02

# Notificando empresas via e-mail

## Notificando empresas via e-mail

Durante o capítulo 5 aprendemos a enviar formulários utilizando AJAX através da integração do Rails com Javascript. No decorrer deste capítulo vamos notificar as empresas via e-mail quando seus *jobs* recebem comentários, e aprenderemos a enviar e-mails em diferentes formatos tornando-os mais compatíveis com diferentes clientes de e-mail.

## Envio de e-mails com Action Mailer

Para começar vamos utilizar um *generator* do Rails para criar um **mailer** para nós. Os *mailers* são os responsáveis por enviar e-mails em aplicações Rails. No terminal, execute:

```
$ rails g mailer CompanyMailer new_comment
```

E você deverá ver o *mailer* sendo criado em *app/mailers*:

```
      create  app/mailers/company_mailer.rb
      invoke  erb
      create    app/views/company_mailer
      create    app/views/company_mailer/new_comment.text.erb
      invoke  test_unit
      create    test/functional/company_mailer_test.rb
```

O nosso *mailer* chama-se `CompanyMailer`, é terá um e-mail que denominamos `new_comment`. Abra este *mailer* em seu editor, ele deve se parecer com isso:

```
class CompanyMailer < ActionMailer::Base
  default from: "from@example.com"

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

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

Assim como os *models* do Active Record herdam de `ActiveRecord::Base`, os *mailers* herdam de `ActionMailer::Base`, que provê toda a funcionalidade de envio de e-mails para nós, assim podemos nos preocupar apenas com a parte importante: o conteúdo dos e-mails que queremos enviar.

A primeira linha da classe que define nosso *mailer* configura alguns *defaults* para os e-mails enviados por ele. No mínimo devemos definir o campo **from**, que precisamos modificar para o endereço de origem que aparecerá para quem receber os e-mails. Por agora vamos alterar para nosso endereço de e-mail:

```
default from: "carlos.antonio@plataformatec.com.br"
```

Depois temos o método `new_comment`, que é o nome que demos ao e-mail que enviaremos quando um novo comentário for criado. Esse método funciona como uma *action* de um *controller*: ela deve setar as variáveis de instância que serão utilizadas na *view* que será renderizada por esse *mailer*. A única diferença é que ao final do método chamamos `mail`, que é o responsável por gerar e retornar o objeto do e-mail para conseguirmos enviá-lo posteriormente.

Vamos alterar este método para receber o *job* e o *comment*, armazená-los em variáveis e enviar o e-mail para a empresa associada ao *job*:

```
def new_comment(job, comment)
  @job     = job
  @comment = comment
  @company = job.company

  mail to: @company.email
end
```

Perceba que alteramos o argumento `to` do método `mail` para o e-mail da empresa que receberá nossa notificação. Também podemos definir o **assunto** do e-mail através da opção `subject`:

```
mail to: @company.email, subject: "New comment received"
```

Pronto, vamos agora escrever a mensagem de e-mail que será enviada para a empresa, com o conteúdo do comentário criado. Abra a *view app/views/company_mailer/new_comment.text.erb*, e altere como abaixo:

```
Hello <%= @company.name %>,

You've got a new comment about your <%= @job.title %> job:

Name: <%= @comment.name %>
Body: <%= @comment.body %>

You can see all comments for this job in <%= job_url(@job) %>.

Thank you!
```

É um texto bem simples que exibe o nome e mensagem deixados no comentário, além da url para acessar a página do *job*. É **extremamente** importante que você utilize as *named routes* do Rails terminando em **_url** ao invés de **_path** para gerar as urls, pois quando o usuário estiver lendo os e-mails, ele estará completamente fora do contexto da aplicação, dessa forma os links devem apontar para o endereço completo da página, e não serem apenas endereços relativos. Ou seja, *job_url* gerará uma url como *http://localhost:3000/jobs/1 (http://localhost:3000/jobs/1)*, enquanto que *jobs_path* será apenas */jobs/1*, e dentro do e-mail este último não faz muito sentido.

Tudo pronto! Vamos abrir um console do Rails e testar o envio do e-mail:

```
>> job = Job.find 1
=> #<Job id: 1, title: "Developer", description: "Good guy.", created_at: "2013-02-18 01:24:11"
>> comment = job.comments.first
=> #<Comment id: 2, name: "Plataformatec", body: "Awesome!", job_id: 1, created_at: "2013-02-18
>> CompanyMailer.new_comment(job, comment).deliver
ActionView::Template::Error: Missing host to link to! Please provide the :host parameter, set de
...
```

Estamos chamando o método `new_comment` diretamente no `CompanyMailer`, passando o `job` e o `comment` que desejamos, e chamando `deliver` ao final para efetivamente enviar o e-mail, contudo recebemos um erro! O que acontece é que precisamos gerar urls com o endereço completo como comentamos acima, mas o *mailer* não sabe que endereço usar para montar essas urls. Dentro de uma *view* normal que está sendo processada pela aplicação durante uma requisição, a aplicação reutiliza o endereço atual para montar as urls quando necessário, como por exemplo o endereço *localhost:3000* usado em desenvolvimento. Porém dentro do contexto de um e-mail, informações da requisição não podem ser usadas pois não há garantia de que exista uma requisição acontecendo, por exemplo o e-mail pode estar sendo criado e enviado através de algum processo separado em um segundo momento, onde não existe uma requisição envolvida.

No caso de e-mails, devemos configurar o endereço que será usado para gerar as urls, e isto deve ser definido para cada ambiente, ou seja, configuramos um endereço para desenvolvimento, teste e produção. Vamos setar apenas em desenvolvimento por enquanto, então abra o arquivo *config/environments/development.rb* e adicione o seguinte ao final, dentro do bloco `configure`:

```
config.action_mailer.default_url_options = { host: "localhost:3000" }
```

Isto vai indicar para o Rails que as urls geradas pelos *mailers* usem o endereço *localhost:3000*, que é o endereço padrão do servidor em desenvolvimento. Para o ambiente de produção lembre-se de configurar a url correta de sua aplicação. Com isto, qualquer e-mail gerado conterá urls válidas que apontarão corretamente para nosso servidor. Saia daquela sessão do console e abra uma nova para recarregar essa configuração e repita os comandos anteriores:

```
>> job = Job.find 1
=> #<Job id: 1, title: "Developer", #...>
>> comment = job.comments.first
=> #<Comment id: 2, name: "Plataformatec", body: "Awesome!", #...>
>> CompanyMailer.new_comment(job, comment).deliver
=> #<Mail::Message:70299197293820, Multipart: false, Headers: <Date: Tue, 19 Feb 2013 20:37:59
```

O e-mail foi enviado com sucesso! O último passo é executar esse comando para notificar a empresa quando o comentário é criado. Abra o *app/controllers/comments_controller.rb*, e altere a *action create* conforme abaixo:

```
def create
  @job = Job.find(params[:job_id])
  @comment = @job.comments.build(params[:comment])

  if success = @comment.save
    CompanyMailer.new_comment(@job, @comment).deliver
  end

  respond_to do |format|
```

```
      format.html do
        if success
          flash[:notice] = "Comment was created with success!"
        else
          flash[:alert] = "Please fill in all fields to create a comment."
        end
        redirect_to @job
      end
      format.js
    end
  end
```

Criamos a variável `success` para armazenar se o comentário foi salvo com sucesso ou não, e caso seja, já enviamos o e-mail em seguida da mesma forma que utilizamos no console. Note que utilizamos a variável `success` também no bloco `html` para checar qual mensagem *flash* deve ser setada, e removemos o bloco da resposta `js`, pois já estamos salvando o comentário logo acima.

Agora basta testarmos tudo enviando um novo comentário pelo navegador. Se estiver rodando o servidor do Rails, reinicie-o pois alteramos um arquivo de configuração. Então navegue até um *job* como em *http://localhost:3000/jobs/1 (http://localhost:3000/jobs/1)*, e crie um novo comentário. Depois verifique o log do servidor, que deve exibir o e-mail como abaixo:

```
Sent mail to carlos@plataformatec.com.br (18ms)
Date: Wed, 20 Feb 2013 17:31:23 -0300
From: carlos.antonio@plataformatec.com.br
To: carlos@plataformatec.com.br
Message-ID: <5125329bdf9cf_8f913fc2ae99e8a4945d9@Carloss-MacBook-Pro.local.mail>
Subject: New comment received
Mime-Version: 1.0
Content-Type: text/plain;
 charset=UTF-8
Content-Transfer-Encoding: 7bit

Hello Plataformatec,

You've got a new comment about your Developer job:

Name: Carlos
Body: Comment with email notification.

You can see all comments for this job in http://localhost:3000/jobs/1.

Thank you!
```

Este é o conteúdo completo do e-mail que está sendo enviado, contendo os cabeçalhos e o conteúdo da mensagem. Maravilha, nosso e-mail está funcionando perfeitamente!

*Nota*: Essa informação completa do e-mail aparece apenas no log de desenvolvimento, no de produção apenas a linha `Sent mail to ...` é impressa.

Agora se analisarmos o nome da *view* que é enviada pelo e-mail `new_comment`, notaremos que ela usa o formato **text**, similar à *view* que criamos no capítulo anterior que utilizava o formato *js*. Isto quer dizer que o conteúdo dessa *view*

deverá ser **somente texto**, ou seja, não temos como adicionar nenhum HTML para formatar o e-mail, como por exemplo um link para o *job*.

É bastante aconselhável enviar e-mails em formato texto pois todos os clientes de e-mail disponíveis conseguirão apresentar este e-mail sem problemas, e por isso o Rails gera esse formato por padrão com o *mailer*. Contudo um grande número de clientes atuais consegue ler também e-mails em formato HTML, permitindo que criemos um conteúdo melhor formatado e mais apresentável para ser lido pelos usuários. Com isso em mente, vamos transformar nossa notificação em um e-mail **multipart**, ou seja, um e-mail que contém ambos os formatos `text` e `html`, assim os clientes poderão decidir qual versão utilizar, e conseguiremos entregar um e-mail otimizado para quem aceitar HTML.

## Criando e-mails com diferentes formatos

Para transformarmos nosso e-mail em *multipart* tudo que precisamos fazer é criar uma nova *view* de mesmo nome, mas que utilize o formato `html`. Crie o arquivo *app/views/company_mailer/new_comment.html.erb*, e utilize o mesmo texto adicionando um pouco de HTML para formatá-lo:

```
<h1>Hello <%= @company.name %>,</h1>

<p>You've got a new comment about your <strong><%= @job.title %></strong> job:</p>

<p>
  <em>Name</em>: <%= @comment.name %><br />
  <em>Body</em>: <%= @comment.body %>
</p>

<p>You can see all comments for this job in <%= link_to @job.title, job_url(@job) %>.</p>

<p>Thank you!</p>
```

Adicionamos um pouco de formatação HTML para tornar o e-mail mais apresentável, e um link para o *job* para facilitar o acesso do usuário. Tente criar um novo comentário pelo navegador, e você verá no log a informação do e-mail enviado, que agora é *multipart*, contendo ambos os formatos `text` e `html`:

```
Sent mail to carlos@plataformatec.com.br (42ms)
Date: Wed, 20 Feb 2013 18:05:48 -0300
From: carlos.antonio@plataformatec.com.br
To: carlos@plataformatec.com.br
Message-ID: <51253aacefe6d_8f913fc2ad151b84949bb@Carloss-MacBook-Pro.local.mail>
Subject: New comment received
Mime-Version: 1.0
Content-Type: multipart/alternative;
 boundary="--==_mimepart_51253aace1bb9_8f913fc2ad151b8494651";
 charset=UTF-8
Content-Transfer-Encoding: 7bit



----==_mimepart_51253aace1bb9_8f913fc2ad151b8494651
Date: Wed, 20 Feb 2013 18:05:48 -0300
Mime-Version: 1.0
Content-Type: text/plain;
 charset=UTF-8
```

```
Content-Transfer-Encoding: 7bit
Content-ID: <51253aaceef7d_8f913fc2ad151b849475b@Carloss-MacBook-Pro.local.mail>

Hello Plataformatec,

You've got a new comment about your Developer job:

Name: Carlos
Body: Multipart email!

You can see all comments for this job in http://localhost:3000/jobs/1.

Thank you!

----==_mimepart_51253aace1bb9_8f913fc2ad151b8494651
Date: Wed, 20 Feb 2013 18:05:48 -0300
Mime-Version: 1.0
Content-Type: text/html;
 charset=UTF-8
Content-Transfer-Encoding: 7bit
Content-ID: <51253aacef891_8f913fc2ad151b84948d@Carloss-MacBook-Pro.local.mail>

<h1>Hello Plataformatec</h1>,

<p>You've got a new comment about your <strong>Developer</strong> job:</p>

<p>
  <em>Name</em>: Carlos<br />
  <em>Body</em>: Multipart email!
</p>

<p>You can see all comments for this job in <a href="http://localhost:3000/jobs/1">Developer</a

<p>Thank you!</p>

----==_mimepart_51253aace1bb9_8f913fc2ad151b8494651--
```

Simples assim! Antes de finalizarmos vamos dar uma olhada nas configurações do Rails necessárias para que consigamos efetivamente enviar e-mails em produção.

## Configurando o ambiente de produção do Rails para entregar e-mails

Até então os e-mails que estamos criando no ambiente de desenvolvimento aparecem em nosso log, mas não são efetivamente entregues pois não definimos nenhuma configuração real de envio de e-mails. Em desenvolvimento, por padrão o *Action Mailer* tentará entregar o e-mail através de um serviço rodando na porta 25 da máquina local. Porém nós não temos nenhum tipo de serviço para isso rodando, então um erro será disparado, mas em desenvolvimento esse erro é simplesmente ignorado, pois ainda não nos importamos em efetivamente entregar esse e-mail, afinal estamos apenas desenvolvendo a aplicação e a informação do log é suficiente para sabermos que o e-mail está sendo gerado corretamente. Confira o arquivo de configuração *config/environments/development.rb*, que deve conter o seguinte:

```
# Don't care if the mailer can't send
config.action_mailer.raise_delivery_errors = false
```

Isso diz para o *Action Mailer* não se importar se erros acontecerem ao entregar e-mails, o que está ok para o ambiente de desenvolvimento, mas não para produção. Cheque agora o *config/environments/production.rb*, e você verá que a mesma configuração está comentada, o que indica que erros serão disparados caso a entrega falhe.

```
# Disable delivery errors, bad email addresses will be ignored
# config.action_mailer.raise_delivery_errors = false
```

## Configuração para gerar urls

Da mesma forma que configuramos o *host* padrão para gerar urls no ambiente de desenvolvimento, apontando para *localhost:3000*, devemos configurar o ambiente de produção com a url correta de nossa aplicação. Para isso você deverá adicionar o seguinte código ao *production.rb*:

```
config.action_mailer.default_url_options = { host: "<example.com>" }
```

Substituindo o `<example.com>` pelo domínio real da aplicação.

## SMTP

Bom, sabemos que em desenvolvimento não temos um serviço de entrega de e-mails rodando, mas precisaremos de um em produção: então o que podemos utilizar? Por padrão o Rails utiliza o *SMTP - Simple Mail Transfer Protocol*, o mais comum para envio de e-mails. Para conseguir entregar e-mails precisamos de uma conta em um servidor *SMTP*, como por exemplo uma conta do *Gmail*.

*Nota*: é importante saber que o *Gmail* possui limite para envio de e-mails, mesmo para usuários do *Google Apps* onde o limite é um pouco maior, e portanto não é recomendado para o envio de vários e-mails por aplicações comerciais. Para isso é melhor possuir uma configuração própria de *SMTP* da empresa, que os servidores de hospedagem normalmente disponibilizam, com uma conta específica para o envio através da aplicação.

Com os dados de acesso de uma conta do *Gmail* em mãos, basta adicionar a configuração abaixo no *production.rb*:

```
config.action_mailer.delivery_method = :smtp
config.action_mailer.smtp_settings = {
  :address              => 'smtp.gmail.com',
  :port                 => 587,
  :domain               => '<example.com>',
  :user_name            => '<username>',
  :password             => '<password>',
  :authentication       => 'plain',
  :enable_starttls_auto => true
}
```

Tenha a certeza de substituir o `<example.com>` com o domínio de sua aplicação (normalmente o mesmo *host* configurado para as urls), o `<username>` com o nome de usuário do *Gmail*, e o `<password>` com a senha. Isso seta o método de entrega de e-mails para `:smtp`, e configura o endereço do servidor *SMTP* do *Gmail*, além dos dados de acesso da conta para conseguir entregar os e-mails. E com essa configuração definida, a aplicação estará pronta para enviar e-mails em produção! Se você quiser fazer um teste de envio, pode copiar essa mesma configuração para o

*config/environments/development.rb*, reiniciar a aplicação e criar um novo comentário: o e-mail deve chegar para a empresa cadastrada corretamente.

## Sendmail

Uma outra maneira de se entregar e-mails que o Rails possibilita é utilizando o **Sendmail**, um serviço de envio que pode ser facilmente instalado no mesmo servidor em que a aplicação estará rodando, para entregar e-mails de forma simplificada. A configuração do *sendmail* também é bastante fácil:

```
config.action_mailer.delivery_method = :sendmail
#
# Por padrão o Rails utiliza essa configuração:
#
# config.action_mailer.sendmail_settings = {
#   :location  => '/usr/sbin/sendmail',
#   :arguments => '-i -t'
# }
```

Basta setar o método de entrega para `sendmail` que o Rails já vai utilizar alguns *defaults* para enviar os e-mails utilizando o *sendmail* através de `/usr/sbin/sendmail` . Caso seja necessário modificar os parâmetros, basta configurar a opção `sendmail_settings` como no exemplo acima, alterando conforme necessário.

Com isso fechamos o capítulo 6. Aprendemos como funciona o envio de e-mails no Rails através do *Action Mailer*, e descobrimos como enviar e-mails *multipart* para criar conteúdos mais apresentáveis para clientes de e-mail que suportam HTML. No próximo capítulo retomaremos a partir do *mailer* que criamos aqui para aprendermos como internacionalizar os textos em diferentes idiomas utilizando os *helpers* do Rails, vejo você lá!.

## Para saber mais

- 
- Assim como utilizamos *layouts* para as *views* da aplicação, *mailers* também podem ter seus próprios *layouts* para por exemplo criar uma estrutura HTML geral para todos os e-mails. Por padrão deve-se criar um novo *layout* em *app/views/layouts*, com o nome do *mailer* em questão, como por exemplo *company_mailer.html.erb*, mas isso pode ser customizado. Para saber mais sobre *layouts* em e-mails, verifique o Rails Guides (http://guides.rubyonrails.org/action_mailer_basics.html#action-mailer-layouts) que explica com mais detalhes como criar e selecionar um *layout*. Aproveite a oportunidade para criar o *layout* para o *company mailer* que criamos neste capítulo, contento uma estrutura HTML básica.
- 
- Em algumas aplicações poderá ser necessário enviar e-mails utilizando anexos, como por exemplo para mandar um relatório em PDF ou um arquivo qualquer para um determinado e-mail. O *Action Mailer* já provê essa funcionalidade para nós tornando o uso de anexos bastante simplificado. Cheque o tópico sobre envio de e-mails com anexo no Rails Guides do *Action Mailer* (http://guides.rubyonrails.org/action_mailer_basics.html#sending-emails-with-attachments) para saber mais e ver alguns exemplos.
- 
- Conheça mais configurações disponíveis para o *Action Mailer* (http://guides.rubyonrails.org/action_mailer_basics.html#action-mailer-configuration).
- 
- Em determinadas aplicações o envio de e-mails pode ser uma funcionalidade bastante importante, e portanto os e-mails podem ter *layouts* muito mais elaborados. Nestes casos apenas o log do Rails pode não ser suficiente para

validar o conteúdo dos e-mails. Em situações como esta, existem ferramentas muito úteis como o MailCatcher (http://mailcatcher.me/), que permite rodar um servidor *SMTP* local, permitindo que configuremos o ambiente de desenvolvimento para enviar e-mails para esse servidor. O MailCatcher então se encarregará de receber estes e-mails e exibí-los em uma interface web como se fosse um cliente de e-mail, permitindo que visualizemos ambos os formatos texto e HTML dos e-mails, e portanto vendo como eles estão sendo exibidos para o usuário final de forma bastante simples.