Este documento descreve o processo de testes, ferramente utilizada e comprova a utilização de ferramentas para os testes.

Giovanni M Guidini - 16/0122660 Gabriel Bessa - 16/0120811

Descrição do Processo de Testes

Para os testes decidimos utilizar testes automatizados para testes unitários e testes manuais para integração. A integração foi testada por subsistema (i.e. subsistema de autenticação, subsistema de cadastramento de eventos, etc). O teste de sistema envolve todas as funcionalidades so sistema.

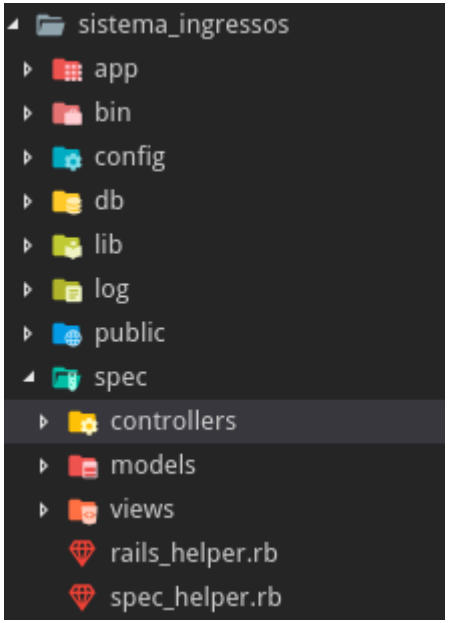
Como "unidade" utilizamos Models, Views e Controllers (os componentes do MVC). Isto é, cada Controller, Model e View foi testado individualmente através de testes unitários automatizados. Nesses testes foi utilizado o framework Rspec. Esse framework é voltado para desenvolvimento utilizando estilo TDD ou BDD, e acreditamos ter sido muito apropriado no desenvolvimento do sistema.

Nós utilizamos uma abordagem híbrida para integração do sistema, que foi feita por subsistemas relacionados, conforme comentado. Essa abordagem consistiu em integrar as partes relacionadas de um subsistema (M, V e C) e testá-los. Ao final desse processo todo o sistema pode ser integrado e o Smoke test realizado. Passado o smoke test podem ser realizados os testes de sistema mais rigorosos.

Além disso foram realizados testes estáticos dos códigos, utilizando a técnica *walkthrough*, e com o auxílio de ferramentas automáticas para detecção de erros sintáticos/semânticos (extensão Ruby para VSCode). Os outros documentos (de gerenciamento) foram revisados pela equipe.

Comprovação de Testes automatizados

Os testes automatizados foram os de unidade. Para esses testes foi utilizado o framework Rspec. Os arquivos de testes estão na pasta `spec` do projeto (ver figura abaixo). Note que existe uma pasta para os arquivos de teste dos Controllers, uma para Models e uma para Views.



Exemplos de casos de testes criados estão abaixo, para um controller e um model, respectivamente:

```ruby
describe "PUT #update" do
  context "with valid params" do
    let(:new_attributes){
      {
        street: "Rua do Toddy",
        neighborhood: "Quebrada do toddynho",
        number: 20
      }
    }

    it "updates the requested address" do
      address = Address.create! valid_attributes
      put :update, params: {id: address.to_param, address: new_attributes}
      address.reload
      expect(address.street).to eq("Rua do Toddy")
    end

    it "redirects to index" do
      address = Address.create! valid_attributes
      put :update, params: {id: address.to_param, address: valid_attributes}
      expect(response).to redirect_to(address_url)
    end
  end
```

```ruby
require 'rails_helper'

RSpec.describe Address, type: :model do

  it 'is valid with valid attributes' do
    ad = Address.new
    ad.street = "Rua Dom Bosco"
    ad.neighborhood = "Quebrada Dom Bosco"
    ad.number = 10
    expect(ad).to be_valid
  end

  it 'is not valid with nil attributes' do
    ad = Address.new
    ad.street = nil
    ad.neighborhood = nil
    ad.number = nil
    expect(ad).to_not be_valid      You, 2 days ago • Adicionando testes as models e 3 c
  end

end
```

Finalmente, os casos de teste podem ser executados:

```
→ sistema_ingressos git:(master) ✗ rspec
.....................................................................................................................................
.....................

Finished in 1.43 seconds (files took 1.07 seconds to load)
170 examples, 0 failures
```