
A
STUDY
IN
DEVISE
THE FIRST ADVENTURE IF
KL RUBY BRIGADE

What is Devise

- Flexible authentication solution for Rails with Warden by Plataformatec
- A gem I've used for a long long time
- A gem that still mystify us (don't they all)

Devise 4.0 works with Rails 4.1 onwards. You can add it to your Gemfile with:

```
gem 'devise'
```

Run the bundle command to install it.

Next, you need to run the generator:

```
$ rails generate devise:install
```

```
$ rails generate devise MODEL
```

Controller filters and helpers

Devise will create some helpers to use inside your controllers and views. To set up a controller with user authentication, just add this before_action (assuming your devise model is 'User'):

```
before_action :authenticate_user!
```

Magic much?

Purpose

- Learn how to dig deep
- Learn how gems are built
- Pick up some tips & tricks

Study Area

- Basic building blocks
- Generators
- Routing
- Modules

Basic Building Blocks

- warden - Rack based Auth framework
- orm_adapter - For DB connections
- bcrypt - For password encryption
- railties - For Rails generator & stuffs
- responders - respond_with + respond_to helpers

Responders

- respond_with + respond_to Controller helpers
- separated from Rails after 4.2.0

```ruby
class Api::V1::ThingsController < ApplicationController
  respond_to :json

  # POST /api/v1/things
  def create
    @thing = Thing.create(thing_params)
    respond_with :api, :v1, @thing
  end
end
```

Warden

The What

Warden is a Rack-based middleware, designed to provide a mechanism for authentication in Ruby web applications. It is a common mechanism that fits into the Rack Machinery to offer powerful options for authentication.

Warden is designed to be lazy. That is, if you don’t use it, it doesn’t do anything, but when you do use it, it will spring into action and provide an underlying mechanism to allow authentication in any Rack-based application.

```ruby
env['warden'].authenticated?                # Ask the question if a request has been
env['warden'].authenticated?(:foo)          # Ask the question if a request is auther
env['warden'].authenticate(:password)       # Try to authenticate via the :password s
env['warden'].authenticate!(:password)      # Ensure authentication via the password
```

```ruby
env['warden'].authenticate(:password)
env['warden'].user # the user object
```

```ruby
warden = env['warden']
if warden.authenticated?(:admin)
  warden.authenticated?(:user) && warden.logout(:user)
  warden.session(:admin)[:redirect_back] = "/admin/path/to/somewhere"
  warden.set_user(@user, scope: :user)
end
```

Demo

Generators

- lib/generators/active_record
- lib/generators/devise
- lib/generators/mongoid
- lib/generators/templates

2 Creating Your First Generator

Since Rails 3.0, generators are built on top of Thor. Thor provides powerful options for parsing and a great API for manipulating files. For instance, let's build a generator that creates an initializer file named `initializer.rb` inside `config/initializers`.

The first step is to create a file at `lib/generators/initializer_generator.rb` with the following content:

```
class InitializerGenerator < Rails::Generators::Base
  def create_initializer_file
    create_file "config/initializers/initializer.rb", "# Add
initialization content here"
  end
end
```

To invoke our new generator, we just need to do:

```
$ bin/rails generate initializer
```

Devise Routes

- lib/devise/rails/routes.rb
- def devise_for

Look at Code

Array#extract_options!

extract_options!() *public*

Extracts options from a set of arguments. Removes and returns the last element in the array if it's a hash, otherwise returns a blank hash.

```
def options(*args)
  args.extract_options!
end

options(1, 2)        # => {}
options(1, 2, :a => :b) # => {:a=>:b}
```

Route Contraints

- http://api.rubyonrails.org/classes/ActionDispatch/Routing/Mapper/Scoping.html#method-i-constraints

- devise_scope

Devise::Mapping

- lib/devise/mapping.rb
- Stores all information about a certain mapping (devise_for object)

Modules

- Database Authenticable
- Omniauthable
- Confirmable
- Recoverable
- Registrable
- Rememberable
- Trackable
- Timetable
- Validatable
- Lockable

Look at Code

Modules

- lib/devise/models.rb

- def devise(*modules)

Splat Operator (*)

You can use a splat in a method definition to gather up any remaining arguments:

```ruby
def say(what, *people)
  people.each{|person| puts "#{person}: #{what}"}
end

say "Hello!", "Alice", "Bob", "Carl"
# Alice: Hello!
# Bob: Hello!
# Carl: Hello!
```

Q&A