



Send mails with Rails (Action Mailer)

Juan “Harek” Urrios
@xharekx33

Sending mail with your app

You'll frequently want to send emails to the users of your app. When they create a new account, so they can recover their lost password, to inform them of important changes to their account (password change), expiration of subscription periods... There are many reasons.

Rails provides an easy way to do it through Action Mailer.



What is Action Mailer

Action Mailer provides a way to create emails using templates in the same way that Action Controller renders views using templates.



Clone a base repo

Clone this repo to use as the starting point for this Unit:

https://github.com/xharekx33/taskly_ironhack_authentication

```
$ git clone https://github.com/xharekx33/taskly_ironhack_authentication
$ bundle install
$ rake db:migrate
$ rake db:seed
```



@xharekx33

Creating a Mailer

Create a mailer using the generator:

`rails generate mailer UserMailer`

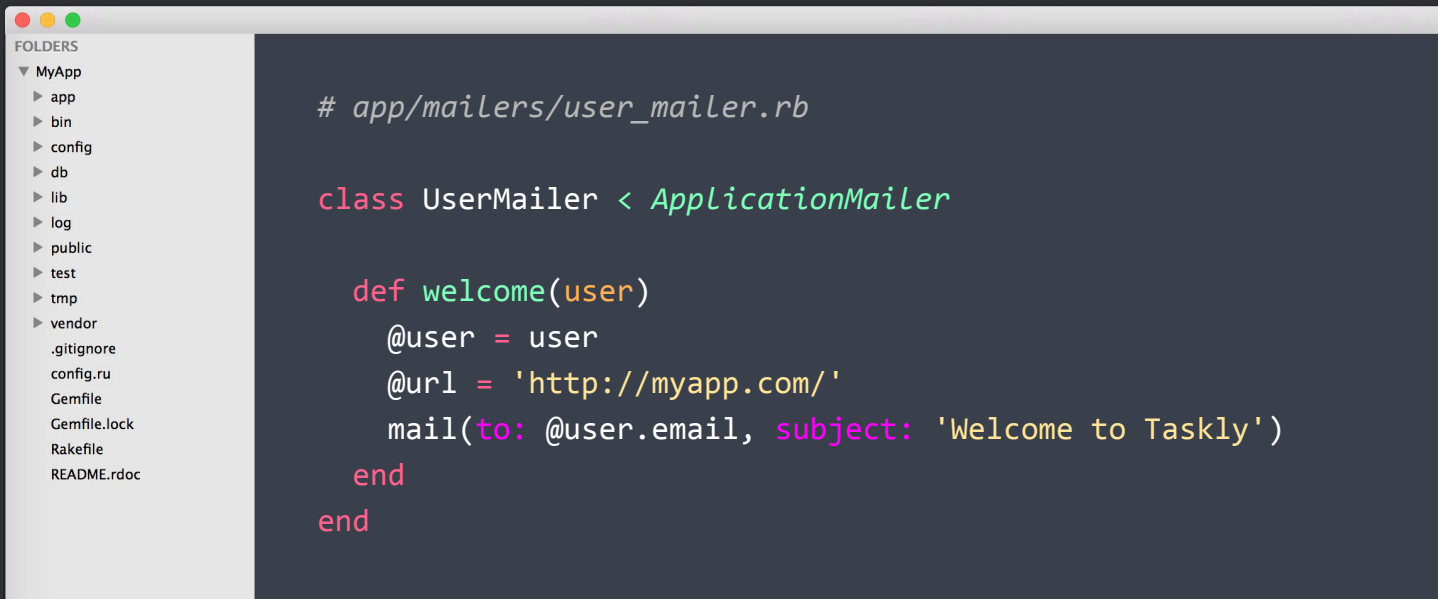
```
$ rails g mailer UserMailer
  create  app/mailers/user_mailer.rb
  invoke  erb
  create  app/views/user_mailer
  invoke  test_unit
  create  test/mailers/user_mailer_test.rb
  create  test/mailers/previews/user_mailer_preview.rb
```



@xharekx33

Add a method to the Mailer

The new mailer has no methods. Add a new method to send a welcome mail to users. You can change default settings in `ApplicationMailer`



The screenshot shows a code editor with a sidebar on the left displaying a file tree under 'FOLDERS'. The tree includes 'MyApp' with subfolders 'app', 'bin', 'config', 'db', 'lib', 'log', 'public', 'test', 'tmp', and 'vendor', as well as files like '.gitignore', 'config.ru', 'Gemfile', 'Gemfile.lock', 'Rakefile', and 'README.rdoc'. The main editor area displays the following Ruby code:

```
# app/mailers/user_mailer.rb

class UserMailer < ApplicationMailer

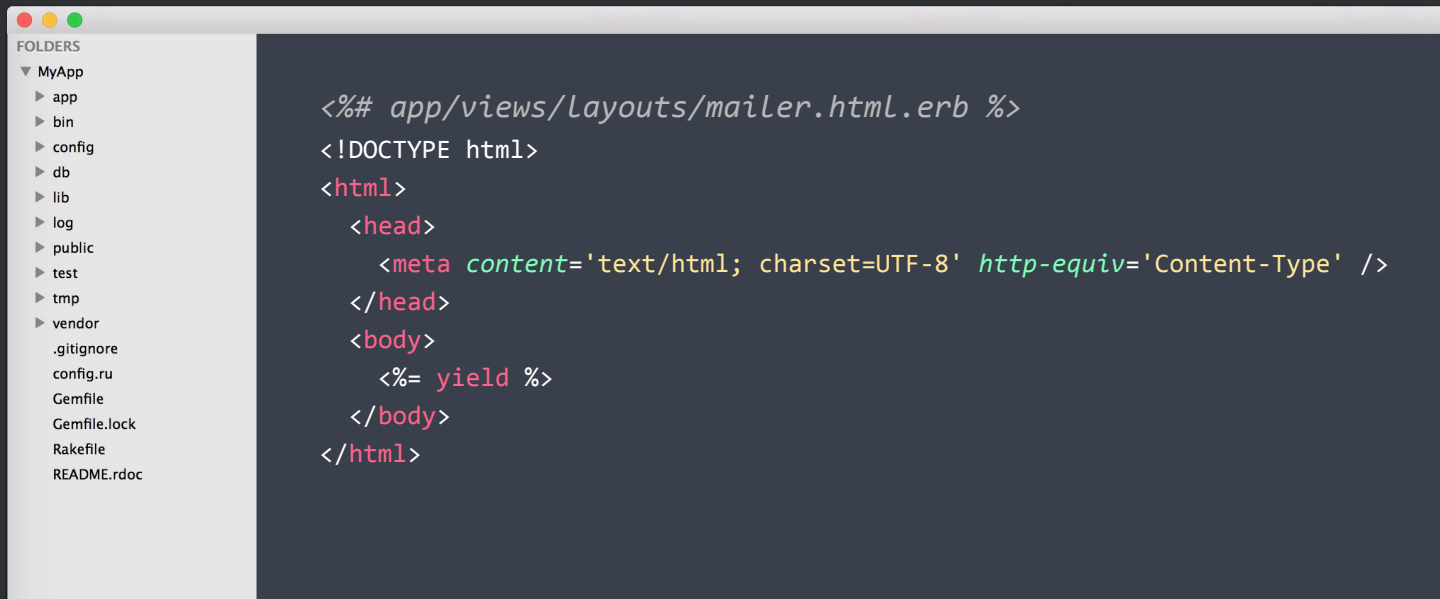
  def welcome(user)
    @user = user
    @url = 'http://myapp.com/'
    mail(to: @user.email, subject: 'Welcome to Taskly')
  end
end
```



@xharekx33

Fill the HTML layout for our emails

Add the doctype and head to the existing HTML mailer template in `app/views/layouts`



The screenshot shows a code editor with a sidebar on the left displaying a file tree under the name 'FOLDERS'. The tree includes a 'MyApp' folder with subfolders 'app', 'bin', 'config', 'db', 'lib', 'log', 'public', 'test', 'tmp', and 'vendor', as well as files '.gitignore', 'config.ru', 'Gemfile', 'Gemfile.lock', 'Rakefile', and 'README.rdoc'. The main editor area displays the content of the file `app/views/layouts/mailer.html.erb`. The code is as follows:

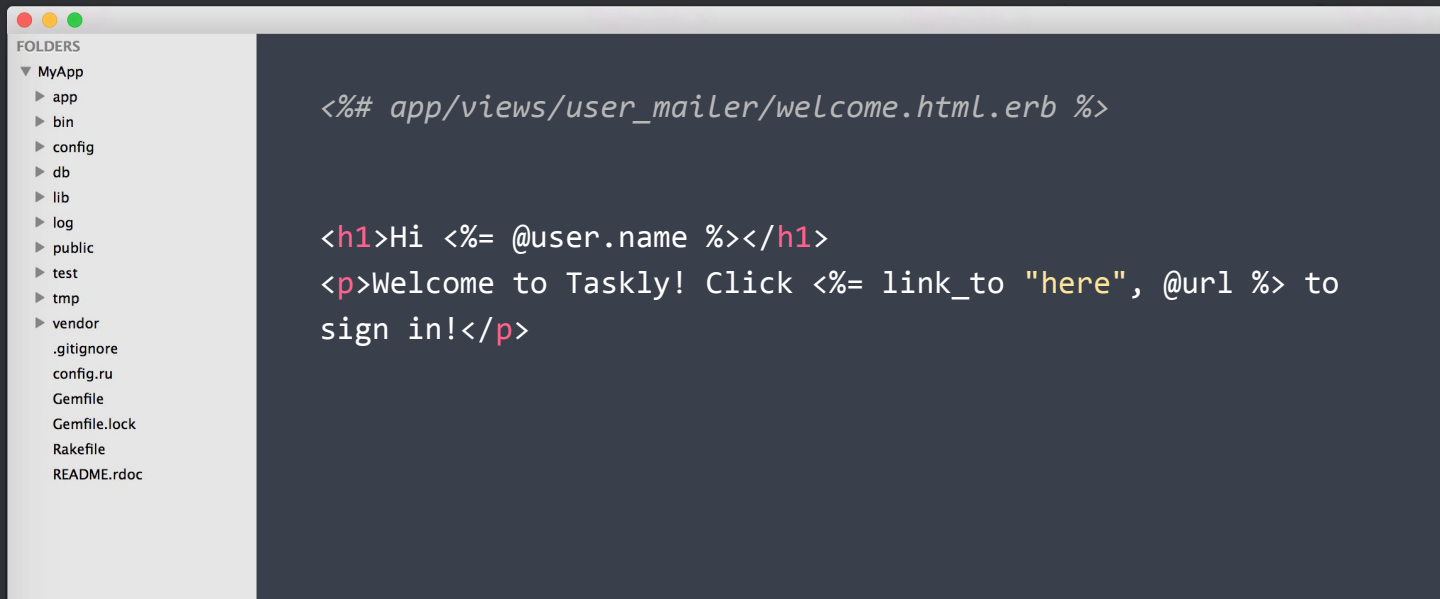
```
<%= app/views/layouts/mailer.html.erb %>
<!DOCTYPE html>
<html>
  <head>
    <meta content='text/html; charset=UTF-8' http-equiv='Content-Type' />
  </head>
  <body>
    <%= yield %>
  </body>
</html>
```



@xharekx33

Create an HTML template for the email

Write the content for the mail we want the method to send. First create the HTML version of the email in `app/views/user_mailer`



The screenshot shows a code editor with a sidebar on the left labeled 'FOLDERS'. The sidebar lists the following structure: MyApp (expanded), app, bin, config, db, lib, log, public, test, tmp, vendor, .gitignore, config.ru, Gemfile, Gemfile.lock, Rakefile, and README.rdoc. The main editor area displays the following ERB code:

```
<%# app/views/user_mailer/welcome.html.erb %>

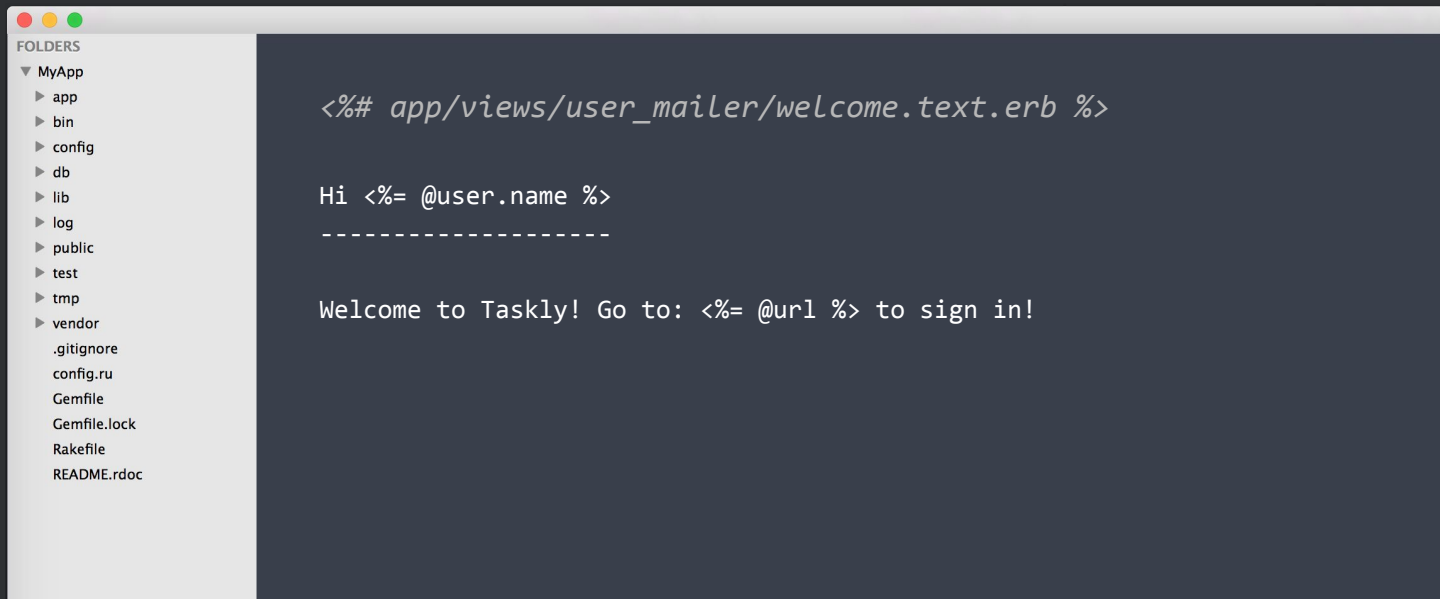
<h1>Hi <%= @user.name %></h1>
<p>Welcome to Taskly! Click <%= link_to "here", @url %> to
sign in!</p>
```



@xharekx33

Create the text version of the email

Not all clients prefer HTML emails, so we'll create a text version of the previous template too.



The screenshot shows a code editor window with a sidebar on the left titled 'FOLDERS'. The sidebar lists a project structure under 'MyApp' including folders like 'app', 'bin', 'config', 'db', 'lib', 'log', 'public', 'test', 'tmp', and 'vendor', as well as files like '.gitignore', 'config.ru', 'Gemfile', 'Gemfile.lock', 'Rakefile', and 'README.rdoc'. The main editor area displays the content of the file 'app/views/user_mailer/welcome.text.erb' in a monospaced font. The code is a text template for an email, using ERB syntax for variable interpolation.

```
<%# app/views/user_mailer/welcome.text.erb %>

Hi <%= @user.name %>
-----

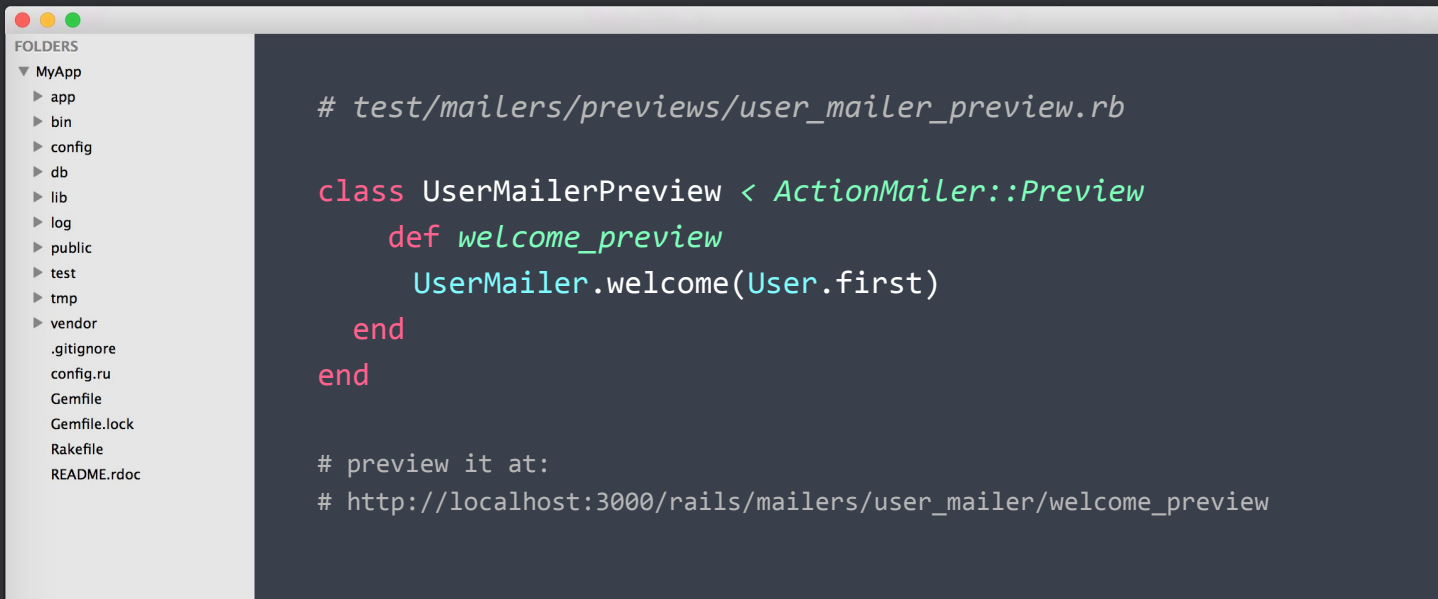
Welcome to Taskly! Go to: <%= @url %> to sign in!
```



@xharekx33

Preview your emails

To quickly test your emails you can use the previews in `test/mailers/previews/`



The screenshot shows a code editor with a sidebar on the left displaying a file tree under 'FOLDERS'. The tree includes 'MyApp' with subfolders 'app', 'bin', 'config', 'db', 'lib', 'log', 'public', 'test', 'tmp', and 'vendor', as well as files like '.gitignore', 'config.ru', 'Gemfile', 'Gemfile.lock', 'Rakefile', and 'README.rdoc'. The main editor area displays the content of `test/mailers/previews/user_mailer_preview.rb`. The code defines a `UserMailerPreview` class that inherits from `ActionMailer::Preview` and includes a `welcome_preview` method that calls `UserMailer.welcome(User.first)`. Comments at the bottom indicate how to preview the email at `localhost:3000/rails/mailers/user_mailer/welcome_preview`.

```
# test/mailers/previews/user_mailer_preview.rb

class UserMailerPreview < ActionMailer::Preview
  def welcome_preview
    UserMailer.welcome(User.first)
  end
end

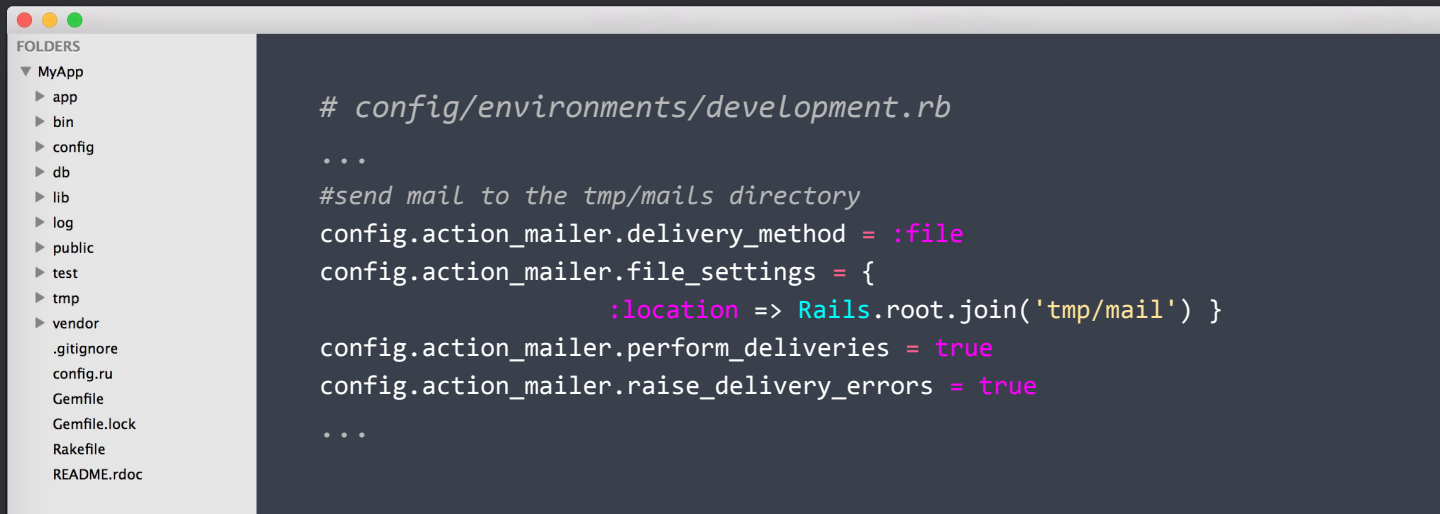
# preview it at:
# http://localhost:3000/rails/mailers/user_mailer/welcome_preview
```



@xharekx33

Sending mails to a file

To make sure that emails are correctly sent on your development machine, without a functional SMTP server and to avoid spamming people it is possible to just send mails to a file.



```
# config/environments/development.rb

...

#send mail to the tmp/ails directory
config.action_mailer.delivery_method = :file
config.action_mailer.file_settings = {
  :location => Rails.root.join('tmp/ail') }

config.action_mailer.perform_deliveries = true
config.action_mailer.raise_delivery_errors = true

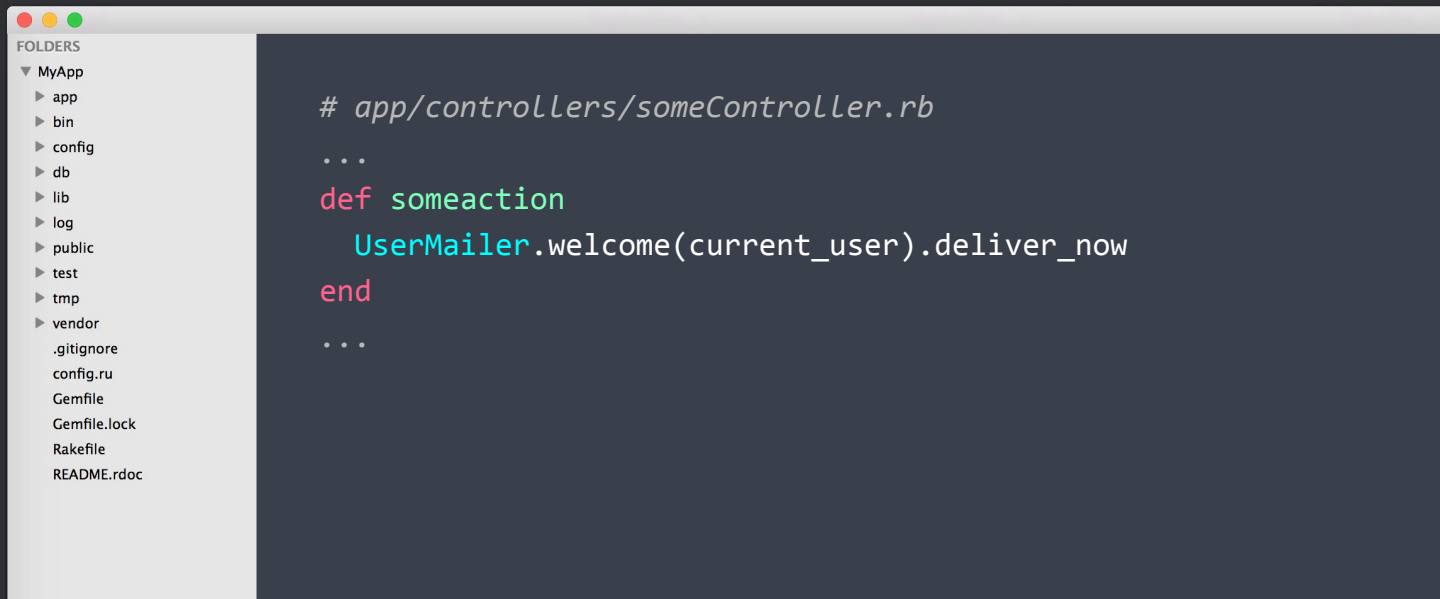
...
```



@xharekx33

Let's send some mail!

Call our newly created mailer from a controller action. Now when you call that action an email will be sent to the current user.



The screenshot shows a code editor with a sidebar on the left displaying a file tree under the heading 'FOLDERS'. The tree includes a 'MyApp' folder with subfolders 'app', 'bin', 'config', 'db', 'lib', 'log', 'public', 'test', 'tmp', and 'vendor', as well as files like '.gitignore', 'config.ru', 'Gemfile', 'Gemfile.lock', 'Rakefile', and 'README.rdoc'. The main editor area shows a Ruby code snippet for a controller action:

```
# app/controllers/someController.rb

...

def someaction
  UserMailer.welcome(current_user).deliver_now
end

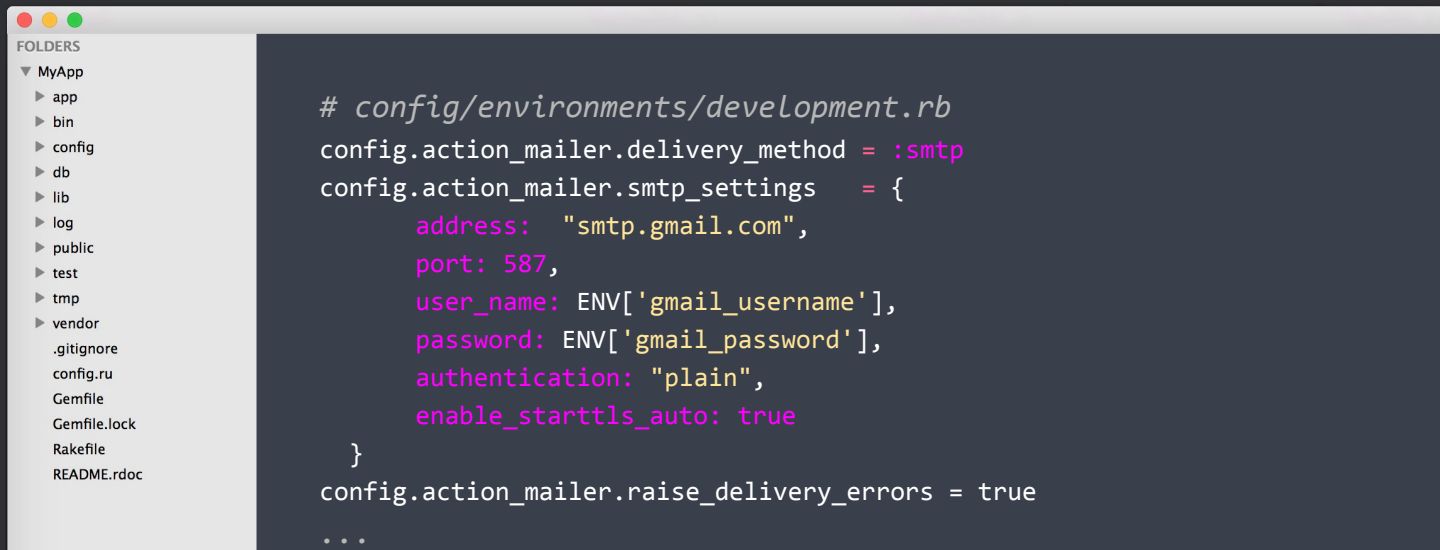
...
```



@xharekx33

Configure SMTP

For your production environment you'll want to actually send mail instead of saving it to a file. Let's try it using Gmail (and the development environment, just this time)



The screenshot shows a code editor with a file explorer on the left. The file explorer lists folders and files for 'MyApp', including 'app', 'bin', 'config', 'db', 'lib', 'log', 'public', 'test', 'tmp', 'vendor', '.gitignore', 'config.ru', 'Gemfile', 'Gemfile.lock', 'Rakefile', and 'README.rdoc'. The main editor area displays the content of 'config/environments/development.rb'. The code configures the action_mailer to use SMTP for email delivery, specifically using Gmail's SMTP service. The settings include the address 'smtp.gmail.com', port 587, and authentication using a username and password from environment variables. The 'enable_starttls_auto' is set to true. The code also sets 'config.action_mailer.raise_delivery_errors = true' and ends with three dots indicating further code.

```
# config/environments/development.rb
config.action_mailer.delivery_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
}
config.action_mailer.raise_delivery_errors = true
...
```



@xharekx33

Environment variables

In the previous slide you saw 2 environment variables being used: `ENV['gmail_username']` and `ENV['gmail_password']`

Environment variables are used to hold sensitive information, instead of hardcoding them into files that might end up uploaded into your version control (git).

These can be stored in your OS so your app can access them, for example in your `.bashrc` file, but that makes things harder to maintain and deploy.



Configuring your app with Figaro

Figaro is an app that makes it easy to securely configure your Rails App.
To install it:

```
$ echo "gem 'figaro'" >> Gemfile
```

```
$ bundle install
```

```
$ figaro install
```

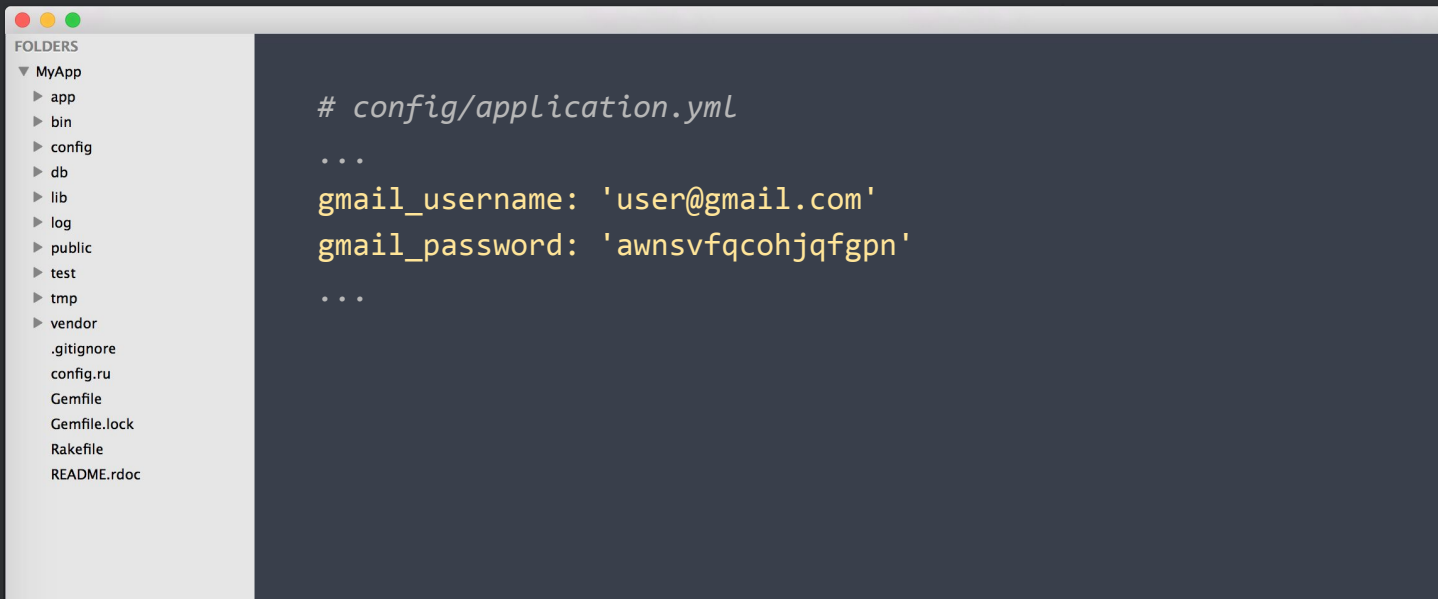
```
  create  config/application.yml
  append  .gitignore
```



@xharekx33

Add environment variables

In the newly created `application.yml` file, just add the variables needed for our SMTP configuration. Figaro will take care of the rest.



The screenshot shows a code editor with a sidebar on the left displaying a file tree. The main editor area shows the content of `config/application.yml`. The file tree includes folders like `MyApp` (with subfolders `app`, `bin`, `config`, `db`, `lib`, `log`, `public`, `test`, `tmp`, and `vendor`) and files like `.gitignore`, `config.ru`, `Gemfile`, `Gemfile.lock`, `Rakefile`, and `README.rdoc`. The main editor shows the following YAML configuration:

```
# config/application.yml

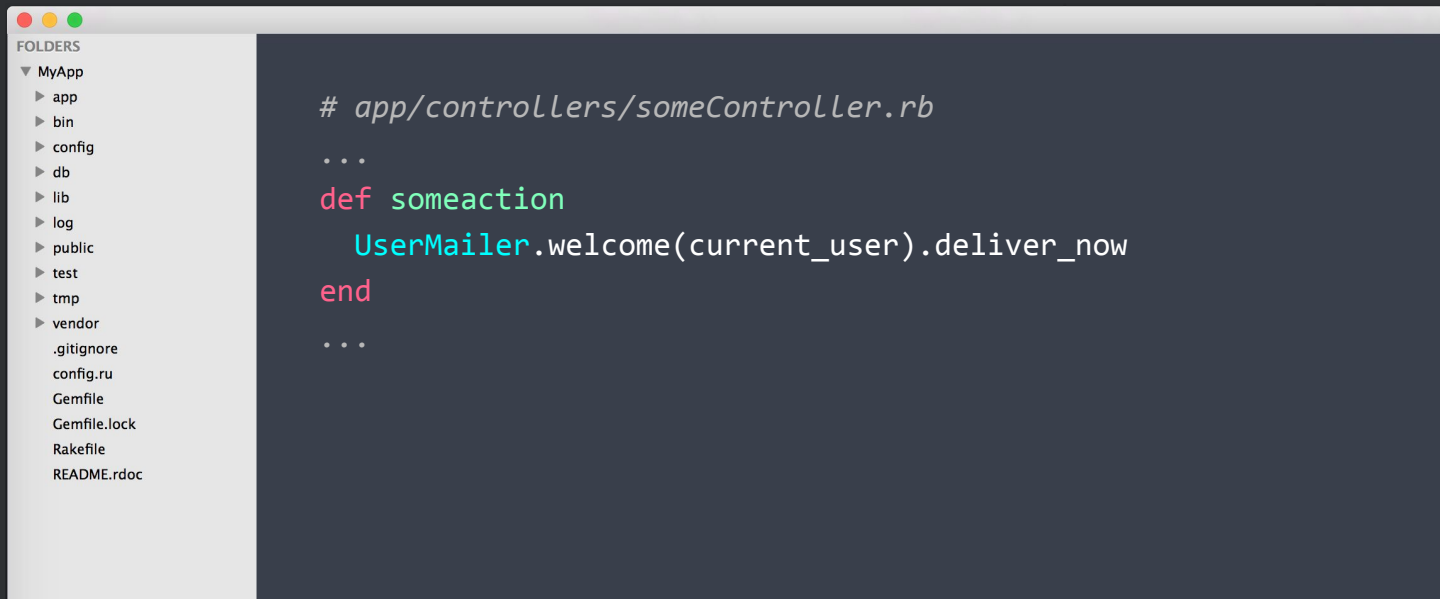
...
gmail_username: 'user@gmail.com'
gmail_password: 'awnsvfqcohjqfgpn'
...
```



@xharekx33

Let's send some mail!

Call our newly created mailer from a controller action. Now when you call that action an email will be sent to the current user.



@xharekx33

Send emails asynchronously

We've used the `deliver_now` method to send our emails.

We can also use Active Job to send it in the background using a processor. (`delayed_job`, `resque`, `sidekiq`)

This way emails are sent outside of the request-response cycle, and the action (and as a result, your user) doesn't have to wait on it.

We'll learn about background jobs next.

