Relatório - Teste de Proficiência (iS) Internet Sistemas – Desenvolvimento de Testes

1. **Introdução**

Existem vários frameworks para desenvolvimento de testes em Rails. Optei por utilizar o RSpec em sua última versão, que é o default da última versão do Rails.

No entanto, saberia também desenvolver testes para Rails utilizando os *asserts* clássicos.

Os arquivos de testes se encontram no diretório *spec* na raíz do projeto.

Foram desenvolvidos no total, 43 casos de teste; que não foram suficentes para atingir 100% de cobertura do código mesmo em um teste de proficiência, mostrando mais uma vez que desenvolvimento de testes é uma área extensa, complexa e de extrema importância no desenvolvimento e posteriormente na manutenção de sistemas.

Para executar a bateria de testes, navegue por linha de comando até a raíz do projeto e execute o comando *rspec*. Recomendo também um *drop* de ambos os *schemas* (bancos de dados de desenvolvimento e teste) e execução do comando *rake db:create db:migrate db:reset*.

2. **Testes de Unidade (Unit Tests)**

Foram desenvolvidos testes de unidade para todos os Modelos e Controladores.

O framework RSpec é bem interessante pois fica fácil de focar no comportamento esperado dos modelos, que já haviam sido definidos anteriormente. Porém, com a implementação dos testes foi possível reforçar ainda mais algumas validações nos modelos.

3. **Testes de Integração (Integration Tests)**

Por questões de tempo, consegui implementar apenas um teste de integração que se encontra no arquivo de teste do Student Controller. Mas ficou claro para mim como poderia extender e implementar outros casos de teste de integração, onde seria possível também validar as regras de negócio definidas anteriormente, principalmente em relação as atualizações de status (ver relatório anterior – Regras de Negócio).

4. **Dificuldades Encontradas**

A primeira foi decidir qual framework para desenvolvimento de testes utilizar. Na última versão do Rails, o diretório *tests* sequer é criado, como aponta a documentação oficial.

Assim que optei por utilizar o RSpec, houve uma curva de aprendizado para utilizá-lo em sua última versão – e com certeza ainda há bastante a ser aprendido. Muito material na Internet indica o uso dos operadores *should* ao invés do *expect*, que é o recomendado atualmente.

Consegui bastante ajuda no site http://betterspecs.org/ e na documentação oficial do RSpec; não tive muitas dificuldades em desenvolver os testes de unidade para os Modelos. Mas por falta de exemplos, tive mais dificuldades com os testes de unidade para Controladores e de integração. No entanto, ao final do prazo desta segunda etapa do teste de proficiência, estava começando a me soltar mais com ambos.

Por fim, o tempo também foi uma dificuldade. Não pelo Rails em si, mas hoje especialmente tive dificuldades de concentração comparado à primeira etapa onde fiz um bom uso das 24 horas fornecidas para a realização do teste. Acabei me dispersando em alguns momentos e não desenvolvi o quanto eu esperava, não menosprezando os aprendizados que obtive durante a realização desta etapa.

5. **Conclusões**

Mesmo não obtendo 100% de cobertura de código, foi possível melhorar a qualidade do sistema com o desenvolvimento de testes, que detectaram possíveis erros de validação e

reforçaram a corretude dos comportamentos esperados por Modelos e Controladores. Não cheguei a desenvolver testes complexos de integração e de unidade em relação aos controladores, mas ao fim da prática estava começando a entender melhor como fazê-los no Ruby On Rails, e sinto que seria capaz de extender estes testes para validar as regras de negócio que envolvem diversos Modelos desenvovlidos anterioremente.

Por fim, o desenvolvimento de testes em RoR não é muito diferente do que já realizei em outros frameworks MVC, e sinto que fiquei aquém do que eu mesmo esperava nesta etapa por ter tido problemas em gerenciar o meu próprio tempo durante a realização da prática.

Geralmente costumo dar prioridade aos testes de integração. Na minha opinião, apesar de serem mais lentos costumam ser mais efetivos que os testes de unidade no que diz respeito em garantir corretude do sistema como um todo. E como desenvolver testes demanda tempo e cuidado, prefiro concentrar os esforços nos testes de integração. Porém, como em Rails ainda sou novo no desenvolvimento de testes, optei por implementar primeiro os de unidade para ganhar mais experiência na prática.

6. Referências

Documentação Ruby On Rails:
http://guides.rubyonrails.org/testing.html
http://api.rubyonrails.org/classes/ActionDispatch/IntegrationTest.html

Documentação RSpec:
https://github.com/rspec/rspec-rails
https://www.relishapp.com/rspec/
https://www.relishapp.com/rspec/rspec-expectations/v/3-1/docs/built-in-matchers
https://www.relishapp.com/rspec/rspec-expectations/v/3-1/docs/built-in-matchers/be-matchers
https://www.relishapp.com/rspec/rspec-expectations/v/3-1/docs/built-in-matchers/predicate-matchers
https://www.relishapp.com/rspec/rspec-expectations/v/3-1/docs/built-in-matchers/equality-matchers
https://www.relishapp.com/rspec/rspec-expectations/v/3-1/docs/built-in-matchers/comparison-matchers
https://www.relishapp.com/rspec/rspec-expectations/v/3-1/docs/built-in-matchers/type-matchers
https://www.relishapp.com/rspec/rspec-rails/v/3-1/docs/request-specs/request-spec

Better Specs:
http://betterspecs.org/

Outros:
https://leanpub.com/everydayrailsrspec/read
http://everydayrails.com/2012/03/19/testing-series-rspec-models-factory-girl.html
http://everydayrails.com/2012/04/07/testing-series-rspec-controllers.html
http://everydayrails.com/2012/04/24/testing-series-rspec-requests.html

Leituras interessantes sobre desenvolvimento de testes:
http://david.heinemeierhansson.com/2014/tdd-is-dead-long-live-testing.html
http://www.rbcs-us.com/documents/Why-Most-Unit-Testing-is-Waste.pdf