Capítulo 6

Ruby on Rails

"Rails is the killer app for Ruby."
— Yukihiro Matsumoto, Criador da linguagem Ruby

Aqui faremos um mergulho no Rails e, em um único capítulo, teremos uma pequena aplicação pronta: com banco de dados, interface web, lógica de negócio e todo o necessário para um CRUD, para que nos capítulos posteriores possamos ver cada parte do Rails com detalhes e profundidade, criando uma aplicação bem mais completa.

6.1 - Ruby On Rails

David Heinemeier Hansson, conhecido pela comunidade como DHH, criou o Ruby on Rails para usar em um de seus projetos na *37signals*, o Basecamp. O Rails foi anunciado oficialmente em setembro de 2004 e em dezembro de 2005 chegou na versão 1.0. Em 2006, o framework começou a ganhar muita atenção da comunidade de desenvolvimento Web e passou a ser bastante utilizado desde então.

O Rails foi criado pensando na praticidade que ele proporcionaria na hora de escrever os aplicativos para Web. No Brasil a Caelum vem utilizando o framework desde 2007, e grandes empresas como Abril e Locaweb adotaram o framework em uma grande quantidade de projetos.

Outro atrativo do framework é que, comparado a outros, ele permite que as funcionalidades de um sistema possam ser implementadas de maneira incremental por conta de alguns padrões e conceitos adotados. Isso tornou o Rails uma das escolhas óbvias para projetos e empresas que adotam metodologias ágeis de desenvolvimento e gerenciamento de projeto.

Para saber mais sobre metodologias ágeis, a Caelum oferece os cursos

Gerenciamento ágil de projetos de Software com Scrum (PM-83) e *Práticas ágeis de desenvolvimento de Software* (PM-87).

Como pilares do Rails estão os conceitos de *Don't Repeat Yourself* (DRY), e *Convention over Configuration* (CoC).

O primeiro conceito nos incentiva a fazer bom uso da **reutilização de código**, que é também uma das principais vantagens da orientação a objetos. Além de podermos aproveitar as características de OO do Ruby, o próprio framework nos incentiva a adotar padrões de projeto mais adequados para essa finalidade.

O segundo conceito nos traz o benefício de poder escrever muito menos código para implementar uma determinada funcionalidade em nossa aplicação, desde que respeitemos alguns padrões de nome e localização de arquivos, nome de classes e métodos, entre outras regras simples e fáceis de serem memorizadas e seguidas.

É possível, porém, contornar as exceções com algumas linhas de código a mais. No geral nossas aplicações apresentam um código bem simples e enxuto por conta desse conceito.

Atualmente o Rails está em sua versão 4.0, lançada em junho de 2013, e continua sendo atualizado constantemente.

Versões do Rails

Histórico de todas as versões oficiais e datas de lançamento:

- 1.0 – 12/2005
- 1.2 – 01/2007
- 2.0 – 12/2007
- 2.1 – 06/2008
- 2.2 – 11/2008
- 2.3 – 03/2009
- 3.0 – 08/2010
- 3.1 – 08/2011

- 3.2 - 01/2012
- 4.0 - 06/2013

6.2 - MVC

A arquitetura principal do Rails é baseada no conceito de separar tudo em três camadas:

A parte responsável por apresentar os resultados na página web é chamado de Apresentação (**View**).

A camada que faz o trabalho de tratar os parâmetros da requisição e direcionar para as demais camadas é chamada de Controladora (**Controller**).

As classes que representam suas entidades e as que te ajudam a armazenar e buscar os dados são chamadas de Modelo (**Model**).

Esses três formam um padrão arquitetural chamado de **MVC**, ou **Model View Controller**. Ele pode sofrer variações de diversas maneiras. O MVC é um padrão arquitetural onde os limites entre os seus modelos, suas lógicas e suas visualizações são bem definidos, garantindo a separação de tarefas e tornando muito mais simples a manutenção do código de alguma parte, já que essas três partes se comunicam de maneira bem desacoplada.

Meta-Framework

Rails não é baseado num único padrão, mas sim um conjunto de padrões, alguns dos quais discutiremos durante o curso.

Outros frameworks que faziam parte do núcleo do Rails antigamente foram removidos desse núcleo para diminuir o acoplamento com ele e permitir que vocês os substitua sem dificuldade, mas continuam funcionando e sendo usados em conjunto. Aqui estão alguns deles:

- ActionMailer
- ActionPack
    - Action View

    - Action Controller
- ActiveRecord
- ActiveSupport

Projetos que usam o Ruby on Rails

Há uma extensa lista de aplicações que usam o Ruby on Rails e, entre elas estão o Twitter, YellowPages, Groupon, Typo (blog open source) e o Spokeo (ferramenta de agrupamento de sites de relacionamentos).

Uma lista de sites usando Ruby on Rails com sucesso pode ser encontrada nesses endereços:

- http://rubyonrails.org/applications
- http://www.rubyonrailsgallery.com
- http://www.opensourcerails.com

6.3 - Ambiente de Desenvolvimento

Por ser uma aplicação em Ruby -- contendo arquivos .rb e alguns arquivos de configuração diversos, todos baseados em texto puro -- o Ruby on Rails não requer nenhuma ferramenta avançada para que possamos criar aplicações. Utilizar um editor de textos simples ou uma IDE (*Integrated Development Environment*) cheia de recursos e atalhos é uma decisão de cada desenvolvedor.

Algumas IDEs podem trazer algumas facilidades como execução automática de testes, *syntax highlighting*, integração com diversas ferramentas, wizards, servidores, autocomplete, logs, perspectivas do banco de dados etc. Existem diversas IDEs para trabalhar com Rails, sendo as mais famosas:

- RubyMine - baseada no IntelliJ IDEA
- Aptana Studio 3 - antes conhecida como RadRails, disponível no modelo *stand-alone* ou como plug-in para Eclipse
- Ruby in Steel - para Visual Studio
- NetBeans

Segundo enquetes em listas de discussões, a maioria dos desenvolvedores Ruby on Rails não utiliza nenhuma IDE, apenas um bom **editor de texto** e um pouco de treino para deixá-lo confortável no uso do **terminal de comando** do sistema operacional é suficiente para ser bem produtivo e ainda por cima não depender de um Software grande e complexo para que você possa desenvolver.

Durante nosso curso iremos utilizar o editor de textos padrão do Ubuntu, o GEdit. Alguns outros editores são bem conhecidos na comunidade:

- TextMate - o editor preferido da comunidade Rails, somente para Mac;
- SublimeText 2 - poderoso e flexível, é compatível com plugins do TextMate. Versões para Windows, Mac OS e Linux;
- NotePad++ - famoso entre usuários de Windows, traz alguns recursos interessantes e é bem flexível;
- SciTE - editor simples, compatível com Windows, Mac e Linux;
- Vi / Vim - editor de textos poderoso, pode ser bem difícil para iniciantes. Compatível com Windows, Mac e Linux;
- Emacs - o todo poderoso editor do mundo Linux (também com versões para Windows e Mac);

6.4 - Criando um novo projeto Rails

Para utilizarmos o Rails, é necessário instalar a gem **rails** em nossa máquina através do comando `gem install`:

```
gem install rails
```

O comando acima, instalará a última versão estável do Rails. Caso queira, é possível também instalarmos uma versão específica passando-a conforme abaixo:

```
gem install rails -v=4.0.0
```

Após a instalação da gem **rails** em nossa máquina, ganhamos o comando que iremos utilizar para gerar os arquivos iniciais de nossa aplicação. Como a maioria dos comandos de terminal, podemos passar uma série de informações para que o projeto gerado seja customizado.

Para conhecer mais sobre as opções disponíveis, podemos imprimir a ajuda no console. Para isso basta executar o seguinte comando:

```
rails --help
```

Atualmente o Rails está na versão 4, que é a versão utilizada no curso. Caso queira saber a versão do Rails instalada na máquina, é só rodar o comando:

```
rails --version
```

O Rails já vem com inúmeros scripts prontos para tornar a vida do programador mais fácil criando tudo que for necessário para desempenhar uma determinada tarefa. Estes scripts são chamados de geradores e um destes é o gerador **new**, que proporciona a criação da estrutura base de uma aplicação Rails. Para usá-lo, é só digitar no terminal o seguinte comando:

```
rails new [caminho-da-app]
```

Caso a pasta apontada como caminho da aplicação não exista, ela será criada pelo gerador do Rails. Caso a pasta exista e contenha arquivos que possam conflitar com os gerados pelo Rails será necessário informar a ação adequada (sobrescrever, não sobrescrever) para cada um deles durante o processo de geração de arquivos.

O comando **new** suporta algumas opções a serem passadas no momento em que é chamado. Para visualizar todas as opções aceitas é só executar:

```
rails new -h
```

Dentre todas as opções, uma das mais úteis é a que prepara nossa aplicação para conexão com um banco de dados. Como o Rails é compatível com uma grande quantidade de bancos de dados podemos escolher aquele que temos mais familiaridade, ou um que já estiver instalado em nossa máquina. Por padrão o

Rails usa o SQLite3 pois é bem simples, versátil e consegue se comportar bem em um ambiente de desenvolvimento, além de possibilitar que as informações sejam salvas em qualquer local do sistema de arquivos (por padrão dentro da aplicação) e está disponível para Windows, Mac e Linux.

Caso queira utilizar, por exemplo, o MySQL, basta passar a opção:

```
rails new meu_projeto -d mysql
```

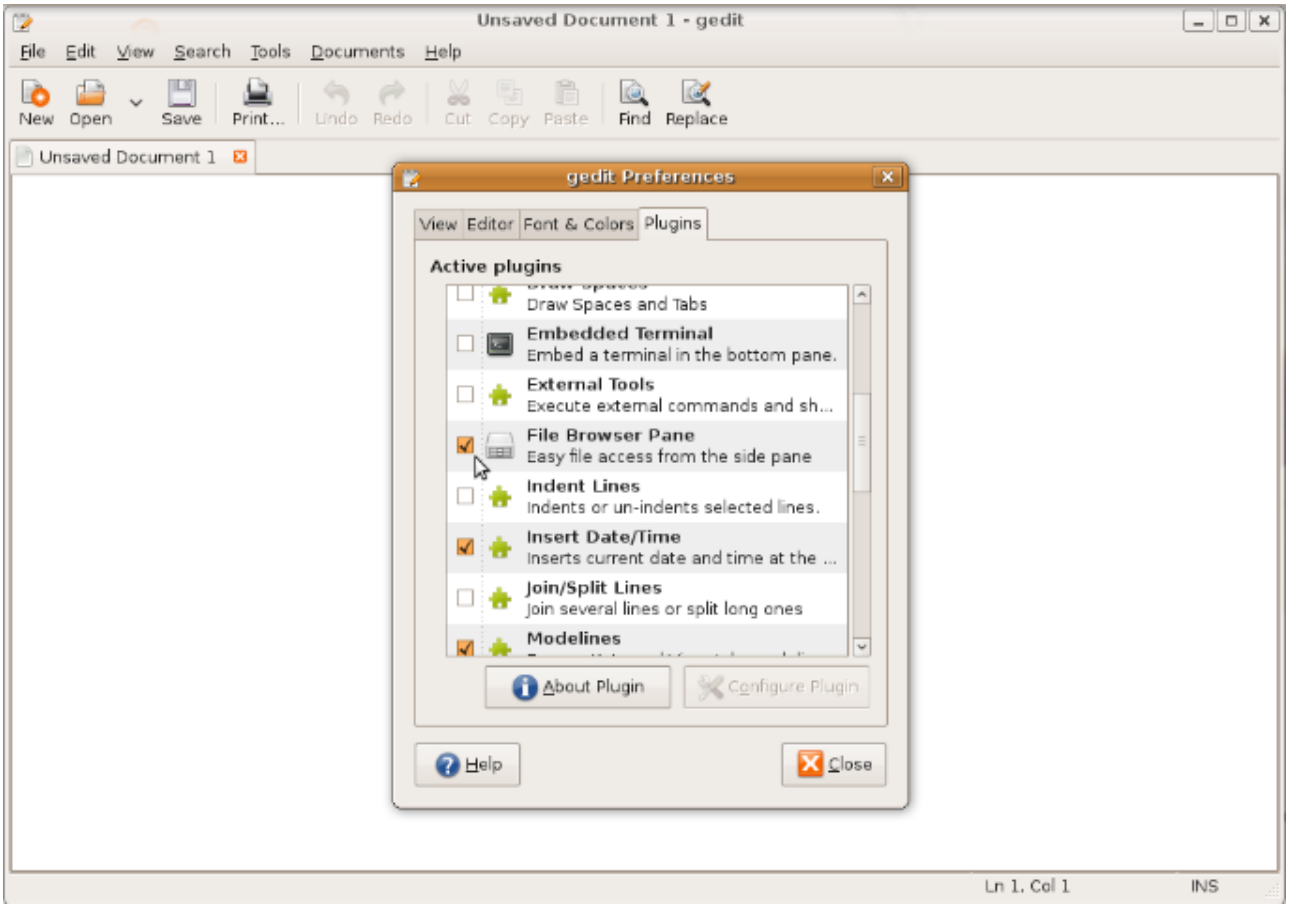
6.5 - Exercícios - Iniciando o projeto

1. Inicie o GEdit:

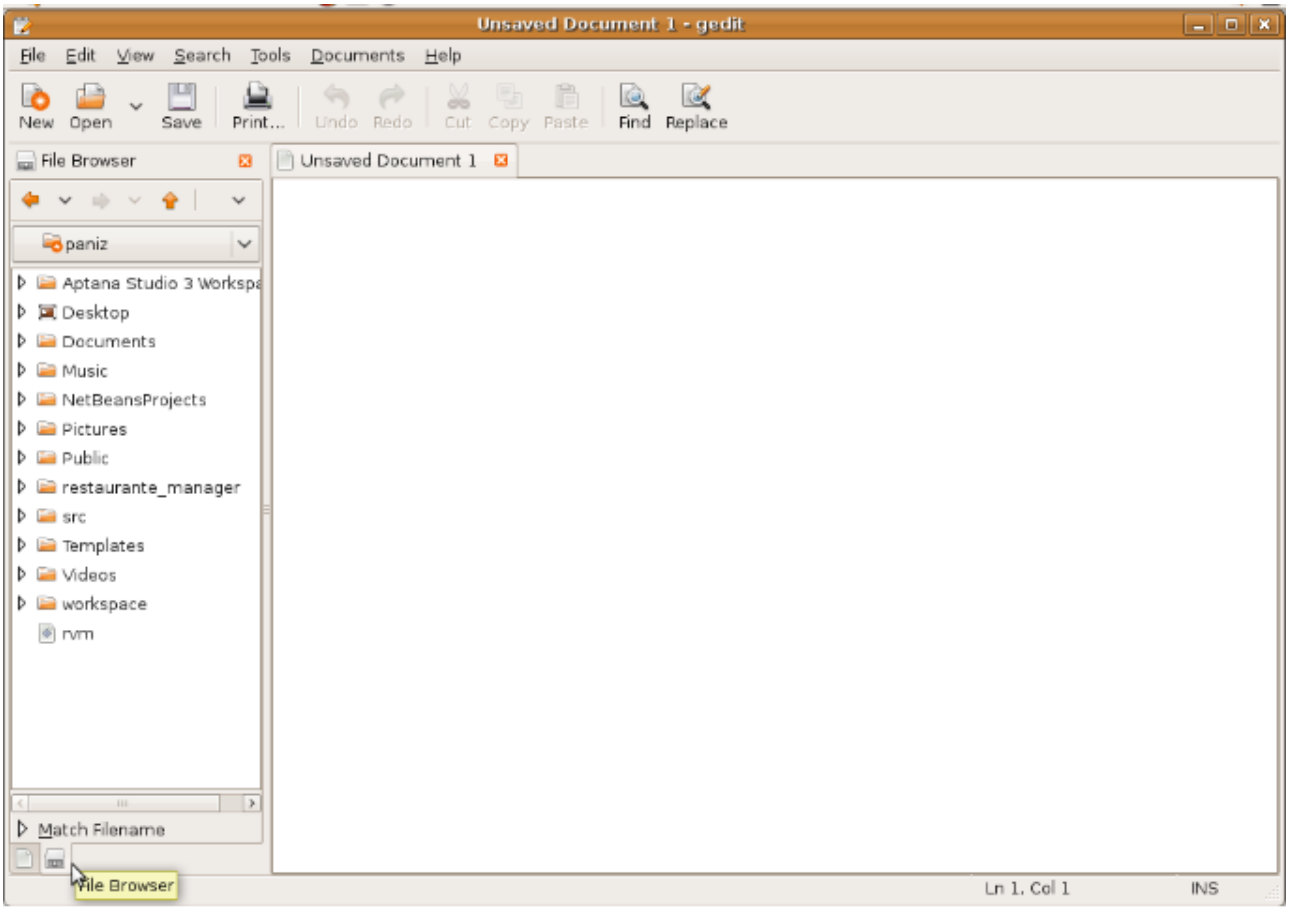
   a. Abra o GEdit através do link "Text Editor" no seu desktop ou barra de ferramentas

   b. Clique no menu **Edit** e escolher a opção **Preferences**

   c. Na nova janela clicar na aba **Plugins** e selecionar **File Browser Pane**



   d. Clicar no menu View e então selecionar **Side Pane**



2. Inicie um novo projeto:

   a. Abra o terminal

   b. Execute o seguinte comando

```
rails new agenda
```

O script informa no terminal quais arquivos foram criados para nossa aplicação e logo na sequência executa o comando para instalação das dependências (bundle install).

6.6 – Estrutura dos diretórios

Cada diretório tem uma função específica e bem clara na aplicação:

- **app** – A maioria dos arquivos específicos da nossa aplicação ficam aqui (inclusive todo o MVC, dividido em diretórios);
- **bin** – Executáveis do Rails e das gems instaladas;
- **config** – Configurações da aplicação;
- **db** – Migrações, esquema e outros arquivos relacionados ao banco de dados;
- **doc** – Documentação do sistema;
- **lib** – Bibliotecas auxiliares;
- **log** – Informações de log;
- **public** – Arquivos estáticos que serão servidos pela WEB;
- **test** – Testes da nossa aplicação;
- **tmp** – Arquivos temporários como cache e informações de sessões;
- **vendor** – Dependências e bibliotecas de terceiros.

6.7 – O Banco de dados

Uma das características do Rails é a facilidade de se comunicar com o banco de dados de modo transparente ao programador.

Um exemplo dessa facilidade é que ao gerar a aplicação, ele criou também um arquivo de configuração do banco, pronto para se conectar. O arquivo está localizado em **config/database.yml**. Como o banco de dados padrão é o SQLite3, a configuração foi gerada para ele. Nada impede, porém, a alteração manual dessa configuração para outros tipos de bancos.

Nesse arquivo, são configuradas três conexões diferentes: *development*, *test* e *production*, associados respectivamente aos bancos `agenda_development`, `agenda_test` e `agenda_production`.

O banco *agenda_development* é usado na fase de desenvolvimento da aplicação

e *agenda_production* em produção. A configuração para *agenda_test* se refere ao banco utilizado para os testes que serão executados, e deve ser mantido separado dos outros pois o framework apagará dados e tabelas constantemente.

Apesar de já estarem configuradas as conexões, o gerador da aplicação não cria os bancos de dados automaticamente. Existe um comando para isso:

```
rake db:create
```

A geração não é automática pois em alguns casos é possível que algum detalhe ou outro de configuração no arquivo **config/database.yml** seja necessário.

6.8 - Exercícios - Criando o banco de dados

1. ◦ Pelo terminal entre no diretório do projeto.

```
cd agenda
```

◦ Execute o script para criação dos bancos de dados.

```
rake db:create
```

Esse comando não deve exibir nenhuma saída no console, não se preocupe. Porém, se alguma mensagem for exibida, verifique se o comando foi digitado exatamente como demonstrado acima.

2. Verifique que as bases de desenvolvimento e teste foram criadas. Pelo explorador de arquivos do GEdit, verifique a pasta **db** da nossa aplicação. O *Rails* cria arquivos com a extensão **.sqlite3** para serem utilizados como base de dados pelo SQLite3, isso permite que você copie o banco de dados para outra máquina sem a necessidade de processos complexos de configuração e importação/exportação de dados.

6.9 - A base da construção: Scaffold

Pensando no propósito de facilitar a execução de tarefas repetitivas, o Rails traz um comando que é muito útil nos casos em que precisamos de um cadastro básico de alguma entidade em nossa aplicação -- um CRUD -- que se chama **scaffold**. *Scaffold* pode ser traduzido para o português como "andaime", ou seja, é uma base, um apoio que facilita a construção de nossa aplicação.

Com esse comando é gerado todo o código necessário para que possamos **listar todos os registros**, **exibir um registro**, **criar um novo registro**, **atualizar um registro** e **excluir um registro** de determinada entidade em nossa aplicação.

6.10 - Exercícios - Scaffold

1. Execute o scaffold do seu modelo "Evento":

   a. No terminal, na pasta da aplicação, execute o comando:

   ```
   rails generate scaffold Evento nome:string local:string inicio:datetime
   termino:datetime
   ```

   Veja os diversos arquivos que são criados.

   b. Note que também são gerados arquivos para criação da tabela no banco de dados, onde serão armazenados nossos "eventos", e um arquivo chamado **routes.rb** foi modificado.

   c. (opcional) Criamos atributos dos tipos *string*, *datetime*. Quais outros tipos são suportados? Abra a documentação do Rails (http://api.rubyonrails.org) e procure pela classe **ActiveRecord*ConnectionAdapters*Table**.

6.11 - Gerar as tabelas

Juntamente com nossos arquivos foi gerado um arquivo de migração, dentro da pasta **db/migrate**. No cenário mais comum, para cada entidade que criamos é necessário que exista uma tabela no banco de dados, para garantir esse padrão o Rails gera automaticamente um script em Ruby para que as informações no banco de dados sejam consistentes com o que declaramos na aplicação.

Quando executamos o script de *scaffold*, implicitamente declaramos que nossa aplicação tem uma entidade **Evento**, então o Rails gerou o arquivo

db/migrate/<timestamp>_create_eventos.rb (`<timestamp>` corresponde ao momento da criação do arquivo) com a definição da nossa tabela `eventos` e dos campos necessários.

Agora precisamos executar o script de geração das tabelas. Assim como temos um comando *rake* que cria o banco de dados, também temos um comando para criar as tabelas necessárias para nossa aplicação:

```
rake db:migrate
```

O Rails gera para todas as tabelas por padrão o campo `id` como chave primária. Toda a lógica de utilização dessa chave primária é tratada automaticamente para que o desenvolvedor possa se preocupar cada vez menos com o banco de dados e mais com os modelos.

Versão do banco de dados

Ao executarmos a migração do banco de dados pela primeira vez, será criada uma outra tabela chamada **schema_migrations**. Essa tabela contém uma única coluna (`version`), que indica quais foram as migrações executadas até agora, utilizando para isso o timestamp da última migration executada.

Database evolution

O Rails adota uma estratégia de evolução para o banco de dados, dessa maneira é possível a aplicação evolua gradativamente e o esquema do banco de dados seja incrementado com tabelas e campos em tabelas já existentes conforme o necessário.

Essa característica vai de encontro com as necessidades das metodologias ágeis de desenvolvimento de Software.

São Paulo, Rio de Janeiro e Brasília, além de turmas incompany.

Consulte as vantagens do curso *Desenv. Ágil para Web com Ruby on Rails*.

6.12 – Exercícios – Migrar tabela

1. Vamos executar o script de migração de banco de dados para que a tabela "eventos" seja criada:

a. Execute o comando pelo terminal, na pasta da nossa aplicação:

```
rake db:migrate
```

b. O Rails inclui uma maneira facilitada de conectarmos ao console interativo do banco de dados para que possamos inspecionar as tabelas e registros:

```
rails dbconsole
```

c. O comando anterior inicia o console interativo, portanto agora será necessário utilizar a sintaxe do SQLite3 para verificar nossas tabelas. Vamos primeiro listar todas as tabelas do nosso banco de dados de desenvolvimento:

```
sqlite> .tables
```

d. Agora vamos solicitar mais informações sobre a tabela *eventos*:

```
sqlite> PRAGMA table_info(eventos);
```

e. Para finalizar a sessão de console de banco de dados, execute o comando a seguir:

```
sqlite> .quit
```

A sintaxe do console interativo do banco de dados depende do tipo de banco utilizado. Se você escolher, por exemplo, utilizar o MySQL os comandos para listar as tabelas do banco de dados é `show tables;`, e para obter detalhes sobre a tabela é `describe eventos;`.

6.13 – Server

Agora que já geramos o código necessário em nossa aplicação, criamos o banco de dados e as tabelas necessárias, precisamos colocar nossa aplicação em um servidor de aplicações Web que suporte aplicações feitas com o Ruby on Rails. Só assim os usuários poderão acessar nossa aplicação.

O próprio conjunto de ferramentas embutidas em uma instalação padrão de um ambiente Rails já inclui um servidor simples, suficiente para que o desenvolvedor possa acessar sua aplicação através do *browser*. Esse servidor, o **WEBrick**, não é indicado para aplicações em produção por ser muito simples e não contar com recursos mais avançados necessários para colocar uma aplicação "no ar" para um grande número de usuários.

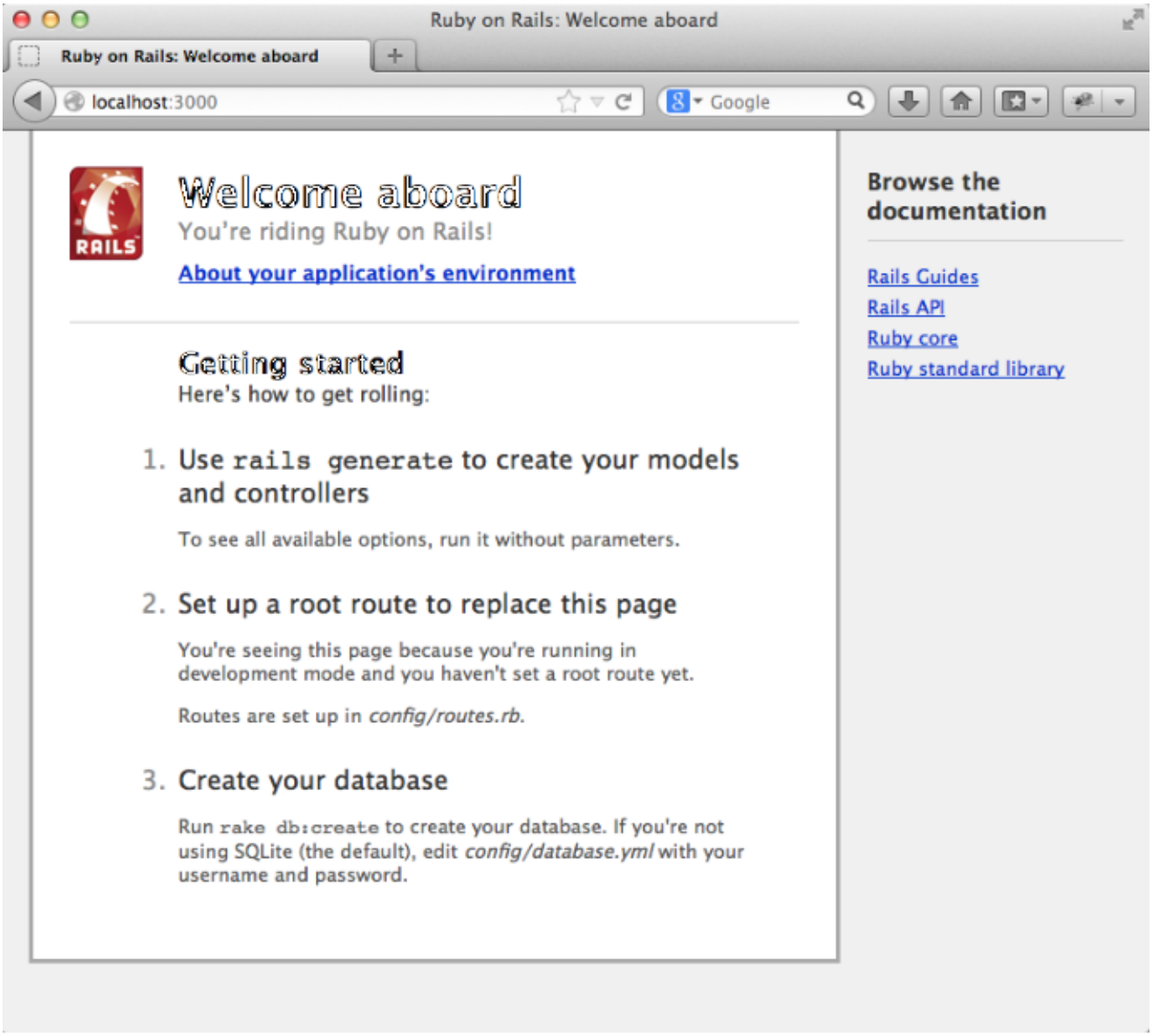
Podemos iniciar o servidor através do comando *rails server* no terminal:

```
rails server
```

A saída será algo como:

```
$ rails server
=> Booting WEBrick
=> Rails 4.0.0 application starting in development on http://0.0.0.0:3000
=> Run `rails server -h` for more startup options
=> Ctrl-C to shutdown server
[2013-12-23 15:49:09] INFO  WEBrick 1.3.1
[2013-12-23 15:49:09] INFO  ruby 2.0.0 (2013-06-27) [x86_64-darwin13.0.0]
[2013-12-23 15:49:09] INFO  WEBrick::HTTPServer#start: pid=3634 port=3000
```

Após o processo de inicialização, podemos acessar nossa aplicação no navegador. Por padrão o servidor disponibiliza nossa aplicação na porta 3000, podemos visitar nossa aplicação em http://localhost:3000.



Quando geramos o *scaffold* de eventos, foram geradas telas simples para listar todos registros de eventos, exibir um evento e os formulários para criar um novo evento e editar um evento existente. As telas são simples, mas são totalmente funcionais.

Ao visitar "http://localhost:3000/eventos", você é redirecionado para a listagem de eventos.

6.14 - Exercícios - Iniciando o servidor

1. a. Inicie o servidor com o comando:

```
rails server
```

b. Acesse a aplicação pela url: http://localhost:3000/eventos

Tire suas dúvidas no novo GUJ Respostas

O GUJ é um dos principais fóruns brasileiros de computação e o maior em português sobre Java. A nova versão do GUJ é baseada em uma ferramenta de *perguntas e respostas* (QA) e tem uma comunidade muito forte. São mais de 150 mil usuários pra ajudar você a esclarecer suas dúvidas.

Faça sua pergunta.

6.15 – Documentação do Rails

O Ruby tem uma ferramenta capaz de gerar documentação do nosso código a partir dos comentários que podemos fazer dentro de uma classe ou imediatamente antes das declarações de métodos, essa ferramenta é o RDoc.

A documentação da versão atual do Rails está disponível em **http://api.rubyonrails.org**.

Há diversos outros sites que fornecem outras versões da documentação, ou modos alternativos de exibição e organização.

- http://www.railsapi.com
- http://www.gotapi.com
- http://apidock.com/rails

O RubyGems também oferece a documentação para cada uma das gems que estão instaladas, para isso basta iniciar o servidor de documentação embutido:

```
gem server
```

Este comando inicia um servidor WebRick na porta 8808. Basta acessar pelo browser para ver o RDoc de todos os gems instalados.

A última alternativa é gerar a documentação através do rake:

```
rails docs
rake doc:rails
```

Depois de executar a task `doc:rails`, a documentação estará disponível no

diretório *docs/api/*. A documentação do Ruby (bem como a biblioteca padrão) pode ser encontrada em **http://ruby-doc.org**.

Existem ainda excelentes guias e tutoriais oficiais disponíveis. É **obrigatório** para todo desenvolvedor Rails passar por estes guias: **http://guides.rubyonrails.org**.

Além desses guias, existe um site que lista as principais gems que podem ser usadas para cada tarefa: **http://ruby-toolbox.com/**.

A comunidade Ruby e Rails também costuma publicar excelentes *screencasts* (vídeos) com aulas e/ou palestras sobre diversos assuntos relacionados a Ruby e Rails. A lista está sempre crescendo. Aqui estão alguns dos principais:

- **http://railscasts.com** - Vídeos pequenos abordando algumas práticas e ferramentas interessantes, mantidos por Ryan Bates. Alguns vídeos são gratuitos.
- **http://peepcode.com** - Verdadeiras vídeo-aulas, algumas chegam a 2h de duração. São pagos, cerca de US$ 9,00 por vídeo.
- **http://www.confreaks.com** - Famosos por gravar diversas conferências sobre Ruby e Rails.

6.16 - Exercício opcional - Utilizando a documentação

No formulário de cadastro de eventos podemos verificar que, apesar de já criar os formulários com os componentes adequados para que o usuário possa escolher uma data correta, a lista de anos disponíveis exibe somente alguns anos recentes.

Um dos requisitos possíveis para nossa aplicação é que o ano inicial do evento seja no mínimo o ano atual, para evitar que algum usuário cadastre um evento com uma data de fim no passado.

Para fazer essa alteração na aplicação será preciso editar o código gerado pelo *Rails* responsável por exibir a lista de anos, comece a listar a partir do ano atual:

1. Verifique no arquivo *app/views/eventos/_form.html.erb* um método chamado datetime_select:

```
<%= f.datetime_select :termino %>
```

Encontre a documentação desse método (em **ActionView*Helpers*DateHelper**). Como podemos indicar que queremos exibir anos a partir do ano atual?

```
<%= f.datetime_select :termino, start_year: 2012 %>
```

CAPÍTULO ANTERIOR:

Metaprogramação

PRÓXIMO CAPÍTULO:

Active Record

Casa do Código

Twitter