TESTING RUBY+RAILS APPLICATIONS

Marco Console
console@dis.uniroma1.it

LABORATORIO

SCENARIO

- **Vogliamo testare una applicazione ruby+rails**
  - Test di Accettazione
  - Test di Integrazione
  - Test Funzionali
  - Test Unitari

- Le funzionalità delle applicazioni sono spesso protette
  - Autenticazione
  - Autorizzazione

- Come implementiamo i test?

PROBLEMI

- Test di accettazione e integrazione
  - I nostri test dovranno necessariamente autenticarsi
  - Come facciamo ad autenticarci con Cucumber+Capybara?

- Test funzionali
  - I test funzionali coinvolgono spesso i controller.
  - Le funzionalità dei controller sono protette
  - Come facciamo ad autenticarci con RSpec?

- Test unitari
  - Si concentrano (spesso) su modelli e classi separate da Rails
  - Solitamente non avremo bisogno di autenticare un utente
  - Possiamo usare le tecniche viste per il testing dei controller

CARATTERISTICHE DEI TEST

- Test FIRST:
  - **Fast**: veloci da eseguire.
    - I test potrebbero essere lanciati tantissime volte.
  - **Independent**: i test sono indipendenti tra di loro.
    - Potremmo voler lanciare solo un sottoinsieme dei test alla volta.
  - **Repeatable**: il risultato dei test dipende solo dal codice esercitato.
    - Per isolare più chiaramente i bug.
  - **Self-Checking**: il codice dei test può valutarne il risultato in autonomia.
    - La presenza di uno sviluppatore rallenterebbe il processo.
  - **Timely**: i test vanno creati prima del codice.
    - Il codice scritto in questa maniera tende ad essere più chiaro.

CUCUMBER+CAPYBARA

ESEMPIO

- Pe oggi partiamo dalla seguente repository
  - **http://gitlab.com/console.marco/spoiled-potatoes.git**
  - **`Branch: feature_canard-auth`**

GEMME

- Per i test che stiamo per vedere utilizzeremo

- `gem 'capybara', '>= 3.26'`
- `gem 'selenium-webdriver'`
- `gem 'webdrivers'`
- `gem 'rexml'`
- `gem 'database_cleaner'`
- `gem 'cucumber-rails'`

WEBPACKER

- Dobbiamo installare **`webpacker`** nell'environment di test

```
RAILS_ENV=test rails webpacker:install
```

- Questa operazione potrebbe essere necessaria ogni volta che fate clone di una applicazione ruby+rails da git

```
rails webpacker:install
```

ANATOMIA DEI TEST

- Per autenticarci con Cucumber+Capybara possiamo usare direttamente gli URL relativi all'autenticazione

- Con quale utente?
  - I test vengono avviate in un environment separato (test)
  - Se ci fossero utenti nell'applicazione non verrebbero trovati nei test
  - Sarebbe comunque una cattiva pratica di testing (FIRST)

- I passi da eseguire sono i seguenti
  1. **Precondizione: Creare l'utente**
  2. **Precondizione: Autenticare l'utente**
  3. **Eseguire il resto del test**

ESEMPIO

```
Scenario: View Number of Reviews per Movie
   Given I am on the home page
   Then I should see table heading "Number of Reviews"
   Then I should see table heading "Average Score"
```

```
Scenario: View Number of Reviews per Movie
   Given I am authenticated
   Given I am on the home page
   Then I should see table heading "Number of Reviews"
   Then I should see table heading "Average Score"
```

STEP DEFINITION

```
Given(/^I am authenticated$/) do
    u = User.create!(:email=>"banned@spoiledpotatoes.com",
                     :password => "password",
                     :roles_mask => 0 )
    visit "/"
    fill_in "Email", with: "banned@spoiledpotatoes.com"
    fill_in "Password", with: "password"
    click_button "Log in"
    expect(page).to have_text("Logged in as
   banned@spoiledpotatoes.com")
end
```

AUTORIZZAZIONE

- L’utente che abbiamo create è in tutto e per tutto un utente
  - **Soggetto ad autenticazione**
  - **Soggetto ad autorizzazione**

- Per testare funzionalità protette da autorizzazione
  1. **`Precondizione: Creare un utente autorizzato`**
  2. **`Precondizione: Autenticare l’utente`**
  3. **`Eseguire il resto del test`**

ESEMPIO

```
Scenario: Moviegoers cannot edit movies
   Given Movie "TEST_MOVIE" ... is there
   Given I am authenticated as moviegoer
   Given I am on the home page
   Then I should not see link "Edit"
```

```
Scenario: Admins can edit movies
   Given Movie "TEST_MOVIE" ... is there
   Given I am authenticated as admin
   Given I am on the home page
   Then I should see link "Edit"
```

STEP DEFINITION

```
Then(/^I should see link "(.*?)"$/) do |arg1|
   expect(page).to have_link(text:/\A#{arg1}\Z/)
end
```

```
Then(/^I should not see link "(.*?)"$/) do |arg1|
   expect(page).not_to have_link(text:/\A#{arg1}\Z/)
end
```

STEP DEFINITION

```
Given(/^Given I am authenticated as admin$/) do
    u = User.create!(:email =>"admin@spoiledpotatoes.com",
                     :password => "password",
                     :roles_mask => 3)
    visit "/"
    fill_in "Email", with: "admin@spoiledpotatoes.com"
    fill_in "Password", with: "password"
    click_button "Log in"
end
```

STEP DEFINITION

```
Given(/^Given I am authenticated as moviegoer$/) do
    u = User.create!(:email =>"mvg@spoiledpotatoes.com",
                     :password => “password",
                        :roles_mask => 1)
    visit "/"
    fill_in "Email", with: " mvg @spoiledpotatoes.com"
    fill_in "Password", with: " password "
    click_button "Log in"
end
```

RSPEC

GEMME

- Per i test che stiamo per vedere utilizzeremo

- `gem 'rspec-rails'`
- `gem 'rails-controller-testing'`

TEST DEI CONTROLLER

- I test funzionali coincidono spesso col test dei controller
  - **Definiscono le singole funzionalità dell'applicazione**
- Per testare funzionalità dei controller usiamo RSpec
  - Dobbiamo importare le funzionalità per i controller
  - Dobbiamo importare le funzionalità per devise
- La lista delle condizioni utilizzabili è a questo link
  - **https://relishapp.com/rspec/rspec-rails/v/5-0/docs/controller-specs**

SETTING UP – HELPERS

- In **`spec/rails_helper`** aggiungiamo

```
 config.include Devise::Test::ControllerHelpers, type:
:controller
```

- Definiamo i nostri test dichiarandoli per controller

```
describe <ControllerClass>, type: :controller do
   ...
end
```

SETTING UP -- CLEANER

- Dobbiamo pulire la base di dati prima e dopo i test
- Creiamo il seguente file in **spec/support/dbcleaner.rb**

```
RSpec.configure do |config|
  config.before(:suite) do
    DatabaseCleaner.clean_with :truncation
  end
  config.after(:suite) do
    DatabaseCleaner.clean_with :truncation
  end
end
```

- Aggiungiamo il seguente in **spec/rails_helper.rb**

```
require 'support/dbcleaner.rb'
```

ESEMPIO -- FIXTURES

- `spec/fixtures/users.yml`

```
banned:
  email: banned@spoiledpotatoes.com
  roles_mask: 0
```

- `spec/fixtures/movies.yml`

```
one:
  title: Blade Runner
  director: Ridley Scott
  year: 1982
```

ESEMPIO -- GENERALE

```
require 'rails_helper.rb'

describe MoviesController, type: :controller do

      fixtures :all
      it "Should give me three movies from fixtures" do
         adm = users(:admin)
         sign_in adm
         get :index
         expect(assigns(:movies).size).to eql(3)
      end
end
```

ESEMPIO -- AUTORIZZAZIONE

- Testiamo i privilegi di un utente bannato

```
  #Bannned
  context "with Banned privileges" do
     before :each do
        banned = users(:banned)
        sign_in banned
     end
...
   end
```

ESEMPIO -- AUTORIZZAZIONE

- Create

```
      #Create
      it "should not create movies" do
         params = {:movie=>{:title => "Title", :director =>
"Director", :year => 1900}}
         get :create, :params => params
         m_tst = Movie.where(:title => "Title")
         expect(m_tst).to be_empty
      end
```

ESEMPIO -- AUTORIZZAZIONE

- Retrieve

```
    #Retrieve
    it "should retrieve movies" do
       mv = movies(:one)
       params = {:id => mv.id}
       get :show, :params => params
       expect(assigns(:movie)).to eql(mv)
    end
```

ESEMPIO -- AUTORIZZAZIONE

- Update

```
    #Update
    it "should not update movies" do
       mv = movies(:one)
       params = {:id => mv.id, :movie=>{:title =>
"Title"}}
       get :update, :params => params
       m_tst = Movie.find(mv.id)
       expect(m_tst.title).to eql(mv.title)
    end
```

ESEMPIO -- AUTORIZZAZIONE

- Destroy

```
    #Destroy
    it "should not destroy movies" do
       mv = movies(:one)
       params = {:id => mv.id}
       get :destroy, :params => params
       m_tst = Movie.where(:id => mv.id)
       expect(m_tst).not_to be_empty
    end
```

ESERCIZIO 1

- Implementare con cucumber e capybara i test per i seguenti scenari
  - **Un moviegoer non eliminare film**
  - **Un admin può eliminare film**

ESERCIZIO 2

- Implementare con rspect i seguenti test
  - **Funzionalità CRUD su Movie per utenti Moviegoer**
  - **Funzionalità CRUD su Movie per utenti Admin**

SOLUZIONI

- Una soluzione si trova nella seguente repository
  - **http://gitlab.com/console.marco/spoiled-potatoes.git**
  - **`Branch: feature_canard-auth-tests`**