IRON
HACK

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.

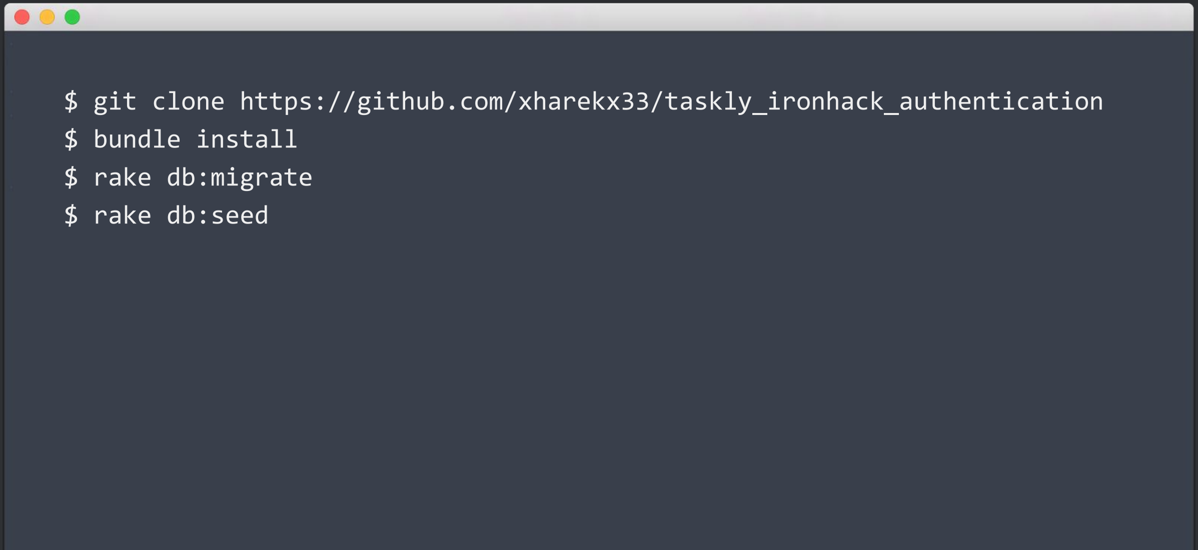
Rails provides an easy way to do it through Action Mailer.

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



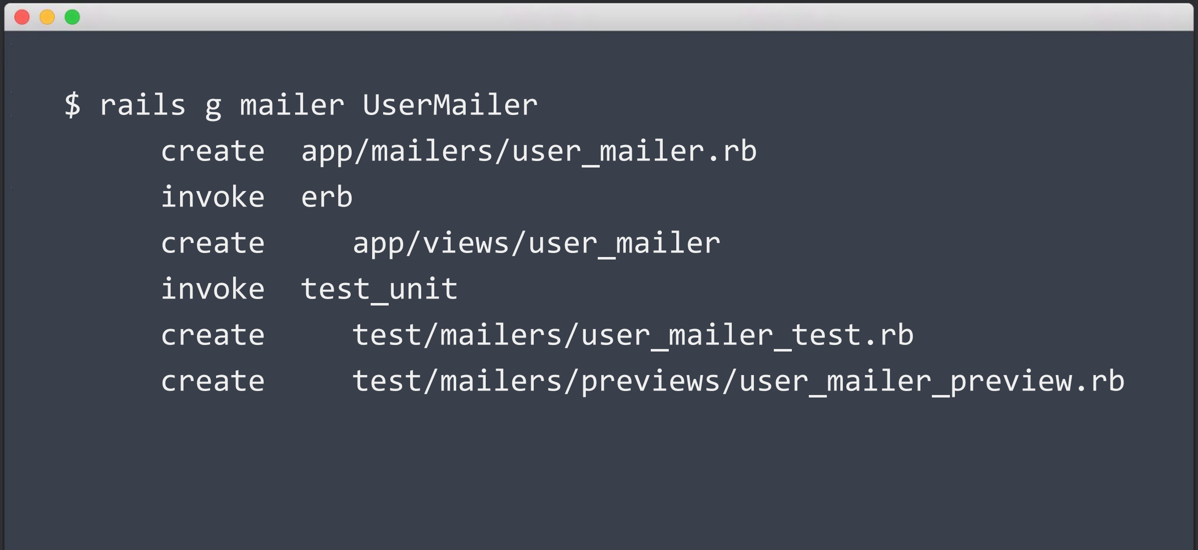
```
$ git clone https://github.com/xharekx33/taskly_ironhack_authentication
$ bundle install
$ rake db:migrate
$ rake db:seed
```

IRON HACK

@xharekx33

Creating a Mailer

Create a mailer using the generator:
`rails generate mailer UserMailer`



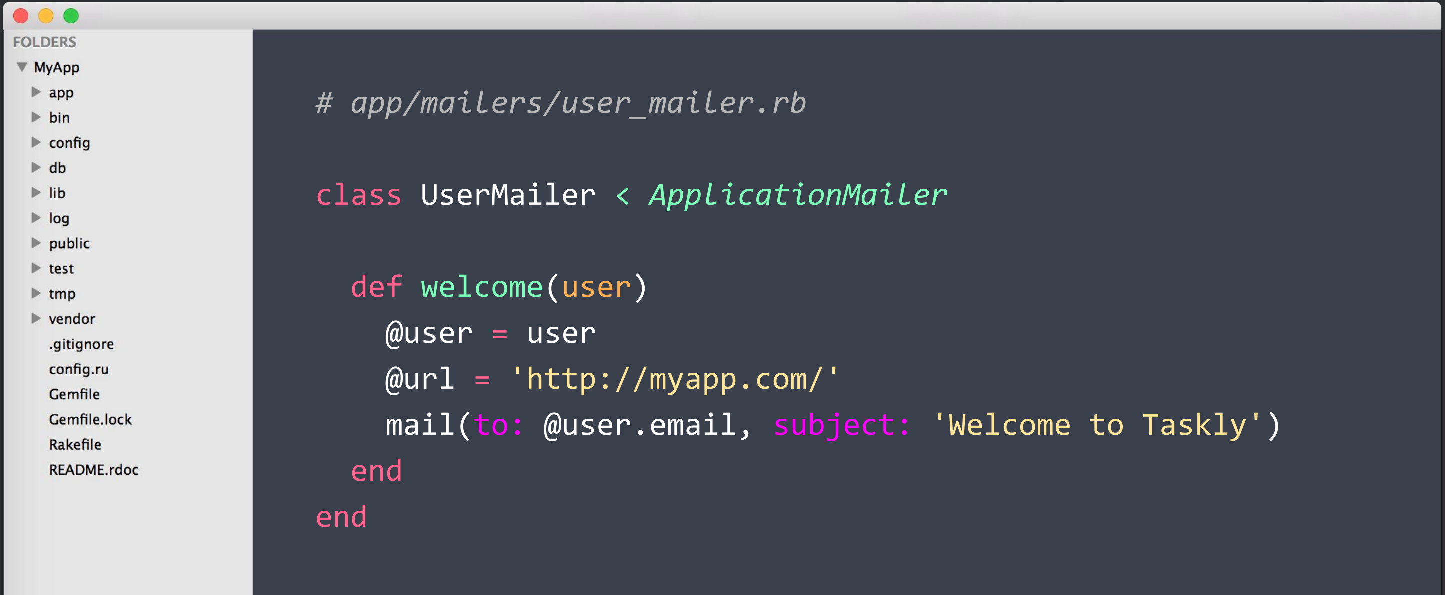
```
$ rails g mailer UserMailer
      create  app/mailers/user_mailer.rb
      invoke  erb
      create    app/views/user_mailer
      invoke  test_unit
      create    test/mailers/user_mailer_test.rb
      create    test/mailers/previews/user_mailer_preview.rb
```

IRON HACK

@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`

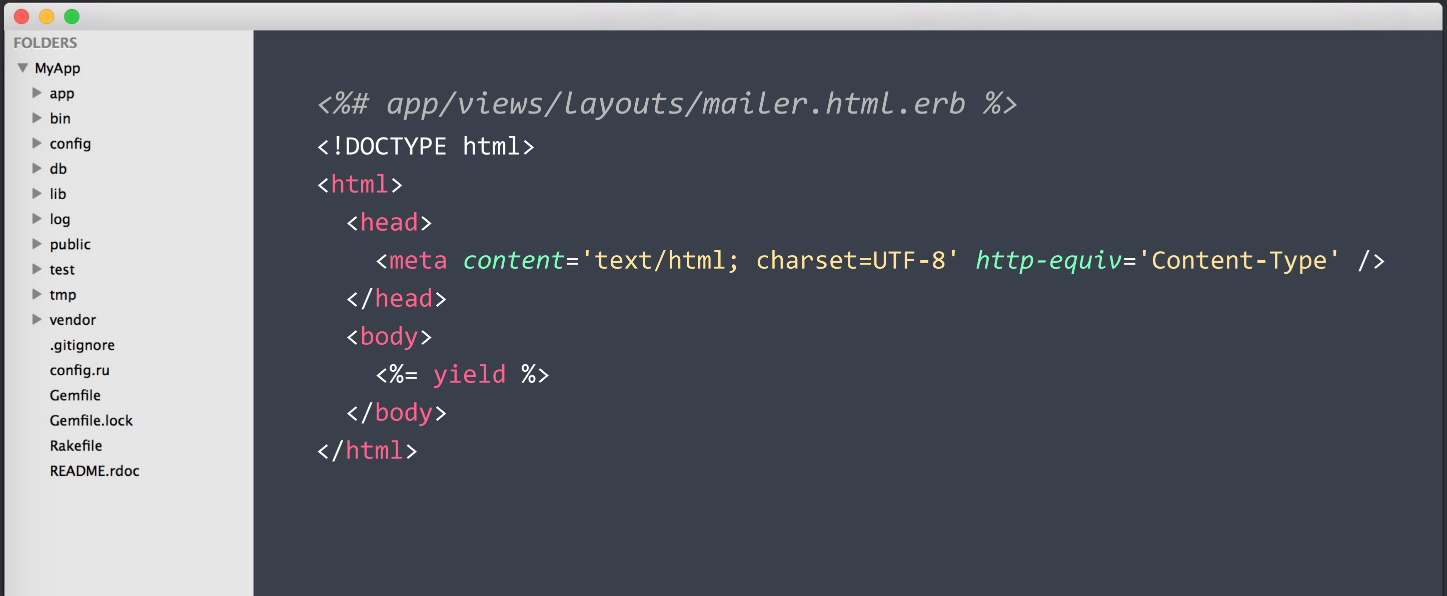


```ruby
# app/mailers/user_mailer.rb

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`

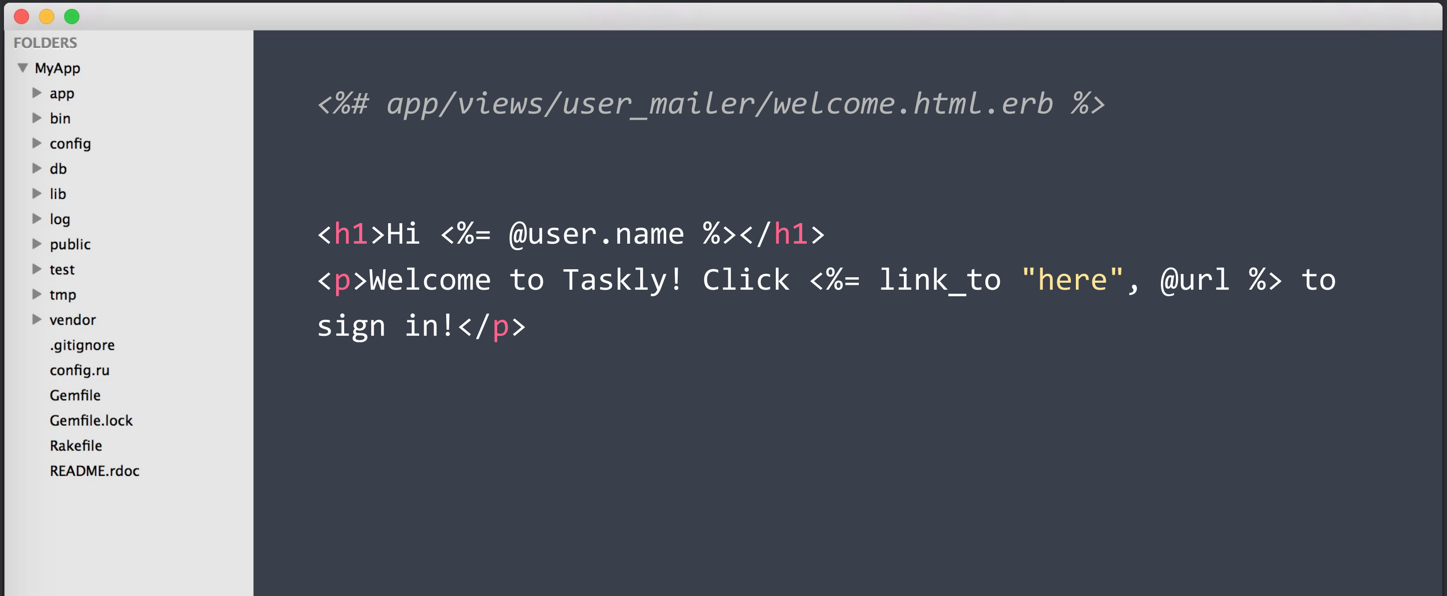


```
<%# app/views/layouts/mailer.html.erb %>
<!DOCTYPE html>
<html>
  <head>
    <meta content='text/html; charset=UTF-8' http-equiv='Content-Type' />
  </head>
  <body>
    <%= yield %>
  </body>
</html>
```

@xharekx33

Create an HTML template for the email

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



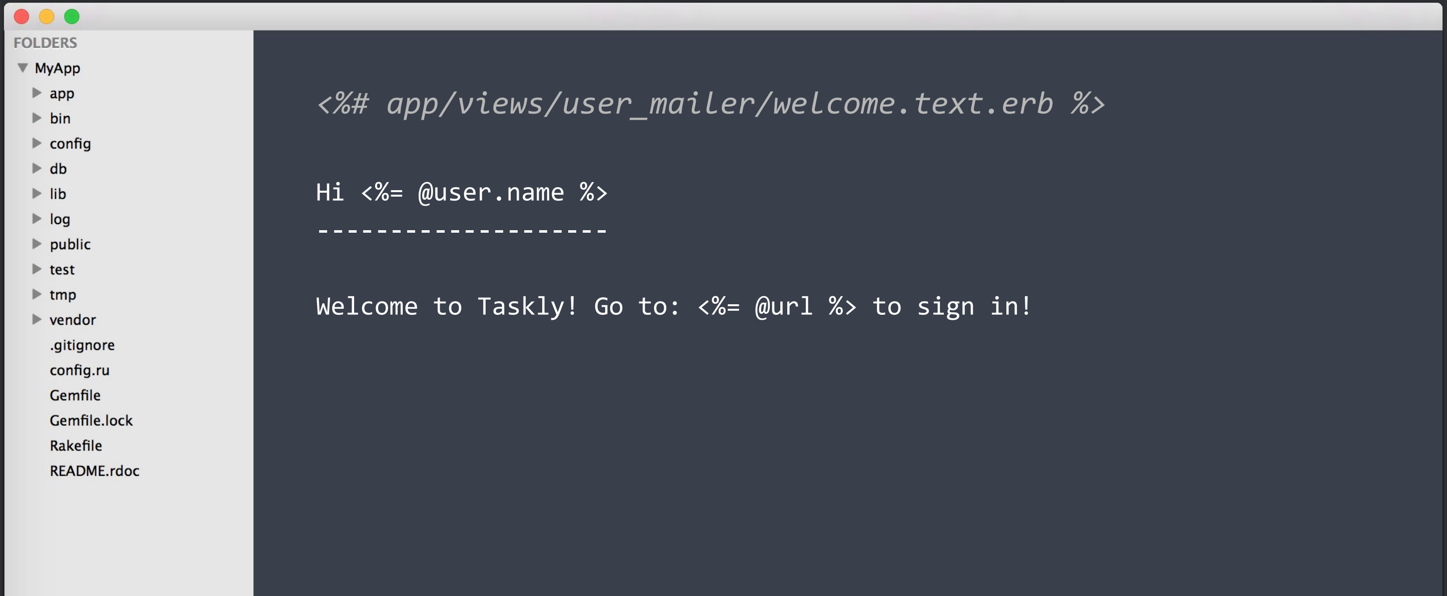
FOLDERS
- MyApp
  - app
  - bin
  - config
  - db
  - lib
  - log
  - public
  - test
  - tmp
  - vendor
  - .gitignore
  - config.ru
  - Gemfile
  - Gemfile.lock
  - Rakefile
  - README.rdoc

```
<%# 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.

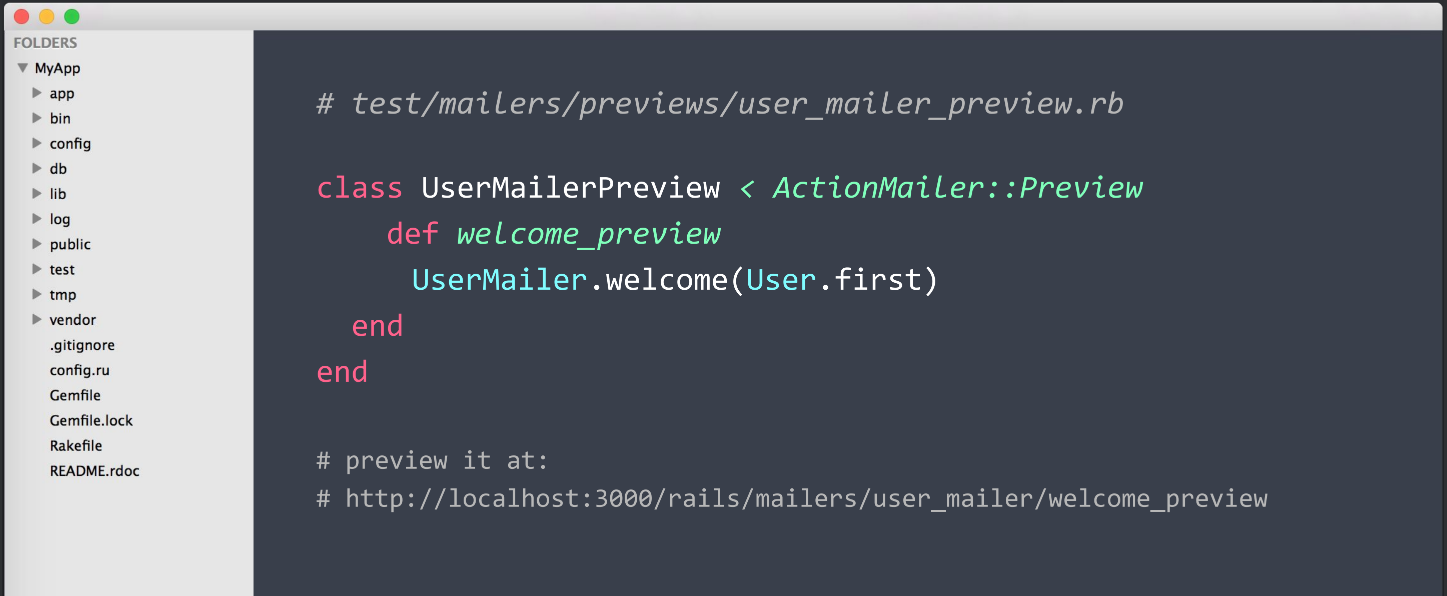


```
<%# app/views/user_mailer/welcome.text.erb %>

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/`

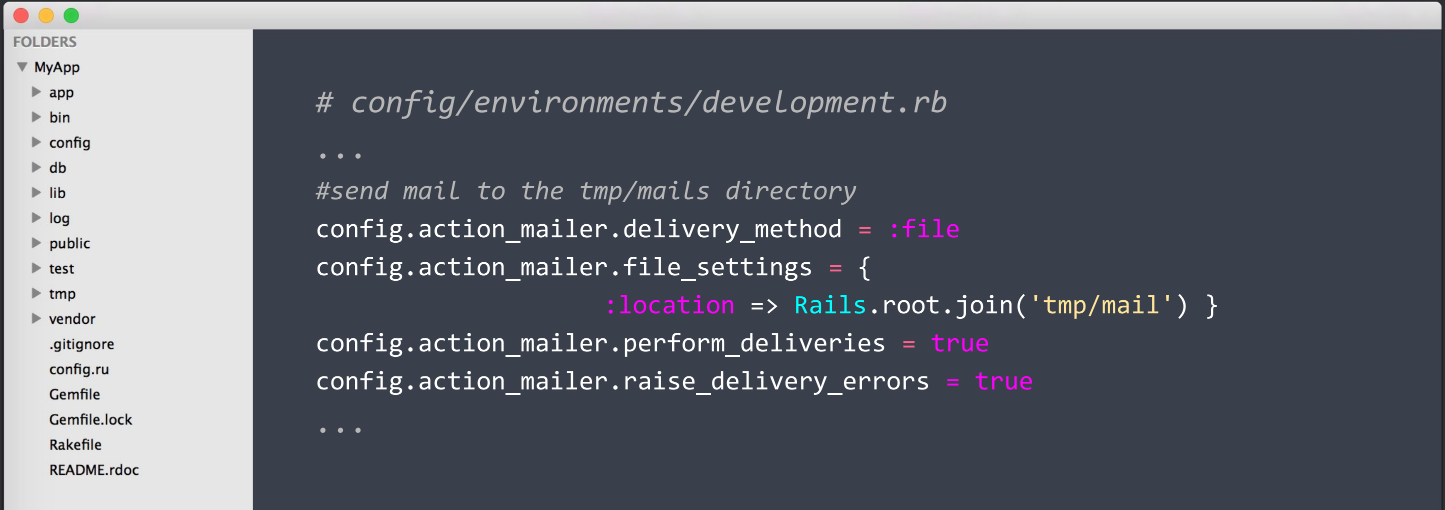


```ruby
# test/mailers/previews/user_mailer_preview.rb

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.

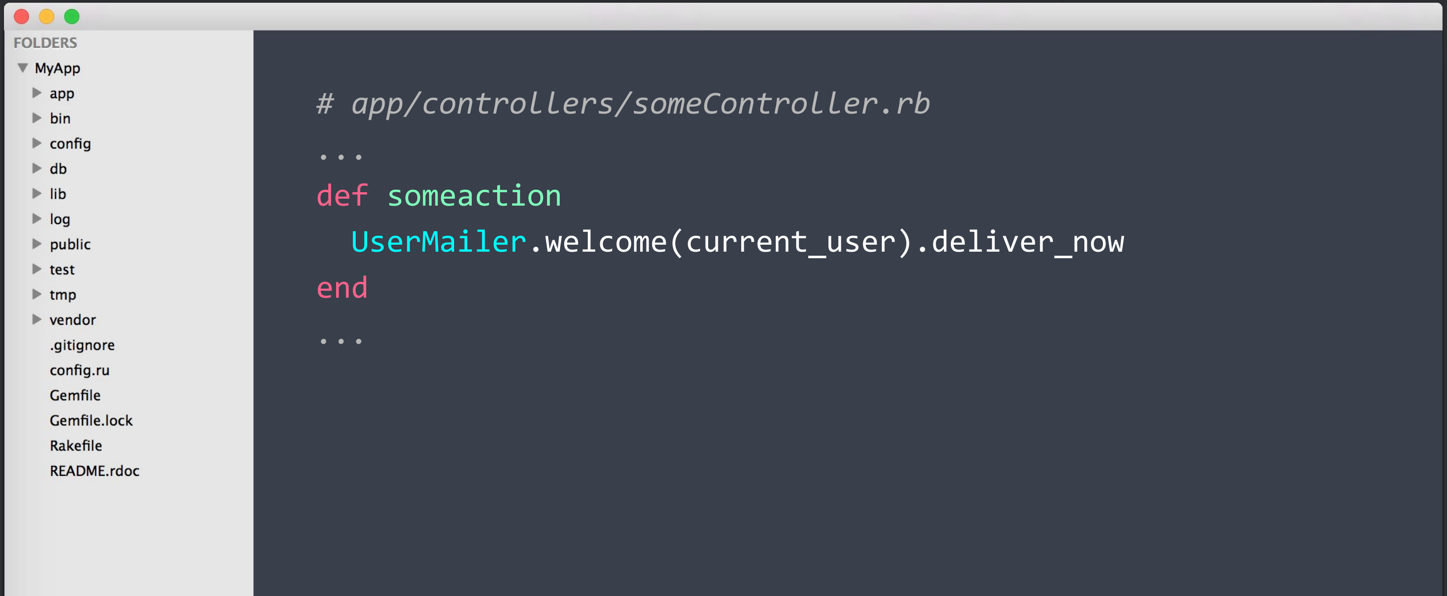


```ruby
# config/environments/development.rb
...
#send mail to the tmp/mails directory
config.action_mailer.delivery_method = :file
config.action_mailer.file_settings = {
                    :location => Rails.root.join('tmp/mail') }
config.action_mailer.perform_deliveries = true
config.action_mailer.raise_delivery_errors = 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.



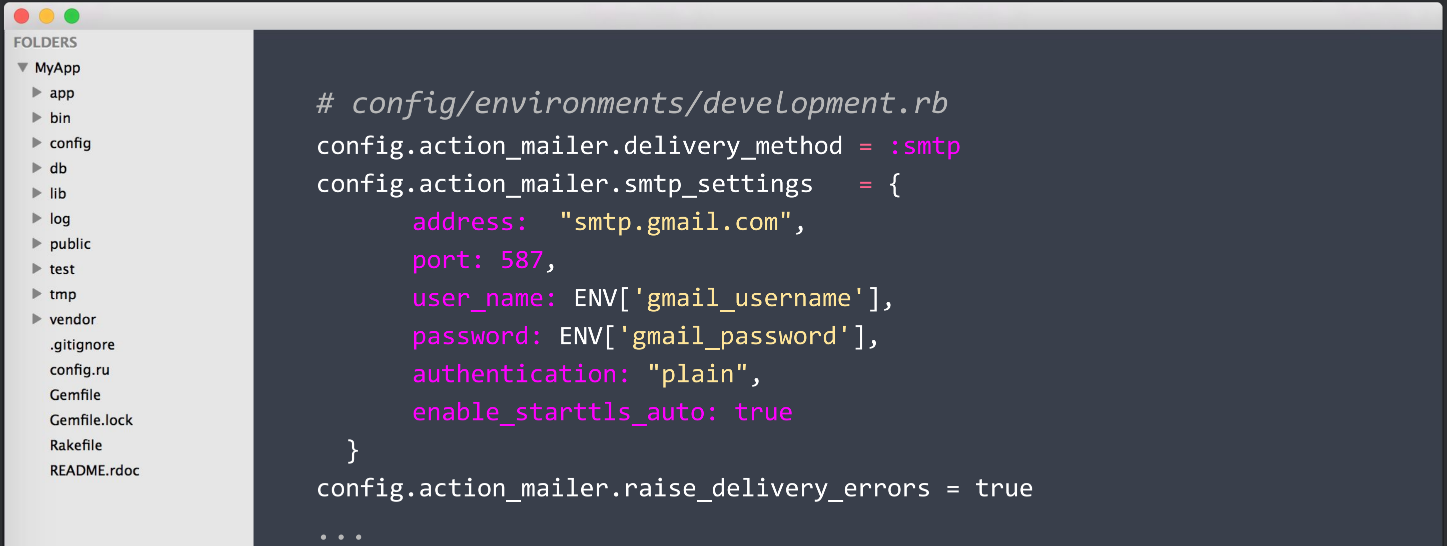
```
# app/controllers/someController.rb
...
def someaction
  UserMailer.welcome(current_user).deliver_now
end
...
```

IRON HACK

@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)



```
# 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).

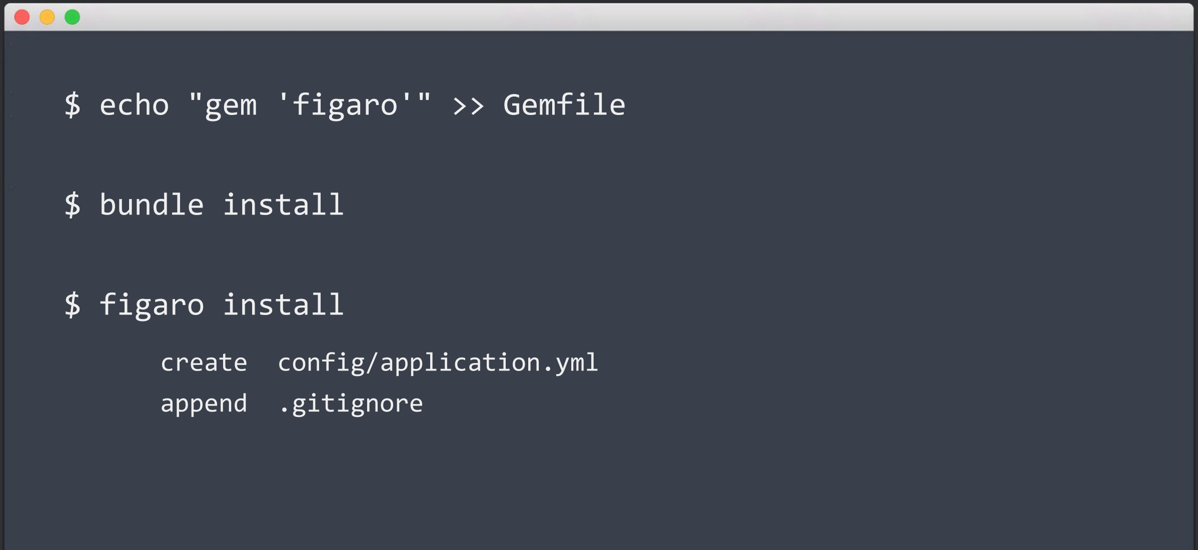
These can be stored in your OS so you app can access them, for example in your `.bashrc` file, but that makes things harder to maintain and deploy.

IRON HACK

@xharekx33

Configuring your app with Figaro

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

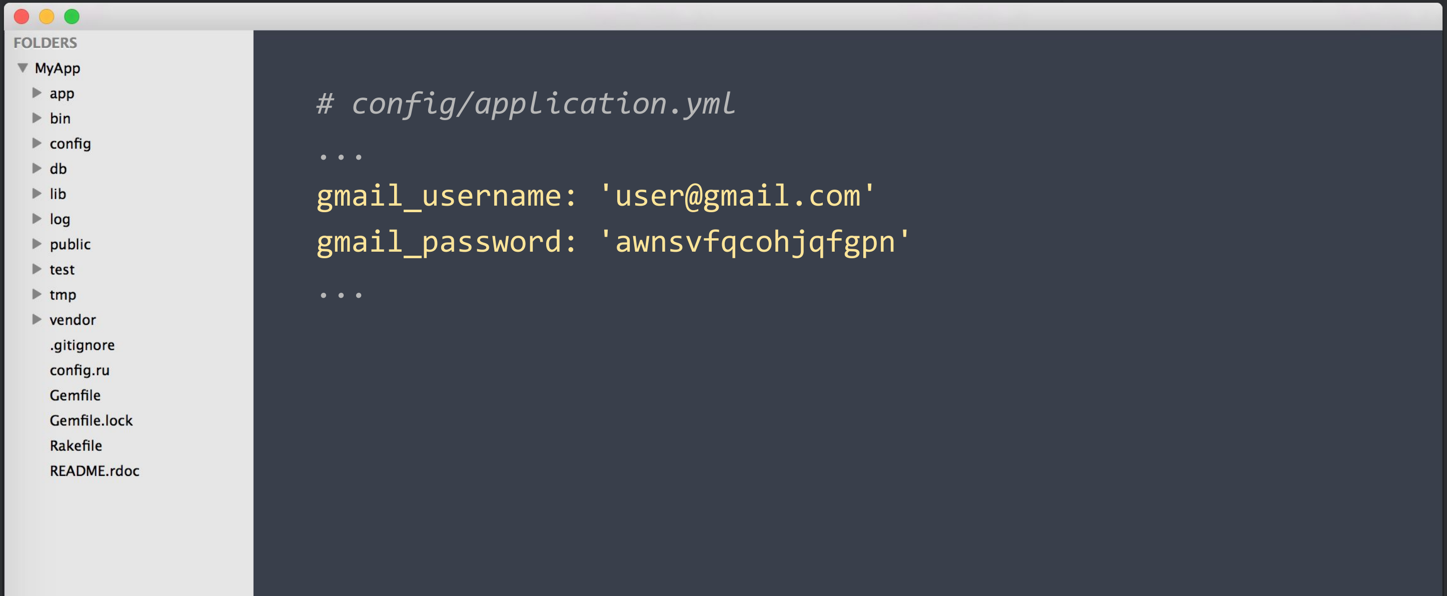


```
$ echo "gem 'figaro'" >> Gemfile

$ 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.

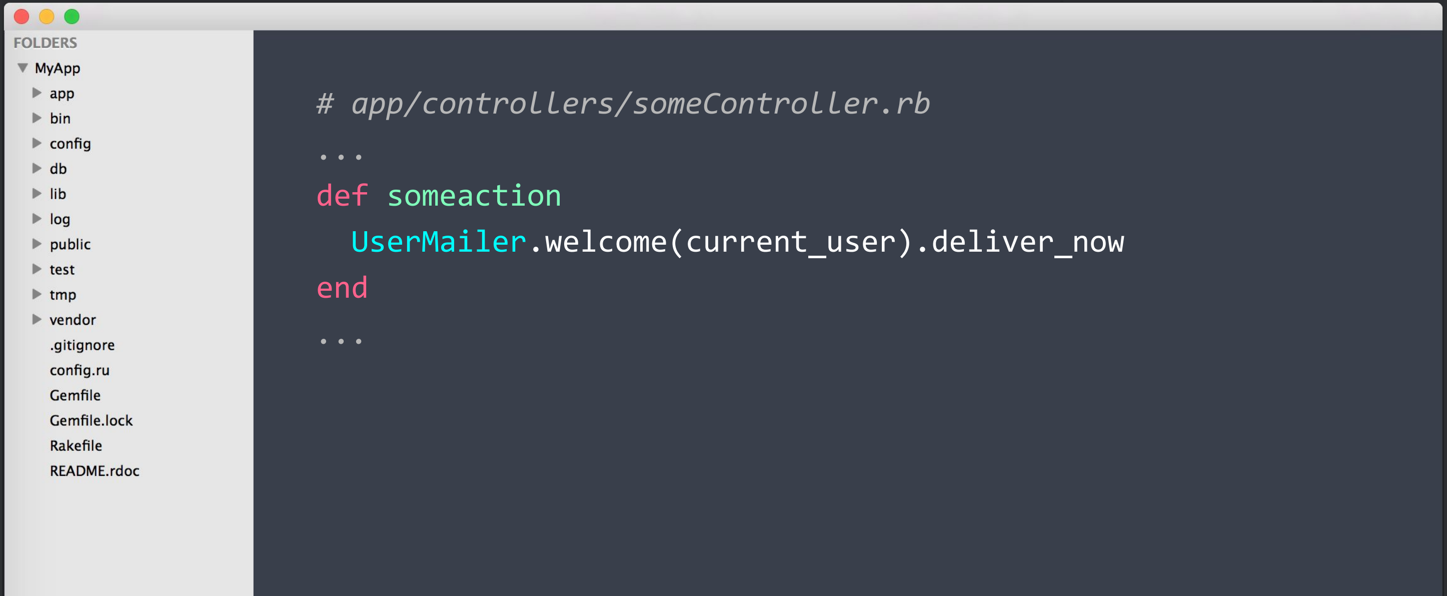


```yaml
# config/application.yml
...
gmail_username: 'user@gmail.com'
gmail_password: 'awnsvfqcohjqfgpn'
...
```

@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.



```
# app/controllers/someController.rb
...
def someaction
  UserMailer.welcome(current_user).deliver_now
end
...
```

IRON HACK

@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.

IRON HACK

@xharekx33