



3. Rails Básico

m. As primeiras views

- i. Representação dos resultados das requisições
- ii. Por padrão ERB
 1. Texto puro misturado com Ruby
 - a. Ruby demarcado por `<% %>`
 - b. Apenas para imprimir algo `<%= %>`

iii. Views geradas

iv. index.html.erb

1.

```
<h1>Listing users</h1>
<table>
<tr>
<th>Login</th>
<th>Password</th>
<th>Email</th>
</tr>
<% @users.each do |user| %>
<tr>
<td><%=h user.login %></td>
<td><%=h user.password %></td>
<td><%=h user.email %></td>
<td><%= link_to 'Show', user %></td>
<td><%= link_to 'Edit', edit_user_path(user) %></td>
<td><%= link_to 'Destroy', user, :confirm => 'Are you sure?',
:method => :delete %></td>
</tr>
<% end %>
</table>
<br />
<%= link_to 'New user', new_user_path %>
```

2. Descrição

- a. `<% for user in @users %>` percorre os itens da lista. É finalizado por `<% end %>`
- b. `<%=h user.login %>` imprimir texto, mas chama um helper (“h”) para fazer escape de html e javascript
- c. `link_to`: helper padrão do rails para criar links. Primeiro parâmetro é o texto do link, o segundo é um hash com as mesmas opções do `url_for`, ou então um model do ActiveRecord, o terceiro parâmetro recebe opções HTML
 - i. `:confirm`: confirmação em javascript



UNIVERSIDADE FEDERAL DO PIAUÍ
CENTRO DE CIÊNCIAS DA NATUREZA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

ii. :method: método de requisição (muda de GET para DELETE)

v. show.html.erb

```
1. <p>
  <b>Login:</b>
  <%=h @user.login %>
</p>
<p>
  <b>Nome:</b>
  <%=h @user.password %>
</p>
<p>
  <b>Email:</b>
  <%=h @user.email %>
</p>
<%= link_to 'Edit', edit_user_path(@user) %> |
<%= link_to 'Back', users_path %>
```

2. Descrição

a. Nada muito diferente do que já foi usado

vi. new.html.erb e edit.html.erb são iguais

```
1. <h1>New user</h1>
<% form_for(@user) do |f| %>
  <%= f.error_messages %>
  <p>
    <%= f.label :login %><br />
    <%= f.text_field :login %>
  </p>
  <p>
    <%= f.label :password %><br />
    <%= f.text_field :password %>
  </p>
  <p>
    <%= f.label :email %><br />
    <%= f.text_field :email %>
  </p>
  <p>
    <%= f.submit 'Create' %>
  </p>
<% end %>
<%= link_to 'Back', users_path %>
```

2. <h1>Editing user</h1>



```
<% form_for(@user) do |f| %>
  <%= f.error_messages %>
  <p>
    <%= f.label :login %><br />
    <%= f.text_field :login %>
  </p>
  <p>
    <%= f.label :password %><br />
    <%= f.text_field :password %>
  </p>
  <p>
    <%= f.label :email %><br />
    <%= f.text_field :email %>
  </p>
  <p>
    <%= f.submit 'Update' %>
  </p>
<% end %>
<%= link_to 'Show', @user %> |
<%= link_to 'Back', users_path %>
```

3. Descrição

- a. Altera apenas o link de confirmação
 - b. `<% form_for(model) do |f| %>` helper para criar um FormBuilder (formulários). Métodos:
 - i. `label: <label>`
 - ii. `text_field: <input type="text">`
 - iii. `text_area: <textarea>`
 - iv. `radio_button: <input type="radio">`
 - v. `password_field: <input type="password">`
 - vi. `hidden_field: <input type="hidden">`
 - vii. `check_box: <input type="checkbox">`
 - viii. `fields_for`: FormBuilder interno com escopo definido no parâmetro
 - ix. `error_messages`: imprime mensagens de erro
- vii. Modularização de views: partial
1. Arquivos ERB começados por `_`
 2. `render :partial => <nome>`
 3. Remoção de código duplicado
 - a. Crie um arquivo `"_form.html.erb"` com o conteúdo dos formulários (linha depois do `form_for` até linha antes do `submit`)



UNIVERSIDADE FEDERAL DO PIAUÍ
CENTRO DE CIÊNCIAS DA NATUREZA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

- b. Substituir nas views por
 - i. `render :partial => 'form', :locals => { :f => f }`
- 4. Bem mais DRY
- viii. Layouts
 - 1. Se não for passado layout para o render, o rails procura em `apps/views/layouts` por `<controlador>.html.erb`
 - a. Se não encontrar procura `application.html.erb`
 - 2. Scaffold cria layout padrão
 - 3.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en"
lang="en">
<head>
  <meta http-equiv="content-type"
content="text/html;charset=UTF-8" />
  <title>Users: <%= controller.action_name %></title>
  <%= stylesheet_link_tag 'scaffold' %>
</head>
<body>
<p style="color: green"><%= flash[:notice] %></p>
<%= yield %>
</body>
</html>
```
 - 4. O controlador renderiza no ponto onde está o `yield`
- n. Precisando de ajuda para limpar o código das views?
 - i. Helpers são mixins que disponibilizam métodos para as views
 - 1. Normalmente retornam texto
 - 2. Para cada controlador existe um helper
 - 3. Existe ainda o `application_helper`
 - 4. Por padrão é criado em branco
 - 5.

```
module UserHelper
end
```
 - ii. Vamos adicionar um método no helper criado e utilizá-lo nas views do controlador `users`
 - 1.

```
module Userhelper
def my_input form, field
  %Q{
    <p>
      #{form.label field}<br />
```



UNIVERSIDADE FEDERAL DO PIAUÍ
CENTRO DE CIÊNCIAS DA NATUREZA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

- ```
 #{form.text_field field}
 </p>
 }
 end
end
```
- iii. Alteramos o partial (mais DRY)
    - 1. `<%= f.error_messages %>`
    - `<%= my_input f, :login %>`
    - `<%= my_input f, :password %>`
    - `<%= my_input f, :name %>`
    - `<%= my_input f, :email %>`
  - o. Configuração de rotas, um nome bonito para URLs
    - i. Devemos, ainda, entender como funciona o mapeamento de URLs
      - 1. Arquivo `config/routes.rb`
    - ii. Indicar qual controlador deve atender a qual requisição
    - iii. 2 formas
      - 1. Rotas REST do rails
      - 2. Mapeando explicitamente a rota, nomeando ou não
    - iv. `map.resource :users`
      - 1. Mapeia um recurso REST (Representational State Transfer)
      - 2. `rake routes`
      - 3. Gera algumas rotas nomeadas para um recurso
    - v. `map.connect ':controller/:action/:id'`  
`map.connect ':controller/:action/:id.:format'`
      - 1. Definição de rotas específicas para controladores
    - vi. `map.root`
      - 1. Rota para a página inicial
  - p. Continuando o desenvolvimento
    - i. Sistema: gerenciador de horas trabalhadas
      - 1. Já temos o modelo `user`. O que mais é necessário?
    - ii. Ponto de partida: `models` e `migrations`
    - iii. Precisaremos de `Project`, `TaskType`, `ProjectMembership` e `TimeLog`
    - iv. Comandos:
      - 1. `ruby script/generate model Project name:string description:text`
      - 2. `ruby script/generate model TaskType name:string project:belongs_to`
      - 3. `ruby script/generate model ProjectMembership joined:date leaved:date user:belongs_to project:belongs_to`
      - 4. `ruby script/generate model TimeLog user:belongs_to task_type:belongs_to description:text tak_date:date`



UNIVERSIDADE FEDERAL DO PIAUÍ  
CENTRO DE CIÊNCIAS DA NATUREZA  
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA

---

- v. São gerados models, migrations, tests e fixtures para cada model
  - 1. Por enquanto daremos prioridade a models e migrations
- q. Conferindo as migrations geradas
- r. Escrevendo modelos
- s. Gerando todo o código
- t. Um pouco de segurança na aplicação
- u. Um layout menos confuso para a aplicação
- v. Associando usuários a projetos
- w. Adicionando tipos de tarefas a um projeto
- x. Cadastro das horas trabalhadas
- y. Um relatório para a aplicação
- z. Limpando um pouco o código e se livrando um pouco do “inglês”