

CAPÍTULO 10

Completando o Sistema

"O êxito parece doce a quem não o alcança"
— Dickinson, Emily

10.1 – UM POUCO MAIS SOBRE O SCAFFOLD

Vimos que quando usamos o gerador `scaffold` o Rails cria os arquivos necessários em todas as camadas. **Controller**, **Model**, **Views** e até mesmo arquivos de testes e a **Migration**. Para concluir nosso projeto precisamos criar apenas **Controller + Views** pois tanto o modelo quanto as migrations já estão prontos. Para isso vamos usar o mesmo gerador `scaffold` mas vamos passar os parâmetros: `--migration=false` para ele ignorar a criação da migration e o parâmetro `-s` que é a abreviação para `--skip` que faz com que o rails "pule" os arquivos já existentes.

Para concluir os modelos que já começamos vamos executar o gerador da seguinte maneira: **`rails generate scaffold cliente nome:string idade:integer --migration=false -s`**

Outros parâmetros nos geradores

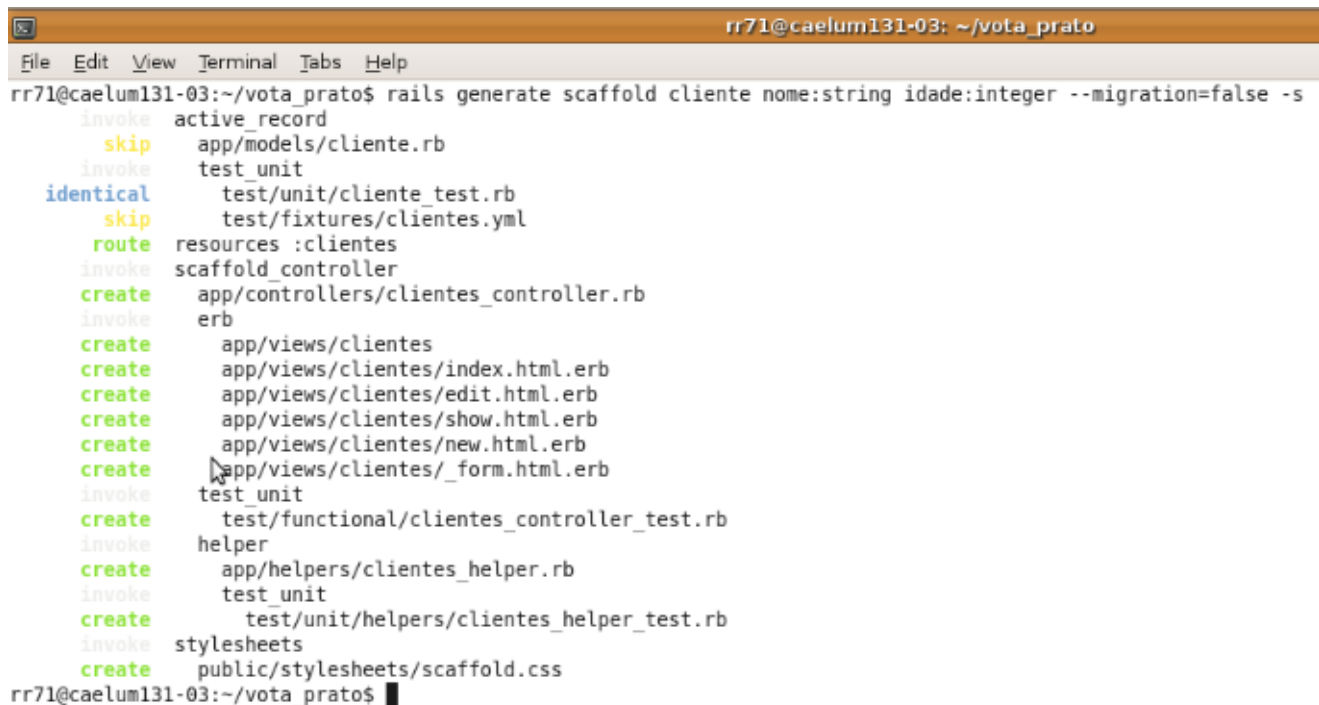
Para conhecer todas as opções de parâmetros que um gerador pode receber tente executá-lo passando o parâmetro `-h` Ex. **`rails generate scaffold -h`**

10.2 – EXERCÍCIOS: COMPLETANDO NOSSO DOMÍNIO

Vamos gerar os outros controllers e views usando o `scaffold`:

1. Primeiro vamos gerar o scaffold para **cliente**, no terminal execute:

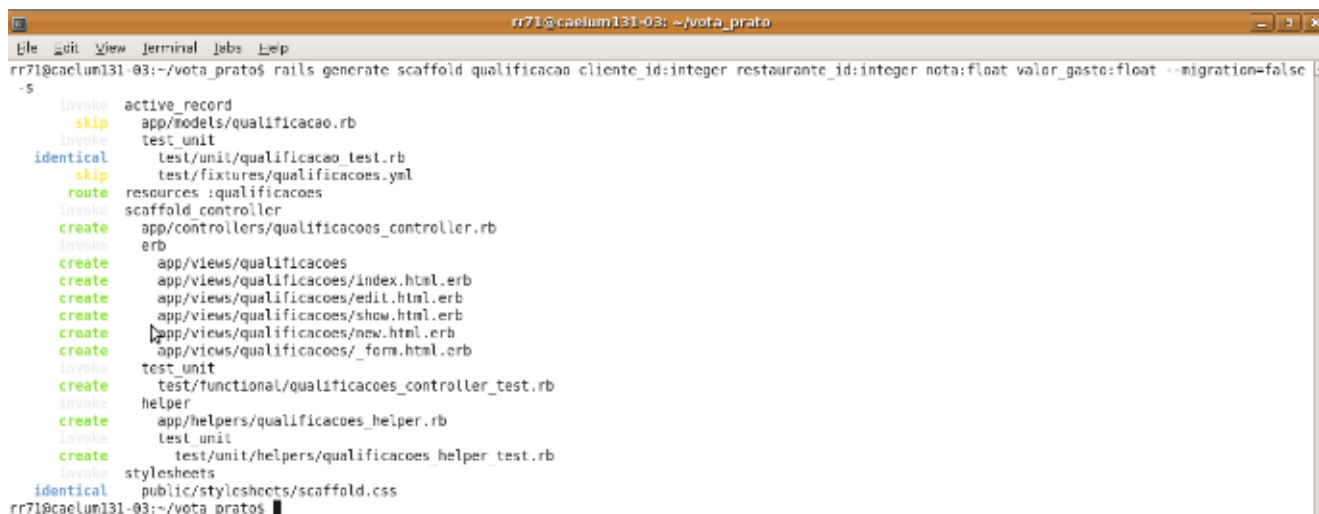
```
$ rails generate scaffold cliente nome idade:integer --migration=false -s
```



```
rr71@caelum131-03: ~/vota_prato
File Edit View Terminal Tabs Help
rr71@caelum131-03:~/vota_prato$ rails generate scaffold cliente nome:string idade:integer --migration=false -s
  invoke  active_record
  skip    app/models/cliente.rb
  invoke  test_unit
  identical test/unit/cliente_test.rb
  skip    test/fixtures/clientes.yml
  route   resources :clientes
  invoke  scaffold_controller
  create   app/controllers/clientes_controller.rb
  invoke  erb
  create   app/views/clientes
  create   app/views/clientes/index.html.erb
  create   app/views/clientes/edit.html.erb
  create   app/views/clientes/show.html.erb
  create   app/views/clientes/new.html.erb
  create   app/views/clientes/_form.html.erb
  invoke  test_unit
  create   test/functional/clientes_controller_test.rb
  invoke  helper
  create   app/helpers/clientes_helper.rb
  invoke  test_unit
  create   test/unit/helpers/clientes_helper_test.rb
  invoke  stylesheets
  create   public/stylesheets/scaffold.css
rr71@caelum131-03:~/vota_prato$
```

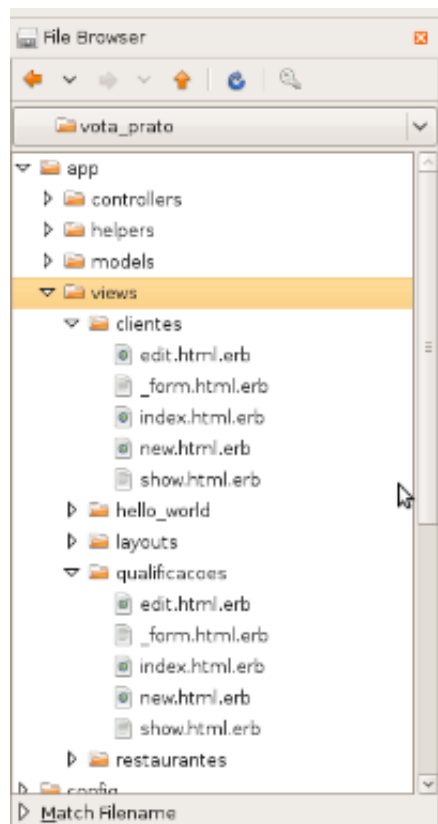
2. Agora vamos gerar o scaffold de **qualificacao**, execute:

```
$ rails generate scaffold qualificacao cliente_id:integer
  restaurante_id:integer nota:float valor_gasto:float
  --migration=false -s
```

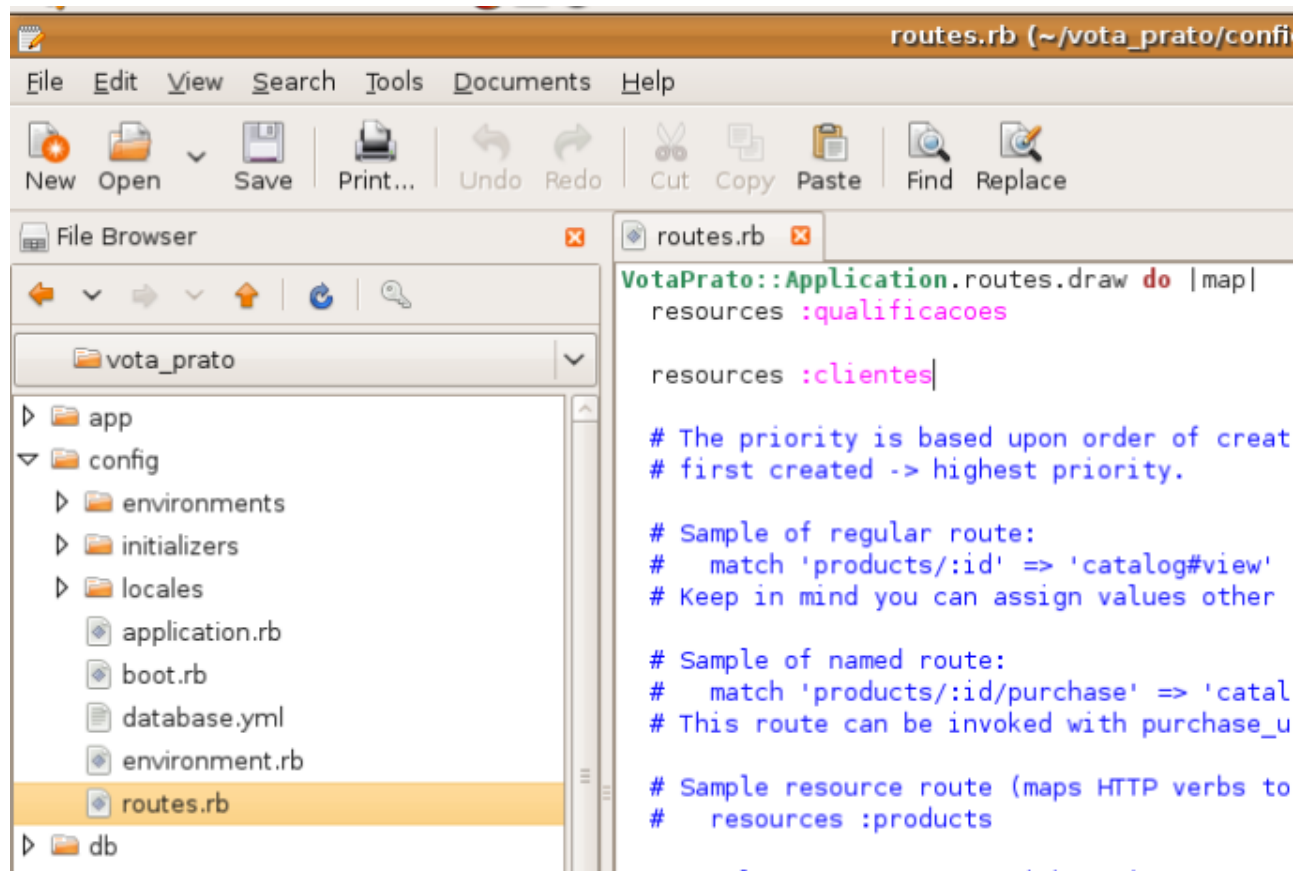


```
rr71@caelum131-03: ~/vota_prato
File Edit View Terminal Tabs Help
rr71@caelum131-03:~/vota_prato$ rails generate scaffold qualificacao cliente_id:integer restaurante_id:integer nota:float valor_gasto:float --migration=false -s
  invoke  active_record
  skip    app/models/qualificacao.rb
  invoke  test_unit
  identical test/unit/qualificacao_test.rb
  skip    test/fixtures/qualificacoes.yml
  route   resources :qualificacoes
  invoke  scaffold_controller
  create   app/controllers/qualificacoes_controller.rb
  invoke  erb
  create   app/views/qualificacoes
  create   app/views/qualificacoes/index.html.erb
  create   app/views/qualificacoes/edit.html.erb
  create   app/views/qualificacoes/show.html.erb
  create   app/views/qualificacoes/new.html.erb
  create   app/views/qualificacoes/_form.html.erb
  invoke  test_unit
  create   test/functional/qualificacoes_controller_test.rb
  invoke  helper
  create   app/helpers/qualificacoes_helper.rb
  invoke  test_unit
  create   test/unit/helpers/qualificacoes_helper_test.rb
  invoke  stylesheets
  identical public/stylesheets/scaffold.css
rr71@caelum131-03:~/vota_prato$
```

3. a. Olhe as views criadas (**app/views/clientes** e **app/views/qualificacoes**)



b. Olhe as rotas criadas (**config/routes.rb**)



c. Abra os arquivos **app/views/clientes/index.html.erb** e **app/views/restaurantes/index.html.erb** e apague as linhas que chamam a action **destroy** Lembre-se de que não queremos inconsistências na nossa tabela de qualificações

d. Reinicie o servidor

e. Teste: <http://localhost:3000/clientes> e <http://localhost:3000/qualificacoes>

Note que precisamos utilizar a opção "--migration=false" no comando scaffold, além de informar manualmente os atributos utilizados em nossas migrations. Isso foi necessário, pois já tínhamos um migration pronto, e queríamos que o Rails gerasse os formulários das views para nós, e para isso ele precisaria conhecer os atributos que queríamos utilizar.

css scaffold

O comando scaffold, quando executado, gera um css mais bonito para nossa aplicação. Se quiser utilizá-lo, edite nosso layout (`app/views/layouts/application.html.erb`) e adicione a seguinte linha logo abaixo da tag `<title>`:

```
<%= stylesheet_link_tag 'scaffold' %>
```

Nova editora Casa do Código com livros de uma forma diferente



Editoras tradicionais pouco ligam para ebooks e novas tecnologias. Não conhecem programação para revisar os livros tecnicamente a fundo. Não têm anos de experiência em didáticas com cursos.

Conheça a **Casa do Código**, uma editora diferente, com curadoria da **Caelum** e obsessão por livros de qualidade a preços justos.

[Casa do Código, ebook com preço de ebook.](#)

10.3 – SELECIONANDO CLIENTES E RESTAURANTE NO FORM DE QUALIFICAÇÕES

Você já deve ter reparado que nossa view de adição e edição de qualificações está um tanto quanto estranha: precisamos digitar os IDs do cliente e do restaurante manualmente.

Para corrigir isso, podemos utilizar o **FormHelper** select, inserindo o seguinte código nas nossas views de adição e edição de qualificações:

```
<%= select('qualificacao', 'cliente_id',  
          Cliente.order(:nome)  
          {|p| [ p.nome, p.id]}) %>
```

em substituição ao:

```
<%= f.number_field :cliente_id %>
```

Mas existe um outro **FormHelper** mais elegante, que produz o mesmo efeito, o `collection_select`:

```
<%= collection_select(:qualificacao, :cliente_id,  
  Cliente.order(:nome),  
  :id, :nome, :prompt => true) %>
```

Como estamos dentro de um `form_for`, podemos usar do fato de que o formulário sabe qual o nosso ActiveRecord, e com isso fazer apenas:

```
<%= f.collection_select(:cliente_id,  
  Cliente.order(:nome),  
  :id, :nome, :prompt => true) %>
```

10.4 – EXERCÍCIOS: FORMULÁRIO COM COLLECTION_SELECT

1. Vamos utilizar o **FormHelper** `collection_select` para exibirmos o nome dos clientes e restaurantes nas nossas views da qualificação:

a. Abra o arquivo `app/views/qualificacoes/_form.html.erb`

b. Troque a linha:

```
<%= f.number_field :cliente_id %>
```

por:

```
<%= f.collection_select(:cliente_id, Cliente.order(:nome),  
  :id, :nome, :prompt => true) %>
```

c. Troque a linha:

```
<%= f.number_field :restaurante_id %>
```

por:

```
<%= f.collection_select(:restaurante_id, Restaurante.order(:nome),  
  :id, :nome, :prompt => true) %>
```

d. Teste: <http://localhost:3000/qualificacoes>

10.5 – EXERCÍCIOS OPCIONAIS: REFATORANDO PARA RESPEITARMOS O MVC

A forma como implementamos o seletor de restaurante e cliente não é a mais adequada pois ela fere o modelo arquitetural que o rails se baseia, o MVC. Isso ocorre, pois estamos invocando `Cliente.order(:nome)` dentro da view, ou seja, a view está se comunicando com o modelo.

1. Vamos arrumar isso, criando as variáveis de instância `@restaurantes` e `@clientes` nas actions **new** e **edit** do **QualificacoesController**:

a. Abra o **QualificacoesController**.

(**app/controllers/qualificacoes_controller.rb**)

b. Crie as variáveis de instância nas primeiras linhas da action **new**:

```
def new
  @clientes = Cliente.order :nome
  @restaurantes = Restaurante.order :nome

  # resto do código
end
```

c. Crie as variáveis de instância nas primeiras linhas da action **edit**:

```
def edit
  @clientes = Cliente.order :nome
  @restaurantes = Restaurante.order :nome

  # resto do código
end
```

2. Agora podemos usar as variáveis de instância no nosso partial **form**.

a. Abra o partial **form** (**app/views/qualificacoes/_form.html.erb**)

b. Substitua as antigas chamadas do `select_collection` por:

```
f.collection_select(:restaurante_id, @restaurantes,
                  :id, :nome, :prompt => true)
```

e

```
f.collection_select(:restaurante_id, @clientes,
                  :id, :nome, :prompt => true)
```

3. Observe que temos código duplicado no nosso **QualificacoesController**, vamos

refatorar extraíndo a criação das variáveis de instância para um método privado,

a. Abra o controller de qualificacoes

(**app/controllers/qualificacoes_controller.rb**).

b. Ao final da classe, crie um método privado que cria as variáveis de instância:

```
class QualificacoesController < ApplicationController
  # todas as actions

  private
  def preparar_form
    @clientes = Cliente.order :nome
    @restaurantes = Restaurante.order :nome
  end
end
```

c. Ainda no controller de qualificacoes, vamos usar o preparar_form na action **new**. Dentro da action **new**, substitua:

```
@clientes = Cliente.order :nome
@restaurantes = Restaurante.order :nome
```

por:

```
preparar_form
```

d. Repita o exercício acima na action **edit**.

4. As actions **new** e **edit** estão funcionando perfeitamente. Vamos analisar a action **create**. Observe que caso dê erro de validação, ela também irá renderizar a view **new.html.erb**. Logo, é necessário criarmos as variáveis de instância **@restaurantes** e **@clientes** também na action **create**.

a. Abra o controller de qualificacoes

(**app/controllers/qualificacoes_controller.rb**).

b. Altere a action **create** para chamar o **preparar_form**:

```
def create
  @qualificacao = Qualificacao.new(qualificacao_params)

  respond_to do |format|
    if @qualificacao.save
      format.html { redirect_to @qualificacao,
        notice: 'Qualificacao was successfully created.' }
      format.json { render json: @qualificacao,
        status: :created, location: @qualificacao }
    else
      preparar_form
      format.html { render action: "new" }
      format.json { render json: @qualificacao.errors, status:
```

```
:unprocessable_entity }  
  end  
end  
end
```

5. O mesmo problema da action **create** ocorre na action **update**, portanto vamos alterá-la.

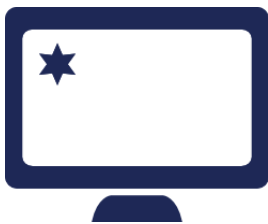
a. Abra o controller de qualificacoes
(**app/controllers/qualificacoes_controller.rb**).

b. Altere a action **update** para chamar o **preparar_form**:

```
def update  
  @qualificacao = Qualificacao.find(params[:id])  
  
  respond_to do |format|  
    if @qualificacao.update_attributes(qualificacao_params)  
      format.html { redirect_to @qualificacao,  
        notice: 'Qualificacao was successfully updated.' }  
      format.json { head :no_content }  
    else  
      preparar_form  
      format.html { render action: "edit" }  
      format.json { render json: @qualificacao.errors,  
        status: :unprocessable_entity }  
    end  
  end  
end  
end
```

6. Inicie o servidor (rails server) e tente criar uma qualificação inválida para testar nossa correção.

Já conhece os cursos online Alura?



A **Alura** oferece dezenas de **cursos online** em sua plataforma exclusiva de ensino que favorece o aprendizado com a **qualidade** reconhecida da Caelum. Você pode escolher um curso nas áreas de Java, Ruby, Web, Mobile, .NET e outros, com uma **assinatura** que dá acesso a todos os cursos.

[Conheça os cursos online Alura.](#)

10.6 – EXERCÍCIOS OPCIONAIS: EXIBINDO NOMES AO INVÉS DE NÚMEROS

1. Agora vamos exibir o nome dos restaurantes e clientes nas views **index** e **show** de qualificações:

a. Abra o arquivo **app/views/qualificacoes/show.html.erb**

b. Troque a linha:

```
<%= @qualificacao.cliente_id %>
```

por:

```
<%= @qualificacao.cliente.nome %>
```

c. Troque a linha:

```
<%= @qualificacao.restaurante_id %>
```

por:

```
<%= @qualificacao.restaurante.nome %>
```

d. Abra o arquivo **app/views/qualificacoes/index.html.erb**

e. Troque as linhas:

```
<td><%= qualificacao.cliente_id %></td>
<td><%= qualificacao.restaurante_id %></td>
```

por:

```
<td><%= qualificacao.cliente.nome %></td>
<td><%= qualificacao.restaurante.nome %></td>
```

f. Teste: <http://localhost:3000/qualificacoes>

2. Por fim, vamos utilizar o FormHelper `hidden_field` para permitir a qualificação de um restaurante a partir da view **show** de um cliente ou de um restaurante. No entanto, ao fazer isso, queremos que não seja necessário a escolha de cliente ou restaurante. Para isso:

a. Abra o arquivo **app/views/qualificacoes/_form.html.erb**

b. Troque as linhas

```
<p>
  <%= f.label :cliente_id %><br />
  <%= f.collection_select(:cliente_id,
    @clientes,
    :id, :nome, :prompt => true) %>
</p>
```

por:

```
<% if @qualificacao.cliente %>
  <%= f.hidden_field 'cliente_id' %>
<% else %>
  <p><%= f.label :cliente_id %><br />
  <%= f.collection_select(:cliente_id, @clientes,
                        :id, :nome, :prompt => true) %></p>
<% end %>
```

c. Troque as linhas

```
<p>
  <%= f.label :restaurante_id %><br />
  <%= f.collection_select(:restaurante_id, @restaurantes,
                        :id, :nome, :prompt => true) %>
</p>
```

por:

```
<% if @qualificacao.restaurante %>
  <%= f.hidden_field 'restaurante_id' %>
<% else %>
  <p><%= f.label :restaurante_id %><br />
  <%= f.collection_select(:restaurante_id, @restaurantes,
                        :id, :nome, :prompt => true) %>
</p>
<% end %>
```

d. Adicione a seguinte linha na view **show** do cliente
(**app/views/clientes/show.html.erb**):

```
<%= link_to "Nova qualificação", controller: "qualificacoes",
          action: "new",
          cliente: @cliente %>
```

e. Adicione a seguinte linha na view **show** do restaurante
(**app/views/restaurantes/show.html.erb**):

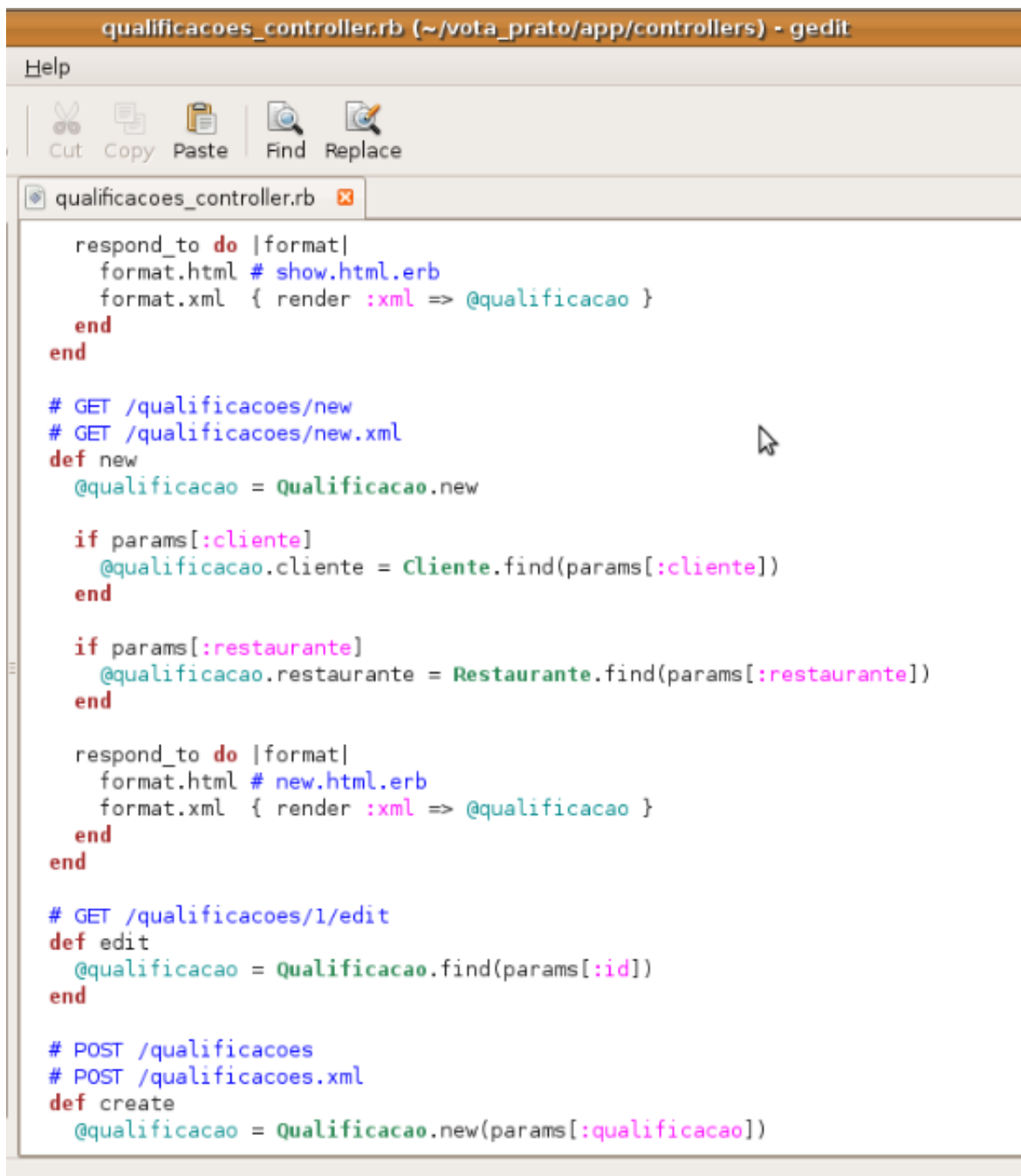
```
<%= link_to "Qualificar este restaurante", controller: "qualificacoes",
          action: "new",
          restaurante: @restaurante %>
```

f. Por fim, precisamos fazer com que o controlador da action `new` das qualificações receba os parâmetros para preenchimento automático. Abra o controller **app/controllers/qualificacoes_controller.rb**

g. Adicione as seguintes linhas à nossa action `new`:

```
if params[:cliente]
  @qualificacao.cliente = Cliente.find(params[:cliente])
end
if params[:restaurante]
  @qualificacao.restaurante = Restaurante.find(params[:restaurante])
end
```

end



```
qualifikacoes_controller.rb (~/vota_prato/app/controllers) - gedit
Help
Cut Copy Paste Find Replace
qualifikacoes_controller.rb
respond_to do |format|
  format.html # show.html.erb
  format.xml { render :xml => @qualificacao }
end
end

# GET /qualifikacoes/new
# GET /qualifikacoes/new.xml
def new
  @qualificacao = Qualificacao.new

  if params[:cliente]
    @qualificacao.cliente = Cliente.find(params[:cliente])
  end

  if params[:restaurante]
    @qualificacao.restaurante = Restaurante.find(params[:restaurante])
  end

  respond_to do |format|
    format.html # new.html.erb
    format.xml { render :xml => @qualificacao }
  end
end

# GET /qualifikacoes/1/edit
def edit
  @qualificacao = Qualificacao.find(params[:id])
end

# POST /qualifikacoes
# POST /qualifikacoes.xml
def create
  @qualificacao = Qualificacao.new(params[:qualificacao])
```

h. Teste: <http://localhost:3000/clientes>, entre na página Show de um cliente e faça uma nova qualificação.

10.7 – MAIS SOBRE OS CONTROLLERS

Podemos notar que nossas actions, por exemplo a index, fica muito parecida com a action index de outros controladores, mudando apenas o nome do modelo em questão.

```
#qualifikacoes_controller
def index
  @qualifikacoes = Qualificacao.all
```

```

    respond_to do |format|
      format.html # index.html.erb
      format.xml { render :xml => @qualificacoes }
    end
  end

#clientes_controller
def index
  @clientes = Cliente.all

  respond_to do |format|
    format.html # index.html.erb
    format.xml { render :xml => @clientes }
  end
end

```

Normalmente precisamos fazer exatamente a mesma ação para formatos iguais e por isso acabamos repetindo o mesmo bloco de `respond_to` nas actions. Para solucionar esse problema, no rails 3 acrescentaram o método `respond_to` **nos controllers** e o método `respond_with` **nas actions**. Veja o exemplo:

```

class ClientesController < ApplicationController

  respond_to :html, :xml

  # GET /clientes
  # GET /clientes.xml
  def index
    @clientes = Cliente.all

    respond_with @clientes
  end

  # GET /clientes/1
  # GET /clientes/1.xml
  def show
    @cliente = Cliente.find(params[:id])

    respond_with @cliente
  end
  ...
end

```

Dessa forma estamos dizendo para o rails que esse controller irá responder para os formatos html e xml, dentro da action basta eu dizer qual objeto é pra ser usado. No caso da action **index**, se a requisição pedir o formato html, o rails simplesmente vai enviar a chamada para o arquivo `views/clientes/index.html.erb` e a variável `@clientes` estará disponível lá. Se a requisição pedir o formato xml, o rails fará exatamente o mesmo que estava no bloco de `respond_to` que o scaffold criou, vai renderizar a variável `@clientes` em xml. O bloco de código acima é equivalente ao gerado pelo scaffold:

```

class ClientesController < ApplicationController
  # GET /clientes
  # GET /clientes.xml
  def index
    @clientes = Cliente.all

    respond_to do |format|
      format.html # index.html.erb
      format.xml { render :xml => @clientes }
    end
  end

  # GET /clientes/1
  # GET /clientes/1.xml
  def show
    @cliente = Cliente.find(params[:id])

    respond_to do |format|
      format.html # show.html.erb
      format.xml { render :xml => @cliente }
    end
  end
  ...
end

```

Veja na imagem como ficaria o **ClientesController** usando essa outra maneira de configurar os controllers.

```
clientes_controller.rb
class ClientesController < ApplicationController

  respond_to :html, :xml

  # GET /clientes
  # GET /clientes.xml
  def index
    @clientes = Cliente.all

    respond_with @clientes
  end

  # GET /clientes/1
  # GET /clientes/1.xml
  def show
    @cliente = Cliente.find(params[:id])

    respond_with @cliente
  end

  # GET /clientes/new
  # GET /clientes/new.xml
  def new
    @cliente = Cliente.new

    respond_with @cliente
  end

  # GET /clientes/1/edit
  def edit
    @cliente = Cliente.find(params[:id])
  end

  # POST /clientes
```

CAPÍTULO ANTERIOR:

[Controllers e Views](#)

PRÓXIMO CAPÍTULO:

[Calculations](#)

Você encontra a Caelum também em:

Blog Caelum

Cursos Online

Facebook

Newsletter

Casa do Código

Twitter