquement, ce motif est basé sur le diagramme suivant :



Les *associations directes* correspondent ici au contrôle direct des instances de la classe pointée par les instances de la classe maître, par exemple à travers des appels de méthodes. Les *associations indirectes* se réfèrent à des messages sous forme de notifications, par exemple en utilisant le pattern "observateur". Le framework auquel nous allons nous intéresser se nomme Rails, et est implémenté en langage Ruby. Du diagramme précédent, il élimine les associations indirectes [1].

Arborescence Rails

Une application Rails se présente sous la forme d'une arborescence de la forme suivante :

- `app` le code relié au MVC, découpé en :
  - `models` les modèles ;
  - `controllers` les contrôleurs ;
  - `views` les vues ;
  - `helpers` le code auxiliaire pour générer les vues ;
- `config` les fichiers de configuration ;
- `db` les fichiers liés à la base de données ;
- `lib` le code indépendant du MVC ;
- `log` les fichiers de log ;
- `script` les scripts de démarrage du serveur et de la console ;
- `test` les fichiers de test ;
- `vendor` le code fourni par des fournisseurs externes.

Seul le code appartenant aux répertoires `app` et `lib` doit être testé.

1. Pour plus d'informations sur les différents motifs de programmation reliés au MVC, jeter un coup d'oeil à la page suivante écrite par M. Fowler : `http://www.martinfowler.com/eaaDev/uiArchs.html`

Description fonctionnelle de l'application

L'application est censée permettre de créer des groupes d'utilisateurs de l'ENSEIRB. Elle contient des utilisateurs qui sont soit des élèves, soit des enseignants. Pour se connecter à l'application, il faut s'authentifier, puis l'application permet d'accéder à la liste des élèves que l'on peut manipuler. Si l'on possède les droits (cf. `config/authorization_rules.rb`), il est possible de créer de nouveaux groupes, d'ajouter de nouveaux utilisateurs et d'obtenir les adresses mail correspondantes.

Lancement du serveur

Dans le répertoire `script`, il existe plusieurs scripts Ruby permettant de gérer l'application :

- script / rails server −e <env> : lance le serveur sur l'adresse `http://localhost:3000`, en utilisant l'environnement décrit par env ;
- script / rails console <env> : lance une console de diagnostic sur laquelle il est possible d'exécuter des commandes Ruby, en utilisant l'environnement décrit par env.

Les différents environnements disponibles sont test, development et production.
La base de développement est chargée avec les personnes du groupe de TD, et possède un utilisateur `root` dont le mot de passe est `root`, avec le rôle `manager`.
La base de test est chargée avec les *fixtures* placées dans `test/fixtures`.

Description des modèles

• User :	représente un utilisateur de l'application, qui possède un certain nombre de rôles possibles. Les utilisateurs peuvent être des Student ou des Teacher, qui sont tous stockés dans la même table en utilisant le principe de la *single table inheritance*.
• Student :	représente les élèves utilisateurs.
• Teacher :	représente les enseignants utilisateurs.
• AuthSource :	représente les sources d'authentification possibles des utilisateurs (pour l'instant uniquement les serveurs LDAP).
• Role :	représente les rôles possible d'un utilisateur.
• Group :	représente des groupes de Student.
• GroupMembership :	modèle jointure (en terme de base de données) entre Group et Student.

Utilisation de la console

A l'intérieur de la console, il est possible de lancer n'importe quelle commande Ruby valide. La console permet ainsi de tester le contenu courant de la base de données, et d'exécuter

certaines fonctions (principalement celles présentes dans les modèles). Voici quelques exemples d'utilisations :

- Liste des élèves

```
Student. all
```

- Source d'authentification d'id 1

```
AuthSource.find(1)
```

- Liste des groupes par id et nom

```
Group. all . collect  { |g|  [g.id ,  g.name] }
```

- Liste des GroupMemberships correspondant au groupe d'id 1

```
GroupMembership.all. select  { |gm| gm.group.id == 1 }
Group. find (1). group_memberships
```

Références

[Bur92] S. Burbeck. Application programming in smalltalk-80 : How to use model-view-controller. *University of Illinois in Urbana-Champaign (UIUC) Smalltalk Archive*, 1992. Original article : 1987. Available online at `http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html`.

[EGV94] R. Johnson E. Gamma, R. Helm and J. M. Vlissides. *Design Patterns : Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.