

# Actionmailer / Delayed Jobs

Devpoint Labs - Dave Jungst / Jake Sorce

# Create a rails app

```
rails new mailing_list -d postgresql
```

```
Gemfile.rb  
gem 'haml-rails'
```


```
bundle
```

```
rails g controller signup index
```

```
config/routes.rb  
root 'signup#index'
```

```
rails g model user name email
```

```
bundle exec rake db:create db:migrate
```



# Generate a mailer

rails g mailer signup\_mailer

```
create  app/mailers/signup_mailer.rb
create  app/mailers/application_mailer.rb
invoke  haml
create  app/views/signup_mailer
create  app/views/layouts/mailer.text.haml
create  app/views/layouts/mailer.html.haml
invoke  test_unit
create  test/mailers/signup_mailer_test.rb
create  test/mailers/previews/signup_mailer_preview.rb
```

Notice the test/mailers/previews/ directory

## app/mailers/signup\_mailer.rb

```
1 class SignupMailer < ApplicationMailer
2   default from: ENV['MAIL_FROM']
3
4   def new_signup(user)
5     @user = user
6     mail(to: @user.email, subject: 'Thank you for signing up!')
7   end
8
9 end
```

## app/views/signup\_mailer/new\_signup.html.haml

```
1 !!!  
2 %html  
3   %head  
4     %meta{content: 'text/html; charset=UTF-8'}  
5   %body  
6     %h1  
7       Hi #{@user.name}  
8     %p  
9       Thank you for signing up
```

# app/views/signup\_mailer/new\_signup.text.haml

Not all HTML clients prefer HTML emails so we need create a text backup

Hi #{@user.name}

Thank you for signing up



# Previewing emails

test/mailers/previews/signup\_mailer\_preview.rb

```
1 # Preview all emails at http://localhost:3000/rails/mailers/signup_mailer
2 class SignupMailerPreview < ActionMailer::Preview
3   def new_signup_preview
4     SignupMailer.new_signup(User.first)
5   end
6 end
```

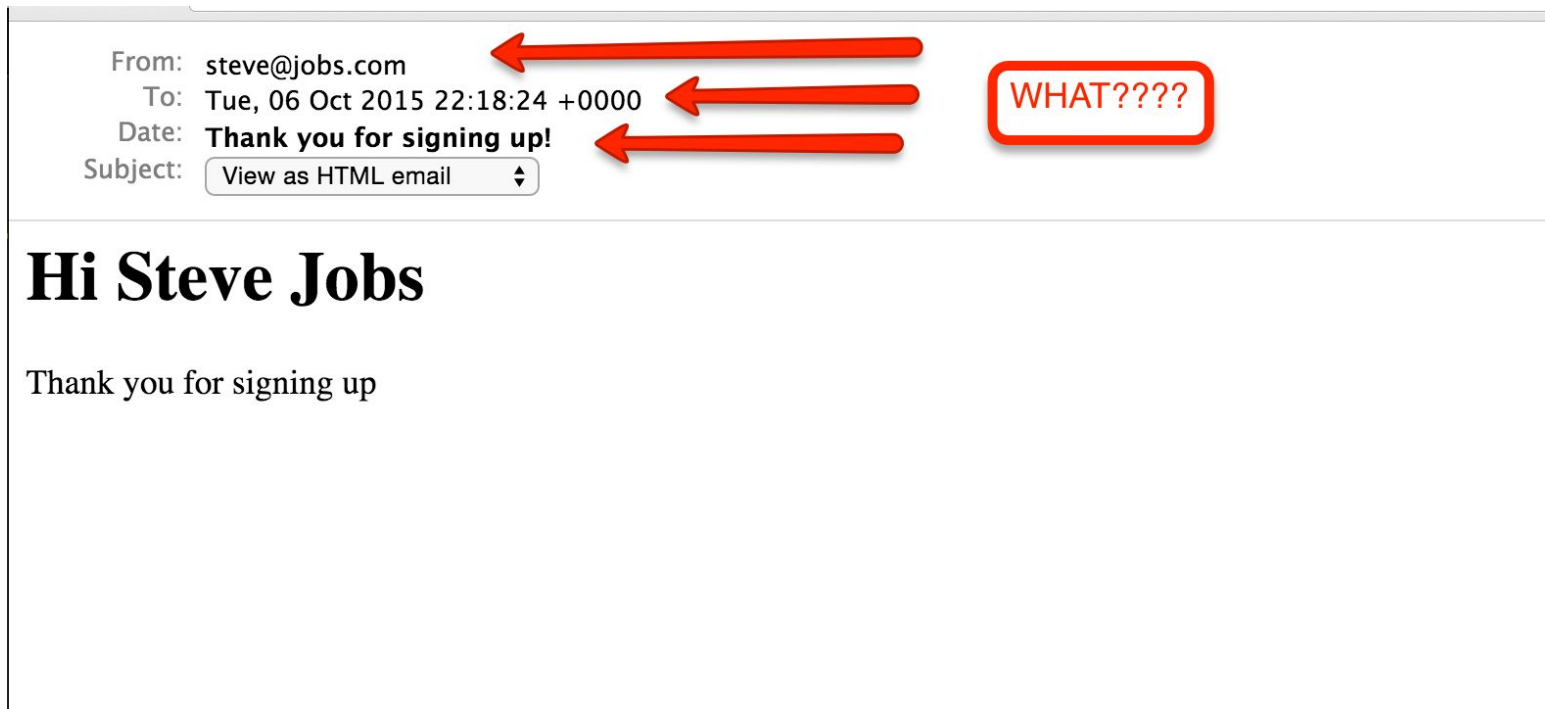
bundle exec rails c

User.create(name: 'Steve Jobs', email: '[stevejobs@apple.com](mailto:stevejobs@apple.com)')

bundle exec rails s

localhost:3000/rails/mailers

# This doesn't look right



Why did this happen?



# ENV['MAIL\_FROM']

config/mail.yml

MAIL\_FROM: 'myemail@gmail.com'

config/initializers/mail.rb

```
1 mail = "#{Rails.root}/config/mail.yml"
2 YAML.load_file(mail).each { |key, value| ENV[key] ||= ENV[key] = value.to_s } if File.exists? mail
3
```

---

From: djungst@gmail.com

To: steve@jobs.com

Date: Tue, 06 Oct 2015 22:34:39 +0000

Subject: **Thank you for signing up!**

View as HTML email



---

## Hi Steve Jobs

Thank you for signing up

# Configuring Mail client for gmail

config/mail.yml

```
1 MAIL_FROM: 'me@gmail.com'  
2 GMAIL_USERNAME: 'my_gmail_username'  
3 GMAIL_PASSWORD: 'my gmail password'
```

config/environments/production.rb

```
config.action_mailer.deliver_method = :smtp  
config.action_mailer.smtp_settings = {  
  address: 'smtp.gmail.com',  
  port: 587,  
  user_name: ENV['GMAIL_USERNAME'],  
  password: ENV['GMAIL_PASSWORD'],  
  authentication: 'plain',  
  enable_starttls_auto: true  
}
```

This code goes in production.rb  
so that emails don't accidentally  
go out in development

# Sending mail in development

Gemfile

```
gem 'letter_opener', group: :development
```

bundle

```
config/environments/development.rb
```

```
config.action_mailer.delivery_method = :letter_opener
```

Now mail in development can be viewed in the browser and is stored in

```
tmp/letter_opener
```



```

1 -# app/views/signup/index.html.haml
2 %h2 Sign up for the newsletter
3 %hr
4 = form_for :user, url: url_for(controller: 'signup', action: 'create') do |f|
5   = f.label :name
6   = f.text_field :name
7   = f.label :email
8   = f.email_field :email
9   = f.submit :submit

```

```

1 #config/routes.rb
2 Rails.application.routes.draw do
3   root 'signup#index'
4   post 'signup/create', to: 'signup#create', as: 'signup'

```

In the view we are using the :user model but directing to a different route for simplicity

In the controller after creating a user you can simply call the proper method on your Mailer and then call deliver

Notice what happens in development when mail is sent

```

1 class SignupController < ApplicationController
2   def index
3   end
4
5   def create
6     user = User.create(user_params)
7     SignupMailer.new_signup(user).deliver
8     redirect_to root_path
9   end
10
11   private
12   def user_params
13     params.require(:user).permit(:name, :email)
14   end
15 end

```

# Delayed Jobs

There are many requests that take place in a web app that can take a long time to process.

Long running thread can cause several issues:

1. Bad user experience while they wait for the action to process
2. Heroku can has a response timer and if a subsequent response isn't made within a certain amount of time the request will be terminated

The answer is to process these actions in a background job



# delayed\_job setup

Gemfile

```
gem 'delayed_job'
```

```
gem 'delayed_job_active_record'
```

```
bundle install
```

```
rails g delayed_job:active_record
```

```
bundle exec rake db:migrate
```

```
config/application.rb
```

```
config.active_job.queue_adapter = :delayed_job
```

# Delayed Jobs

Calling `delay.method(params)` on any method will put it in the job queue and run it in the order it was called

Methods that should always be run in a delayed can be setup by adding code after the method

```
def send_some_mail
```

```
  #code here
```

```
end
```

```
handle_asynchronously :send_some_mail
```



# handle\_asynchronously & :delay

Both of these methods take 3 params:

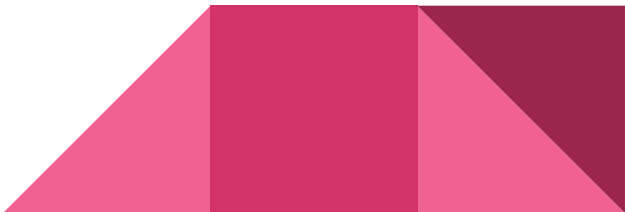
priority: lower numbers run first (default is 0)

`handle_asynchronously :send_mail, priority: 10`

run\_at: Run the job after this time

`handle_asynchronously :send_mail, run_at: Proc.new { 5.minutes.from.now }`

queue: named queue to put job in





## Sending mail with delayed job

```
1 class SignupMailer < ApplicationMailer
2   default from: ENV['MAIL_FROM']
3
4   def new_signup(user)
5     @user = user
6     mail(to: @user.email, subject: 'Thank you for signing up!')
7   end
8   handle_asynchronously :new_signup
9
10 end
```

The jobs run on their own server. In another terminal in the app directory call

bundle exec rake jobs:work

# Update the controller

```
[Worker(host:dave-laptop.local pid:53803)] Job Delayed::PerformableMethod (id=6) RUNNING  
[Worker(host:dave-laptop.local pid:53803)] 1 jobs processed at 11.8016 j/s, 0 failed
```

Default

```
1 class SignupController < ApplicationController  
2   def index  
3   end  
4  
5   def create  
6     user = User.create(user_params)  
7     SignupMailer.delay.new_signup(user)  
8     redirect_to root_path  
9   end
```

# Add procfile for production

If you are going to have delayed\_jobs running on Heroku (\$\$\$ for more than 1 worker)

Create Procfile in the root of the app

Procfile

```
worker: bundle exec rake jobs:work
```



# Flash

Now that mail is sending in the background it would improve the user experience to have some notification.

Rails flash is just a key value pair that persists through requests

```
2 <html>
3 <head>
4   <title>MailingList</title>
5   <%= stylesheet_link_tag 'application', media: 'all', 'data-turbolinks-track' => true %>
6   <%= javascript_include_tag 'application', 'data-turbolinks-track' => true %>
7   <%= csrf_meta_tags %>
8 </head>
9 <body>
10  <%= flash.each do |key, value| %>
11    <div class="alert alert-<%= key %>"><%= value %></div>
12  <%= end %>
13  <%= yield %>
14
15 </body>
16 </html>
```

app/views/layouts/application.html.erb  
Add the flash logic

## Now call flash in the controller

```
4
5  def create
6    if user = User.create(user_params)
7      SignupMailer.delay.new_signup(user)
8      flash[:success] = "Thank you an email will be sent shortly"
9    else
10     flash[:error] = "A user with this email already exists"
11    end
12  -
13    redirect_to root_path
14  end
```

# The last step is to style the flash messages

```
1 .alert {  
2   width: 100%;  
3   color: white;  
4   text-align: center;  
5 }  
6  
7 .alert-success {  
8   background-color: green;  
9 }  
10  
11 .alert-error {  
12   background-color: red;  
13 }
```

