plataformatec / devise

- Watch Unwatch
- Fork
- 5,685
- 667

- Code
- Network
- Pull Requests 2
- Issues 6
- Wiki 69
- Stats & Graphs

Flexible authentication solution for Rails with Warden. — Read more

http://blog.plataformatec.com.br/tag/devise/

- Clone in Mac
- ZIP

- HTTP
- Git Read-Only

https://github.com/platafor

Read-Only access

- Tags 63
- Downloads 0

- *branch:* master
  Switch Branches/Tags
  Filter branches/tags
  - Branches
  - Tags

  adding-ci-status-to-readme

  gh-pages

  master

  rails-3-2-fixes

  v1.0

  v1.1

  v1.2

  v1.2.oauth

  v1.3

  v1.4

  v1.5

- Files

- Commits
- Branches 11

Latest commit to the **master** branch

Show a warning message in case routes are not mounted in the main app.

commit dc37b82298
 josevalim authored 1 day ago
devise /

name	age	history message
app	February 01, 2012	Extract auth_options into its own method. [josevalim]
config	January 24, 2012	Add deprecations, update changelog. [josevalim]
gemfiles	January 10, 2012	Use global Gemfile in travis-ci [rafaelfranca]
lib	1 day ago	Show a warning message in case routes are not mounted in the main app. [josevalim]
test	January 24, 2012	Merge pull request #1565 from joliss/unconfirmed-message [josevalim]
.gitignore	December 12, 2011	Add Gemfile back to repository. [josevalim]
.travis.yml	January 23, 2012	fix travis build [nashby]
CHANGELOG.rdoc	January 27, 2012	Release 2.0.0 [josevalim]
Gemfile	January 21, 2012	Update to 3.2.0 [josevalim]
Gemfile.lock	January 28, 2012	Improve the message for case_insensitive_keys. [josevalim]
MIT-LICENSE	January 25, 2012	2012 [ci skip] [carlosantoniodasilva]
README.md	February 03, 2012	Added a documentup link on README [rodrigoflores]
Rakefile	July 02, 2011	Removed a rdoc deprecation warning [rodrigoflores]
devise.gemspec	January 10, 2012	Change gemspec to avoid subshells and remove unneeded files [rafaelfranca]

README.md

IMPORTANT: Devise 2.0.0 is out. If you are upgrading, please read: https://github.com/plataformatec/devise/wiki/How-To:-Upgrade-to-Devise-2.0

Devise

INFO: This README is also available in a friendly navigable format.

Devise is a flexible authentication solution for Rails based on Warden. It:

- Is Rack based;
- Is a complete MVC solution based on Rails engines;
- Allows you to have multiple roles (or models/scopes) signed in at the same time;
- Is based on a modularity concept: use just what you really need.

It's comprised of 12 modules:

- Database Authenticatable: encrypts and stores a password in the database to validate the authenticity of a user while signing in. The authentication can be done both through POST requests or HTTP Basic Authentication.
- Token Authenticatable: signs in a user based on an authentication token (also known as "single access token"). The token can be given both through query string or HTTP Basic Authentication.
- Omniauthable: adds Omniauth (github.com/intridea/omniauth) support;
- Confirmable: sends emails with confirmation instructions and verifies whether an account is already confirmed during sign in.
- Recoverable: resets the user password and sends reset instructions.
- Registerable: handles signing up users through a registration process, also allowing them to edit and destroy their account.
- Rememberable: manages generating and clearing a token for remembering the user from a saved cookie.
- Trackable: tracks sign in count, timestamps and IP address.
- Timeoutable: expires sessions that have no activity in a specified period of time.
- Validatable: provides validations of email and password. It's optional and can be customized, so you're able to define your own validations.
- Lockable: locks an account after a specified number of failed sign-in attempts. Can unlock via email or after a specified time period.
- Encryptable: adds support of other authentication mechanisms besides the built-in Bcrypt (the default).

Information

The Devise wiki

The Devise Wiki has lots of additional information about Devise including many "how-to" articles and answers to the most frequently asked questions. Please browse the Wiki after finishing this README:

https://wiki.github.com/plataformatec/devise

Bug reports

If you discover a problem with Devise, we would like to know about it. However, we ask that you please review these guidelines before submitting a bug report:

https://github.com/plataformatec/devise/wiki/Bug-reports

If you found a security bug, do *NOT* use the GitHub issue tracker. Send email or a private GitHub message to the maintainers listed at the bottom of the README.

Mailing list

If you have any questions, comments, or concerns, please use the Google Group instead of the GitHub issue tracker:

https://groups.google.com/group/plataformatec-devise

RDocs

You can view the Devise documentation in RDoc format here:

http://rubydoc.info/github/plataformatec/devise/master/frames

If you need to use Devise with Rails 2.3, you can always run "gem server" from the command line after you install the gem to access the old documentation.

Example applications

There are a few example applications available on GitHub that demonstrate various features of Devise with different versions of Rails. You can view them here:

https://github.com/plataformatec/devise/wiki/Example-Applications

Extensions

Our community has created a number of extensions that add functionality above and beyond what is included with Devise. You can view a list of available extensions and add your own here:

https://github.com/plataformatec/devise/wiki/Extensions

Contributing

We hope that you will consider contributing to Devise. Please read this short overview for some information about how to get started:

https://github.com/plataformatec/devise/wiki/Contributing

You will usually want to write tests for your changes. To run the test suite, go into Devise's top-level directory and run "bundle install" and "rake". For the tests to pass, you will need to have a MongoDB server (version 2.0 or newer) running on your system.

Starting with Rails?

If you are building your first Rails application, we recommend you to *not* use Devise. Devise requires a good understanding of the Rails Framework. In such cases, we advise you to start a simple authentication system from scratch, today we have two resources:

- Michael Hartl's online book: http://railstutorial.org/chapters/modeling-and-viewing-users-two#top
- Ryan Bates' Railscast: http://railscasts.com/episodes/250-authentication-from-scratch

Once you have solidified your understanding of Rails and authentication mechanisms, we assure you Devise will be very pleasant to work with. :)

Getting started

Devise 2.0 works with Rails 3.1 onwards. You can install it with:

```
gem install devise
```

After you install Devise and add it to your Gemfile, you need to run the generator:

```
rails generate devise:install
```

The generator will install an initializer which describes ALL Devise's configuration options and you MUST take a look at it. When you are done, you are ready to add Devise to any of your models using the generator:

```
rails generate devise MODEL
```

Replace MODEL by the class name used for the applications users, it's frequently 'User' but could also be 'Admin'. This will create a model (if one does not exist) and configure it with default Devise modules. Next, you'll usually run "rake db:migrate" as the generator will have created a migration file (if your ORM supports them). This generator also configures your config/routes.rb file to point to Devise controller.

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_filter:

```
before_filter :authenticate_user!
```

To verify if a user is signed in, use the following helper:

```
user_signed_in?
```

For the current signed-in user, this helper is available:

```
current_user
```

You can access the session for this scope:

```
user_session
```

After signing in a user, confirming the account or updating the password, Devise will look for a scoped root path to redirect. Example: For a :user resource, it will use +user_root_path+ if it exists, otherwise default +root_path+ will be used. This means that you need to set the root inside your routes:

```
root :to => "home#index"
```

You can also overwrite +after_sign_in_path_for+ and +after_sign_out_path_for+ to customize your redirect hooks.

Finally, you need to set up default url options for the mailer in each environment. Here is the configuration for "config/environments/development.rb":

```
config.action_mailer.default_url_options = { :host => 'localhost:3000' }
```

Notice that if your devise model is not called "user" but "member", then the helpers you should use are:

```
before_filter :authenticate_member!

member_signed_in?

current_member

member_session
```

Configuring Models

The devise method in your models also accepts some options to configure its modules. For example, you can choose the cost of the encryption algorithm with:

```
devise :database_authenticatable, :registerable, :confirmable, :recoverable, :stretches => 20
```

Besides :stretches, you can define :pepper, :encryptor, :confirm_within, :remember_for, :timeout_in, :unlock_in and other values. For details, see the initializer file that was created when you invoked the "devise:install" generator described above.

Configuring multiple models

Devise allows you to set up as many roles as you want. For example, you may have a User model and also want an Admin model with just authentication and timeoutable features. If so, just follow these steps:

```
# Create a migration with the required fields
create_table :admins do |t|
  t.string :email
  t.string :encrypted_password
  t.timestamps
end

# Inside your Admin model
devise :database_authenticatable, :timeoutable

# Inside your routes
devise_for :admins

# Inside your protected controller
before_filter :authenticate_admin!

# Inside your controllers and views
admin_signed_in?
current_admin
admin_session
```

On the other hand, you can simply run the generator!

Configuring views

We built Devise to help you quickly develop an application that uses authentication. However, we don't want to be in your way when you need to customize it.

Since Devise is an engine, all its views are packaged inside the gem. These views will help you get started, but after some time you may want to change them. If this is the case, you just need to invoke the following generator, and it will copy all views to your application:

```
rails generate devise:views
```

If you have more than one role in your application (such as "User" and "Admin"), you will notice that Devise uses the same views for all roles. Fortunately, Devise offers an easy way to customize views. All you need to do is set "config.scoped_views = true" inside "config/initializers/devise.rb".

After doing so, you will be able to have views based on the role like "users/sessions/new" and "admins/sessions/new". If no view is found within the scope, Devise will use the default view at "devise/sessions/new". You can also use the generator to generate scoped views:

```
rails generate devise:views users
```

Configuring controllers

If the customization at the views level is not enough, you can customize each controller by following these steps:

1) Create your custom controller, for example a Admins::SessionsController:

```
class Admins::SessionsController < Devise::SessionsController
end
```

2) Tell the router to use this controller:

```
devise_for :admins, :controllers => { :sessions => "admins/sessions" }
```

3) And since we changed the controller, it won't use the "devise/sessions" views, so remember to copy "devise/sessions" to "admin/sessions".

Remember that Devise uses flash messages to let users know if sign in was successful or failed. Devise expects your application to call "flash[:notice]" and "flash[:alert]" as appropriate.

Configuring routes

Devise also ships with default routes. If you need to customize them, you should probably be able to do it through the devise_for method. It accepts several options like :class_name, :path_prefix and so on, including the possibility to change path names for I18n:

```
devise_for :users, :path => "usuarios", :path_names => { :sign_in => 'login', :sign_out => 'logout', :password => 'secret', :confirmation =
```

Be sure to check +devise_for+ documentation for details.

If you have the need for more deep customization, for instance to also allow "/sign_in" besides "/users/sign_in", all you need to do is to create your routes normally and wrap them in a +devise_scope+ block in the router:

```
devise_scope :user do
  get "sign_in", :to => "devise/sessions#new"
end
```

This way you tell devise to use the scope :user when "/sign_in" is accessed. Notice +devise_scope+ is also aliased as +as+ and you can also give a block to +devise_for+, resulting in the same behavior:

```
devise_for :users do
  get "sign_in", :to => "devise/sessions#new"
end
```

Feel free to choose the one you prefer!

I18n

Devise uses flash messages with I18n with the flash keys :notice and :alert. To customize your app, you can set up your locale file:

```
en:
  devise:
    sessions:
      signed_in: 'Signed in successfully.'
```

You can also create distinct messages based on the resource you've configured using the singular name given in routes:

```
en:
  devise:
    sessions:
      user:
        signed_in: 'Welcome user, you are signed in.'
      admin:
        signed_in: 'Hello admin!'
```

The Devise mailer uses a similar pattern to create subject messages:

```
en:
  devise:
    mailer:
      confirmation_instructions:
        subject: 'Hello everybody!'
        user_subject: 'Hello User! Please confirm your email'
      reset_password_instructions:
        subject: 'Reset instructions'
```

Take a look at our locale file to check all available messages. You may also be interested in one of the many translations that are available on our wiki:

https://github.com/plataformatec/devise/wiki/I18n

Test helpers

Devise includes some tests helpers for functional specs. To use them, you just need to include Devise::TestHelpers in your test class and use the sign_in and sign_out method. Such methods have the same signature as in controllers:

```
sign_in :user, @user   # sign_in(scope, resource)
sign_in @user          # sign_in(resource)

sign_out :user         # sign_out(scope)
sign_out @user         # sign_out(resource)
```

You can include the Devise Test Helpers in all of your tests by adding the following to the bottom of your test/test_helper.rb file:

```
class ActionController::TestCase
  include Devise::TestHelpers
end
```

If you're using RSpec and want the helpers automatically included within all +describe+ blocks, add a file called spec/support/devise.rb with the following contents:

```
RSpec.configure do |config|
  config.include Devise::TestHelpers, :type => :controller
end
```

Do not use such helpers for integration tests such as Cucumber or Webrat. Instead, fill in the form or explicitly set the user in session. For more tips, check the wiki (https://wiki.github.com/plataformatec/devise).

Omniauth

Devise comes with Omniauth support out of the box to authenticate from other providers. You can read more about Omniauth support in the wiki:

- https://github.com/plataformatec/devise/wiki/OmniAuth:-Overview

Other ORMs

Devise supports ActiveRecord (default) and Mongoid. To choose other ORM, you just need to require it in the initializer file.

Migrating from other solutions

Devise implements encryption strategies for Clearance, Authlogic and Restful-Authentication. To make use of these strategies, you need set the desired encryptor in the encryptor initializer config option and add :encryptable to your model. You might also need to rename your encrypted password and salt columns to match Devise's fields (encrypted_password and password_salt).

Troubleshooting

Heroku

Using devise on Heroku with Ruby on Rails 3.1 requires setting:

```
config.assets.initialize_on_precompile = false
```

Read more about the potential issues at http://guides.rubyonrails.org/asset_pipeline.html

Additional information

Warden

Devise is based on Warden, which is a general Rack authentication framework created by Daniel Neighman. We encourage you to read more about Warden here:

https://github.com/hassox/warden

Contributors

We have a long list of valued contributors. Check them all at:

https://github.com/plataformatec/devise/contributors

Maintainers

- José Valim (https://github.com/josevalim)
- Carlos Antônio da Silva (https://github.com/carlosantoniodasilva)
- Rodrigo Flores (https://github.com/rodrigoflores)

License

MIT License. Copyright 2012 Plataforma Tecnologia. http://blog.plataformatec.com.br

Markdown Cheat Sheet

Format Text

Headers

```
# This is an <h1> tag
## This is an <h2> tag
###### This is an <h6> tag
```

Text styles

```
*This text will be italic*
_This will also be italic_
**This text will be bold**
__This will also be bold__

*You **can** combine them*
```

Lists

Unordered

```
* Item 1
* Item 2
  * Item 2a
  * Item 2b
```

Ordered

```
1. Item 1
2. Item 2
3. Item 3
   * Item 3a
   * Item 3b
```

Miscellaneous

Images

```
![GitHub Logo](/images/logo.png)
Format: ![Alt Text](url)
```

Links

```
http://github.com - automatic!
[GitHub](http://github.com)
```

Blockquotes

```
As Kanye West said:

> We're living the future so
> the present is our past.
```

Code Examples in Markdown

Syntax highlighting with GFM

````
```javascript
function fancyAlert(arg) {
  if(arg) {
    $.facebox({div:'#foo'})
  }
}
```
````

Or, indent your code 4 spaces

```
Here is a Python code example
without syntax highlighting:

    def foo:
      if not bar:
        return true
```

Inline code for comments

```
I think you should use an
`<addr>` element here instead.
```

Something went wrong with that request. Please try again. Dismiss