< Codificando nossa aplicação - E-Book />

Vamos gerar uma nova rvm:

```
$ rvm --rvmrc --create ruby-1.9.3-p392@ebooks
```

Agora vamos instalar a gem do rails:

```
$ gem install rails
```

Crie um novo projeto configurado para o mysql atravéz desse comando:

```
$ rails new ebooks -d mysql
```

Vamos gerar nosso banco de dados no mysql:

```
$ rake db:create
```

Execute o comando para gerar nossa página de bem vindo:

```
$ rails generate controller welcome index
```

Vamos mapear a página criada acima para que ela seja a nossa home:

```
root to: 'welcome#index'
```

< Criando um CRUD dos usuários />

Vamos incluir algumas gems que sera necéssaria para o andamento do projeto e a versão do ruby que vamos usar:

```
ruby '1.9.3'

gem 'compass-rails', '2.0.alpha.0'
gem 'devise', '3.1.0'

group :development do
  gem 'quiet_assets'
end
```

Rode o bundle para instalar as gems:

```
$ bundle install
```

Usando o generate vamos criar a migration e o modelo de user:
```
$ rails generate model User name:string birthdate:date
```

Vamos gerar o arquivo de configuração do device:
```
$ rails generate devise:install
```

Agora iremos gerar todas as views do device:
```
$ rails generate devise:views
```

Vamos adicionar as migrations que carrega os atributos necessários do device e apontar qual modelo ele vai usar:
```
$ rails generate devise User
```

Criaremos a tabela de usuário em nosso banco de dados:
```
$ rake db:migrate
```

Validaremos para que o nome e data de aniversário sejam validadas suas presenças:

```
validates_presence_of :name, :birthdate
validates_uniqueness_of :name
```

Vamos incluir em */app/views/device/registrations/new.html.erb* e */app/views/device/registrations/edit.html.erb* o restante dos atributos que não é padrão do device mas que achamos necessário conter:

```
<%= devise_error_messages! %>

<div><%= f.label :name %></br>
<%= f.text_field :name%></div>
<div><%= f.label :birthdate %></br>
<%= f.date_field :birthdate%></div>
```

No final do */app/views/device/registrations/edit.html.erb* vamos alterar o html e retirar todos < br >:

```
<div class='bottom'>
  <%= link_to "Cancel my account",
registration_path(resource_name), :data => { :confirm => "Are you
sure?" }, :method => :delete %>
  <%= link_to "Back", :back %>
</div>
```

Em *app/controllers/application_controller.rb* vamos permitir nossos atributos do device de persistir no banco de dados inserindo:

```
before_filter :configure_permitted_parameters, if: :devise_controller?


protected
def configure_permitted_parameters
  devise_parameter_sanitizer.for(:sign_up) << [:name, :birthdate]
  devise_parameter_sanitizer.for(:sign_in) { |u|
u.permit(:name, :email) }
  devise_parameter_sanitizer.for(:account_update) <<
[:name, :birthdate]
end
```

Vamos customizar nossos erros de cadastro começando pelo device, em *app/helpers/application_helper.rb* vamos inserir:

```
def error_messages(record)
  return if record.errors.empty?
  record.errors.full_messages.map{ |message| content_tag(:span,
message, class: 'error') }.join.html_safe
end
```

e em *app/helpers/devise_helper.rb* adicionaremos o método devise_error_messages! responsável pela mensagem de erro:

```
def devise_error_messages!
  error_messages(resource)
end
```

Agora iremos em *app/views/shared/_nav.html.erb* definir nosso menu da applicação:

```
<ul class='nav'>
  <li>
    <%= link_to 'home', root_path %>
  </li>

  <%- unless user_signed_in?%>
    <li>
      <%= link_to 'subscribe', new_user_registration_path %>
    </li>

    <li>
      <%= link_to 'login', new_user_session_path %>
    </li>
  <% end %>
</ul>

<%- if user_signed_in?%>
  <ul class='user'>
    <li>
      <%= link_to image_tag(current_user.avatar),
edit_user_registration_path %>
      <%= link_to 'exit', destroy_user_session_path, method: :delete,
class: 'exit'%>
    </li>
  </ul>
```

```
<% end %>

<%- unless notice.blank? %>
  <div class='notice'>
    <b><%= notice%></b>
  </div>
<% end %>

<%- unless alert.blank? %>
  <div class='alert'>
    <b><%= alert%></b>
  </div>
<% end %>
```

Como vimos estamos fazendo current_user.avatar, mas não temos esse método então em *app/models/user.rb* vamos implementar a lógica para acessarmos nosso avatar que fica no gravatar:

```
def avatar
  md5 = Digest::MD5.hexdigest(email).to_s
  "http://gravatar.com/avatar/#{md5}.png"
end
```

< Criando um CRUD dos books />

Vamos gerar nossa migration e modelo do book:

```
$ rails generate model Book user:references title:string author:string edition:integer published_at:date published:boolean publishing_house:string issuu_id:string
```

Vamos validar no modelo book os atributos que foram gerados:

```
validates_presence_of :user, :author, :title, :published_at
validates_uniqueness_of :title
```

Vamos criar a tabela de book em nosso banco de dado:

```
$ rake db:migrate
```

Vamos gerar nossos controllers e views para o book:

```
$ rails generate controller books index show new create edit update destroy
```

Delete em app/views/books os aquivos: destroy.html.erb, update.html.erb e create.html.erb!

Agora em config/routes vamos apagar as rotas geradas pelo generate e criar apenas uma:

```
resources :books
```

Vamos alterar nosso menu que esta em *app/views/shared/_nav.html.erb* para:

```
<ul class='nav'>
  <li>
    <%= link_to 'home', root_path %>
  </li>

  <li>
    <%= link_to 'books', books_path%>
```

```
  </li>

  <%- unless user_signed_in?%>
    <li>
      <%= link_to 'subscribe', new_user_registration_path %>
    </li>

    <li>
      <%= link_to 'login', new_user_session_path %>
    </li>
  <%- else%>
    <li>
      <%= link_to 'new book', new_book_path%>
    </li>
  <% end %>
</ul>

<%- if user_signed_in?%>
  <ul class='user'>
    <li>
      <%= link_to image_tag(current_user.avatar),
edit_user_registration_path %>
      <%= link_to 'exit', destroy_user_session_path, method: :delete,
class: 'exit'%>
    </li>
  </ul>
<% end %>

<%- unless notice.blank? %>
  <div class='notice'>
    <b><%= notice%></b>
```

```
  </div>
<% end %>

<%- unless alert.blank? %>
  <div class='alert'>
    <b><%= alert%></b>
  </div>
<% end %>
```

Vamos colocar no modelo user.rb o relacionamento com books:

```
has_many :books, dependent: :destroy
```

Com nosso model validando e se relacionando podemos em *app/controllers/books_controller.rb* instanciar o objeto book na action new:

```
def new
  @book = current_user.books.new
end
```

Agora iremos escrever o form quer irá salvar um novo book:

```
<h2>Create new book</h2>

<%= form_for @book, url: books_path do |f|%>
<%= error_message(@book) %>

<div>
  <%= f.label :published%>
  <%= f.check_box :published%>
</div>

<div>
```

```
  <%= f.label :title%>
  <%= f.text_field :title%>
</div>

<div>
  <%= f.label :author%>
  <%= f.text_field :author%>
</div>

<div>
  <%= f.label :edition%>
  <%= f.number_field :edition%>
</div>

<div>
  <%= f.label :published_at%>
  <%= f.date_field :published_at%>
</div>

<div>
  <%= f.label :publishing_house%>
  <%= f.text_field :publishing_house%>
</div>
<div><%= f.submit %></div>
<% end %>
```

Nosso form esta apontando para a rota que redireciona para a action create, mas não estamos fazendo nada nela, o que temos que fazer exatamente então? E pegar os parâmetros do nosso formulário que esta vindo de uma requisição post atribuir na instância do objeto book e salvar ficando apenas, lembrando que não podemos esquecer de permitir os attributos:

```ruby
def create
  @book = current_user.books.new(book_params)
  if @book.save
    redirect_to book_path(@book)
  else
    render action: :new
  end
end

private
def book_params
  params.require(:book).permit!
end
```

A nossa tabela de book possui um atributo issuu_id, ele sera responsável em relacionar um book que esta postado no issuu com nosso website; primeiro precisamos criar um cadastro em http://issuu.com/home, depois so começarmos a parte 1 da nossa integração com o sistema:

Gemfile:

```ruby
gem 'issuu', '0.3.0'
```

config/initializer/issuu.rb:

```ruby
Issuu.configure do |c|
  c.api_key    = 'sua key'
  c.secret     = 'seu secret'
end
```

Agora em *app/models/book.rb* faça:

```
validates_presence_of :issuu_id

# callbacks
before_validation :set_issuu_id
private
def set_issuu_id
  Issuu::Document.list.each do |file|
    self.issuu_id = file.documentId if file.title == title
  end
  errors.add(:issuu_id, 'There is no such issuu') if issuu_id.blank?
end
```

Agora com o issuu integrado em nosso sistema é so acessarmos o *localhost:3000/books/new* e testarmos!

Com nosso cadastro de books funcionando agora é hora de desacoplar nosso form para um partial para utilizarmos tanto no cadastro quanto na edição ficando assim:

new.html.erb

```
<h2>Create new book</h2>
<%= form_for book, url: books_path do |f|%>
  <%= render partial: 'books/form', locals: {f: f}%>
<% end %>
```

edit.html.erb

```
<h2>Edit <%= @book.title%></h2>
<%= form_for book, url: book_path do |f|%>
  <%= render partial: 'books/form', locals: {f: f}%>
<% end %>
```

_form.html.erb

```
<%= error_messages(@book) %>

<div>
  <%= f.label :published%>
  <%= f.check_box :published%>
</div>

<div>
  <%= f.label :title%>
  <%= f.text_field :title%>
</div>

<div>
  <%= f.label :author%>
  <%= f.text_field :author%>
</div>

<div>
  <%= f.label :edition%>
  <%= f.number_field :edition%>
</div>

<div>
  <%= f.label :published_at%>
  <%= f.date_field :published_at%>
</div>

<div>
  <%= f.label :publishing_house%>
  <%= f.text_field :publishing_house%>
</div>
<div><%= f.submit %></div>
```

Aproveitando que o form foi desacoplado em um partial vamos criar ações para os actions edit e update do controller, sendo que o edit so vai fazer um find no banco procurando o book pelo parâmetro id e o update ira buscar no banco o registro e utilizar

um método do rails chamado de **update_attributes** passando os parâmetros novos para salvar o registro no banco, ficando assim:

```
def edit
  @book = Book.find(params[:id])
end



def update
  @book = Book.find(params[:id])

  if @book.update_attributes(book_params)
    redirect_to book_path(@book)
  else
    render action: :edit
  end
end
```

Depois de conseguirmos criar e editar um book, chegou a hora de criarmos uma view para listar todos os books cadastrados, para isso vamos incluir o seguinte código em *app/controllers/books_controller.rb* na action **index** o seguinte:

```
def index
  @books = Book.all
end
```

So que temos um problemas, queremos todos books que esteja marcado como publicado, para isso vamos criar um escopo no model de book.rb, isso fará que todas as vezes que chamamos o objeto Book.all ele irá passar pela condição definida no default_scope :

```
default_scope conditions: { published: true }
```

Agora que nossa index retorna todo books publicados é hora de fazer o html, em `app/views/books/index.html.erb`:

```
<div id='books'>
  <%- @books.each do |book| %>
    <div class='book'>
      <%= link_to book do%>
        <span class='title'> <%= book.title %> </span>
        <span class='author'> <%= book.author %> </span>
        <span class='border'></span>
        <%= image_tag book.url%>
      <% end %>
    </div>
  <% end %>
</div>
```

Em nossa index do book estamos chamando um método url para book que não existe, devemos fazer esse getter em app/model/book.rb que basicamente vai setar path da imagem do book postado no issuu ficando assim:

```
def url
  return if issuu_id.blank?
  "http://image.issuu.com/#{ issuu_id }/jpg/page_1_thumb_large.jpg"
end
```

Queremos que os books tenha apenas 6 por página para isso iremos utilizar uma gem chamada will paginate coloque em:

`Gemfile:`

```
gem 'will_paginate', '3.0.4'
```

app/controllers/books_controller.rb:

```
def index
  @books = Book.paginate(page: params[:page], per_page: 6
)
end
```

app/views/books/index.html.erb:

```
<%= will_paginate @books %>
```

routes.rb:

```
get '/books/page/:page', controller: 'books', action: 'index'
```

Antes de implementarmos a view de show, vamos implementar a parte 2 da integração com o issuu.

Primeiro vamos precisar usar uma lib de javascript para inserir um swf do book iniciamos com o download do arquivo em *vendor/assets/javascripts*:

```
$ cd vendor/assets/javascripts && wget http://goo.gl/ztNxvP
```

2 - Descompacte e deixe somente o swfobject.js, depois disso vamos fazer o require em *app/assets/javascripts/application.js* desse arquivo:

```
//= require swfobject
```

3- Vamos contruir nosso html que fica em *app/views/books/show.html.erb* da seguinte maneira:

```
<h2>Books <%= @book.title %></h2>

<div class='content_book_show'>
  <div id='book'>
    <div class= 'file'>
      <div id="issuu-<%= @book.id %>-viewer">
        <%= book_viewer @book, 600, 400 %>
      </div>
    </div>
  </div>

  <div class='info'>
    <span class='edition'>
      <b>Edition:</b>
      <%= @book.edition%>
    </span>
    <span class='published_at'>
      <b>Published at:</b>
      <%= @book.published_at%>
    </span>
  </div>
</div>
```

Ao executarmos essa página ela tera dois erros:

1 - **Não existe** o a variável de instância **@book,** vamos cria-la dentro do nosso controller `app/controllers/books_controller.rb`:

```
def show
  @book = Book.find(params[:id])
end
```

2 - O segundo erro é que estamos chamando um método **book_viewer** passando **três parâmetros** e ele não existe, para isso vamos utilizar um helper para ajudar nossa view a fazer essa lógica, para isso o melhor lugar para se fazer é em `app/helpers/book_helper.rb`:

```
def book_viewer(book, width, height)
  attributes = { id: 'issuu_viewer' }
  params = { allowfullscreen: true, allowScriptAccess: 'always', menu:
false,
    wmode: 'transparent'}
  flashvars = { mode: 'mini', documentId: book.issuu_id }

  content_for :javascript do
    javascript_tag
"swfobject.embedSWF('http://static.issuu.com/webembed/viewers/style1/v
2/IssuuReader.swf', 'issuu-#{@book.id}-viewer', '#{width}',
'#{height}', '10.0.2', '/expressinstall.swf', #{flashvars.to_json},
#{params.to_json}, #{attributes.to_json})"
  end
end
```

Em nosso layout precisamos fazer um yeld para colocarmos todo nosso javascript embaixo do de todos html:

```
<%= yield :javascript %>
```

Já estamos quase la :D. Porém falta aprimorarmos um pouco mais nossas views. Na `index.html.erb` de books vamos deixar exposto para o usuário dono do book um link para editar:

```
<%- if current_user && book.user_id == current_user.id %>
  <div class='edition'>
    <%= link_to 'edit', edit_book_path(book) %>
  </div>
<% end %>
```

E agora dentro do edit.html.erb vamos colocar um link para deletar:

```
<%= link_to 'destroy', book_path(@book), method: :delete, confirm:
'You want to delete?' %>
```

E finalmente para nosso link de deletar funcionar devemos dentro do controller books_controller na action destroy implementar a lógica para destruir esse book, ficando assim:

```
def destroy
  @book = Book.find(params[:id])
  if @book.destroy
    redirect_to books_path
  else
    render action: :edit
  end
end
```

Depois do nosso CRUD do book quase concluído, notamos que há uma grande falha de segurança…**COMO ASSIM?** Se outro usuário quiser editar e deletar um book que não seja dele ele conseguirá...**PORQUÊ?** Por que não estamos verificando se o usuário pertence o book antes de fazer a ação, para isso vamos acrescentar apenas uma verificação em `app/controllers/books_controller.rb` e fazer a verificação para a action new para não deixar o usuário deslogado criar nenhum book.

```ruby
before_filter :authenticate_user!, only: [:new, :create]
before_filter :authorized_book!, only: [:edit, :update, :destroy]

def authorized_book!
  book = Book.find(params[:id])
  if book.user_id != current_user.id
    redirect_to books_path, alert: 'This book is not your!'
  end
end
```

< Criando categorias para o book />

Para encerrarmos nossa aplicação precisamos relacionar nosso book a uma categoria para isso vamos gerar nossas migrations:

$ rails generate model Category title:string && rails generate model books_categories category:references book:references

Fazer uma alteração na linha 3 da migration books_categories para:

```
create_table :books_categories, id: false do |t|
```

Agora vamos executar nossas migrations criadas:
$ rake db:migrate

Vamos fazer algumas validações e relacionamento nos models e apagar o modelo gerado pela migration books_categories.rb:

app/models/category:

```
# relations
has_and_belongs_to_many :books

# validations
validates_presence_of :title

accepts_nested_attributes_for :books, allow_destroy: true
```

app/models/book:

```
has_and_belongs_to_many :categories

accepts_nested_attributes_for :categories, allow_destroy: true,
reject_if: proc{ |attrs| attrs['id'].blank? }
```

Desta vez vamos pedir auxilío de uma gem que ajudará a fazer nossos formulários aninhados para isso no *Gemfile* inclua:

```
gem 'awesome_nested_fields', '0.6.1'
```

Rode o bundler:

```
$ bundle install
```

Em `app/assets/javascripts/application.js` vamos fazer require do js:

```
//= require jquery.nested-fields
```

Precisamos que no nosso form de cadastro do book tenha um espaço para colocarmos a categoria, para isso vamos criar um input novo em `app/views/books/_form.html.erb`:

```
<h2> Categories </h2>
<div id='category'>
  <div class="items">
    <%= f.nested_fields_for :categories do |f| %>
      <fieldset class="item">
        <%= f.label :title %>
        <%= f.text_field :title, name: 'book[category_titles][]' %>

        <a href="#" class="remove">remove</a>

        <%= f.hidden_field :id %>
        <%= f.hidden_field :_destroy %>
      </fieldset>
    <% end %>
  </div>
```

```
  <a href="#" class="add">add categories</a>
</div>


<% content_for :javascript do %>
  <script type='text/javascript'>
    $ ('#category').nestedFields();
  </script>
<% end %>
```

Como podem observar estou utilizando um attributo category_titles, ele sera nosso getter e setter onde caso não houver nenhuma categoria com o título digitado ele criará uma nova categoria, em `app/models/books.rb` vamos inserir o seguinte trecho de código:

```
# attr_accessor
attr_accessor :category_titles

def category_titles=(titles)
  self.category_ids = titles.map do |category_title|
    Category.where(title: category_title).first_or_create.id
  end
end

def category_titles
  Category.where(id: category_ids) unless self.category_ids.blank?
end
```

Vamos gerar as view de show e index das categorias:

```
$ rails generate controller categories index show
```

Vamos alterar nossas rotas para mapear as categorias corretamente:

```
get "/categories" => 'categories#index'
get "/category/:id" => 'categories#show', as: :category
get '/categories/page/:page', controller: 'categories', action:
'index'
get '/categories/:id/page/:page', controller: 'categories', action:
'show'
```

No controller gerado app/controllers/categories_controller.rb incluimos os seguintes códigos:

```
def index
  @categories = Category.paginate(page: params[:page], per_page: 10)
end

def show
  @category = Category.find(params[:id])
  @books = @category.books.paginate(page: params[:page], per_page: 6)
end
```

Agora falta fazer somente as views:

app/views/categories/index.html.erb:

```
<%- @categories.each do |category| %>
  <div class='category'>
    <%= link_to category.title, category_path(category) %>
  </div>
<% end %>

<%= will_paginate @categories %>
```

app/views/categories/show.html.erb:

```
<h2> Category <%= @category.title%> </h2>

<%= render 'books/lists' %>
```

E alterarmos as seguintes views:

app/views/books/index.html.erb:

```
<h1>Books list</h1>

<%= render 'books/lists' %>
```

app/views/books/_list.html.erb:

```
<div id='#books'>
  <%- @books.each do |book| %>
    <%= link_to book do%>
      <div class='book'>
        <%= image_tag book.url %>
```

```
        <br/>
        <b> <%= book.title %> </b>
        <b> <%= book.author %> </b>
        <br/>
        <b> Edition: <%= book.author %> </b>
      </div>
    <% end %>


    <div id='category'>
          <%- if book.categories.present? %>
          <%- book.categories.limit(4).each do |category| %>
            <span class='categories'>
                <%= category.title %>
            </span>
          <% end %>
   </div>

      <%- if current_user && book.user_id == current_user.id  %>
        <div class='edition'>
          <%= link_to 'edit', edit_book_path(book) %>
        </div>
      <% end %>
    <% end %>
  <% end %>

  <%= will_paginate @books %>
</div>
```

Inclua a categoria no menu:

```
<li>
  <%= link_to 'categories', categories_path%>
</li>
```

Por último vamos fazer o footer e apagar arquivos que foi gerado que não iremo utilizar:

app/shared/_footer.html.erb:

```
<footer>
  <span>Desenvolvido durante o workshop na universidade Toledo no dia 18/09/2013.</span>
  <span class='social'>
    <%= link_to image_tag(image_path('github.png')), 'http://goo.gl/gLDNci', target: '_blank'%>
    <%= link_to image_tag(image_path('facebook.png')), 'http://goo.gl/M7tER', target: '_blank'%>
    <%= link_to image_tag(image_path('twitter.png')), 'http://goo.gl/S6Rey', target: '_blank'%>
  </span>
</footer>
```

app/layouts/application.html.erb:

```
<%= render 'shared/footer' %>
```

Vamos remover os seguintes arquivos:

app/assets/javascripts/books.js.coffee

app/assets/javascripts/categories.js.coffee

app/assets/javascripts/welcome.js.coffee

app/assets/stylesheets/books.css.scss

app/assets/stylesheets/categories.css.scss

app/assets/stylesheets/welcome.css.scss

app/controllers/welcome_controller.rb

`app/helpers/categories_helper.rb`

`app/helpers/welcome_helper.rb`

`app/views/welcome/index.html.erb`

`test/controllers/.keep`

`test/controllers/books_controller_test.rb`

`test/controllers/categories_controller_test.rb`

`test/controllers/welcome_controller_test.rb`

`test/fixtures/.keep`

`test/fixtures/books.yml`

`test/fixtures/books_categories.yml`

`test/fixtures/categories.yml`

`test/fixtures/users.yml`

`test/helpers/.keep`

`test/helpers/books_helper_test.rb`

`test/helpers/categories_helper_test.rb`

`test/helpers/welcome_helper_test.rb`

`test/integration/.keep`

`test/mailers/.keep`

`test/models/.keep`

`test/models/book_test.rb`

`test/models/books_categories_test.rb`

`test/models/category_test.rb`

`test/models/user_test.rb`

`test/test_helper.rb`

Parabéns a você que chegou até aqui. Finalmente temos nossa aplicação pronta!
Mas não terminou, vamos utilizar o heroku para hospedarmos nosso site.
Mas antes vamos aos ajustes.

`Gemfile:`

```
group :development do
  gem 'quiet_assets'
  gem 'mysql2'
end

group :production do
  gem 'rails_12factor'
  gem 'pg', '0.16.0'
end
```

config/database.yml:

```
production:
  adapter: postgresql
  encoding: utf8
  reconnect: false
  database: nome do seu database
  pool: 5
  username: seu username
```

Depois de criar sua conta, e ajustar nossa app execute no terminal:

```
$ heroku login
$ heroku create nome-da-sua-app

$ git push origin heroku master
$ heroku run rake db:migrate
```

Agora é so acessar a url que o heroku criou e veras que a nossa aplicação esta no ar!