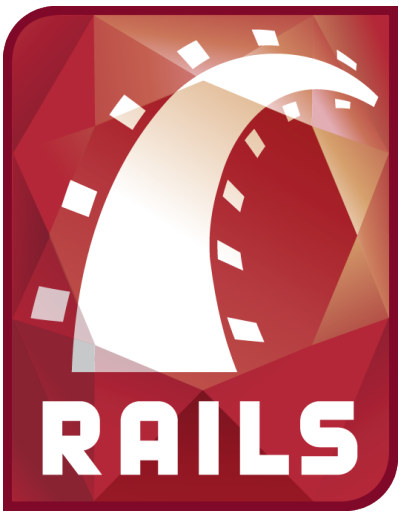
Monografia
Ruby on Rails



Alunos:
Alexandre Oki Takinami
Bruno Henrique Yoshimura
Marcelo de Rezende Martins
Márcio Vinicius dos Santos

1) Framework Rails

Ruby on Rails é um "framework de desenvolvimento web gratuito focado na produtividade e felicidade do programador", utilizando-se da extensibilidade da linguagem Ruby e da padronização das configurações.

Analisando as tecnologias existentes para desenvolver uma aplicação web, poderíamos escolher o **PHP** pela simplicidade e rapidez do desenvimento inicial. Mas no final, poderíamos ter problemas na metade do desenvolvimento ao descobrir que a falta de convenções acaba duplicando trabalho. Por outro lado, uma aplicação em **Java** nos obrigaria a fazer muitas escolhas prematuras sobre qual padrões ou tecnologias utilizar.

Ilustração que mostra a quantidade de tecnologias que o Rails engloba do Java

O Ruby on Rails mistura a simplicidade e agilidade da programação do PHP com a robustes do Java, eliminando a necessidade de decisões importantes no início do projeto (pois já estão convencionadas). Além disso, oferece uma ferramenta (*scaffold*) que constrói uma base completa para o projeto em apenas alguns minutos.

Don't Repeat Yourself

Por ser focado na felicidade do programador, o Rails tem grande preocupação em evitar trabalhos repetitivos. Aplicações Web dinâmicas são em boa parte baseadas em operações CRUD (Create, Read, Update e Delete). Muitas vezes o processo de criação dos modelos, visualizações e controladores é repetitivo, pois as entidades são muito parecidas fisicamente e as operações sobre elas são parecidas. Para evitar essa repetição, o Rails possui o comando *scaffold* que gera uma base MVC completa para entidades com as operações CRUD.

Do ponto de vista de duplicação de código, o Rails permite a implementação de blocos reusáveis, chamados de *helpers*. Além de poder utilizar helpers da comunidade, também é possível desenvolver novos.

Convention Over Configuration

A solução eficiente que o Rails implementa para evitar trabalhos duplicados e facilitar a vida do programador é a convenção das configurações dos programas. Por padrão, o Rails já entende o óbvio: na definição dos modelos, por exemplo, você não precisa falar que existe uma chave primária "id" e que a tabela do banco de dados que representa o modelo *Post* vai se chamar *posts*. Isso torna a programação mais agradável e deixa o código mais limpo, eliminando informações que na realidade são ruídos. Outra convenção importante é a utilização da filosofia REST, que é comprendida naturalmente pelo *dispatcher* do Rails.

Performance do Rails
Uma preocupação que os desenvolvedores têm ao adotar um framework Web é a performance das consulta ao banco de dados geradas automaticamente. Por possuirem muitos *joins* e por serem geradas automaticamentes, as consultas podem ser lerdas.

A primeira solução seria ajudar o Rails a utilizar o método find() de forma a adicionar parâmetros como Group By, Limit, Joins, eliminar relacionamentos em tempo de execução, etc. A segunda solução seria criar as consultas mais pesadas na mão, como em um sistema normal.

Mas o Rails já oferece duas soluções nativas para o problema
1) *Query Cache*: As consultas ao banco de dados podem ser cacheadas. Ao analisar o código SQL da consulta gerada, o Rails verifica se já existe um consulta no cache. Este cache é eliminado ao encontrar operações como INSERT, UPDATE, DELETE.
2) *Page Cache:* O Rails também oferece cache de páginas e blocos, que são guardados inteiros em uma pasta temporária.

Pela simplicidade e rapidez com que se podem ser configurado, as soluções de cache do Rails podem ter performance melhor do que otimizações comum de banco de dados. Em alguns casos, essa otimização pode ser complementada por índices de banco de dados e cache nativo.

Conclusão e Vantagens do Rails
Uma das principais vantagems do Rails é a rapidez com que pode ser desenvolvida a base de um projeto (e por isso o nome *scaffold* = Andaime). Por existirem padrões de desenvolvimento, não é necessário gastar tempo no início do projeto para escolher quais tecnologias usar. A única configuração inicial seria a do banco de dados.

Em poucas horas, o desenvolvedor pode gerar uma base MVC completa para um sistema Web, mesmo que utilize relacionamentos completos de banco de dados. Desta forma, sobra tempo para focar no mais importante: o projeto. Na prática, isso torna elimina o trabalho repetitivo, reduzindo a necessidade de desenvolvedores com baixo nível de experiência.

2) Como fazer um Blog em Ruby on Rails

Parte 1:
O objetivo é criar um blog que tenha post e comentários

Crie um projeto Rail com o nome blog
>rais blog

Edite o arquivo database.yml com as configurações do seu banco de dados

Excute o comando scaffold para gerar a estrutura MVC de um Post. Um post terá um título e um corpo.
>./script/generate scaffold Post title:string body:text

No arquivo model/post.rb adicione a validação do campo título.
validates_presence_of :title

Crie um banco da dados e execute o comando rake para gerar a tabela posts.
Rode o script para iniciar o servidor do rails.
>rake db:migrate
>./script/server

Teste sue post no caminho http://localhost:3000/posts

Crie a estrutura para um comentário
>./script/generate scaffolding Comment post_id:integer body:text

Gerar a tabela de comentários
>rake db:migrate

No Post Model (models/post.rb) adicionar
has_many :comments

No Comment Model (models/comment.rb) adicionar
belongs_to :post

Testes seus comentários
http://localhost:3000/comments

Parte 2

O objetivo da segunda parte é fazer o relacionamento post e comentário no formato REST. (/posts/1/commnents)

Edtiar o arquivo config/route.rb

- Adicionar: map.resources :post, :has_many => :comments
- Retirar: map.resources :comments
- Retirar: map.resources :post

Testar a relação Post e Comentários. Para isso abra o script console

```
>./script/console
>>Post.find(:all)
>>post = Post.find(:first)
>>c = Comment.new(:body => "Novo Comentario")
>>post.comments << c
>>post.save
>>quit
```

Relacionando o comentário ao post

Editar o arquivo controllers/comments_controller.rb

- find "Comment." replace "@post.comments."
- before_filter :load_post
- Criar o método para buscar o post do comentário

```
def load_post
  @post = Post.find(params[:post_id])
end
```

- Find ".new" Replace ".build"
- mudar o redirect_to(@comment) para redirect_to([@post, @comment])
- mudar o if @@post para @post.save

Em /views/comments/edit.html.erb do Comment View

- form_for([@post, @comment])
- Link show - ([@post, @comment])
- Link back - post_comments_path(@post)

No /views/comments/index.html.erb do Comment

- Link show - ([@post, comment])

- Link Destroy - ([@post, comment])
- De edit_comment_path => edit_post_comment_path(@post, comment)
- Em new = new_post_commenent_path(@post)

Nos arquivos /views/comments/new.html.erb

- form_for([@post, @comment])
- Link back - post_comments_path(@post)

No /views/comments/show.html.erb fazer

- edit_post_comment_path(@post, @comment)
- back : post_comments_path(@post)

No /views/posts/show.html.erb

```
<%= link_to 'Commnets', post_comments_path(@post) %>
<% unless @post.comments.empty? %>
<h3>Comentarios</h3>
<% @post.comments.each do |comment| %>
<p><%= h comment.body %></p>
<% end %>
<% end %>
```

Referências

Tutorial do Ruby iterativo - Aprenda ruby em 15 minutos
http://tryruby.hobix.com/

Aprenda Ruby em 20 minutos
http://www.ruby-lang.org/pt/documentacao/ruby-em-vinte-minutos

Ruby On Rails
http://rubyonrails.org/

Instruções de como instar o Rails
http://rubyonrails.org/download

Vídeo de como fazer um blog
http://www.vimeo.com/425800