RAILS AUTH - DIY

AUTHENTICATION IN RAILS & API, DIY INSTEAD OF DEVISE

Created by Jesse Wolgamott / @jwo

AUTHENTICATION & AUTHORIZATION

- Authentication: Knowing who you are
- Authorization: Granting access based on conditions

WHY NOT JUST USE DEVISE?

- Difficult to understand
- Difficult to customize
- Difficult to extend

GENERAL FLOW:

1. Request Page
2. Redirect to Sign in if not signed in
3. Sign in
4. Request Page
5. Now we view page!

REDIRECT TO SIGN IN IF NOT SIGNED IN

```
class YourController < ApplicationController
  before_action do
    if @current_user.nil?
      redirect_to sign_in_path, alert: "Please Sign In"
    end
  end
end
```

```
if @current_user.nil?
```

So, we need something that sets `@current_user`

```
redirect_to sign_in_path
```

So, we need a `sign_in_path`

```
class ApplicationController < ActionController::Base
  protect_from_forgery with: :exception

  before_action do
    @current_user = User.find_by id: session[:user_id]
  end
end
```

- Before every single action is executed, Rails will look at the session and get the user_id
- It will try to find a User by that id, so `current_user` may be nil
- Every action now has access to `@current_user`

SessionController:

```
class SessionsController < ApplicationController
  def new
  end
  def create
  end
  def delete
  end
end
```

Routes

```
get 'sign_in' => 'sessions#new', as: :sign_in
post 'sign_in' => 'sessions#create'
delete 'sign_in' => 'sessions#delete'
```

We'll end up at `/sign_in` if we are not already signed in. We need a form:

```
<%= form_tag do %>

  <div>
    <%= label_tag :username %>
    <%= text_field_tag :username, params[:username] %>
  </div>

  <div>
    <%= label_tag :password %>
    <%= password_field_tag :password, "" %>
  </div>

  <div>
    <%= submit_tag "Sign In", class: "btn"%>
  </div>

<% end %>
```

Since we did not specify a location, Rails will use `POST` and the `/sign_in` path. That matches our Sessions#create. Yay!

Now we get to the core of the matter. How can we store a user's password, verify the user's password is correct, but never be able to reverse engineer the user's password?

BCRYPT

HAS_SECURE_PASSWORD

```ruby
class User < ApplicationRecord
  has_secure_password
  validates :username, presence: true, uniqueness: true
end
```

This requires us to have a database field named `password_digest`.

THIS REQUIRES US TO HAVE A DATABASE FIELD NAMED `PASSWORD_DIGEST`.

- when you set a user's password `@user.password = '12345'`, it will encrypt it into `password_digest`
- You cannot reverse engineer `12345`
- You must give the password again to see if it's correct:

```
@user.authenticate("12345")
=> #<User id="3"..../>
@user.authenticate("42")
=> nil
```

```ruby
class SessionsController < ApplicationController

  def create

    user = User.find_by username: params[:username]
    if user && user.authenticate(params[:password])
      session[:user_id] = user.id
      redirect_to root_path, notice: "Signed in!"
    else
      flash.now[:alert] = "Something is wrong with your username and/or p
      render :new
    end
  end
end
```

PROTIPS

```ruby
class ApplicationController < ActionController::Base
  protect_from_forgery with: :exception

  before_action do
    @current_user = User.find_by id: session[:user_id]
  end

  def authenticate_user!
    unless @current_user do
      redirect_to sign_in_path, notice: "Please Sign In"
    end
  end

  def current_user
    @current_user
  end
  helper_method :current_user
```

Allow you to in your controllers:

```
class YourController < ApplicationController
  before_action :authenticate_user!
end
```

Allows you to use in your ERB views:

```erb
<%= if user_signed_in? %>
  Hi <%= current_user %>.
<% else %>
  <%= link_to 'Sign Up', new_user_path %>
<% end %>
```

SOOO BASICALLY, WE CREATED DEVISE.

- It's Secure
- It's rather easy to implement
- It's **Easy to Extend and Customize**

WHAT TO TAKE AWAY FOR BOTH DIY *AND* DEVISE

- `@current_user` is not special. It's an ActiveRecord Object
- All the routes and controllers aren't special: you can customize then.

SECURING AN API

HAS_SECURE_TOKEN

Requires you to have a `token` field.

```ruby
class User < ActiveRecord::Base
  has_secure_token
end

user = User.new
user.save
user.token # => "pX27zsMN2ViQKta1bGfLmVJE"
```

Each web request could then pass a token as an auth_token parameter.

```
class Api::UsersController < ApiController
  before_action do
    @current_user = User.find_by token: params[:auth_token]

    render "Auth Token Required", status: 401 unless @current_user
  end
end
```

SIGNING IN TO API

```ruby
class Api::SessionsController < ApiController
  def create
    @current_user = User.find_by username: params[:username]
    if @current_user && @current_user.authenticate(params[:password])
      render json: {user: @current_user, auth_token: @current_user.token
    else
      render errors: ["Username or Password is Invalid"], status: 422
    end
  end
end
```

DOWNSIDES

- There's only one token per user
- If regenerated, would sign out all phones/sessions/etc

DOORKEEPER

DOORKEEPER IS AN OAUTH 2 PROVIDER FOR RAILS

ENABLING PASSWORD GRANT.

`/config/initializers/doorkeeper.rb`

```
Doorkeeper.configure do
  orm :active_record

  resource_owner_from_credentials do
    User.find_by(email: params[:username]).try(:authenticate, params[:pas
  end

  access_token_methods :from_bearer_authorization,:from_access_token_para


  grant_flows %w(password)

end
Doorkeeper.configuration.token_grant_types << "password"
```

- from_bearer_authorization (header): `'Authorization': 'Bearer TOKENHERE'`
- from_access_token_param : `?access_token=TOKENHERE`

IN YOUR CONTROLLER

```
class Api::BooksController < Api::V1::ApiController
  before_action :doorkeeper_authorize!

  def index
   render json: {books: current_user.books}
  end

  private

  def current_user
    User.find(doorkeeper_token.resource_owner_id) if doorkeeper_token
  end
end
```

1. Already have User with secure password
2. add to gemfile `doorkeeper`
3. `bundle install`
4. Add file `config/initializers/doorkeeper.rb`
5. Add to routes: `use_doorkeeper`
6. `rails generate doorkeeper:migration`
7. `rails db:migrate`

SIGNING IN

POST JSON to `/oauth/token`

```
{
  "grant_type": "password",
  "username": "jwo",
  "password": "12345"
}
```

Result will have

```
{
  "auth_token": "eebdaddb2c2de2817dbd6bebe06b0a7ffa34ffd38adeb7d0",
  "expires": 1234567890
}
```

BENEFITS OF DOORKEEPER

1. Multiple signing per account
2. Can be expanded later
3. Fits into Grape

LIVE DEMO

WE'LL DO IT LIVE

Links:

- t: @jwo / g: @jwo
- Slides
- Code (Before and After)