



3. Rails Básico

i. O primeiro controlador

- i. Publicar os models na internet
- ii. Não deve haver muito código nos controladores
 1. Chamar, no máximo, 1 ou 2 métodos dos models e encaminhar o resultado para uma view
 2. Pode mandar para views diferentes dependendo do resultado

iii. Código gerado pelo scaffold

1.

```
class UserController < ApplicationController
```

```
# GET /users
```

```
# GET /users.xml
```

```
def index
```

```
  @users = User.find(:all)
```

```
  respond_to do |format|
```

```
    format.html #index.html.erb
```

```
    format.xml { render :xml => @users }
```

```
  end
```

```
end
```

```
#GET /users/1
```

```
#GET /users/1.xml
```

```
def show
```

```
  @user = User.find(params[:id])
```

```
  respond_to do |format|
```

```
    format.html #show.html.erb
```

```
    format.xml { render :xml => @user }
```

```
  end
```

```
end
```

```
#GET /user/new
```

```
#GET /user/new/xml
```

```
def new
```

```
  @user = User.new
```

```
  respond_to do |format|
```

```
    format.html #new.html.erb
```

```
    format.xml { render :xml => @user }
```

```
  end
```

```
end
```

```
#GET /users/1/edit
```

```
def edit
```

```
  @user = User.find(params[:id])
```

```
end
```



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

```
#POST /users
#POST /users.xml
def create
  @user = User.new(params[:user])
  respond_to do |format|
    if @user.save
      flash[:notice] = "User was successfully created"
      format.html { redirect_to(@user) }
      format.xml { render :xml => @user, :status =>
:created, :location => @user }
    else
      format.html { render :action => "new" }
      format.xml { render :xml => @user.errores,
:status =>unprocessable_entity }
    end
  end
end
#PUT /users/1
#PUT /users/1.xml
def update
  @user = Post.find(params[:id])
  respond_to do |format|
    if @user.update_attributes(params[:user])
      flash[:notice] = 'User was successfully updated.'
      format.html { redirect_to(@user) }
      format.xml { head :ok }
    else
      format.html { render :action => "edit" }
      format.xml { render :xml => @user.errors,
:status => :unprocessable_entity }
    end
  end
end
# DELETE /users/1
# DELETE /users/1.xml
def destroy
  @user = Post.find(params[:id])
  @user.destroy
  respond_to do |format|
    format.html { redirect_to(users_url) }
    format.xml { head :ok }
  end
end
```



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

- end
- end
- end
- iv. Sete métodos-padrão
 1. index: GET ao endereço do controlador (busca a lista de usuários)
 2. show: GET ao controlador/id (busca o usuário do id recebido e encaminha para a view)
 3. new: GET ao controlador/new (cria um novo usuário e encaminha para a view)
 4. edit: GET ao controlador/id/edit (semelhante ao show, apenas mostra uma view diferente)
 5. create: POST ao controlador (cria um novo usuário e, caso sucesso vai para show, caso contrário, para new)
 6. update: PUT para controlador/id (atualiza um usuário, caso sucesso, vai para show, caso contrário, para edit)
 7. destroy: DELETE para controlador/id (apaga um usuário, caso ocorram problemas, mostra mensagem de erro)
- j. Métodos básicos do ActiveRecord
 - i. Models são implementados pelo ActiveRecord
 - ii. Importante conhecer funções
 1. User.find: consulta ao banco
 - a. Parâmetros: um id, :first, :last, :all
 - b. Opções
 - i. :conditions: WHERE do SQL
 - ii. :order: ORDER BY
 2. User.new: cria um novo user, se receber hash já preenche
 3. @user.destroy: elimina a instância
 4. @user.save: salva ou atualiza um registro
 5. @user.update_attributes: atualiza os valores de propriedades com valores recebidos no hash
- k. Recebendo parâmetros nos controladores
 - i. Hash "params"
 1. Valores recebidos no request
 - a. Valores simples: params[:id]
 - b. Hash de valores: params[:user]
- l. Respondendo a requisições
 - i. Métodos do controlador para responder a requisições
 1. respond_to: informa qual tipo de resposta será dada e para quem irá responder



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

2. `format.<tipo mime>`: como responder a cada tipo de requisição. Mesma action pode devolver html, js, xml, etc
3. `flash[:notice]`: salva valores em memória para usar no próximo request. Mensagens
4. `redirect_to`: enviar um redirect para o browser realizar uma nova requisição. Parâmetros:
 - a. `registro ActiveRecord`: redireciona para “show” do registro
 - b. URL absoluta: (`http://, /algo`, hash contendo `url_for (:action ou :controller)`)
 - c. `:back`: de volta de onde veio (`HTTP_REFERER`)
 - d. `:render`: renderiza views. Sem parâmetro vai para a view padrão `/app/views/<controlador>/<metodo>.<tipo mime>.erb`
 - i. `:action`: método específico
 - ii. `:layout`: substitui layout padrão
 - iii. `:template`: caminho completo do template
 - iv. `:text`: recebe o texto a ser renderizado
 - v. `:inline`: recebe código ERB para ser processado e renderizado
 - vi. `:xml`: renderiza XML como resposta
 - vii. `users_url`: nomes de rotas (`routes.rb`)
- m. As primeiras views
- n. Precisando de ajuda para limpar o código das views?
- o. Configuração de rotas, um nome bonito para URLs
- p. Continuando o desenvolvimento
- 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”