INSTITUTO FEDERAL DE
EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
RIO GRANDE DO NORTE

Enviando E-mail
(Tarefa H)

Prof. Fellipe Aleixo (*fellipe.aleixo@ifrn.edu.br*)

Feedback do Cliente

- Fechamos o ciclo até a finalização da venda
  - Mas, que tal habilitarmos o envio de e-mail para alguém quando certo eventos acontecerem?

- Por exemplo:
  - Avisar ao administrador ao ocorrer uma exceção
  - Notificar sobre um feedback do usuário
  - Confirmar o registro de um pedido de venda

Iteração H1:

ENVIANDO E-MAILS DE CONFIRMAÇÃO

E-mails em Rails

- Três etapas básicas:
  - (1) Configurar como o e-mail será enviado
  - (2) Determinar quando enviar o e-mail
  - (3) Especificar o que deseja-se notificar

Configurando o Envio de E-mails

- Tal configuração é parte das configurações de uma aplicação Rails
  - No caso específico - **Depot::Application.configure**
- Atente para as necessidades de configurações específicas para (i) desenvolvimento, (ii) teste e (iii) produção
  - Arquivo (configuração única) - **environment.rb**
  - Demais arquivos - **config/environments**

Configurando o Envio de E-mails

- Como deseja-se enviar e-mails:
  - **config.action_mailer.delivery_method = :smtp**

- Algumas alternativas:
  - **sendmail**
  - **test**
    - Simula o envio de e-mail, os quais são armazenados **ActionMailer::Base.deliveries**

Configurando o Envio de E-mails

- Configurações adicionais – como acessar o servidor SMTP para o envido dos e-mails

```
Depot::Application.configure do
  config.action_mailer.delivery_method = :smtp

  config.action_mailer.smtp_settings = {
    address:        "smtp.gmail.com",
    port:           587,
    domain:         "domain.of.sender.net",
    authentication: "plain",
    user_name:      "dave",
    password:       "secret",
    enable_starttls_auto: true
  }
end
```

Enviando um E-mail

- Rails possui um gerador para criar *mailers*
  - Armazenado em **app/mailers**
  - Pode possui um ou mais métodos – cada método representando um *template* de e-mail
  - Para criar o corpo de um e-mail, esses métodos se utilizam de **visões** específicas
- Na aplicação exemplo será criado um *mailer* que envia dois tipos de e-mails diferentes

Gerando um *Mailer*

- Script gerador de *mailer*

```
depot> rails generate mailer OrderNotifier received shipped
  create  app/mailers/order_notifier.rb
  invoke  erb
  create    app/views/order_notifier
  create    app/views/order_notifier/received.text.erb
  create    app/views/order_notifier/shipped.text.erb
  invoke  test_unit
  create    test/mailers/order_notifier_test.rb
```

Alterando o *Mailer*

Download rails40/depot_q/app/mailers/order_notifier.rb

```
class OrderNotifier < ActionMailer::Base
  default from: 'Sam Ruby <depot@example.com>'
  # Subject can be set in your I18n file at config/locales/en.yml
  # with the following lookup:
  #
  #   en.order_notifier.received.subject
  #
  def received
    @greeting = "Hi"

    mail to: "to@example.org"
  end
  ...
```

Alterando o *Mailer*

```
  # Subject can be set in your I18n file at config/locales/en.yml
  # with the following lookup:
  #
  #   en.order_notifier.shipped.subject
  #
  def shipped
    @greeting = "Hi"

    mail to: "to@example.org"
  end
end
```

Alterando o *Mailer*

- Similar a um controlador
  - Ao invés de invocar **render** – invoca-se **mail**
  - O método mail aceita vários parâmetros – **:to**, **:cc**, **:from**, **:subject**, entre outros

Templates de E-mail

- As “visões” geradas são *templates* segundo a extensão **.erb**
  - Eles são utilizados para criar arquivos de texto
    - Combinação de elementos estáticos e dinâmicos
  - Também podem ser utilizados para criar e-mails no formato HTML

Templates de E-mail

Download rails40/depot_q/app/views/order_notifier/received.text.erb

```
Dear <%= @order.name %>

Thank you for your recent order from The Pragmatic Store.

You ordered the following items:

<%= render @order.line_items -%>

We'll send you a separate e-mail when your order ships.
```

- Também existe a figura de *templates* parciais, específicos para a geração de texto

Templates de E-mail

- Parcial de linha de item (texto)

Download rails40/depot_q/app/views/line_items/_line_item.text.erb

```
<%= sprintf("%2d x %s",
            line_item.quantity,
            truncate(line_item.product.title, length: 50)) %>
```

- Tais visões e parciais podem fazer uso dos métodos auxiliares
  - ex.: **trucate**

Alterando o *Mailer*

- Alterando o método **received**

Download rails40/depot_r/app/mailers/order_notifier.rb

```
def received(order)
  @order = order

  mail to: order.email, subject: 'Pragmatic Store Order Confirmation'
end
```

Gerando um E-mail

Download rails40/depot_r/app/controllers/orders_controller.rb

```
def create
  @order = Order.new(order_params)
  @order.add_line_items_from_cart(@cart)

  respond_to do |format|
    if @order.save
      Cart.destroy(session[:cart_id])
      session[:cart_id] = nil
➤     OrderNotifier.received(@order).deliver
      format.html { redirect_to store_url, notice:
        'Thank you for your order.' }
      format.json { render action: 'show', status: :created,
        location: @order }
    else
      format.html { render action: 'new' }
      format.json { render json: @order.errors,
        status: :unprocessable_entity }
    end
  end
end
```

Alterando o *Mailer*

- Alterando o método **shipped**

Download rails40/depot_r/app/mailers/order_notifier.rb

```
def shipped(order)
  @order = order

  mail to: order.email, subject: 'Pragmatic Store Order Shipped'
end
```

Múltiplos Tipos de Conteúdo

- Podem ser utilizados além do formato de texto (*text/plain*), o formato HTML ou outros

Download rails40/depot_r/app/views/order_notifier/shipped.html.erb

```
<h3>Pragmatic Order Shipped</h3>
<p>
  This is just to let you know that we've shipped your recent order:
</p>

<table>
  <tr><th colspan="2">Qty</th><th>Description</th></tr>
<%= render @order.line_items -%>
</table>
```

Múltiplos Tipos de Conteúdo

- Nesse caso é preciso alterar a parcial de linha de item (HTML), pois vai ser usada a original

Download rails40/depot_r/app/views/line_items/_line_item.html.erb

```
<% if line_item == @current_item %>
<tr id="current_item">
<% else %>
<tr>
<% end %>
  <td><%= line_item.quantity %>&times;</td>
  <td><%= line_item.product.title %></td>
  <td class="item_price"><%= number_to_currency(line_item.total_price) %></td>
</tr>
```

Testando o E-mail

- Adaptando o teste criado pelo script gerador

Download rails40/depot_r/test/mailers/order_notifier_test.rb

```
require 'test_helper'

class OrderNotifierTest < ActionMailer::TestCase
  test "received" do
    mail = OrderNotifier.received(orders(:one))
    assert_equal "Pragmatic Store Order Confirmation", mail.subject
    assert_equal ["dave@example.org"], mail.to
    assert_equal ["depot@example.com"], mail.from
    assert_match /1 x Programming Ruby 1.9/, mail.body.encoded
  end

  ...
```

Testando o E-mail

```
  test "shipped" do
➤   mail = OrderNotifier.shipped(orders(:one))
➤   assert_equal "Pragmatic Store Order Shipped", mail.subject
➤   assert_equal ["dave@example.org"], mail.to
➤   assert_equal ["depot@example.com"], mail.from
➤   assert_match /<td>1&times;<\/td>\s*<td>Programming Ruby 1.9<\/td>/,
➤     mail.body.encoded
  end

end
```

Iteração H2:

TESTE DE INTEGRAÇÃO DE APLICAÇÕES

Teste de Integração

- Rails possibilita os testes (**i**) de modelo (unitários), (**ii**) dos controladores (funcional) e (**iii**) de integração
  - Teste unitário das classes de modelo
    - Classe com a lógica do negócio
  - Teste funcional dos controladores
    - Controladores coordenam o fluxo da aplicação
  - Próximo nível – testar o "fluxo" **através** da aplicação – teste das estórias do usuário

Teste de Integração

- Exemplo (de estória do usuário):
  - Um usuário acessa a página inicial da aplicação (Depot). Ele seleciona um produto, adicionado o mesmo ao carrinho de compras. Na sequência, realiza o *checkout*, preenchendo os detalhes do pedido de compra, no formulário apropriado. Ao submeter, um pedido é criado no banco de dados, contendo a linha de item associada ao carrinho de compras. Uma vez que o pedido foi efetivado, um e-mail é enviado confirmando a compra.

Teste de Integração

- Um teste de integração simula uma sessão contínua de um ou mais usuários virtuais
  - Estes são utilizados para enviar requisições, monitorar as respostas, seguir os redirecionamentos e assim por diante

- Não são criados automaticamente, mas é possível fazê-lo através de um script gerador

Teste de Integração

- Criando o teste de integração

```
depot> rails generate integration_test user_stories
      invoke  test_unit
      create    test/integration/user_stories_test.rb
```

```ruby
require 'test_helper'

class UserStoriesTest < ActionDispatch::IntegrationTest
  # test "the truth" do
  #   assert true
  # end
end
```

Teste de Integração

- Para testar a “estória do usuário” é necessário carregar os dados (fixtures) necessários

```
fixtures :products
```

- Criando o teste “comprando um produto”
  - Finalizar com uma ordem criada na tabela correspondente, associada à linha de item desejada

Teste de Integração

- Apagando o banco para assegurar o teste

Download rails40/depot_r/test/integration/user_stories_test.rb

```
LineItem.delete_all
Order.delete_all
ruby_book = products(:ruby)
```

- Atacando a primeira sentença da “estória”

Download rails40/depot_r/test/integration/user_stories_test.rb

```
get "/"
assert_response :success
assert_template "index"
```

Teste de Integração

- Até esse ponto, ele se assemelha a um teste funcional – mas, não para por aí...
  - Diferença: acesso pela URL (relativa) completa
- Testando a segunda sentença da estória

Download rails40/depot_r/test/integration/user_stories_test.rb

```
xml_http_request :post, '/line_items', product_id: ruby_book.id
assert_response :success

cart = Cart.find(session[:cart_id])
assert_equal 1, cart.line_items.size
assert_equal ruby_book, cart.line_items[0].product
```

Teste de Integração

- Solicitando o “checkout”

Download rails40/depot_r/test/integration/user_stories_test.rb

```
get "/orders/new"
assert_response :success
assert_template "new"
```

  - Nesse ponto, o usuário precisa preencher os detalhes do pedido de compra

Teste de Integração

- Simulando a submissão de um formulário

Download rails40/depot_r/test/integration/user_stories_test.rb

```
post_via_redirect "/orders",
                  order: { name:     "Dave Thomas",
                           address:  "123 The Street",
                           email:    "dave@example.com",
                           pay_type: "Check" }
assert_response :success
assert_template "index"
cart = Cart.find(session[:cart_id])
assert_equal 0, cart.line_items.size
```

Teste de Integração

- Nesse ponto, verificamos se os dados foram armazenados de forma correta em banco

Download rails40/depot_r/test/integration/user_stories_test.rb

```
orders = Order.all
assert_equal 1, orders.size
order = orders[0]

assert_equal "Dave Thomas",      order.name
assert_equal "123 The Street",   order.address
assert_equal "dave@example.com", order.email
assert_equal "Check",            order.pay_type

assert_equal 1, order.line_items.size
line_item = order.line_items[0]
assert_equal ruby_book, line_item.product
```

Teste de Integração

- Por fim, verificamos se o e-mail foi endereçado e tem o assunto corretos

Download rails40/depot_r/test/integration/user_stories_test.rb

```
mail = ActionMailer::Base.deliveries.last
assert_equal ["dave@example.com"], mail.to
assert_equal 'Sam Ruby <depot@example.com>', mail[:from].value
assert_equal "Pragmatic Store Order Confirmation", mail.subject
```

Testes

- Trabalhando juntos, os testes unitários, funcionais e de integração
  - Oferecem a flexibilidade de testar aspectos diferentes da aplicação com isolamento e combinação

EXTRAS a Serem Tentados

1. Adicionar a coluna data_de_envio (ship_date) na tabela de pedido – e enviar uma notificação (e-mail) quando este valor for alterado pelo controlador de pedido
2. Possibilitar o envio de um e-mail ao administrador do sistema quando um erro for capturado pela aplicação
3. Incluir testes de integração para os itens anteriores