



# Security

Juan “Harek” Urrios  
@xharekx33

# Securing an app

Apps contain resources that can be accessed by many users. Unless we secure our app, they will be able to access all of them with no restriction whatsoever.

Sometimes we might want to allow that (in our public parts of the site) but for the most part we will want to control access to our app, through authentication and authorization.

It is your responsibility to properly secure your app.



# What is “authentication”

The process of determining whether someone or something is who/what it claims to be.

Authentication deals with establishing WHO you are.



# What is “authorization”

---

The process of determining if someone has the permission to perform an action or access a certain resource.

Authorization deals with WHAT you can do and see, once WHO you are it's been established through authentication.



# How to authenticate someone?

In the real world you use IDs, driver licenses, legal documents or certificates to identify people.

In apps we ask for usernames and passwords, provide methods for users to sign in and out and store their sessions.



# Let's get started

Let's clone the repo for our Taskly app:

[https://github.com/xharekx33/taskly\\_ironhack.git](https://github.com/xharekx33/taskly_ironhack.git)

```
$ git clone https://github.com/xharekx33/taskly_ironhack.git
$ cd taskly_ironhack
$ bundle install
$ rake db:migrate
$ rake db:seed
$ rails s
```



@xharekx33

# What needs to change in our app?

If you go to <http://localhost:3000/users> or <http://localhost:3000/tasks> you'll see that anyone can see, edit and destroy all users and tasks.

We are going to:

- Make sure users can only see, edit and delete their own tasks.
- Make sure users can only see and edit their own information.



# Adding a secure Password

User users only have a name and an email. For authentication to work they'll need to have a secure password. That means that the actual password will not be stored in the database. A hashed version of it will be saved instead.

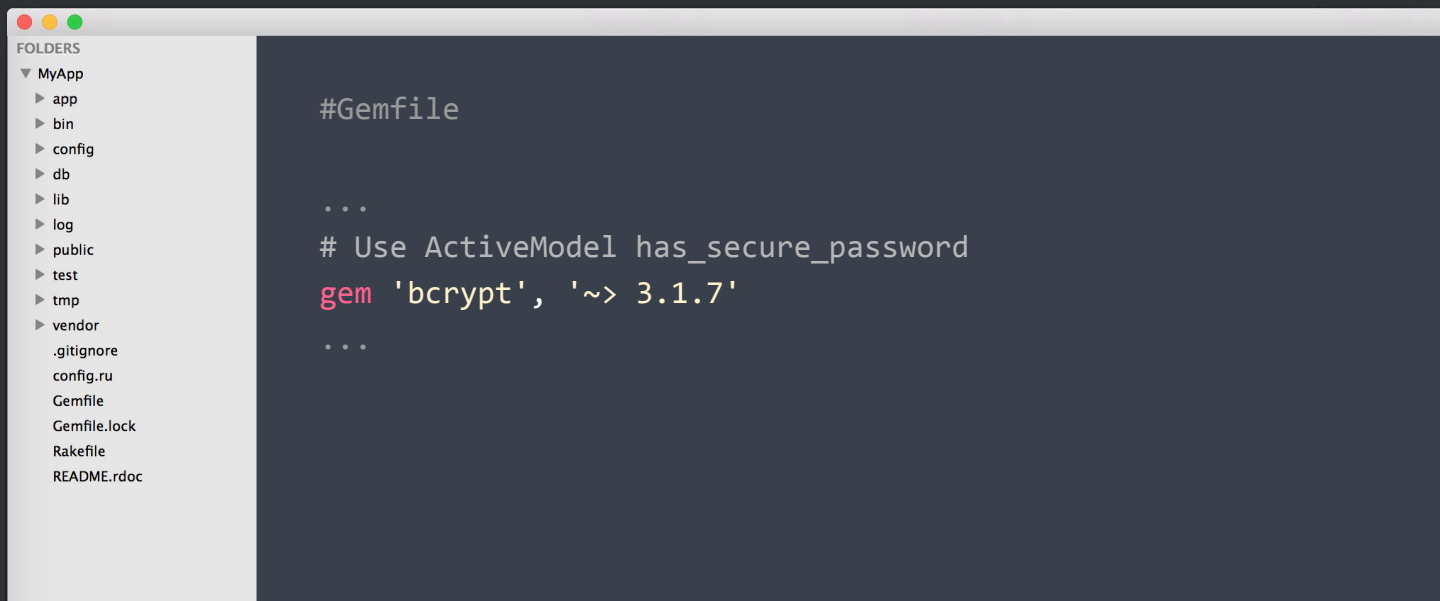
A hashed password is the result of applying an irreversible hash function to the actual password. It will be then be said that the password has been encrypted





# Adding the bcrypt gem

Open your Gemfile and uncomment the the line with the bcrypt gem.  
Run **bundle install**



@xharekx33

# What does the bcrypt gem do?

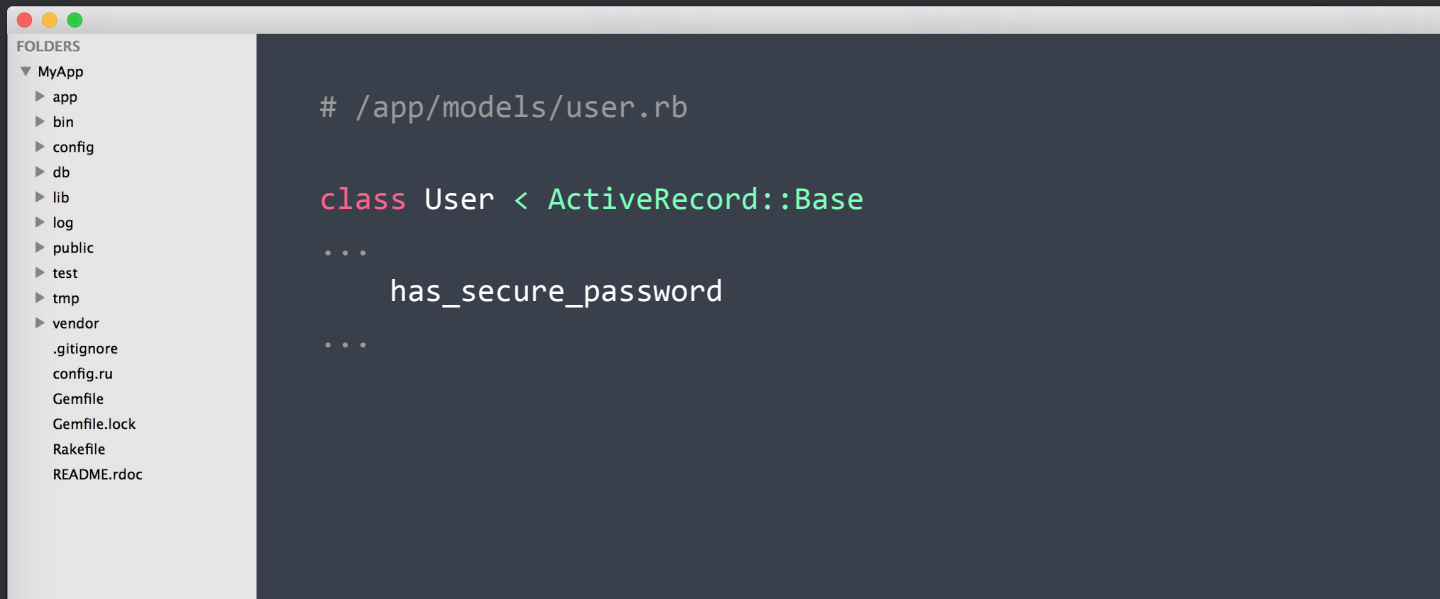
It is the Ruby binding for the OpenBSD `bcrypt()` password hashing algorithm that we will use to encrypt passwords in our app.

By design, the bcrypt algorithm produces a salted hash, that is; it adds an extra string of random characters to each password (called “salt”) before encrypting them, to protect from dictionary attacks and rainbow table attacks.



# Add the has\_secure\_password method

Include the `has_secure_password` method in your User model



The screenshot shows a code editor with a sidebar on the left displaying a file tree under 'FOLDERS'. The main editor area shows the following code:

```
# /app/models/user.rb

class User < ActiveRecord::Base
  ...
  has_secure_password
  ...
end
```

The sidebar lists the following folders and files:

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



@xharekx33

# What does has\_secure\_password do?

When included in a model the `has_secure_password` method adds:

- The ability to save a hashed `password_digest` attribute to the database
- A pair of virtual attributes: `password` and `password_confirmation`
- Presence validations on create for these, and it also validates that they match
- An authenticate method that returns the user when the password is correct (and false otherwise)

The only requirement `has_secure_password` has to work is that the model needs to have an attribute called `password_digest`



# Add a password\_digest

Create a new migration to add the `password_digest` attribute.  
Run `rake db:migrate`

```
$ rails g migration add_password_digest_to_users  
password_digest:string
```

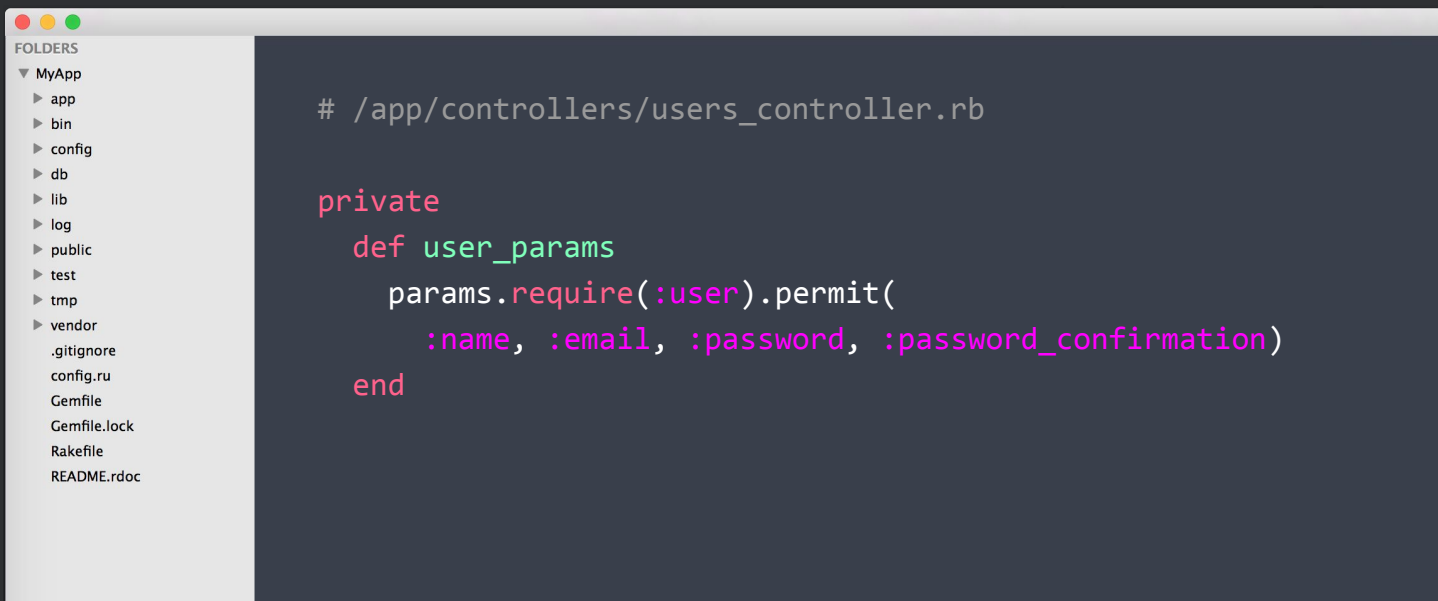
```
$ rake db:migrate
```



@xharekx33

# Strong parameters

Add `:password` and `:password_confirmation` to the list of permitted fields in our Users controller



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer, titled 'FOLDERS', shows a project structure with a 'MyApp' folder containing subfolders like 'app', 'bin', 'config', 'db', 'lib', 'log', 'public', 'test', 'tmp', and 'vendor'. It also lists files like '.gitignore', 'config.ru', 'Gemfile', 'Gemfile.lock', 'Rakefile', and 'README.rdoc'. The code editor on the right shows the content of the file `/app/controllers/users_controller.rb`. The code is as follows:

```
# /app/controllers/users_controller.rb

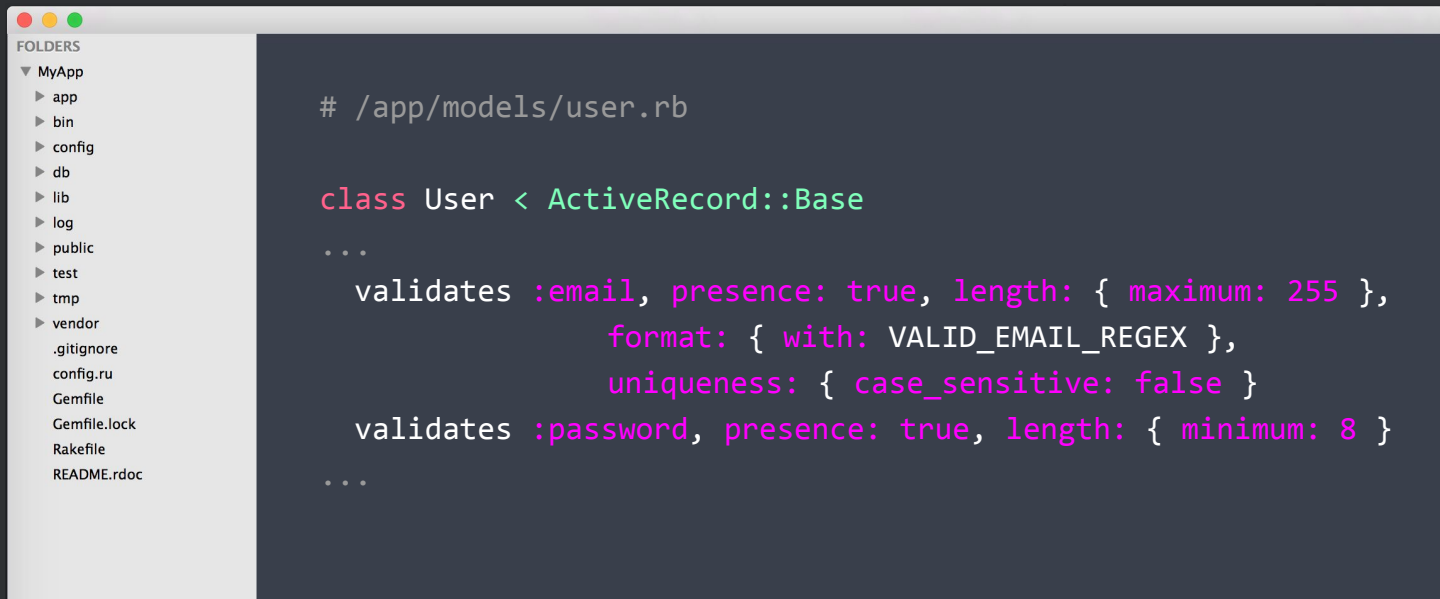
private
def user_params
  params.require(:user).permit(
    :name, :email, :password, :password_confirmation)
end
```



@xharekx33

# Add password validation

Add validations for our `:password` attribute in our User model.



The screenshot shows a code editor with a sidebar on the left displaying a file tree under the name 'FOLDERS'. The tree includes 'MyApp' 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 shows the file path '# /app/models/user.rb' and the following Ruby code:

```
# /app/models/user.rb

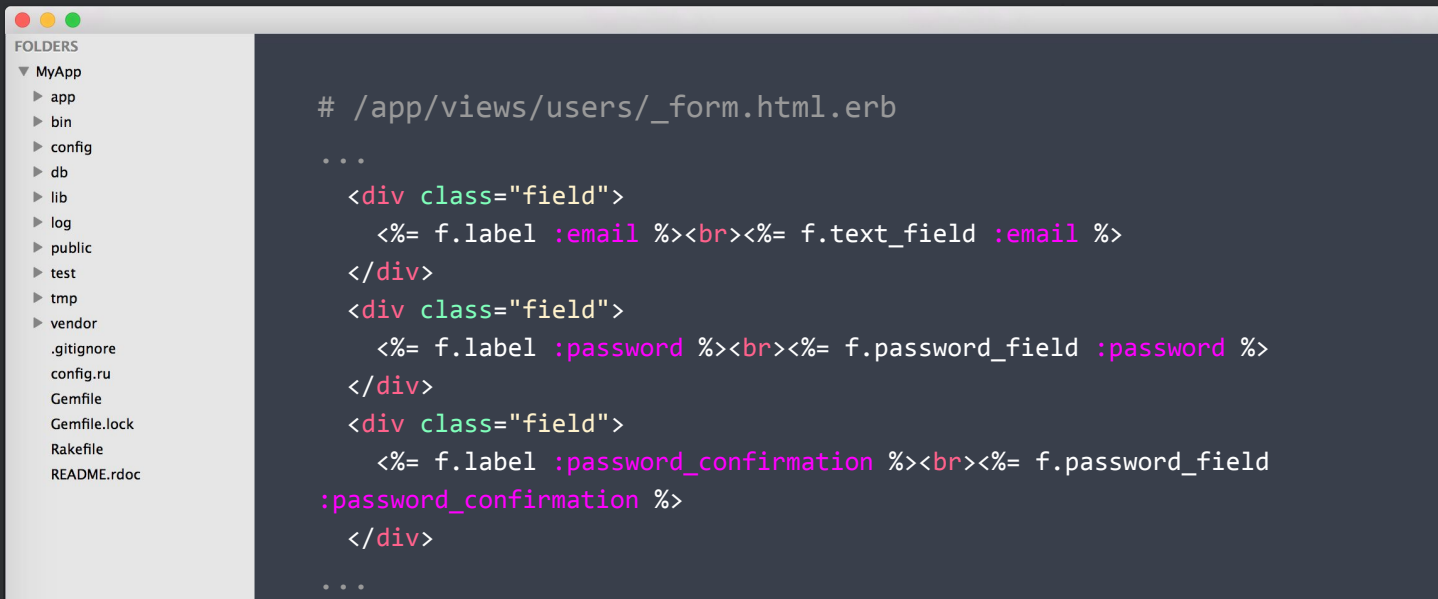
class User < ActiveRecord::Base
  ...
  validates :email, presence: true, length: { maximum: 255 },
            format: { with: VALID_EMAIL_REGEX },
            uniqueness: { case_sensitive: false }
  validates :password, presence: true, length: { minimum: 8 }
  ...
end
```



@xharekx33

# Update the signup form

Add the `:password` and `:password_confirmation` fields to the signup form.



```
# /app/views/users/_form.html.erb

...

<div class="field">
  <%= f.label :email %><br><%= f.text_field :email %>
</div>

<div class="field">
  <%= f.label :password %><br><%= f.password_field :password %>
</div>

<div class="field">
  <%= f.label :password_confirmation %><br><%= f.password_field
:password_confirmation %>
</div>

...
```



@xharekx33



# Create a new user

---

Go to the signup page and create a new user. Open **rails console** and check it was all saved correctly.

```
$ rails console
Loading development environment (Rails 4.2.3)
2.0.0-p598 :001 > User.last
  User Load (0.1ms)  SELECT "users".* FROM "users" ORDER BY
"users"."id" DESC LIMIT 1
=> #<User id: 6, name: "Harek", email: "xharekx33@gmail.com",
created_at: "2015-08-12 01:41:56", updated_at: "2015-08-12 01:41:56",
password_digest: "$2a$10$/YzTNK43q0Zua.OjL4srU00dVDGUX4eecRSLF/xVSQ8...">
2.0.0-p598 :002 >
```



@xharekx33

# Delicious cookies: Sessions

A “cookie” is small piece of data sent from a website and stored in the user’s browser.

In Rails, sessions store bits of data in a cookie, which means you access the data in it until the cookie expires or until you clear it.

HTTP is a stateless protocol. Sessions make it stateful.

Using sessions we can remember a user’s identity from page to page.



# Login & logout

Using sessions we'll give users the ability to login and logout from our app so we don't have to constantly ask for their password.

When the user logs in we'll create a session to save the user id in a cookie and we will destroy the session when the user logs out.

This way we can easily check our authenticated user's identity in every page.



# Sessions controller

---

We'll create a new **sessions controller** to handle our /login and /logout routes, that will match its new, create and destroy actions

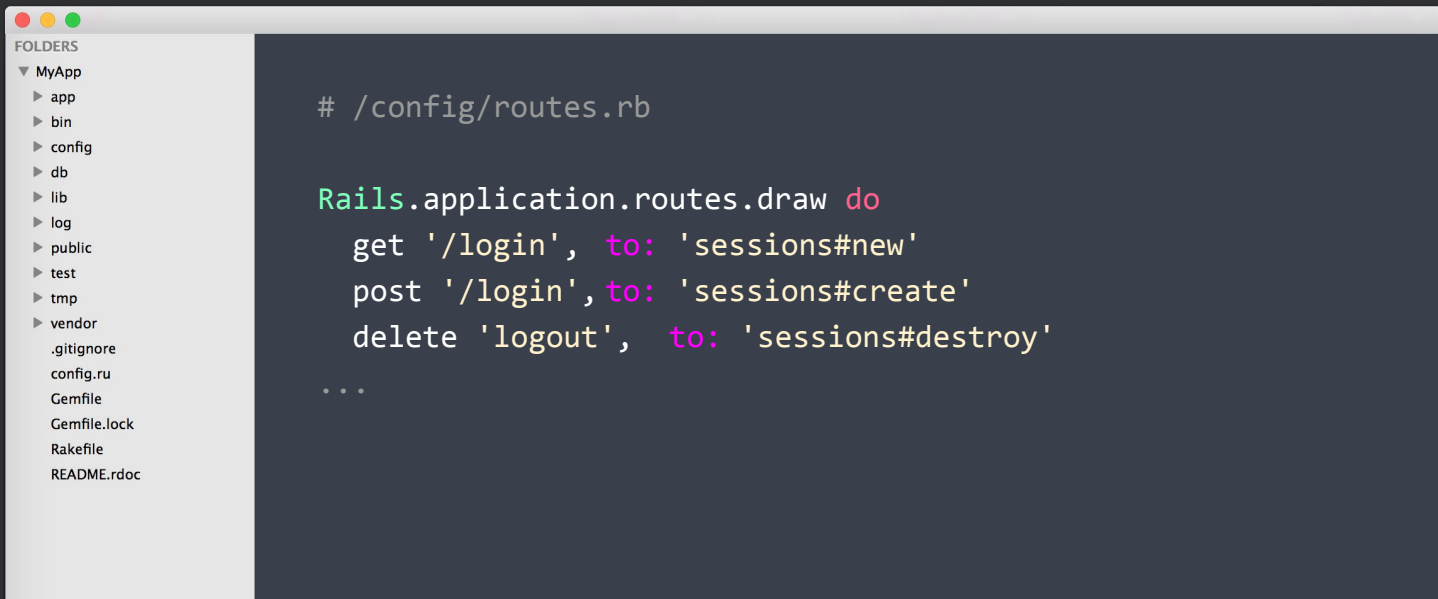
```
$ rails g controller Sessions new create destroy
  create  app/controllers/sessions_controller.rb
  route   get 'sessions/destroy'
  route   get 'sessions/create'
  route   get 'sessions/new'
  invoke  erb
  create   app/views/sessions
  create   app/views/sessions/new.html.erb
  create   app/views/sessions/create.html.erb
  create   app/views/sessions/destroy.html.erb
```



@xharekx33

# Login and logout routes

Add the necessary routes for our sessions controller actions



The screenshot shows a code editor with a sidebar on the left displaying the project structure under 'FOLDERS'. The main editor area shows the content of `config/routes.rb`. The code defines routes for the sessions controller: a GET route for `/login` pointing to `sessions#new`, a POST route for `/login` pointing to `sessions#create`, and a DELETE route for `/logout` pointing to `sessions#destroy`. The code is as follows:

```
# /config/routes.rb

Rails.application.routes.draw do
  get '/login', to: 'sessions#new'
  post '/login', to: 'sessions#create'
  delete 'logout', to: 'sessions#destroy'
  ...
end
```

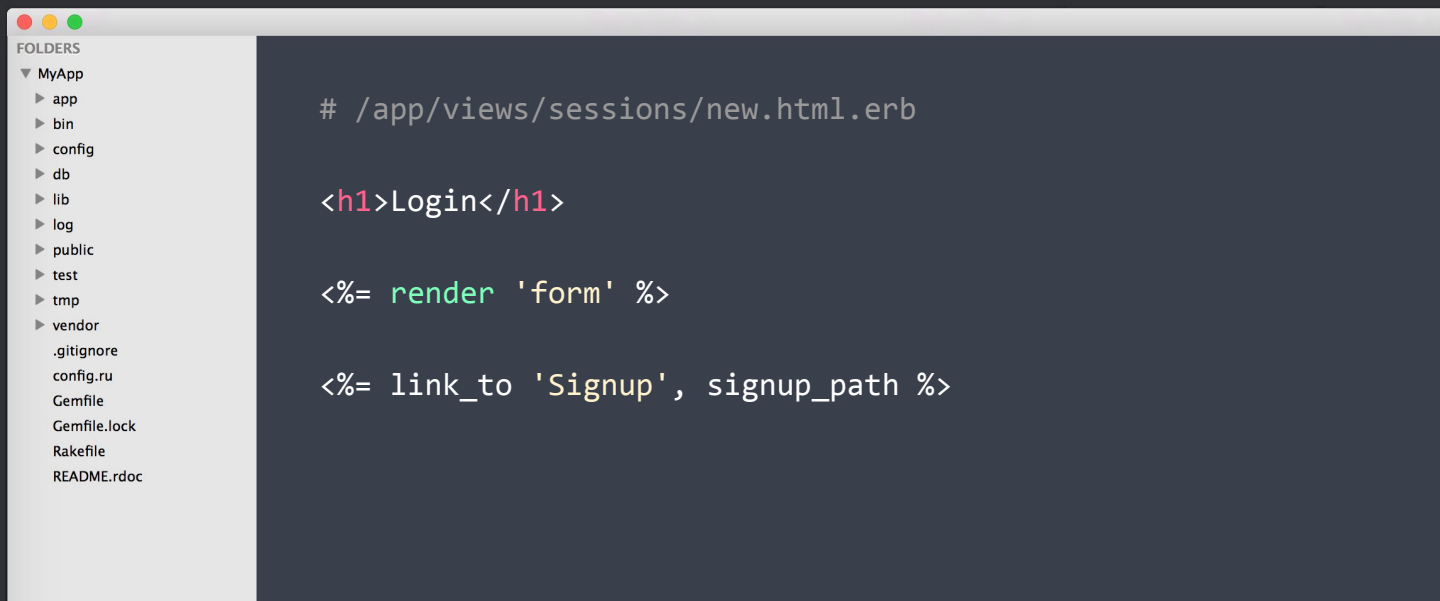


@xharekx33

# Sessions new view

---

The view in `app/views/sessions/new.html.erb` will be similar to the one for creating new users



The image shows a code editor window with a sidebar on the left displaying a file tree under 'FOLDERS'. The main editor area contains the following code:

```
# /app/views/sessions/new.html.erb

<h1>Login</h1>

<%= render 'form' %>

<%= link_to 'Signup', signup_path %>
```

The sidebar file tree includes:

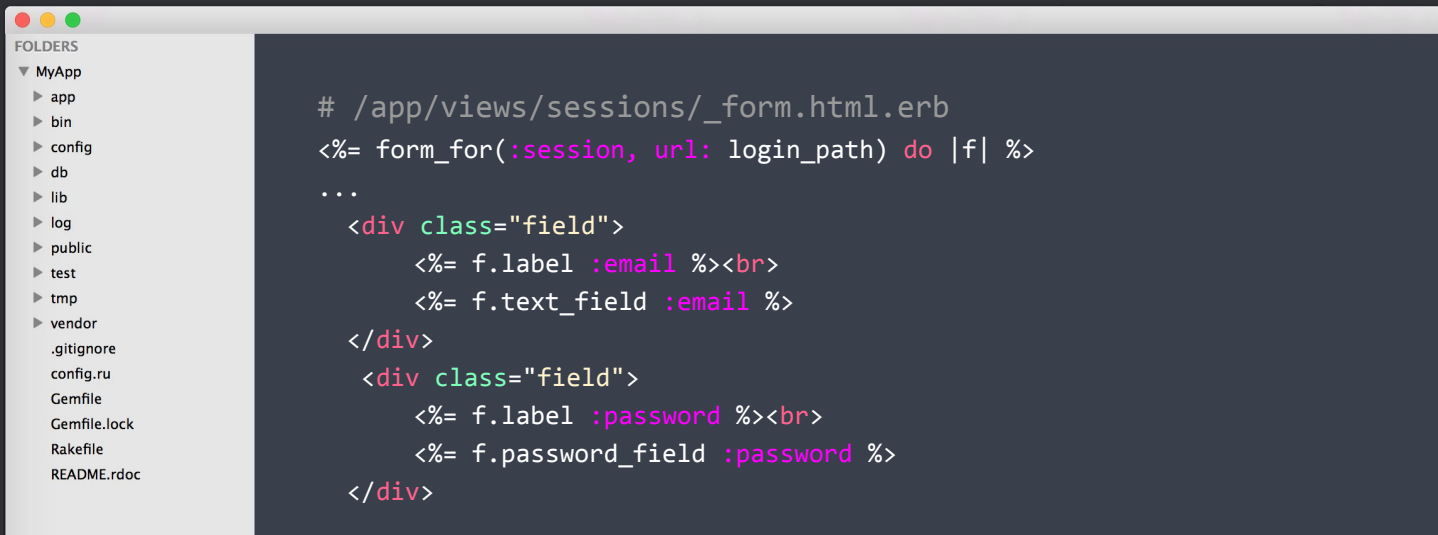
- ▼ MyApp
  - ▶ app
  - ▶ bin
  - ▶ config
  - ▶ db
  - ▶ lib
  - ▶ log
  - ▶ public
  - ▶ test
  - ▶ tmp
  - ▶ vendor
- .gitignore
- config.ru
- Gemfile
- Gemfile.lock
- Rakefile
- README.rdoc



@xharekx33

# Login form

The form partial in `app/views/sessions/_form.html.erb` will be similar to the one for creating new users, but will only include `:email` and `:password` fields



```
# /app/views/sessions/_form.html.erb
<%= form_for(:session, url: login_path) do |f| %>
  ...
  <div class="field">
    <%= f.label :email %><br>
    <%= f.text_field :email %>
  </div>
  <div class="field">
    <%= f.label :password %><br>
    <%= f.password_field :password %>
  </div>
end
```

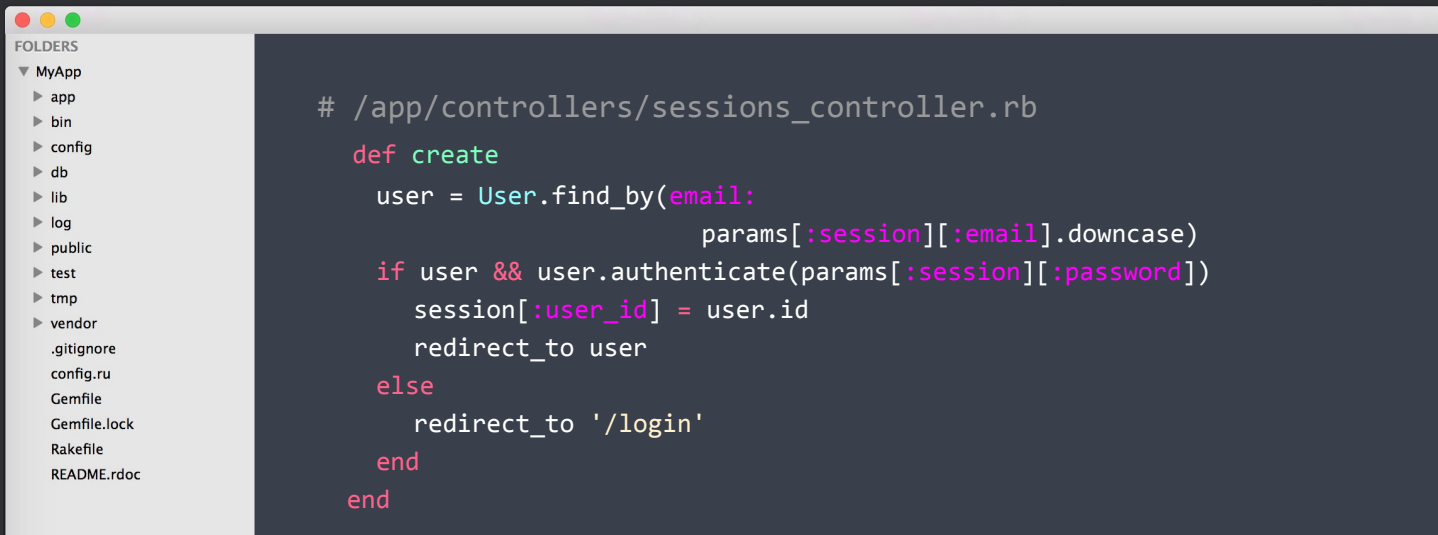


@xharekx33

# Sessions controller: create

---

Fill in the **create** action in the sessions controller so it creates the cookie with the `user_id` in the session if the password is correct and redirects to the login form if it isn't.



```
# /app/controllers/sessions_controller.rb

def create
  user = User.find_by(email:
                        params[:session][:email].downcase)
  if user && user.authenticate(params[:session][:password])
    session[:user_id] = user.id
    redirect_to user
  else
    redirect_to '/login'
  end
end
```

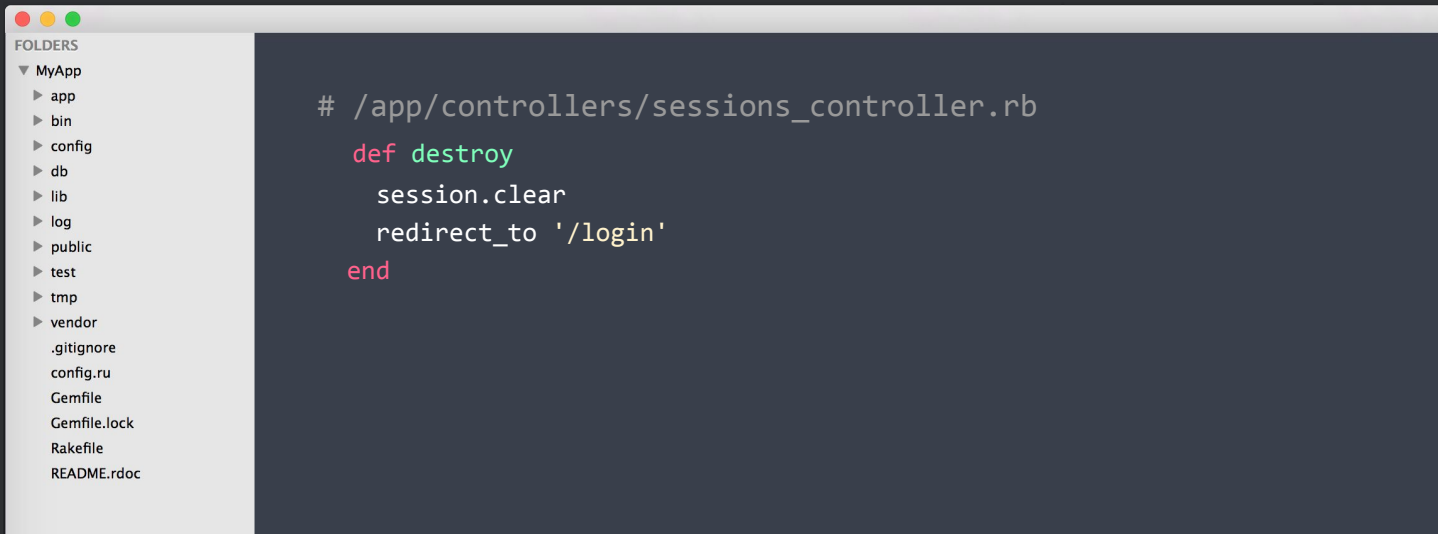


@xharekx33



# Sessions controller: destroy

Fill in the **destroy** action in the sessions controller so it clears the user data saved in the session.



@xharekx33

# Searching for the current user

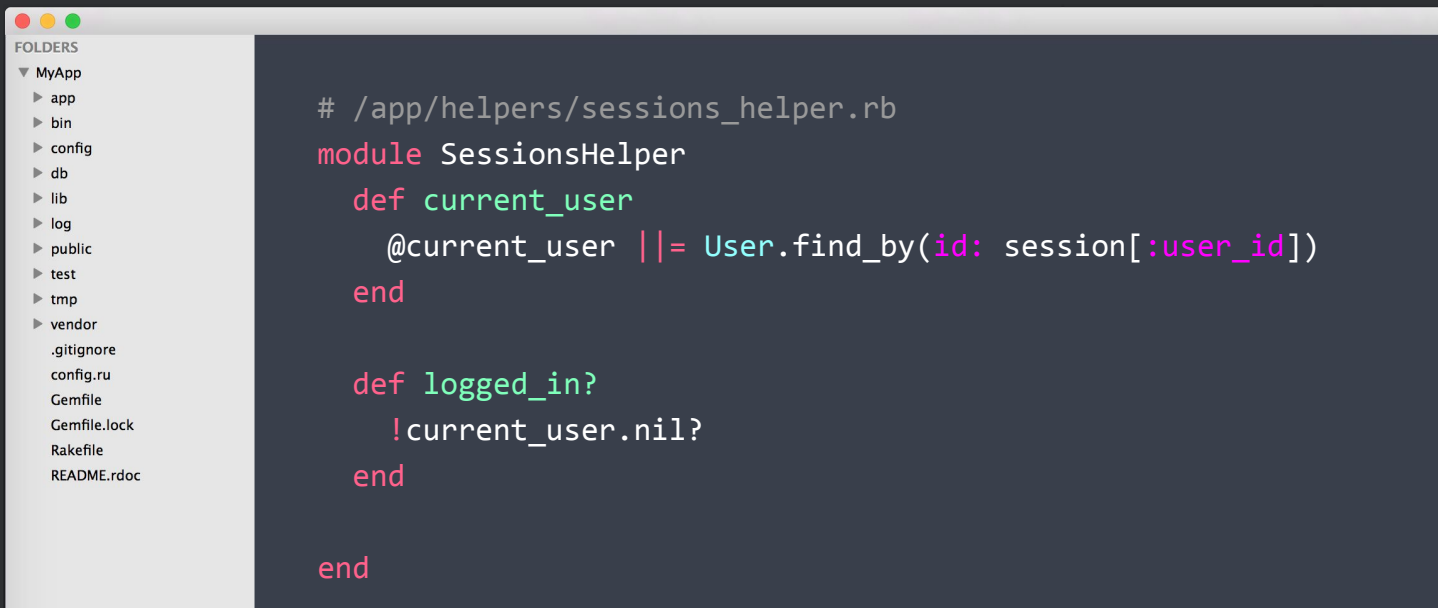
Now that our users can log in, we should do something with their session data.

To avoid having to search for the current user with `session[:user_id]` we will extract that code into a `current_user` helper method so it's available for us to use anywhere in our application.



# Add sessions helpers

We'll add helpers to check who the current user is and whether they're logged in



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' directory 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 the content of a file named '# /app/helpers/sessions\_helper.rb'. The code defines a 'SessionsHelper' module with two methods: 'current\_user' and 'logged\_in?'. The 'current\_user' method uses 'User.find\_by(id: session[:user\_id])' to retrieve the current user. The 'logged\_in?' method checks if 'current\_user' is nil. The code is color-coded: comments are grey, module names are red, method names are green, and variable names are magenta.

```
# /app/helpers/sessions_helper.rb
module SessionsHelper
  def current_user
    @current_user ||= User.find_by(id: session[:user_id])
  end

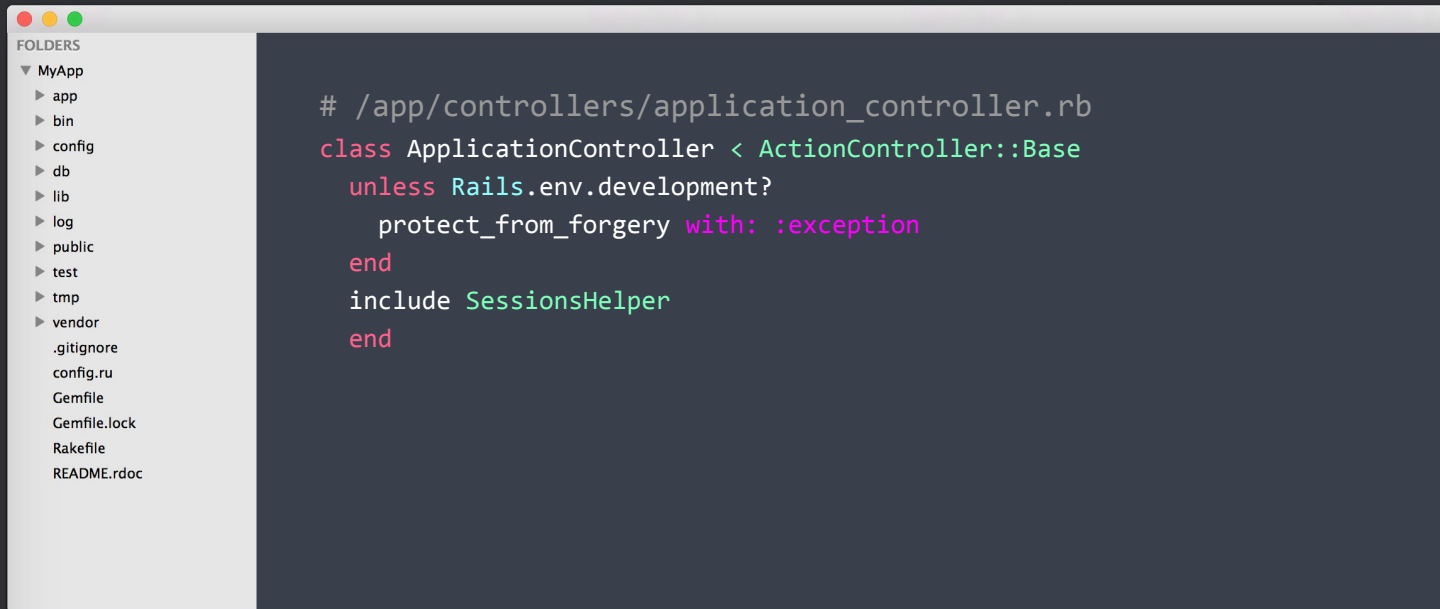
  def logged_in?
    !current_user.nil?
  end
end
```



@xharekx33

# Make session helpers available to controllers

Add it these to the application controller so controllers can use them too



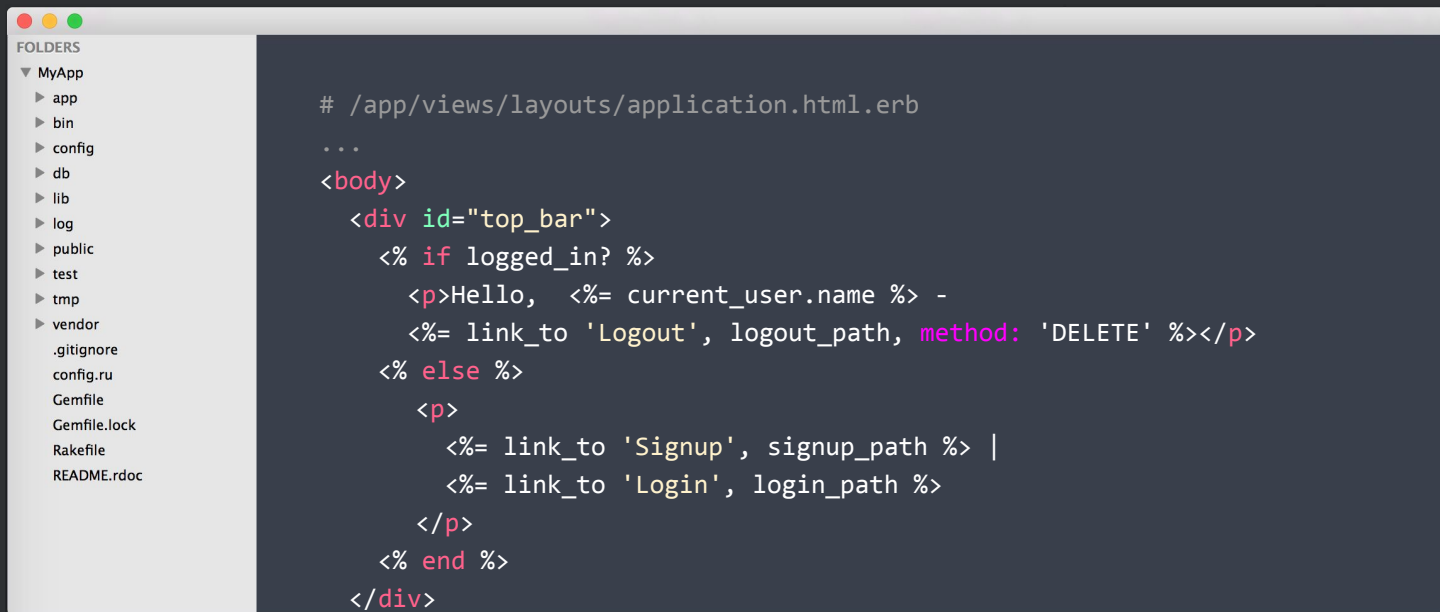
```
# /app/controllers/application_controller.rb
class ApplicationController < ActionController::Base
  unless Rails.env.development?
    protect_from_forgery with: :exception
  end
  include SessionsHelper
end
```



@xharekx33

# Add login and logout links

Now, using our view helpers, we can show the appropriate logout link to authenticated users, and signup and login links to anonymous users



```
# /app/views/layouts/application.html.erb
...
<body>
  <div id="top_bar">
    <% if logged_in? %>
      <p>Hello, <%= current_user.name %> -
      <%= link_to 'Logout', logout_path, method: 'DELETE' %></p>
    <% else %>
      <p>
        <%= link_to 'Signup', signup_path %> |
        <%= link_to 'Login', login_path %>
      </p>
    <% end %>
  </div>
```



@xharekx33

# Give it a try

- Create a user in the signup page
- Use it to login
- Make sure the correct links are shown on the top bar



# Exercise

- Change the Tasks controller index action so authenticated users are only able to list their own tasks.
- Make a before\_action filter that checks if the current user is the owner of a task so users can only edit and delete their own tasks.
- Users should only be able to edit their own name, email and password. Users won't be able to delete other users
- Make all routes in the app are restricted to authenticated users except for the login and signup pages.
- Instead of using `session[:user_id]` use a more secure session token for the current\_user helper

