Test/spécification et framework MVC

À tester en isolation (tests unitaires)

- routage
- modèle
- contrôleur
- vue

Le fonctionnement global est également à tester : tests d'intégration

Exemples avec Rails

Tous les exemples seront données avec Rails, en utilisant la syntaxe rspec.

Rails fournit un environnement séparé pour les tests, avec en particulier une base de données remise à zéro avant chaque test.

rspec-rails est utilisé pour l'intégration de rspec au framework :

```
gem install rspec-rails
rails generate rspec:install # génère la configuration nécessaire
# conf : spec/spec_helpers.rb
# rake spec : joue tous les tests
```

Les générateurs model/controller/view créent automatiquement la spécification associée.

```
rails g model Post title:string body:text
# generate spec/model/post_spec.rb
```

Modèles

spécification basique

Rails

Les spécifications sont dans `spec/models`, utilisez de préférence le générateur rails pour créer un modèle, cela vous créera automatiquement la spécification associée.

```
rails g model ModelName attribute1:type attribute2:type
```

Tester la validation

1. partir d'un modèle valide
2. changer un attribut pour rendre le modèle invalide
3. `model.should_not be_valid`

Routage

- Entrée du test : méthode HTTP, PATH, query string
- Test sur la sortie : quel est le code activé

Exemple en français

La requête HTTP `GET /people/:id` doit être traitée par la méthode `show` du contrôleur `people_controller`.

Exemple rails

dans `spec/routing` :

```
1 describe "routing to profiles" do
2   it "routes /profile/:username to profile#show for username" do
3     { :get => "/profiles/jsmith" }.should route_to(
4       :controller => "profiles",
5       :action => "show",
6       :username => "jsmith"
7     )
8   end
```

```
 9
10   it "does not expose a list of profiles" do
11     { :get => "/profiles" }.should_not be_routable
12   end
13 end
```

Contrôleur :

- Entrée : appel d'une méthode d'un contrôleur
- Entrée : paramètres provenant de la requête HTTP
- Entrée : état de la session
- Sortie : variables assignées pour le rendu des vues
- Sortie : vue rendue, redirection HTTP
- Interaction : avec la couche modèle

Test d'un contrôleur dans Rails avec Rspec

dans spec/controllers

```
get|post|put|delete route_path, params
assigns(:widgets).should eq(expected_value) # teste que le controlleur a affecté @widgets
response.should redirect_to(path)
response.should render_template(template_name)
```

Vues

- Entrée : un template (fichier)
- Entrée : des variables (nom et valeur associée)
- Sortie : une chaîne de caractère, souvent du HTML
- Interaction : utilisation d'une vue partielle

Tester le contenu de la chaîne résultat :

- pattern matching
- parcourir le document comme un document HTML/XML

Test d'une vue dans Rails avec Rspec

dans spec/views

```
stub_model
assign(:key, value)
render # @key is set
rendered.should contain("contenu attendu")
view.should render_template(:partial => "_event", :count => 2)
```

Test d'intégration

Le but d'un test d'intégration est de simuler l'interaction d'un utilisateur avec l'application. Ce n'est pas un test unitaire, il va mettre en jeu les 3 couches MVC de l'application.

- Entrée : état de la couche de persistence
- Entrée : état du navigateur client/état de la session
- Entrée : interactions utilisateurs (interaction avec un navigateur)
- Test sur la sortie : résultat renvoyé dans le navigateur de l'utilisateur

Test d'intégration dans Rails avec rspec

dans `spec/requests`

```
rails g integration_test
```

Test qui traverse les 3 couches MVC : pas un test unitaire. On simule un navigateur complet.

Exemple avec Capybara pour la simulation du navigateur :

```ruby
describe "home page" do
  it "diplays the user's username after successful login" do
    user = User.create(:username => "jdoe", :password => "secret")
    visit "/login"
    fill_in "Username", :with => "jdoe"
    fill_in "Password", :with => "secret"
    click_buton "Log in"

    page.should have_selector(".header .username", :content => "jdoe")
  end
end
```

Méthodologie

BDD pour développer une application web

1. Conception avec le client sur papier du lot n. Le point de départ est un schéma du résultat visuel voulu par le client.
2. écriture du test d'intégration pour une des fonctionnalités décrites en 1 avec le client.
3. On descend dans le code : routage, controlleur, vue, modèle : test d'abord, code minimal ensuite. On itère jusqu'à ce que le test d'intégration passe.
4. On recommence en 2 avec une autre fonctionnalité
5. Lot n+1 : on recommence en 1

Tests d'intégration : simuler l'utilisation d'un navigateur

Capybara

Instalation de Capybara

```
sudo apt-get install libxml2 libxml2-dev libxslt1-dev # pour nokogiri
```

Gemfile

```
group :test, :development do
  gem 'rspec-rails'
  gem 'capybara'
end

bundle install
```

Configuration : spec/spec_helper.rb

```
require 'capybara/rspec'
```

Capybara : utilisation

À utiliser dans les tests d'intégration (`spec/requests/`)

Navigating

```
visit articles_path
visit('/projects')
visit(post_comments_path(post))
```

Clicking links and buttons

```
click 'Link Text'
click_button('Save')
click_link
```

Interacting with forms

```
fill_in('First Name', :with => 'John')
fill_in('Password', :with => 'Seekrit')
fill_in('Description', :with => 'Really Long Text...')
choose('A Radio Button')
```

```
check('A Checkbox')
uncheck('A Checkbox')
attach_file('Image', '/path/to/image.jpg')
select('Option', :from => 'Select Box')
```

Capybara : utilisation

Querying

```
page.has_selector?('table tr')
page.has_selector?(:xpath, '//table/tr')
page.has_no_selector?(:content)

page.has_xpath?('//table/tr')
page.has_css?('table tr.foo')
page.has_content?('foo')
page.has_link?('link test')
```

conjointement à Rspec :

```
page.should have_selector('table tr')
page.should have_selector(:xpath, '//table/tr')
page.should have_no_selector(:content)

page.should have_xpath('//table/tr')
page.should have_css('table tr.foo')
page.should have_content('foo')
page.should have_no_content('foo')
page.should have_link('link text')
```

current_path

`current_path` renvoie le chemin courant. Permet de tester où on arrive après avoir cliqué sur quelquechose dans une page.