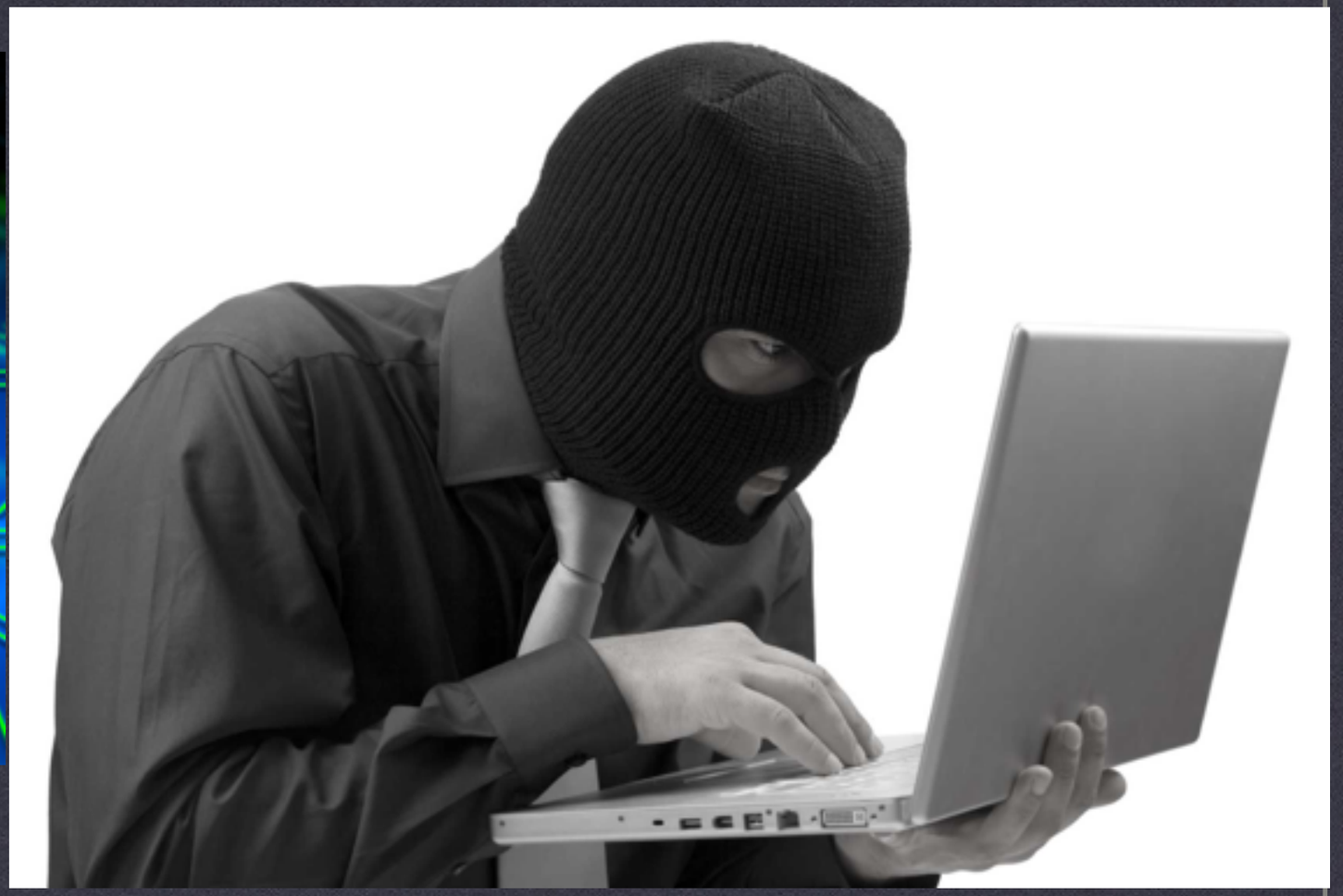


PROJECT

AUTHENTICATION AND SECURITY — HARD THINGS TO FIND A SANE IMAGE FOR.

DATE

JAN 5, 2016

CLIENT

RECAP FROM LAST LESSON

The road ahead

- We have just SIX more classes before you present final projects
- We will have ample time in class to work on our projects. Lessons will be a bit shorter.
- We will learn about auth today, then give you some time to start on your final projects in class, using what we learn in class.
- Still to come: Bootstrap, Heroku, and third party APIs!

Homework

- I did my best to tally where you all are homework wise. If you got an email from me, you are/were missing something.
- You need to complete 8 out of 10 homeworks to get your certificates. I am counting both versions of happitails as two each.

Final projects

- I still need to talk about final projects with some of you, so let's do that.
- Everyone else can continue on HappiTails on Rails

AUTHENTICATION, THE EASY WAY

Soon or later, your app will have to learn who is using it

- Right now, any rando can just walk into our app and change data all willy nilly
- Users should be able to only update stuff that belongs to them.
- Today we will learn how to do that.

We will not do this from scratch, however

- There are hard ways to do things and easy ways to do things. Rails embraces the easy way.
- I personally see to reason to reinvent the wheel, or for that matter, user authentication.
- Just as Rails makes the business of deploying a web application fiendishly easy, there is a handy gem for handling auth called Devise.
- Devise is spearheaded by Jose Valim of Plataformatec, one of the most prolific programmers in the Rails community.

Let's make an app

- Head to your class folder and create a new rails app called `blogr`
- Open it up in Sublime Text. See that file at the root of the project called `Gemfile`?

Gemfile

- This is where you tell rails which gems to use. All you need is gem 'gemname' and it grabs the gem from rubygems.org

```
# Bundle edge Rails instead: gem 'rails', github: 'rails/rails'
gem 'rails', '4.2.4'
# Use sqlite3 as the database for Active Record
gem 'sqlite3'
# Use SCSS for stylesheets
gem 'sass-rails', '~> 5.0'
# Use Uglifier as compressor for JavaScript assets
gem 'uglifier', '>= 1.3.0'
# Use CoffeeScript for .coffee assets and views
gem 'coffee-rails', '~> 4.1.0'
# See https://github.com/rails/execjs#readme for more supported runtimes
# gem 'therubyracer', platforms: :ruby

# Use jquery as the JavaScript library
gem 'jquery-rails'
# Turbolinks makes following links in your web application faster. Read
/rails/turbolinks
gem 'turbolinks'
# Build JSON APIs with ease. Read more: https://github.com/rails/jbuilde
gem 'jbuilder', '~> 2.0'
# bundle exec rake doc:rails generates the API under doc/api.
gem 'sdoc', '~> 0.4.0', group: :doc

# Use ActiveModel has_secure_password
# gem 'bcrypt', '~> 3.1.7'

# Use Unicorn as the app server
# gem 'unicorn'

# Use Capistrano for deployment
# gem 'capistrano-rails', group: :development

group :development, :test do
```

Gemfile

- There is a section for development, test, and production. If you put a gem in one of those blocks, it will only work in that mode.

- For instance, there are tons of apps that make development easier such as the default gem web-console. You don’t want this in production at all!

- You can specify a particular *release* of the gem, or even point to a github repository to get the most bleeding edge new version.

Installing gems

- Put the gem 'gemname' code in the appropriate place in the app.
- Go to your terminal and run bundle install.
- This command executes the gemfile and changes Gemfile.lock, which you shouldn’t edit yourself.
- Now you have devise available in your app!

Install devise

- Your app now has access to devise's generators.
- Run `rails g devise:install`
- Now 10 amazing modules are installed in your app, and the console output even tells you what you have to do next!
- Let's do what it says.

Devise setup

- Give the development environment a default URL.
- Define a root route for "welcome#index" and create view
- Add notice and alert messages to the application layout.

```
<p class="notice"><%= notice %></p>
<p class="alert"><%= alert %></p>
```

- #4 we don’t need.
- #5 lets you customize the auth-related views.

Devise setup

- Now, we tell devise which model to include its modules (remember modules?). The obvious one is User. If User doesn't exist, devise will generate one.
- run `rails g devise user`
- Then `rake db:migrate`

What happened?

- Look at rake routes.
- What are sessions? Registrations?

In the browser

- Run the server
- Go to http://localhost:3000/users/sign_up
- Nice! We can create users with secure passwords just like that.

Devise notices

- Devise comes with flash messages. Let's put them in the application layout so we can see a confirmation or warning after signing up/in

```
<% if notice %>
  <p><%= notice %></p>
<% end %>
<% if alert %>
  <p><%= alert %></p>
<% end %>
```

Throw in some links to those Devise routes

- Let's put some links in so the user can click to get to the login page.
- Put this in application.html.erb and give it a try.

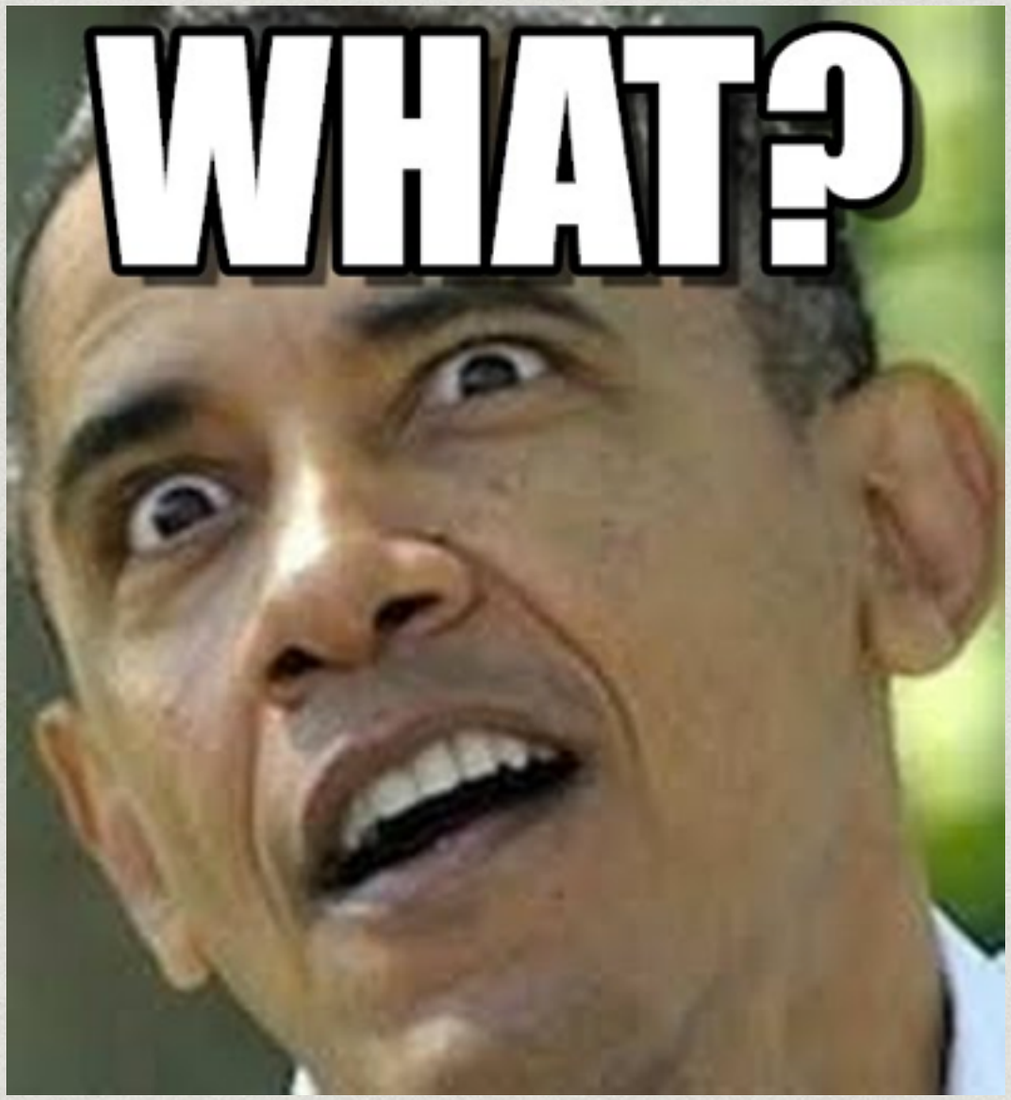
```
<%= link_to 'Edit profile', edit_user_registration_path %>
<%= link_to "Logout", destroy_user_session_path, method: :delete %>
<%= link_to "Sign up", new_user_registration_path %>
<%= link_to "Login", new_user_session_path %>
```

See how this is crappy?

- Even if I am not logged in, I see a link to log out.

See how this is crappy?

- Even if I am not logged in, I see a link to log out.



See how this is crappy?

- Even if I am not logged in, I see a link to log out.
- Fortunately, devise provides you with a suite of handy view helpers so we can put in if statements to display the right stuff in the views.

View helpers

- What would you do with these?

```
<% user_signed_in? %>

<% current_user %>
```

- Use your brain, docs, and your fellow students to display only edit profile and logout if signed in, and display sign up and log in if you aren't.
- Bonus: Next to those links, put a message telling the email address that he or she is signed in as.

CODE ALONG:
ADDING OUR OWN STUFF TO THE DEVISE USER

Before actions

- We only want signed-in users to see posts.
- It would be madness to put if statements all over the place.
- We can use a before_action on posts to force users to authenticate before viewing posts.

More code along

- Now that we can make users, let's make posts.
- Go ahead and make a show and index action for users' posts. I'll give you some time to do this yourselves.
- Think about how you can use current_user to make sure that a user can only make posts for herself
- We'll do the new action together

If you haven't yet

- Let's start those projects! Virtually all of you will have a user or perhaps an admin model. Let's use what we just learned and get started!
- If you already started and there is a user model, you can still install devise for it.
- If you need help, look to the docs, and call us over if that doesn't answer your question.