Pratique d'un cycle de développement

- rails (routage, contrôleur et vue)
- cycle BDD : scénario global

1. spécification d'un scénario utilisateur : dans `spec/requests`
2. Rajout dès que nécessaise d'un élément MVC avec sa spécification.
3. `app/models/xxx.rb => spec/models/xxx_spec.rb`
4. `app/controllers/xxx_controller.rb => spec/controllers/xxx_controller_spec.rb`
5. `app/views/xxx/yyy.html.erb => spec/views/xxx/yyy.html.erb_spec.rb`

Mise en place d'un projet rails

Avec utilisation de :

- `rspec` : pour les spécifications
- `capybara` : pour simuler un navigateur dans les tests d'intégration, et disposer de machers efficaces pour les spécifications des vues
- `guard` : pour automatiser la vérification des tests/spécifications

création du projet, sans utilisation de test-unit, sans installation des librairies

```
rails new blog --skip-test-unit --skip-bundle
```

Spécification des librairies du projet : Gemfile

```
gem 'therubyracer'

# :development pour profiter des tâches rake
group :test, :development do
  gem 'rspec-rails'
  gem 'capybara'
  gem 'guard-rspec'
end

```

Installation des librairies :

```
bundle install # créé Gemfile.lock, à versionner
```

Initialisation de la configuration rspec :

```
rails g rspec:install
```

Utilisation de `capybara` : dans `spec/spec_helper.rb`, rajoutez :

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

Commit git initial

```
git init .
git add .
git commit -m "projet créé"
```

Chaque commit doit être sain : les test doivent passer.

1er scénario : liste des posts

1. En français : en visitant `/posts` ou `/` avec mon navigateur, je dois voir la liste des posts.
2. Au tableau : on définit approximativement le résultat attendu
3. Codage de ce scénario : création d'un test d'intégration

```
rails g integration_test listing_posts
```

`spec/requests/listing_posts_spec.rb`

```
1  require 'spec_helper'
2
3  describe "ListingPosts" do
4    describe "GET /posts" do
5      before(:each) do
6        @post1 = Post.create(:title => "sujet1", :body => "bla bla")
7        @post2 = Post.create(:title => "sujet2", :body => "bla bla")
8      end
9
10     describe "GET /posts" do
11       it "generates a listing of posts" do
12         visit posts_path
13         page.should have_content @post1.title
14         page.should have_content @post2.title
15       end
16     end
17   end
18 end
```

Lancement de la spécification :

```
rspec spec/requests/listing_posts_spec.rb # OU
guard -c # voir plus loin la configuration de guard
```

On va développer jusqu'à ce que cette spécification soit validée.

Gestion des erreurs de type : symbole inconnu

La spécification du scénario signale alors que `Post` est inconnu : on génère le modèle.

```
rails g model Post title:string body:text
  invoke  active_record
  create    db/migrate/20120315095932_create_posts.rb
  create    app/models/post.rb
  invoke    rspec
  create      spec/models/post_spec.rb
```

Pensez à appliquer les migrations dans l'environnement de développement (par défaut) ET dans l'environnement de test :

```
rake db:migrate               # => db/development.sqlite3
RAILS_ENV=test rake db:migrate # => db/test.sqlite3
```

Nous n'avons de spécification à faire à ce stade sur le modèle.

Gestion des erreurs de type : symbole inconnu

La spécification du scénario signale alors que `posts_path` est inconnu :

Il faut spécifier et coder la route générant cet alias.

Le routage est lié à un contrôleur. La route voulue ici :

```
posts GET /posts  posts#index
```

Spécification de la route :

```
 mkdir spec/routing
```

`spec/routing/posts_routing_spec.rb`

```
1 require 'spec_helper'
2 describe PostsController do
3   it "routes to #index" do
4     get('/posts').should route_to("posts#index")
5   end
6
7   it "should provide the aliast post_path for /posts" do
8     posts_path.should == '/posts'
9   end
10 end
```

Erreurs successives sur `spec/routing/posts_routing_spec.rb` :

1. `PostsController` inconnu : `rails g controller Posts`
2. `No route matches "/posts"` : `match '/posts' => 'posts#index'` dans `config/routes`
3. `undefined posts_path` : rajouter `:as => 'posts'` à la route précédente.

Remarques :

- Visualiser la route générée avec : `rake routes`.
- plutôt qu'une route générale, préférez utiliser des ressources REST, éventuellement en restreignant les accès :

  ressources :posts, :only => [:index]

Gestion des erreurs de type : symbole inconnu

Le routage est ok, on revient au scénario d'intégration :

`The action 'index' could not be found for PostsController` : on rajoute la définition de la méthode `index` dans `app/controllers/posts_controller.rb`

```
1 class PostsController < ApplicationController
2   def index
3   end
4 end
```

`Missing template posts/index` : on créé la vue en utilisant le générateur du contrôleur :

```
rails g controller Posts index -s # -s : skip existing files
```

Enlevez la route rajoutée par le générateur.

Spécifier et coder le résultat

L'erreur finale porte (enfin !) sur le contenu

```
expected there to be content "sujet1" in "Blog\n\nPosts#index\nFind me in app/views/posts/index.html.erb\n\n\n"
```

Il faut maintenant spécifier le contrôleur et la vue.

Spécification de l'action index du contrôleur :

`spec/controllers/posts_controller_spec.rb`

```
require 'spec_helper'
describe PostsController do
  describe "GET 'index'" do
    before(:each) do
      @posts = [stub_model(Post,:title => "1"), stub_model(Post, :title => "2")]
      Post.stub(:all){ @posts }
    end
    it "assigns a list of posts" do
      Post.should_receive(:all).and_return(@posts)
      get 'index'
      assigns(:posts).should eq @posts
      response.should be_success
    end

    it "renders the template list" do
      get 'index'
      response.should render_template(:index)
    end
  end
end
```

Remarques :

- `assigns` après l'appel de l'action, pour tester les variables assignées
- `should_receive` avant l'appel de l'action, pour tester les interactions avec la couche métier
- `should render_template` ou `should redirect_to` après l'appel de l'action
- `stub_model` : véritable instance du modèle, non sauvés dans la base néanmoins
- `mock_model` : double qui se comporte comme une instance d'`ActiveModel`, sans en être une

Implémentation de `PostsController#index` :

```
  def index
    @posts = Post.all
  end

```

Spécification de la vue

`spec/views/post/index.html.erb_spec.rb`

```
require 'spec_helper'
describe "posts/index.html.erb" do
  it "displays all the posts" do
    assign(:posts, [
           stub_model(Post, :title => "sujet 1"),
           stub_model(Post, :title => "sujet 2")
         ])
    render
    rendered.should =~ /sujet 1/
    rendered.should =~ /sujet 2/
  end
end
```

Remarques :

- `assign` : définit les variables disponibles dans le template, à utiliser avant `render`
- `render` : calcul du template
- `rendererd`: résultat
- normalement : décrire finement la vue, en particulier : sémantique HTML/CSS à utiliser

Implémentation de la vue :

```
<h1>Posts#index</h1>

<ul>
<% @posts.each do |post| %>
  <li><%= post.title %></li>
<% end %>
</ul>

```

Il reste à rajouter que le chemin par défaut `/` route aussi vers `/posts`. À faire en exercice.

Le scénario d'intégration passe ! On commite.

Scénario 2 : création d'un post

En français :

- À partir de la page qui liste les posts, je clique sur le lien `Créer un nouveau Post`.
- Un formulaire me permet alors de rentrer un nouveau post. Quand je clique sur le bouton `Créer`, je me retrouve de nouveau sur la liste des posts, et le nouveau post apparaît alors dans la liste.

Reprenez les étapes précédents, et spécifiez et implémentez ce scénario.

1. Distinguez bien les étapes à spécifier dans le scénario d'intégration.
2. Faites une spécification par étape : chaque bloc `it` ne doit comporter qu'une ou deux expectations.
3. Utilisez des routes REST !

Utilisation de Guard

création du Guardfile

guard init rspec

```
guard 'rspec', :version => 2 do
  watch(%r{^spec/.+_spec\.rb$})
  watch(%r{^lib/(.+)\.rb$})     { |m| "spec/lib/#{m[1]}_spec.rb" }
  watch('spec/spec_helper.rb')  { "spec" }

  # Rails example
  watch(%r{^app/(.+)\.rb$})                           { |m| "spec/#{m[1]}_spec.rb" }
  watch(%r{^app/(.*)(\.erb|\.haml)$})                 { |m| "spec/#{m[1]}#{m[2]}_spec.rb" }
  watch(%r{^app/controllers/(.+)_(controller)\.rb$})  { |m| ["spec/routing/#{m[1]}_routing_spec.rb", "spec/#{m[2]}s/#{m[1]}_#{m[2]}_spec.rb", "spec/acceptance/#{m[1]}_spec.rb"] }
  watch(%r{^spec/support/(.+)\.rb$})                  { "spec" }
  watch('config/routes.rb')                           { "spec/routing" }
  watch('app/controllers/application_controller.rb')  { "spec/controllers" }
  # Capybara request specs
  watch(%r{^app/views/(.+)/.*\.(erb|haml)$})          { |m| "spec/requests/#{m[1]}_spec.rb" }
end
```

Utilisation

guard -c # -c : efface l'écran à chaque fois