Relatório - Teste de Proficiência (iS) Internet Sistemas

1. **Plataforma Computacional e execução do Projeto Esqueleto**

Para o desenvolvimento do sistema teste, foi utilizado um computador com sistema operacional Manjaro Linux (distribuição de Arch Linux), com o Ruby na versão 2.1.2p95 (2014-05-08 revision 45877) e Rails na versão 4.1.4; para codificação foi utilizado o programa Sublime Text 3.

Ao descompactar o projeto, tentei executá-lo com o comando *rails server*. Houveram erros de dependência e foi indicado a execução do comando *bundle install*, que também resultou em um erro pois não havia o PostgreSQL instalado.

Nunca havia usado tal banco de dados antes, mas não tive muitos problemas em instalá-lo e configurá-lo de acordo com as informações contidas no arquivo *database.yml*.

Feito isto, foi possível inicializar o banco de dados e executar o comando *rails server* com sucesso. Também utilizei o programa pgAdmin III para auxiliar a visualização do banco de dados, uma interface gráfica para o PostgreSQL.

O teste realizado também encontra-se no Github: https://github.com/gabrieljt/is-test

2. **Modelo Relacional e Arquitetura MVC**

Comecei o teste implementando um CRUD simples para os modelos *Student* e *Course*, seguindo os modelos indicados. Para os atributos *status* do tipo inteiro, utilizei Enums.

Foram implementados também validações nos modelos, e para a classe *Student*, um trigger foi criado para gerar o atributo *register_number*, que é único e aleatório.

Feito isto, dei início a implementação dos relacionamentos para que fosse possível vincular um estudante a um curso. A tabela intermediária *classrooms* foi criada através de uma relação *many-to-many* / *through*, que pode ser vista nos arquivos *course.rb*, *student.rb* e *classroom.rb*.

Não houve necessidade de criar um controlador para o modelo *Classroom*, de modo que durante a criação ou edição de um *Student* ou *Course* é possível realizar os vínculos exijidos pelo sistema atavés de um seletor múltiplo.

O recurso *classrooms* foi aninhado aos recursos *student* e *course* no arquivo *routes.rb*.

Num primeiro momento foi criado o atributo *entry_at* para o modelo *Classroom*, mas gerei posteriormente um arquivo de migração para remover esta coluna pois optei por utilizar o atributo padrão do Rails *created_at*. Durante a criação deste mesmo modelo, esqueci de adicionar as relações no comando *rails generate model*, e também tive que criar arquivos de migração para gerar tais relações.

As tabelas no banco de dados ficaram de acordo com o modelo relacional especificado.

3. **Regras de Negócio**

Fiz uso dos Enums para implementar simples regras de negócio, simulando melhor uma aplicação real.

O curso pode possuir 3 status: *enrolling, ongoing, closed;* e o estudante 2: *idle* e *enrolled.*

Durante a criação ou edição de um *Course,* só é possível vincular ou desvincular estudantes caso o curso esteja aberto para inscrições (*enrolling*) ou esteja em andamento (*ongoing*).

No caso de criação ou edição de um estudante, apenas os cursos abertos para inscrições aparecem no seletor múltiplo. Quanto ao *status* do estudante, também foram criados triggers para implementar uma nova regra de negócio: caso o estudante esteja vinculado a um curso *enrolling* ou *ongoing*, seu *status* é atualizado para *enrolled*. Caso não esteja matriculado em

nenhum curso, ou apenas participou de cursos que já foram encerrados (*closed*), seu status é alterado para *idle*.

O arquivo *seeds.rb* foi alimentado gerando exemplos que demonstram diversos casos.

Seria possível expandir ainda mais a aplicação tornando-a mais adequada ao mundo real caso fossem utilizados mecanismos de autenticação e papéis de usuário; mas a complexidade iria aumentar e fiquei inseguro quanto ao tempo.

4. Usabilidade, layout e validações client side

Até o momento presente, apenas o mínimo necessário de usabilidade foi feito em relação a estes itens. O desenvolvimento do back-end acabou tomando bastante tempo, uma vez que tenho pouca experiência com o Ruby On Rails e tive que aprender muitos conceitos do framework para poder concluir o teste.

Tenho conhecimentos em Javascript e JQuery, e já implementei validações client side em outros sistemas. Tenho noções de usabilidade e já li um livro interessante sobre o assunto, “Não me faça pensar”, de Steve Krug. No entanto, não creio que eu seja a pessoa mais indicada para este tipo de desenvolvimento, apesar de ser importante para qualquer desenvolvedor ter estas questões em mente.

Quanto ao HTML e ao CSS, sei apenas o básico e nunca tive interesse em me aprofundar, até mesmo pelo fato de comumente trabalhar com bons web designers que podem focar no front-end.

5. Conclusões

Foi possível implementar o requisito principal do teste seguindo boas práticas de desenvolvimento.

A modelagem ficou de acordo com o especificado, regras de negócios foram atribuídas aos modelos quando possível evitando duplicação de código nos controladores, e o uso de *partials* nas views também evitou duplicações. Foi possível fazer bom uso das rotas que implementam o protocolo REST através do aninhamento de recursos. O uso de triggers auxiliou na consistência da informação, removendo dependências inclusive quando necessário. Foram seguidas as convenções adotadas pelo framework facilitando a automatização de algumas tarefas, como por exemplo a captura de parâmetros nos controladores e criação de novos objetos.

Ruby On Rails chamou mais ainda minha atenção; é um framework leve e intuitivo, com excelente documentação e comunidade ativa. As únicas desvantagens que pude notar até então em relação ao Grails é o esquema de migração de banco de dados, que parece ser mais trabalhoso de manter (no Grails a migração é totalmente automática a partir apenas dos modelos especificados), e o procedimento de deploy, que apesar de não ter feito em produção parece ser mais trabalhoso também (difícil ser mais simples que o Tomcat e um arquivo WAR).

Me interessei também pelo PostgreSQL, que parece ser mais seguro e robusto que o MySQL.

Gostaria também de parabenizar a iS pelo teste proposto. Achei uma excelente e inovadora maneira de avaliar os candidatos. Foi uma grande oportunidade para aprender mais sobre RoR, e em apenas 24 horas sinto que meu conhecimento foi de superficial para básico neste framework. Pude por em prática a criação de modelos, controladores e views, associações entre modelos, aninhamento de atributos, migrações de banco de dados, diversos tipos de buscas no banco de dados, validações, triggers, rotas entre outros conceitos presentes nos principais frameworks de desenvolvimento web; sinto que já seria possível construir vários tipos de aplicações utilizando RoR.

6. **Referências**

Documentação Ruby On Rails:
http://guides.rubyonrails.org/getting_started.html
http://guides.rubyonrails.org/association_basics.html
http://guides.rubyonrails.org/active_record_basics.html
http://guides.rubyonrails.org/active_record_validations.html
http://guides.rubyonrails.org/routing.html
http://edgeapi.rubyonrails.org/classes/ActiveRecord/Enum.html
http://guides.rubyonrails.org/migrations.html
http://api.rubyonrails.org/classes/ActiveRecord/NestedAttributes/ClassMethods.html
http://guides.rubyonrails.org/v2.3.11/active_record_querying.html
http://api.rubyonrails.org/classes/ActiveRecord/FinderMethods.html
http://api.rubyonrails.org/classes/ActiveRecord/Callbacks.html
http://api.rubyonrails.org/classes/ActiveRecord/ConnectionAdapters/TableDefinition.html

Diversos blogs da Internet e tópicos no Stack Overflow.