Rails-Foundation

Last updated 16 Apr 2014

GitHub Repository (https://github.com/RailsApps/rails-foundation) · Issues (https://github.com/RailsApps/rails-foundation/issues)

Foundation Quickstart Guide

Introduction

This guide shows how to integrate Foundation (http://foundation.zurb.com/) with Rails. Versions:

- Foundation 5.2
- Rails 4.1

A complete rails-foundation (https://github.com/RailsApps/rails-foundation) example application is available from the RailsApps project.

You will need:

- The Ruby language – version 2.1
- The Rails gem – version 4.1

See the article Installing Rails (http://railsapps.github.io/installing-rails.html) for instructions about setting up Rails and your development environment.

Is It for You?

This guide is for experienced Rails developers who want to integrate Foundation with Rails. It shows the quickest path to set up Foundation with Rails. If you are a beginner, start with:

- Learn Ruby on Rails (https://tutorials.railsapps.org/tutorials/learn-rails)

What's Here

You'll find basics here:

- installation of Foundation
- the application layout
- flash messages
- navigation links

In summary, here are the steps for adding Foundation to a Rails application:

- add a gem to the **Gemfile**
- modify the file **app/assets/javascripts/application.js** to add Foundation's Javascript files
- add the file **app/assets/stylesheets/framework_and_overrides.css.scss** to add Foundation's CSS files
- modify the file **app/views/layouts/application.html.erb** to change the application layout

This guide uses the rails_layout (https://github.com/RailsApps/rails_layout) gem to set up Foundation and the necessary files.

Rails Composer

You can use the Rails Composer (http://railsapps.github.io/rails-composer/) tool to build a Foundation-based starter app in minutes.

To build the example application, Rails 4.1 must be installed in your development environment. Run the command:

```
$ rails new rails-foundation -m https://raw.github.com/RailsApps/rails-composer/master/compos
er.rb
```

Follow the prompts. The README from the rails-foundation (https://github.com/RailsApps/rails-foundation) example application has more information about the options.

If you'd like to add Foundation to an existing application, or learn how to integrate Foundation with Rails, read the rest of this guide.

Gems

The foundation-rails (https://github.com/zurb/foundation-rails) gem adds Foundation CSS and JavaScript files to the Rails asset pipeline (http://guides.rubyonrails.org/asset_pipeline.html). The foundation-rails gem is the official gem maintained by Zurb.

We'll add the rails_layout (https://github.com/RailsApps/rails_layout) gem to generate files for an application layout, navigation links, and flash messages styled with Foundation CSS classes and layout.

In your **Gemfile**, add:

```
gem 'foundation-rails'
group :development do
  gem 'rails_layout'
end
```

You don't need the rails_layout (https://github.com/RailsApps/rails_layout) gem deployed to production, so put it in the `development` group.

Run `$ bundle install` in the Terminal.

Rails Layout

A generator provided by the rails_layout (https://github.com/RailsApps/rails_layout) gem will set up Foundation and add the necessary files. Run:

```
$ rails generate layout:install foundation5
```

You can add the `--force` argument to replace existing files.

The rails_layout generator will rename the file:

- **app/assets/stylesheets/application.css**

to:

- **app/assets/stylesheets/application.css.scss**

This will allow you to use Sass syntax in your application stylesheet. Stylesheets can use variables, mixins, and nesting of CSS rules when you use Sass.

The rails_layout generator will create the file:

- **app/assets/stylesheets/framework_and_overrides.css.scss**

and modify the file:

- **app/assets/javascripts/application.js**

The gem will create or replace four files:

- **app/views/layouts/application.html.erb**
- **app/views/layouts/_messages.html.erb**

- **app/views/layouts/_navigation.html.erb**
- **app/views/layouts/_navigation_links.html.erb**

We'll examine each of these files and explain their purpose.

config/application.rb

The rails_layout gem will make a necessary change to the **config/application.rb** file. It will add:

```
module RailsFoundation
  class Application < Rails::Application
    .
    .
    .
    # For Foundation 5
    config.assets.precompile += %w( vendor/modernizr )

  end
end
```

Later, in the application layout file, you'll see Modernizer included with a `javascript_include_tag`. All assets are compiled automatically in development but in production, any assets not included in the **assets/** folder must be specified with the `config.assets.precompile` configuration setting.

Without this change, you'll get an error:

```
Sprockets::Rails::Helper::AssetFilteredError at /
Asset filtered out and will not be served: add `config.assets.precompile += %w( vendor/modern
izr )` to `config/application.rb` and restart your server
```

This is a sanity check to reveal asset pipeline errors that you would see in production (details (https://github.com/rails/sprockets-rails/pull/84)).

Application CSS

The Rails asset pipeline will concatenate and compact CSS stylesheets for delivery to the browser when you add them to this directory:

- **app/assets/stylesheets/**

The asset pipeline helps web pages display faster in the browser by combining all CSS files into a single file (it does the same for JavaScript).

Let's examine the file **app/assets/stylesheets/application.css.scss**:

```
/*
 * This is a manifest file that'll be compiled into application.css, which will include all t
he files
 * listed below.
 *
 * Any CSS and SCSS file within this directory, lib/assets/stylesheets, vendor/assets/stylesh
eets,
 * or vendor/assets/stylesheets of plugins, if any, can be referenced here using a relative p
ath.
 *
 * You're free to add application-wide styles to this file and they'll appear at the bottom o
f the
 * compiled file so the styles you add here take precedence over styles defined in any styles
 * defined in the other CSS/SCSS files in this directory. It is generally better to create a
new
 * file per style scope.
 *
 *= require_tree .
 *= require_self
 */
```

The **app/assets/stylesheets/application.css.scss** file serves two purposes.

First, you can add any CSS rules to the file that you want to use anywhere on your website. Second, the file serves as a *manifest*, providing a list of files that should be concatenated and included in the single CSS file that is delivered to the browser.

A Global CSS File

Any CSS style rules that you add to the **app/assets/stylesheets/application.css.scss** file will be available to any view in the application. You could use this file for any style rules that are used on every page, particularly simple utility rules such as highlighting or resetting the appearance of links. However, in practice, you are more likely to modify the style rules provided by Foundation. These modifications don't belong in the **app/assets/stylesheets/application.css.scss** file; they will go in the **app/assets/stylesheets/framework_and_overrides.css.scss** file.

In general, it's bad practice to place a lot of CSS in the **app/assets/stylesheets/application.css.scss** file (unless your CSS is very limited). Instead, structure your CSS in multiple files. CSS that is used on only a single page can go in a file with a name that matches the page. Or, if sections of the website share common elements, such as themes for landing pages or administrative pages, make a file for each theme. How you organize your CSS is up to you; the asset pipeline lets you organize your CSS so it is easier to develop and maintain. Just add the files to the **app/assets/stylesheets/** folder.

A Manifest File

It's not obvious from the name of the **app/assets/stylesheets/application.css.scss** file that it serves as a *manifest file* as well as a location for miscellaneous CSS rules. For most websites, you can ignore its role as a manifest file. In the comments at the top of the file, the `*= require_self` directive indicates that any CSS in the file should be delivered to the browser. The `*= require_tree .` directive (note the Unix "dot operator") indicates any files in the same folder, including files in subfolders, should be combined into a single file for delivery to the browser.

If your website is large and complex, you can remove the `*= require_tree .` directive and specify individual files to be included in the file that is generated by the asset pipeline. This gives you the option of reducing the size of the application-wide CSS file that is delivered to the browser. For example, you might segregate a file that includes CSS that is used only in the site's administrative section. In general, only large and complex sites need this optimization. The speed of rendering a single large CSS file is faster than fetching multiple files.

Foundation JavaScript

Foundation provides both CSS and JavaScript libraries.

Like the **application.css.scss** file, the **application.js** file is a manifest that allows a developer to designate the JavaScript files that will be combined for delivery to the browser.

The rails_layout gem modified the file **app/assets/javascripts/application.js** to include the Foundation JavaScript libraries:

```
//= require jquery
//= require jquery_ujs
//= require turbolinks
//= require foundation
//= require_tree .
$(function() {
  $(document).foundation();
});
```

It added the directive `//= require foundation` before `//= require_tree .`.

The last three lines use jQuery to load the Foundation JavaScript libraries after the browser has fired a "DOM ready" event (which means the page is fully rendered and not waiting for additional files to download).

```
$(function() {
  $(document).foundation();
});
```

Foundation CSS

The rails_layout gem added a file **app/assets/stylesheets/framework_and_overrides.css.scss** containing:

```
// import the CSS framework
@import "foundation";
.
.
.
```

The file **app/assets/stylesheets/framework_and_overrides.css.scss** is automatically included and compiled into your Rails application.css file by the `*= require_tree .` statement in the **app/assets/stylesheets/application.css.scss** file.

The `@import "foundation";` directive will import the Foundation CSS rules from the Foundation gem.

You could add the Foundation `@import` code to the **app/assets/stylesheets/application.css.scss** file. However, it is better to have a separate **app/assets/stylesheets/framework_and_overrides.css.scss** file. You may wish to modify the Foundation CSS rules; placing changes to Foundation CSS rules in the **framework_and_overrides.css.scss** file will keep your CSS better organized.

Overriding Foundation Classes

The file **app/assets/stylesheets/framework_and_overrides.css.scss** shows how to customize Foundation classes.

```
// override for the 'Home' navigation link
.top-bar .name {
  font-size: rem-calc(13);
  line-height: 45px; }
.top-bar .name a {
  font-weight: normal;
  color: white;
  padding: 0 15px; }
```

The example style rules adjust the “Home” navigation link to match the other links in the navigation bar.

The Sass function `rem-calc()` provided by Foundation converts pixels to “root ems” so you can specify font size in pixels but give the browser the font size in scalable ems (rems are explained here (http://snook.ca/archives/html_and_css/font-size-with-rem)).

Using Sass Mixins with Foundation

In addition to the simple `@import "foundation";` directive, the **app/assets/stylesheets/framework_and_overrides.css.scss** contains a collection of Sass mixins. These are examples that you can remove.

You can use Sass mixins to map Foundation class names to your own semantic class names. The rails_layout gem provides examples of Sass mixins that apply CSS style rules to the default application layout. In doing so, the default application layout is free of framework-specific code and can be used with Bootstrap, Foundation, or other front-end frameworks.

Application Layout

Rails uses the layout defined in the file **app/views/layouts/application.html.erb** as a default for rendering any page.

Let's look at the application layout file created by the rails_layout gem:

Examine the contents of the file **app/views/layouts/application.html.erb**:

```
<!DOCTYPE html>
<html>
  <head>
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title><%= content_for?(:title) ? yield(:title) : "Rails Foundation" %></title>
    <meta name="description" content="<%= content_for?(:description) ? yield(:description) :
"Rails Foundation" %>">
    <%= stylesheet_link_tag 'application', media: 'all', 'data-turbolinks-track' => true %>
    <%# Modernizr is required for Zurb Foundation %>
    <%= javascript_include_tag 'vendor/modernizr' %>
    <%= javascript_include_tag 'application', 'data-turbolinks-track' => true %>
    <%= csrf_meta_tags %>
  </head>
  <body>
    <header>
      <%= render 'layouts/navigation' %>
    </header>
    <main role="main">
       <%= render 'layouts/messages' %>
       <%= yield %>
    </main>
  </body>
</html>
```

The Sass mixins in the **app/assets/stylesheets/framework_and_overrides.css.scss** file will apply Foundation classes to the HTML element `main`. You can add Foundation classes directly to the application layout instead of using Sass mixins.

Modernizr JavaScript Library

The application layout file includes:

```
.
.
.
<%# Modernizr is required for Zurb Foundation %>
<%= javascript_include_tag "vendor/modernizr" %>
.
.
.
```

The Modernizr (http://modernizr.com/) JavaScript library is a prerequisite for Foundation. Modernizr makes it possible for older browsers to use HTML5 elements. It also detects mobile devices. It must be loaded before Foundation, so it is included above `javascript_include_tag "application"`.

Foundation Grid

You can add Foundation classes directly to the application layout.

You can organize your layout in horizontal sections using `row` classes. A row can contain a single column or you can split it into multiple columns. A row contains a maximum of 12 columns.

The width of columns automatically adjusts to the width of the browser window. Here's a table that shows the width of the page on different devices:

	Phones	**Tablets**	**Desktops**
Browser width	0 – 640px	641 – 1023px	1024px and up
Class prefix	.small-	.medium-	.large-

You can use Foundation for a "mobile-first" design. That means you design a layout that looks good on small devices, using the `small-` column sizes. For larger screens, you can add an alternative layout using the `medium` or `large` grid classes. For simple designs, just use `small-` and it will look fine on larger screens.

Here's how to add Foundation classes so all your pages display within a single full-width column:

```
<main role="main" class="row">
  <div class="small-12 columns">
    <%= render 'layouts/messages' %>
    <%= yield %>
  </div>
</main>
```

Here's a footer presented as a row with two sections that adapt to small browser widths:

```
<footer class="row">
  <section class="small-12 medium-4 columns">
    Copyright 2014
  </section>
  <section class="small-12 medium-8 columns">
    All rights reserved.
  </section>
</footer>
```

The Foundation `row` class will create a horizontal break. On desktops and tablets, the footer will contain two side-by-side sections. The first section will be four columns wide; the second section will be eight columns wide. On phones, each section will expand to take the full browser width, appearing as stacked rows.

To see the grid in action, watch what happens when you resize the browser on a desktop computer. For any width narrower than 640px, the sections will display as stacked rows.

See the documentation for the Foundation Grid (http://foundation.zurb.com/docs/components/grid.html) to learn about the Foundation grid classes.

Flash Messages

Rails provides a standard convention to display alerts (including error messages) and other notices (including success messages), called a *flash message*.

You can include code to display flash messages directly in your application layout file or you can create a partial template (http://guides.rubyonrails.org/layouts_and_rendering.html#using-partials) – a "partial" – to better organize the default application layout.

The application layout file includes a messages partial:

```
<%= render 'layouts/messages' %>
```

Examine the file **app/views/layouts/_messages.html.erb**:

```
<%# Rails flash messages styled for Zurb Foundation %>
<% flash.each do |name, msg| %>
  <% if msg.is_a?(String) %>
    <div data-alert class="alert-box round <%= name.to_s == 'notice' ? 'success' : 'alert' %>
">
      <%= content_tag :div, msg %>
      <a href="#" class="close">&amp;times;</a>
    </div>
  <% end %>
<% end %>
```

Rails uses `:notice` and `:alert` as flash message keys. Foundation provides a base class `alert-box` with additional classes `success` and `alert`. A bit of parsing is required to get a Rails "notice" message to be styled with the Foundation `success` style. Any other message, including a Rails "alert" message, will be styled with the Foundation `alert` style.

We use `each` to iterate through the flash hash, retrieving a `name` and `msg` that are passed to a block to be output as a string. The expression `if msg.is_a?(String)` serves as a test to make sure we only display messages that are strings. We construct a div that applies Foundation CSS styling around the message. Foundation recognizes a class `alert-box` and `round` (for rounded corners). A class of either `success` or `alert` styles the message. Rails `notice` messages will get styled with the Foundation `success` class. Any other Rails messages, including `alert` messages, will get styled with the Foundation `alert` class.

We use the Rails `content_tag` view helper to create a div containing the message.

Finally, we create a "close" icon by applying the class `close` to a link. We use the HTML entity `&times;` (a big "X" character) for the link; it could be the word "close" or anything else we like. Foundation's integrated JavaScript library will hide the alert box when the "close" link is clicked.

Foundation provides detailed documentation (http://foundation.zurb.com/docs/components/alert-boxes.html) if you want to change the styling of the alert boxes.

Beyond Red and Green

For simplicity, it's wise to stick with the Rails convention of using only "alert" and "notice." However, if you wish, you can accommodate an additional "info" class.

By default, Foundation applies a green background to `success` and a red background to `alert`. Foundation provides additional classes `info` (blue) and `warning` (amber). With a little hacking, it's possible to accommodate a Rails flash message with a third name, such as `:info`. Here's an example.

You can replace `<%= name.to_s == 'notice' ? 'success' : 'alert' %>">` with this:

```
<%= name.to_s == 'notice' ? 'success' : (name.to_s == 'info' ? 'info' : 'alert' )%>
```

This will style any message with the red `alert` class unless it is "info" (blue) or "success" (green).

Navigation Bar

You'll likely need navigation links on every page of your web application.

The layout and styling required for the Foundation navigation bar are in the navigation partial file.

The application layout file includes a navigation partial:

```
<%= render 'layouts/navigation' %>
```

Examine the file **app/views/layouts/_navigation.html.erb**:

```
<%# navigation styled for Zurb Foundation 5 %>
<nav class="top-bar" data-topbar>
  <ul class="title-area">
    <li class="name"><%= link_to 'Home', root_path %></li>
    <li class="toggle-topbar menu-icon"><a href="#">Menu</a></li>
  </ul>
  <div class="top-bar-section">
    <ul>
      <%= render 'layouts/navigation_links' %>
    </ul>
  </div>
</nav>
```

The navigation partial includes layout and Foundation classes needed to produce a responsive navigation bar.

The responsive navigation bar adjusts to different browser widths. At small sizes, the navigation links will disappear and be replaced by an icon labeled "Menu." Clicking the icon will reveal a vertical menu of navigation links. The navigation menu is a great demonstration of the ability of Foundation to adjust to the small screen size of a smartphone.

If you'd like to add a site name or logo, you can replace the link helper `<%= link_to 'Home', root_path %>`. It is important to preserve the enclosing layout and classes, even if you don't want to display a site name or logo. The enclosing layout is used to generate the navigation menu when the browser window shrinks to accommodate a smartphone.

We wrap the nested partial `render 'layouts/navigation_links'` with Foundation layout and classes to complete the navigation bar.

Navigation Links Partial

The file **app/views/layouts/_navigation_links.html.erb** is very simple:

```
<%# add navigation links to this file %>
```

You can add links to this file, for example:

```
<%# add navigation links to this file %>
<li><%= link_to 'About', page_path('about') %></li>
<li><%= link_to 'Contact', new_contact_path %></li>
```

The navigation links partial is simply a list of navigation links. It doesn't require additional CSS styling. By separating the links from the styling that creates the navigation bar, we segregate the code that is unique to Foundation. In the future, if the Foundation layout or CSS classes change,

we can make changes without touching the navigation links.

Troubleshooting

The behavior of the navigation bar will show you if Foundation JavaScript is working correctly.

When you test your application, reduce the browser window to a narrow width. The navigation links should be replaced by an icon. If clicking the menu icon doesn't reveal a drop-down menu, the application may not be loading the Foundation JavaScript library. For more troubleshooting, open the browser JavaScript console and look for error messages.

Form Helpers

Rails provides a set of view helpers for forms. They are described in the RailsGuides: Rails Form Helpers (http://guides.rubyonrails.org/form_helpers.html) document. Many developers use an alternative set of form helpers named SimpleForm, provided by the SimpleForm gem (https://github.com/plataformatec/simple_form). The SimpleForm helpers are more powerful, easier to use, and offer an option for styling with Foundation.

The SimpleForm README (https://github.com/plataformatec/simple_form) indicates that SimpleForm provides form helpers that are compatible with Zurb Foundation 3. In fact, the form helpers are also compatible with Zurb Foundation 5.

In your **Gemfile**, add:

```
gem 'simple_form'
```

Run `$ bundle install .`

Run the generator to install SimpleForm with a Foundation option:

```
$ rails generate simple_form:install --foundation
```

which installs several configuration files:

```
config/initializers/simple_form.rb
config/initializers/simple_form_foundation.rb
config/locales/simple_form.en.yml
lib/templates/erb/scaffold/_form.html.erb
```

Here the SimpleForm gem uses the `rails generate` command to create files for initialization and localization (language translation). SimpleForm can be customized with settings in the initialization file.

Resources for Foundation

Zurb maintains a list of Foundation third-party tools and projects:

- Foundation Tools (http://foundation.zurb.com/develop/tools.html)

Resources from Zurb

Zurb provides a collection of example layouts:

- HTML Templates (http://foundation.zurb.com/templates.html)

Find contributed examples of Foundation layouts and design elements here:

- Pattern Tap (http://patterntap.com/code)

Themes for Foundation

Here are sites that offer pre-built themes for sale:

- FoundationMade (http://www.foundationmade.com)
- RailsThemes (https://railsthemes.com/)

Online Design Tools

You can design your pages using Foundation with a drag-and-drop design tool:

- Divshot (http://www.divshot.com/)

Getting Help for Foundation

Zurb has a community forum for questions and answers:

- Foundation Forum (http://foundation.zurb.com/forum)

Stack Overflow (http://stackoverflow.com/questions/tagged/zurb-foundation) is a popular site for questions and answers about Zurb Foundation.

Comments

Credits

Daniel Kehoe implemented the application and wrote the tutorial.

Did You Like This Guide?

Was the article useful to you? Follow @rails_apps (http://twitter.com/rails_apps) on Twitter and tweet some praise. I'd love to know you were helped out by the article.

You can also find me on Facebook (https://www.facebook.com/daniel.kehoe.sf) or Google+ (https://plus.google.com/+DanielKehoe/).