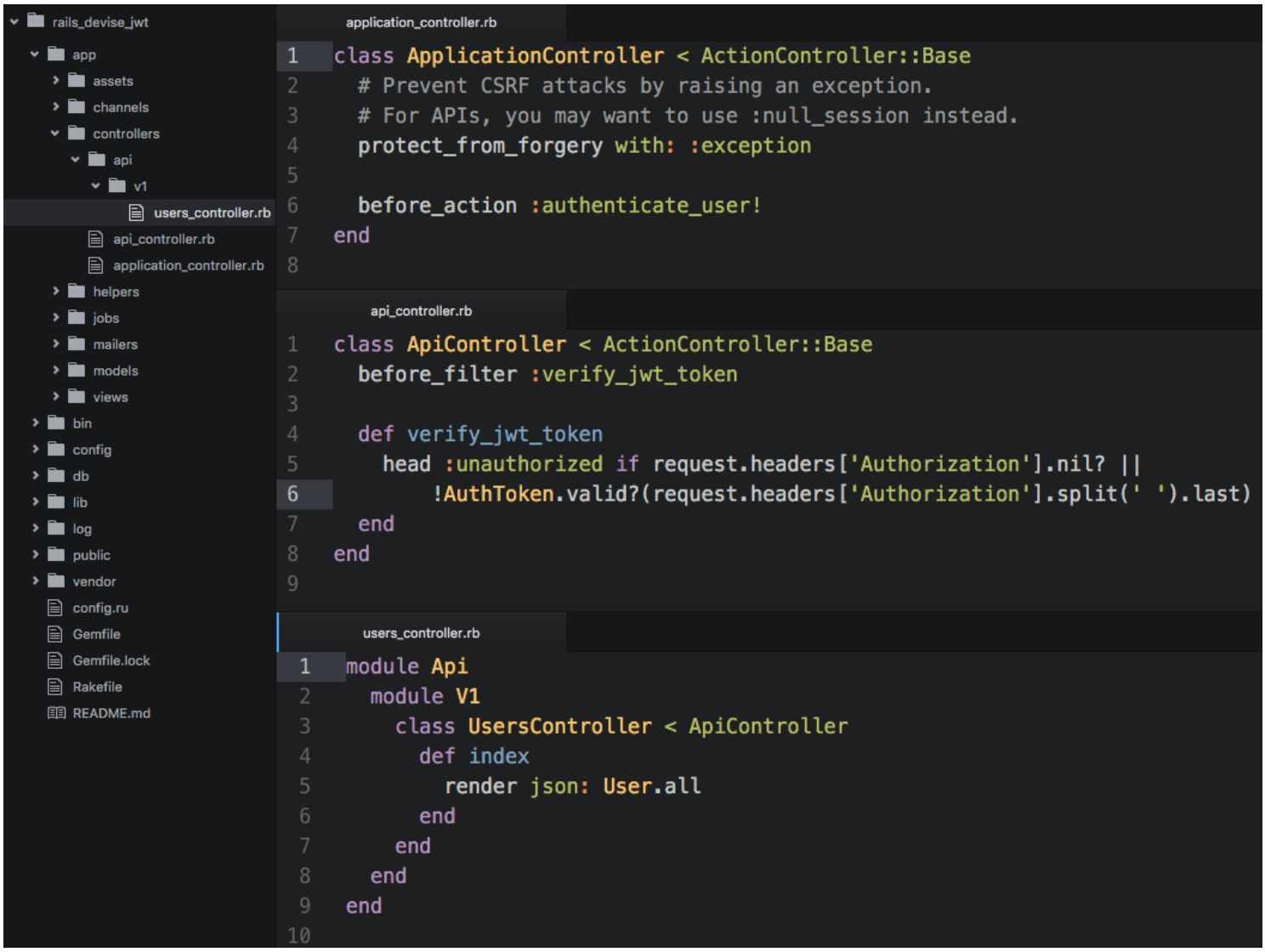
João Gonçalves Follow
Apr 15 · 2 min read

# Rails, Devise, JWT and the forgotten Warden

Long story short: Most of the examples that we found under the *Rails devise jwt examples* google search do not make full use of **devise's** capabilities.

Using a custom **before_action** under the **ApiController** to verify if the request contains a valid **JWT token** and ignoring the already existing **before_action :authenticate_user!** is not the best "devise example".




not cool

It will render all of **Devise**'s helpers like **current_user, user_signed_in?** useless and is actually ignoring all the work that the authors' had to make **Devise** into a:

> *"Flexible authentication solution for Rails with Warden"* ***devise gemspec description***

## So, what is Warden, you ask.

**Warden** is a ruby gem that does the actual authentication through an array of strategies; when the first strategy fails to authenticate the user, it uses the next and so forth.

**Devise** adds several strategies to **Warden**, according to your **User Model** configuration (if it is **rememberable**, **database_authenticatable**, etc.) and delegates the real authentication process to **Warden**, if one of the strategies (user cookie session, user password credentials, etc.) turns out ok, **Devise** will trust **Warden**'s judgment and behave accordingly.

### Cool, lets build our own JWT Strategy and add it to Warden's list

With this new strategy added to the top of the list, if you make a request with a **JWT token** in the **HTTP Authorization Header** that strategy will kick in and **Warden** will tell **Devise** that the user is logged in, just like when you make a request with a **cookie session**.

Adding a new way to authenticate your user should not require you to change your app, this authentication middleware (JWT strategy) will allow your app to trust **Devise** like it should, **before_action :authenticate_user!** will enforce both your **JWT authentication** and your **User Model** configuration, **current_user** will return the authenticated user, **user_signed_in?** will correctly return true or false and all of the other helpers will behave like they should.

So without further ado…

. . .

## Changes you need to make in order to make Devise use JWT Header Authentication

```ruby
# Add somewhere to your Devise configuration
# "config/initializers/devise.rb"

Devise.setup do |config|
  # ...

  config.warden do |manager|
    # Registering your new Strategy
    manager.strategies.add(:jwt, Devise::Strategies::JsonWe

    # Adding the new JWT Strategy to the top of Warden's li
    # Scoped by what Devise would scope (typically :user)
    manager.default_strategies(scope: :user).unshift :jwt
  end

  # ...
end
```

**devise.rb** hosted with ♥ by **GitHub** **view raw**

```ruby
# Add the "https://github.com/jwt/ruby-jwt" gem to your "Gem
gem 'jwt'
```

**Gemfile** hosted with ♥ by **GitHub** **view raw**

```ruby
# Add the following code to a view
# It will show you an example of how to make a
# curl HTTP request with the proper authentication headers.
# Be sure to actually use a working route and not "http://lo

<% if user_signed_in? %>
  curl -X GET --header 'Authorization: Bearer <%= JWTWrapper
<% end %>

```

**index.html.erb** hosted with ♥ by **GitHub** **view raw**

```ruby
# Your actual JWT Strategy
# "config/initializers/core_extensions/devise/strategies/js

```

```ruby
module Devise
  module Strategies
    class JsonWebToken < Base
      def valid?
        request.headers['Authorization'].present?
      end

      def authenticate!
        return fail! unless claims
        return fail! unless claims.has_key?('user_id')

        success! User.find_by_id claims['user_id']
      end

      protected ######################## PROTECTED ########

      def claims
        strategy, token = request.headers['Authorization'].

        return nil if (strategy || '').downcase != 'bearer'

        JWTWrapper.decode(token) rescue nil
      end
    end
  end
```

Cool, hope it helps anyone.