Programowanie w Ruby

Wykład 13

Marcin Młotkowski

21 stycznia 2019

Plan wykładu

Testowanie

Rodzaje testów:

- Testy jednostkowe
- Testy funkcjonalne
- Testy integracyjne
- Testy wydajnościowe (profilowanie)

Dobra wiadomość

Generatory modeli i kontrolerów generują też szkielety testów.

Dobra wiadomość

Generatory modeli i kontrolerów generują też szkielety testów.

```
../test/
```

Testy jednostkowe

Testy sprawdzające pojedyncze funkcjonalności (metodę, funkcję etc.).

Środowisko testowe

- Testy są metodami podklasy klasy ActiveSupport::TestCase, o nazwie test_* (przypadki testowe)
- Klasa jest katalogu test/unit/
- Uruchomienie testów: najprościej ruby test/unit/...rb.

Testy w Railsach

Railsy oferują

- Automatycznie wygenerowane pliki z prototypami testów
- Bazę danych do testów
- Wsparcie tworzenia zmiennych tymczasowych

Przykład

Należy zajrzeć do notatek z wykładu 6

Testowanie modeli

W modelach możemy definiować walidację, tj. warunki poprawności danych sprawdzane podczas wprowadzania danych

Czego potrzebujemy

- Testowej bazy danych
- Testowych danych
- Zestawu testów
- Asercji

Przygotowanie testowej bazy danych

Zajrzenie do config/database.yml

Przygotowanie testowej bazy danych

Zajrzenie do config/database.yml

Wygenerowanie odpowiednich tabel w bazie:

```
rake db:test:prepare
```

Przygotowanie danych testowych

- Dane są w katalogu test/fixtures
- Dane są w plikach typu CSV lub YAML
- Pliki z danymi powinne się nazywać tak jak modele

Przykład danych testowych dla modelu

wyklads.yml

```
ruby:
  id: 1
  title: Kurs języka Ruby
ldi:
  id: 1
  title: Logika dla informatyków
```

Walidacja modeli

Przypomnienie

```
class Wyklad < ApplicationRecord
    protected
    def validate
        errors.add(:punkty, 'punkty powinne być dodatnie')
                if punkty.nil? || punkty < 0.01
    end
end
```

Zestaw testów

```
class WykladTest < ActiveSupport::TestCase
    fixtures :wyklads

end
```

Zestaw testów

```
class WykladTest < ActiveSupport::TestCase
    fixtures :wyklads

    def test_prosty
        w = Wyklad.new(:title => "AiSD")
        w.ects = -1
        assert !w.valid?
    end
end
```

Korzystanie z infrastruktury testowej

- Deklaracja fixtures :wyklads łączy dane z pliku fixtures/wyklads.yml z tabelą w bazie danych
- Przed każdym wywołaniem metody testowej tabela jest wypełniana danymi z pliku
- Po wykonaniu metody testowej dane są usuwane z tej tabeli

Przykład

```
class WykladTest < ActiveSupport::TestCase
    fixtures :wyklady

    def test_prosty
        w = Wyklad.new(:title => "AiSD")
        w.ects = -1
        assert !w.valid?
    end

    def test_zlozony
        w = Wyklad.new(:title => wyklady(:ruby).title)
        assert !w.save
    end

end
```

Inne asercje

assert_valid(obiekt_activerecord)

sprawdzenie, czy obiekt przechodzi walidację

Inne asercje

assert_valid(obiekt_activerecord)

sprawdzenie, czy obiekt przechodzi walidację

flunk(komunikat)

zawsze kończy się niepowodzeniem

Architektura testów kontrolerów

- Szkielety testów kontrolerów są w katalogu tests/functional
- Testy są zgromadzone w metodach klasy testowej
- Opcjonalna metoda setup wykonywana przed testami

Pola obiektu klasy testującej

- @controller = WykladyController.new
- @request = ActionController::TestRequest.new
- @response = ActionController::TestResponse.new

Metody i funkcje pomocnicze

- get :akcja, opcje_wywołania, sesja
- post :akcja, :opcje_wywołania

Asercje

- assert_response :odpowiedź
- assert_redirected_to :action => "akcja"

Przykłady

Przykład prosty

```
def test_index
    get :index
    assert_response :success # assert_response 200
end
```

Przykłady

Przykład prosty

```
def test_index
    get :index
    assert_response :success # assert_response 200
end
```

Przykład złożony

```
def test_logowania
    get :index
    assert_redirected_to :login
    assert_equal "Niezalogowany", flash[:notice]
end
```

Testowanie edycji

```
post :edit :wyklad => { :title => "Ruby", :ects => 1}
```

Wynik działania akcji (@response)

- Komunikat o powodzeniu/niepowodzeniu/...
- Strona w html'u

Weryfikacja html'a

assert_select *tag* *tekst*

Weryfikacja html'a

assert_select *tag* *tekst*

Np:

```
assert_select "title", "Programowanie w Ruby"
```

Testy integracyjne w RoR

- Kontrola przepływu informacji przez aplikację
- Testy polegają na testowaniu scenariuszy
- tests/integration
- Generowanie szkieletu
  $ bin/rails generate integration_test pierwszak

Przykładowy scenariusz (przypadek użycia)

- Student wchodzi na stronę i jest przekierowany na stronę logowania
- Student loguje się
- Jeśli sukces to jest przekierowany na stronę główną
- Student wybiera dwa przedmioty
- Student jest zapisany dwa przedmioty
- Student się wylogowuje

Implementacja scenariusza

Wstęp

```
Zajęcia.delete_all
ruby = wyklad(:ruby)
ldi = wyklad(:ldi)
```

Wejście na stronę

```
get "/wyklady/index"
assert_redirect_to "/login/haslo"
post_via_redirect "/login/validate" :user =>
    { :name => "pierwszak", :passwd => "*******" }
assert_template "wyklady/index"
assert_response :success
assert_equal session[:user_id], Student(:pierwszak).id
```

Wybranie danych

```
get "/wyklady/" + wyklad(:ruby).id + "/dodaj/"
assert_template "index"
get "/wyklady/" + wyklad(:ldi).id + "/dodaj/"
assert_equal 2, session[:zajecia].dane.size
```

Uruchamianie poleceniem rake

- wszystkie testy: `rake test`
- testy kontrolerów: `rake test:controllers`
- testy jednostkowe: `rake test:units`

Plan wykładu

Minitest

Alternatywa do test-unit

- łatwiejsze asercje;
- testowanie przez specyfikację;
- benchmarking;
- mock objects;
- kolorowanie wyników testów.

Testy

```
require 'test_helper'

class MyprojectTest < Minitest::Test
    def test_that_it_has_a_version_number
        refute_nil ::Myproject::VERSION
    end

    def test_it_does_something_useful
        assert false
    end
end
```

Spec–style testing

```
require 'test_helper'

describe "My Project" do
    it "has a version number" do
        value(::Myproject::VERSION).wont_be_nil
    end

    it "does something useful" do
        value(4 + 4).must_equal 8
    end
end
```

Cucumber

Środowisko testowe dedykowane *testom akceptacyjnym*.
Własny język opisu przypadkow użycia Gherkin

Przykład przypadku użycia

Feature: Pobranie pieniędzy z automatu
Użytkownik posiadający konto w banku chce pobrać pieniądze z bankomatu.
Scenario: Eryk chce wyciągnąć pieniądze z bankomatu
Given Eryk ma ważną kartę debetową lub kredytową
And jego stan konta jest 100 zł
When wsunie kartę
And wskaże 45 zł
Then bankomat powinien wydać 45 zł
And stan konta Eryka zmienia się na 55 zł

Capybara

Część środowiska Cucumber dedykowana do testów aplikacji WWW

```
describe 'UserRegistration' do
  it 'allows a user to register' do
    visit new_user_registration_path
    fill_in 'First name', :with => 'New'
    fill_in 'Last name', :with => 'User'
    fill_in 'Email', :with => 'newuser@example.com'
    fill_in 'Password', :with => 'userpassword'
    fill_in 'Password Confirmation', :with => 'userpassw
    click_button 'Register'
    page.should have_content 'Welcome'
  end
end
```