Curso Redmine

Ementa

O Curso

Neste curso veremos como desenvolver plugins para Redmine 2.6. Com foco na teoria e técnicas de programação da framework Rails e como se aplica ao Redmine.

Não é um curso de Ruby on Rails

Ambiente de Desenvolvimento

IDES

Sublime

O Sublime é um editor de texto poderoso, rápido e multiplataforma. Atualmente está na sua versão 3.0, custando 70 obamas.

Atom

Atom é sem dúvida a minha recomendação para quem quer seguir com desenvolvimento ruby. Um editor estável, open source e desenvolvido/mantido pelo próprio GitHub.

Poderia passar horas falando o porque eu gosto do atom, mas não é o foco do curso.

RubyMine

Como escrito no próprio site da JetBrain, a IDE de ruby mais inteligente. Porque não uso? Primeiro, porque acredito que IDE traga mais distrações do que o benefícios, segundo porque demora mais tempo para iniciar que o Atom, terceiro porque custa 199 obamas no primeiro ano e 99 para renovar a licença. E não acho que esse valor se pague.

Mas se você é desenvolvedor Java e não consegue viver sem autocomplete, vale a pena testar os 30 dias de trial.

PS: Não citei o eclipse porque era bem ruim quando usei para ruby(a uns 2 anos atrás) e nunca mais voltei, mas se você quiser testar, fique a vontade.

Ruby

Introdução

Ruby é uma linguagem de programação de 1995 onde quase tudo é um objeto. Uma linguagem moderna, possuindo tipos dinâmicos, lambda function e altamente influenciada por programação funcional.

Diferente do Java onde o tipo é explícito, em Ruby a tipagem é conhecida como Duck Typing, se um argumento tem todos os métodos que o método precisa para usá-lo, o método vai aceita-lo como argumento. O que não significa que a variável não tenha tipo, todo objeto tem o método .class, que retorna a classe que ele pertence.

Outra diferença com o Java é que as classes em Ruby são abertas, mas o que isso significa? Significa que após declarar uma classe, você pode abri-la novamente e altera-la. Continuou sem entender? Vamos para o Exemplo:

```
class A
  def a
    print 'a'
  end
end

obj = A.new
obj.a
=> a

class A
  def b
    print 'b'
  end
end

obj = A.new
obj.a
=> a
obj.b
=> b
```

Depois de declarar a classe A pela segunda vez, quando iniciei um novo objeto dessa classe, ele passou a ter ambos os métodos. Mas o que ocorreria se eu tivesse declarado o mesmo método novamente?

```
class A
  def a
    print 'novo a'
  end
end

obj = A.new
obj.a
=> novo a
obj.b
=> b
```

Se o mesmo método for declarado duas vezes, a última declaração passa a valer. Essa característica da linguagem, evita as milhões de classe Utils que criamos no Java e facilita a criação de plugins.

E como ruby é altamente influenciada por programação funcinal, toda função tem um retorno, não existe função void em Ruby.

Instalar

Para instalar o Ruby no Linux **(Usem LINUX)**, vamos utilizar um gerenciador de versão do Ruby para conseguirmos ter mais de uma versão rodando na mesma máquina.

Os dois gerênciadores mais famosos são o rbenv e o rvm. Para esse curso vamos utilizar o rvm.

Escolha baseada em gosto pessoal, se quiserem se aventurar no rbenv ele também é muito bom.

```
$ apt-get update
$ apt-get install -y subversion git telnet
$ apt-get install apt-get install -y libmysqlclient-dev freetds-dev imagemagick libmagickcore-dev libmagic

$ gpg --keyserver hkp://keys.gnupg.net --recv-keys 409B6B1796C275462A1703113804BB82D39DC0E3

$ \curl -sSL https://get.rvm.io | bash -s stable

$ rvm requirements
$ rvm install 1.9.3
$ rvm use 1.9.3 --default
```

O Redmine 2.6 usa o ruby 1.9.3, por isso instalamos ele.

Enquanto escrevia esse curso, verifiquei que na vesão 2.6.6 o Redmine passou a suportar o Ruby 2.2, mas como ainda não tive tempo de testar, vou seguir com a 1.9.3

Rails

Introdução

O Rails é uma framework MVC baseado em dois princípios básicos que você deve **SEMPRE** seguir, Convenção sobre Configuração e o Dry(Don't repeat yourself).

Atualmente se encontra na versão 4.2.3, mas no redmine 2.6 utilizaremos a 3.2.

Instalar

Podemos instalar o rails utilizando o RubyGem, uma gem seria o equivalente do jar no Java. O RubyGem é um instalador de *gems* que já é instalado junto com o ruby pelo rvm, ele funciona parecido com o apt-get do ububtu.

```
$ sudo gem install rails -v 3.2.x
```

Vamos instalar a versão 3.2 do rails pois é a última compatível com o redmine 2.6

Redmine

Introdução

O Redmine é um gerenciador de projeto **muito** flexível, extensível e configurável. Como o código é aberto, conseguimos utiliza-lo para atender as mais variádas demandas dos clientes. As principais features nativas são:

- Multiple projects support
- Flexible role based access control
- Flexible issue tracking system
- Gantt chart and calendar
- News, documents & files management
- Feeds & email notifications
- Per project wiki
- Per project forums
- Time tracking
- Custom fields for issues, time-entries, projects and users
- SCM integration (SVN, CVS, Git, Mercurial, Bazaar and Darcs)
- Issue creation via email
- Multiple LDAP authentication support
- User self-registration support
- Multilanguage support
- Multiple databases support

Instalar

Para instalar o Redmine, vamos utilizar a última versão estável do redmine 2.6, que no momento de criação desse curso era a 2.6.6

```
$ svn co https://svn.redmine.org/redmine/branches/2.6.6-stable redmine-2.6.6

$ cd redmine-2.6.6
```

Tendo o código do redmine, precisamos configurar o arquivo do banco de dados e o arquivo de configuração de e-mail. Por enquanto vamos utilizar o examplo do próprio redmine.

```
$ cp ./config/database.yml{.example,}
$ cp ./config/configuration.yml{.example,}
```

Assim como no Java temos o Maven para baixar as dependências, no Ruby temos o Bundle, para utiliza-lo basta fazer:

```
$ bundle install
```

Ele irá olhar o arquivo Gemfile, na pasta raiz do projeto e instalar todas as dependências que lá estiver.

> O Bundle faz tudo que o Maven faz?

NÃO, ele, diferente do Maven, se foca em fazer bem uma única coisa: gerênciar dependências.

Para automatizar tarefas, temos o **rake**, vamos utilizar para gerar o token de segurança de sessão.

```
$ rake generate_secret_token
```

Também precisamos gerar as tebelas do banco de dados que o redmine usa. O Rails por padrão possui migrações, arquivos em ruby(.rb) que descreve as operações que devemos realizar no banco.

Podemos com o rake rodar todas essas migrações e o Rails se encarrega de transformar no sql certo para o banco descrito no database.yml

```
$ rake db:create
$ rake db:migrate
```

Pronto, agora estamos pronto para rodar o Redmine

```
$ rails server
```

Fluxo de Dados

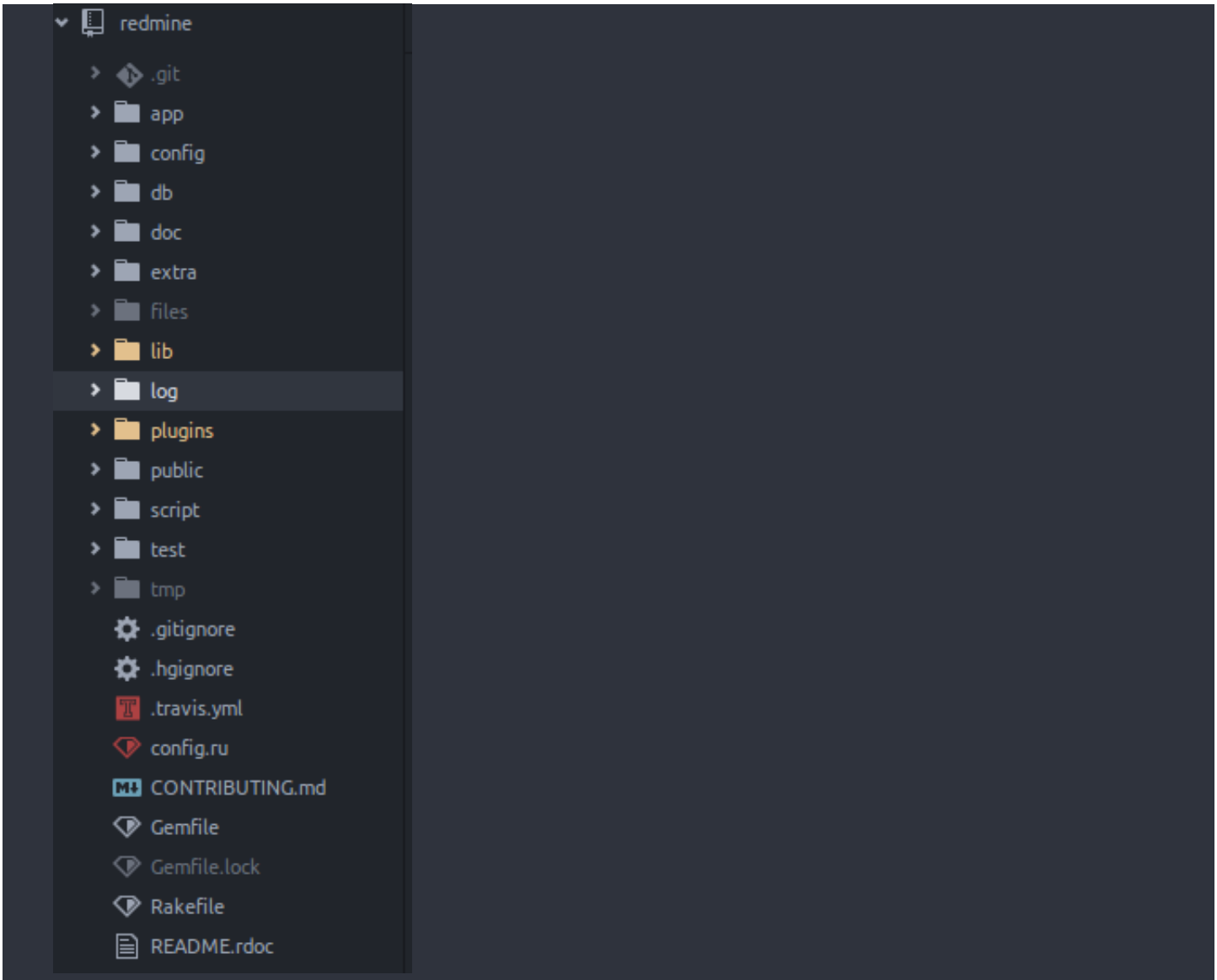
A primeira coisa para se entender como programar utilizando Ruby on Rails é entender o fluxo de dados.

Quando um request chega ele segue o fluxo:

- Tenta encaixar a url do request em algum partner cadastrado nos arquivos de rotas.
  - Podemos verificar todas as rotas cadastradas rodando o comando "*rake routes*" no terminal
- O arquivo de rotas dira para o Rails qual o controller ele deve chamar e qual action ele deve executar.
  - Uma action é o nome de um método de um controller
- Uma action pode redirectionar para outra action ou renderizar uma view.
  - Por padrão o Rails renderiza a view com o mesmo nome da action dentro da pasta com o mesmo nome do controller. **(Convenção sobre Configuração)**
- A view encherga as variáveis de instâncias(@) do controller

Entender e praticar esse fluxo é de extrema importância para saber encontrar o que você deseja modificar no Redmine e saber em qual parte do código está dando erro. Com o tempo você perceberá que as coisas estão onde devem estar.

Estrutura de Pastas

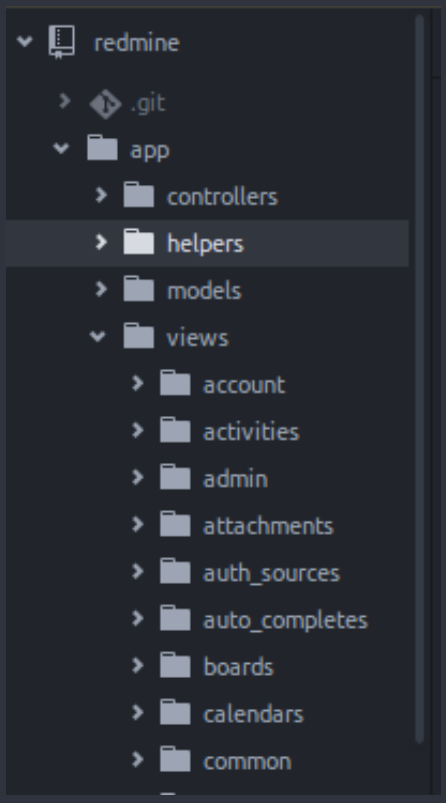


- app
  - Pasta onde ficam os arquivos da aplicação como models, controllers, views, etc
- config
  - Pasta onde ficam os arquivos de configuração de banco, ambiente, **rotas**, locales
- db
  - Pasta onde se encontra as migrações do banco
- doc
  - Pasta para guardar as documentações da aplicação
- extra
  - Pasta com exemplo de plugin
- files
  - Pasta onde o Redmine guarda os arquivos anexados
- lib
  - Pasta onde ficam as bibliotecas de código desenvolvida, como rake taskes, patches, etc

- plugins
  - Pasta onde ficam guardados os plugins desenvolvidos para o Redmine, é aqui onde faremos 90% do desenvolvimento
- public
  - Pasta onde ficam os arquivos estáticos servidos pelo webserver
- script
  - Contêm o script para inicialização do rails
- test
  - Pasta com os testes automatizados do Redmine

Subpastas importantes

App

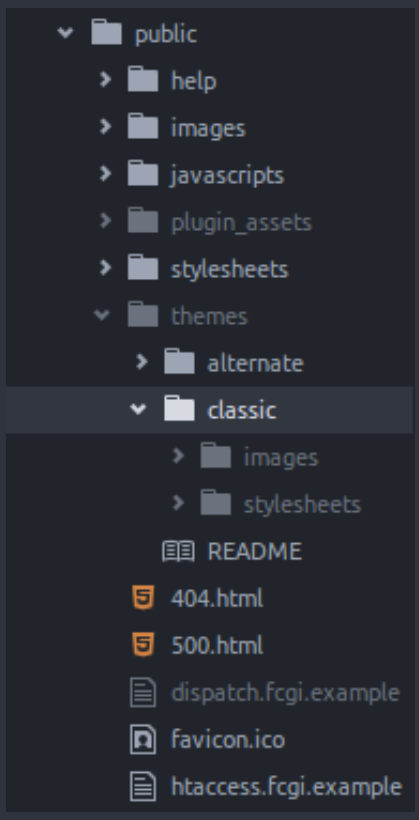


Como mencionado a pasta app contêm os controllers, models e views, mas em qual subpasta eles ficam? Bom, vou deixar você, caro leitor, descobrir sozinho.

Descobriu? Ótimo, agora vamos olhar para as views. As views possuem também subpastas, cada uma delas com o mesmo nome do controller. Assim o rails sabe qual view renderizar quando uma action for chamada. Ele vai dentro da **pasta view > nome_controller > nome_action.{html, js, xml, etcc}.erb**.

Caso ele não encontre o arquivo correspondentes, ele vai buscar na pasta com o mesmo nome da superclasse do controller e assim sucessivamente. Caso ele por fim não encontre, a página 404 do public é renderizada.

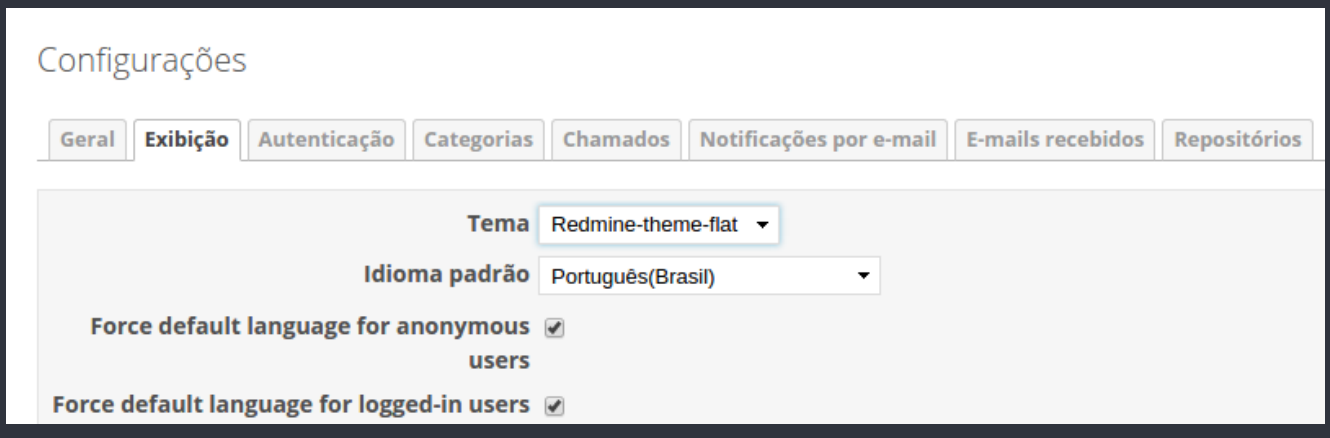
Public



Na pasta public ficam as imagens, javascript e css(pasta stylesheets), nela também ficam os temas do redmine.

O Redmine aceita temas customizados, um tema consiste em um conjunto de css/javascripts e dentro da configuração do Redmine é possível escolher qual tema o usuário vai ver.

O Redmine carrega automaticamente todas as pastas que se encontram dentro da pasta theme e disponibiliza para o administrador escolher qual utilizar.



O Redmine possui uma lista de themas feitos pela comunidade para você não partir do zero.

[Lista de temas da Comunidade]

Plugins

Quando devemos desenvolver um plugin?

Devemos desenvolver um plugin quando:

1. O redmine não faz o que queremos que ele faça **E**
2. Não existe um plugin Open Source que faça o que a gente gostaria que o Redmine fizesse.

Se existir um plugin open source que faça parecido, faça um fork do plugin e contribua com ele. Assim todos ganham =).

Como criar um plugin

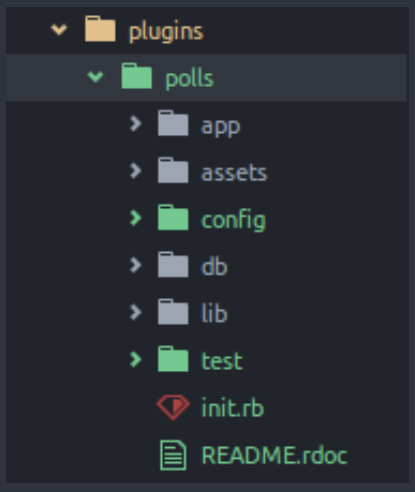
O Redmine provê um generator para criação da estrutura padrão de um plugin

```
$ rails generate redmine_plugin <plugin_name>
```

Rodando por exemplo:

```
$ rails generate redmine_plugin polls
```

Ele irá criar dentro da pasta plugins:



Reparem, ele criou uma estrutura muito parecida com a estrutura do próprio redmine, de diferente temos:

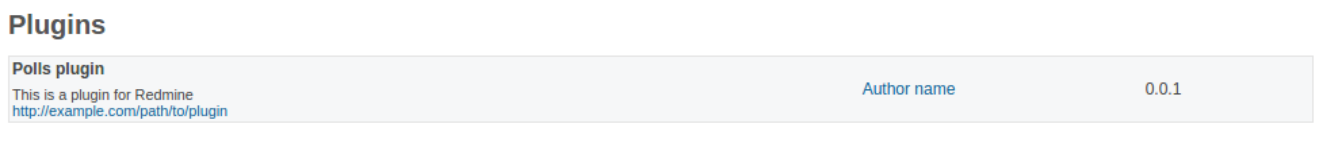
- assets
  - Nessa pasta ficarão as imagens, javascript e css do plugin. O Redmine ao iniciar irá pegar esses arquivos e colocar na pasta plugin_assets dentro da pasta public

Editando o init.rb veremos

```
Redmine::Plugin.register :polls do
  name 'Polls plugin'
  author 'Author name'
  description 'This is a plugin for Redmine'
  version '0.0.1'
  url 'http://example.com/path/to/plugin'
  author_url 'http://example.com/about'
end
```

O init.rb é o arquivo que o redmine chama ao carregar o plugin, iremos utilizar mais ele no futuro. Mas por enquanto podemos somente modificar as informações do plugin.

Essas informações irão aparecer da seguinte forma no redmine:



Criando um modelo

Novamente, para criar um modelo dentro de um plugin, podemos chamar o generator do próprio redmine

```
$ rails generate redmine_plugin_model <plugin_name> <model_name> [field[:type][:index] field[:type][:ir
```

Podendo reparar que é muito parecido com o generator de modelo do rails. Então vamos criar um modelo poll dentro do plugins polls, que vai ter uma question do tipo string, uma contagem de sim e uma contagem de não, ambos do tipo numérico.

```
$ rails generate redmine_plugin_model polls poll question:string yes:integer no:integer
    create  plugins/polls/app/models/poll.rb
    create  plugins/polls/test/unit/poll_test.rb
    create  plugins/polls/db/migrate/001_create_polls.r
```

Reparem, ele criou um modelo na pasta models e uma migração na pasta migrations

Vamos olhar essa migração:

```
class CreatePolls < ActiveRecord::Migration
  def change
    create_table :polls do |t|
      t.string :question
      t.integer :yes
      t.integer :no
    end
  end
end
```

Vamos entender a migração, ela chama uma função *create_table* que recebe como argumento um símbolo (:polls) e um bloco de código que recebe também um argumento(t).

> Primeiro, o que é um símbolo? Você deve estar se perguntando

Um simbolo é parecido com a String do Java. A diferença entre "polls" e :polls é que o primeiro é mutável, se você fizer:

```
a = "a"
a += "b"
print a
  => "ab"
a = nil
```

Quando o GC for executado, não teremos nada alocado na memória.

Já um símbolo, não possui um método para concatenar e se executácemos:

```
a = :a
print a
  => "a"
a = nil
```

O símbolo :a, não seria arrancado da memória. Isso é perigoso quando criamos símbolos dinamicamente e é uma vunerabilidade conhecida do Rails.

Mas porque então usamos ele ao invés de String, a resposta é simples: desempenho.

Se você cria um símbolo de maneira controlada, sempre que você for acessa-lo, não precisará realocar memória e não importa quantas variáveis apontem para ele, elas vão estar consumindo a mesma quantidade de memória, pois estarão apontando sempre para a mesma posição.

> Agora que já entendi o que é um símbolo, o que diabos é passar um bloco de código como argumento?

Bom, em ruby uma função pode receber um bloco de código e executa-lo dentro dela. O equivalente em Java 7 seria instanciar uma interface e preencher os métodos, muito usado nos handlers da vida.

No caso da função create_table ela executa o que tem que executar, instância um objeto e passa para a nossa função anônima. Como o ruby é Duck Typing, se o objeto t recebido pela nossa função anônima tiver todos os métodos necessários para o bloco de código ser executado, então o bloco será executado sem problema.

Esse bloco irá chamar os métodos do objeto t que criam as colunas, o método string cria uma coluna do tipo string, o integer do tipo integer, etc... O método create_table cria sozinho a coluna id e os timestamps.

Para saber mais sobre as migrações do Rails acesse http://guides.rubyonrails.org/v3.2.21/migrations.html

Para executar as migrações dentro dos plugins:

```
$ rake redmine:plugins:migrate
Migrating polls (Polls plugin)...
==  CreatePolls: migrating ====================================================
-- create_table(:polls)
   -> 0.0323s
==  CreatePolls: migrated (0.0324s) ===========================================
```

O rails irá criar a table a as colunas para você, "indendente" do banco que esteja configurado no seu database.yml

Utilizando **convenção sobre configuração**, ele irá atribuir todas as colunas da tabela polls(plural) ao modelo poll(sigular) como métodos, já com getters & setters

```
class Poll < ActiveRecord::Base
  unloadable
end

poll = Poll.new
poll.question = "Question 1"
print poll.question
=> "Question 1"
print poll.yes
=> nil
```

Futuro

A ideia de desenvolver o curso no github é deixar ele colaborativo e expansível, assim como o Redmine. Gerando assim uma apostila completa sobre o assunto, que se mantenha sempre atualizada.

Colaboradores

Quem contribuir com esse material, pesso que mande um pull request adicionado o seu nome na lista abaixo.

- Victor Lima Campos(victorlcampos)

Alunos

Quem utilizar esse material para estudo, pesso que mande um pull request adicionado o seu nome na lista abaixo.

- Victor Lima Campos(victorlcampos)