

# Devise for Rails Guide

Devise is a rails gem that allows you to create a complete user management system. Devise takes care of everything for us, from allowing users to sign up, sign in, sign out, and edit their profiles.

1. To start using Devise, you first need to make sure it's included in your Gemfile.

```
# User authentication system, handling logins and signups
gem 'devise'
```

2. After adding it to your Gemfile, you need to run the bundle install command to install the Devise gem.

3. Next, you need to run the generator:

```
[student1@csc415-server38 src]$ rails generate devise:install
```

4. After running the generator command, devise will give you a list of instructions to follow:

```
Depending on your application's configuration some manual setup may be required:

1. Ensure you have defined default url options in your environments files. Here
   is an example of default_url_options appropriate for a development environment
   in config/environments/development.rb:

   config.action_mailer.default_url_options = { host: 'localhost', port: 3000 }

   In production, :host should be set to the actual host of your application.

   * Required for all applications. *

2. Ensure you have defined root_url to *something* in your config/routes.rb.
   For example:

   root to: "home#index"

   * Not required for API-only Applications *

3. Ensure you have flash messages in app/views/layouts/application.html.erb.
   For example:

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

   * Not required for API-only Applications *

4. You can copy Devise views (for customization) to your app by running:

   rails g devise:views

   * Not required *
```

5. For the first instruction, navigate to your `config/environments/development.rb` file and add the following line (if it is not already there):

```
# Devise host settings
config.action_mailer.default_url_options = { host: 'localhost', port: 3000 }
```

6. For the second instruction, navigate to your `config/routes.rb` file and double check that it has the following line: `root 'home#index'`. This line should have already been added to your `routes.rb` file, but it is a good idea to double check that it's there.

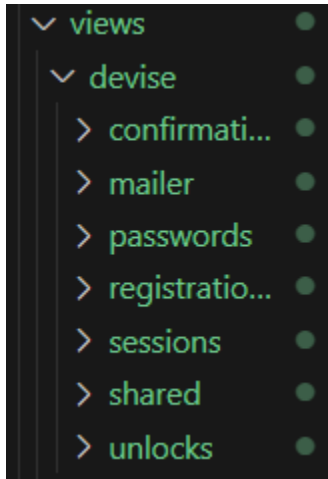
7. For the third instruction, navigate to your `app/views/layouts/application.html.erb` file to ensure you have flash messages. If you don't already have flash messages, you can add the following code:

```
<div class="container">
  <% if notice %>
    <p class="alert alert-success"><%= notice %></p>
  <% end %>
  <% if alert %>
    <p class="alert alert-danger"><%= alert %></p>
  <% end %>
  <%= yield %>
</div>
```

8. For the final instruction, we need to generate the views used by Devise. We can do so by running the following command:

```
=====
[student1@csc415-server38 src]$ rails g devise:views|
```

9. After running this command, you should now have a new folder called “devise” in your views folder:



**THE FOLLOWING TWO COMMANDS CAN BE IGNORED IF YOUR REPOSITORY IS FORKED FROM THE RAILS STARTER KIT (INCLUDING ARMINARM)**

10. Now that we’ve generated the views for Devise, we need to set up a database model called “user” for Devise to store all of the user information. We can do this by running the following command:

```
[student1@csc415-server38 src]$ rails generate devise user
```

11. After running this command, run the database migrate command:

```
[student1@csc415-server38 src]$ rails db:migrate
```

Now that Devise is successfully set up, you can start using the user authentication system by linking buttons to the various Devise views using routes. To see each of the routes for the Devise views, run the command “rails routes”.

```
[student1@csc415-server38 src]$ rails routes
               Prefix Verb   URI Pattern
rails_admin/
rails_admin root GET  /
rails_admin/new_user_session GET  /users/sign_in(.:format)
rails_admin/user_session POST /users/sign_in(.:format)
rails_admin/destroy_user_session DELETE /users/sign_out(.:format)
rails_admin/new_user_password GET  /users/password/new(.:format)
rails_admin/edit_user_password GET  /users/password/edit(.:format)
rails_admin/user_password PATCH /users/password(.:format)
rails_admin/PUT /users/password(.:format)
rails_admin/POST /users/password(.:format)
rails_admin/cancel_user_registration GET  /users/cancel(.:format)
rails_admin/new_user_registration GET  /users/sign_up(.:format)
rails_admin/edit_user_registration GET  /users/edit(.:format)
rails_admin/user_registration PATCH /users(.:format)
rails_admin/PUT /users(.:format)
rails_admin/DELETE /users(.:format)
rails_admin/POST /users(.:format)
```

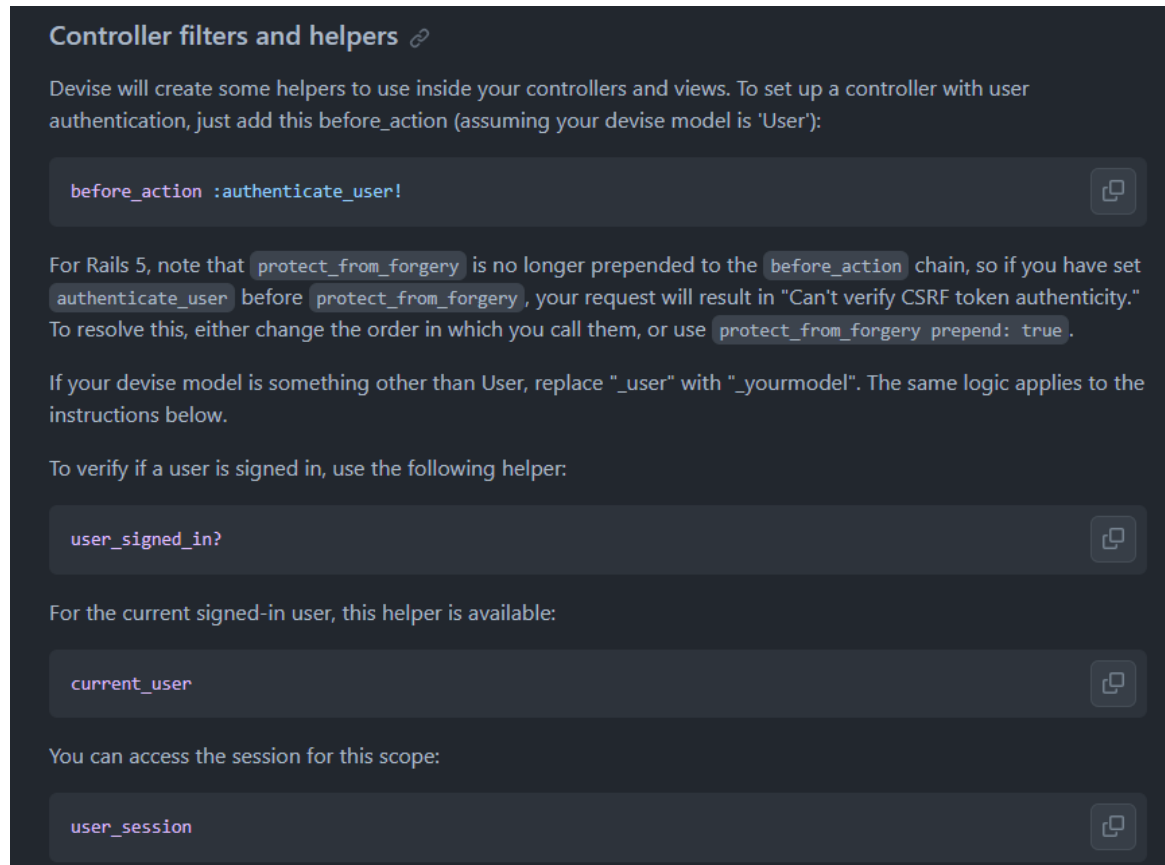
For example, to set up a “Login” button that links to the Devise login page, we can go to our view and add the following code:

```
<li class="nav-item">
  <%= button_to "Login", new_user_session_path, class: "btn btn-outline-primary" %>
</li>
<% end %>
```

This code creates a button that uses the “new\_user\_session\_path” which links to the /users/sign\_in page (as seen in the routes above).

This method for creating a login button can be applied to each of the Devise views. If you want to customize any of the Devise views, you can do so by opening them in the `app/views/devise` directory and making your changes.

Some helper functions that are useful when using Devise are as follows:



The screenshot shows a section of the Devise documentation titled "Controller filters and helpers" with a link icon. The text explains that Devise creates helpers for use in controllers and views. It provides a code example for a controller filter: `before_action :authenticate_user!`. It also notes that in Rails 5, `protect_from_forgery` is no longer prepended to the `before_action` chain, and provides a solution: `protect_from_forgery prepend: true`. Further, it explains how to replace `_user` with `_yourmodel` if the devise model is not 'User'. Below this, it lists three helpers: `user_signed_in?`, `current_user`, and `user_session`, each with a copy icon.

**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!
```

For Rails 5, note that `protect_from_forgery` is no longer prepended to the `before_action` chain, so if you have set `authenticate_user` before `protect_from_forgery`, your request will result in "Can't verify CSRF token authenticity." To resolve this, either change the order in which you call them, or use `protect_from_forgery prepend: true`.

If your devise model is something other than User, replace `"_user"` with `"_yourmodel"`. The same logic applies to the instructions below.

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
```

These helper functions are useful when you want to set up conditions for your login buttons. For example, when a user is signed in, you don't want the user to see the options "sign up" or "sign in". To do so, you can use an if statement and the helper "user\_signed\_in?" to determine whether the current user is already signed in.

The "before\_action :authenticate\_user!" controller filter is especially useful for hiding certain aspects of your application from users who are not currently signed in.

The "current\_user" and "user\_session" helpers are useful for displaying information about the current user and current session.