TESTAREA APLICATIILOR RUBY ON RAILS CU RSPEC

Cezar Hălmăgean

Lucrez cu Ruby on Rails & RSpec din 2007

Articole pe SemaphoreCI, Ruby Weekly, Blog (mixandgo.com/blog)

Consultanță Ruby on Rails

@chalmagean

mixandgo.com

Idei Principale

- **Teste de funcționalitate**

  Orice funcționalitate nouă, începe cu un test de funcționalitate

- **Teste de integrare**

  Cazurile mai puțin evidente sau vizibile sunt acoperite cu teste de integrare

- **Teste unitare**

  Intră în detaliu și testează toate cazurile posibile

De ce să testezi?

- **Nevoile de business se schimbă constant**
  Funcționalitate nouă vs. funcționalitate existentă

- **Documentație**
  Proiectele mici devin mari, uităm repede

- **Upgrade/refactoring mai ușor**
  Rails, Ruby, librării

- **Mai puține bug-uri**
  Lucrezi mai relaxat, la ceva nou

De ce RSpec?

- **Ușor de citit**

  TestUnit/MiniTest vs. RSpec

```ruby
def test_will_return_nil
  result = calculate_result
  assert_nil result
end
```

```ruby
it 'returns nil'
  result = calculate_result
  expect(result).to be_nil
end
```

- **Popularitate**

  RSpec a devenit foarte popular în 2008

- **Comunitate**

  E mai ușor să găsești exemple

Cele 3 tipuri de teste

- **Teste de funcționalitate (UI)**
  Testează doar ce poți vedea

- **Teste de integrare (request-uri)**
  Testează doar ce nu poți vedea

- **Teste unitare (modele, logica de business)**
  Testează izolat și cazuri speciale

Alte tipuri de teste

Routes

Helpers

Views

Controllers

System

Jobs

Email

Mutation

Testele de funcționalitate

- **Testează doar ce poți vedea**
  Formulare, butoane, filtrare, etc.

- **În colaborare cu clientul**
  Ajută la stabilirea obiectivului/cerințelor

- **Durează destul de mult**
  Scrie cât de puține poți și acoperă cât mai mult

- **Două feluri de a le scrie**
  Capybara și Cucumber

Capybara

- **Folosește un limbaj mai neprietenos**

  E ușor de înțeles de programatori

```
RSpec.feature 'Homepage', type: :feature do
  scenario 'displays a welcome message' do
    visit '/'
    expect(page).to have_content 'Welcome'
  end
end
```

- **Poate folosi un browser sau poate rula headless**

  Selenium, poltergeist, capybara-webkit, rack-test

- **Știe să aștepte după evenimente asincron**

  Popup-uri, ajax, etc.

Cucumber

- **Folosește un limbaj prietenos (Gherkin)**

  E ușor de înțeles de non-tehnici

```
Feature: Homepage

  Scenario: Visitor lands on the homepage
    When I go to the homepage
    Then I should see a welcome message
```

- **Te forțează să schimbi perspectiva**

  Să te pui în locul clientului (proprietarul afacerii)

- **Poate folosi ca documentație**

  E mai ușor de citit chiar și de programatori

Cucumber

- **Se instalează separat de RSpec**

  În test environment

```
group :test do
  gem 'cucumber-rails', require: false
  gem 'database_cleaner' # multiple threads
end
```

```
$ rails generate cucumber:install
```

- **În spate folosește Capybara**

  Este opțional dar îmbunătățește colaborarea

Cucumber

- **Exemple pași cucumber**

  Folosește regex ca să găsească pașii

```
When(/^I go to the homepage$/) do
  visit root_path
end

Then(/^I should see a welcome message$/) do
  expect(page).to have_content 'Welcome'
end
```

Introduction to Writing Acceptance Tests with Cucumber

bit.ly/codecamp_cucumber

Testele de integrare

- **Testează doar ce NU poți vedea**
  Email-uri, baze de date, API-uri, etc.

- **Confirmă că unitățile merg bine împreună**
  Că se respectă contractele între obiecte

- **Durează puțin mai mult ca testele unitare**
  E ok să ai mai multe verificări într-un singur test

- **Nu trebuie să fie DRY**
  Au rol de documentație

Testele de integrare

- **Exemplu test de integrare**

  Nu folosește Capybara

```ruby
RSpec.describe 'Homepage', type: :request do
  describe 'GET /' do
    it 'renders the homepage' do
      get root_path
      expect(response.body).to match('Welcome')
    end
  end
end
```

Testele unitare

- **Acoperă cât mai multe cazuri**
  Doar unitatea testată, nu și colaboratorii

- **Acoperă cazurile mai rar întâlnite**
  Fuzz(ing) / property-based

- **Sunt destul de rapide**
  Pot fi mai rapide dacă le separi de Rails

- **Două feluri de a testa modelele**
  Cu AR /models și fără AR /lib

Testele unitare (recomandări)

- **O singură verificare / test**
  Ca să identifici mai ușor sursa erorii

- **Nu testa implementarea**
  Doar rezultatul

- **Prea multe mock-uri**
  Indică o problemă de design

- **Sincronizează mock-urile cu Verifying Doubles**
  bit.ly/codecamp_vd

Concluzie

- **Teste de funcționalitate**
  Testează doar ce POȚI vedea

- **Teste de integrare**
  Testează doar ce NU POȚI vedea

- **Teste unitare**
  Testează izolat și cazuri speciale

DEMO

TESTAREA APLICAȚIILOR RUBY ON RAILS CU RSPEC

Cezar Hălmăgean

@chalmagean

mixandgo.com

github.com/chalmagean/codecamp