Sinatra Recipes

Community contributed recipes and techniques

Select a topic

Chapters

Sinatra-Assetpack ¶ ↑

Install the gem using:

```
gem install sinatra-assetpack
```

and require it in your app file:

```
require 'sinatra/base'
require 'sinatra/assetpack'

class App < Sinatra::Base
  register Sinatra::AssetPack
  assets do
    # read on
  end

  # Rest of your app
end
```

Generic Structure ¶ ↑

Defaults

If you do not use any CSS dev tools such as Compass or Foundation, and have a simple app structure that you generate on a per-project basis, there are certain defaults added to `Sinatra::Assetpack`. By default, the gem assumes your asset files are located under a folder called "app" in your app's root directory.

This is the default structure which when used, makes it possible to use the gem with almost zero configuration.

```
app
  |- js/
       |- jquery.js
       |- app.js
  |- css/
       |- jqueryui.css
       |- reset.css
       |- foundation.sass
       |- app.sass
myapp.rb
```

Some points of note:

- There is no need to stick to this structure. The filepaths can be configured in `sinatra-assetpack` in case you need it.
- There is no need for the `public` folder which is the default asset look-up path for Sinatra.
- The `.sass` files go into the `css` directory. The conversion will be handled by `sinatra-assetpack` automatically. You just need the `sass` gem installed and loaded using `require 'sass'`.

Inside the `myapp.rb` file, inside the `assets do .. end` block, the paths to the files need to be specified:

```
assets do

  js :application, [
    '/js/jquery.js',
    '/js/app.js'
    # You can also do this: 'js/*.js'
  ]

  css :application, [
    '/css/jqueryui.css',
    '/css/reset.css',
    '/css/foundation.sass',
    '/css/app.sass'
   ]

  js_compression :jsmin
  css_compression :sass

end
```

The symbol that is sent to the `js` and `css` methods becomes the access helper in your views. You can use those helpers like so:

```
<%= css :application %>
<%= js :application %>
```

Which gets expanded to:

```
<link rel="stylesheet" href="/css/application.css"/>
<script src="/js/application.js" type="text/javascript"></script>
```

That is it!

Custom structure

Level 2

In case you have a different app directory structure, say:

```
assets
  |- javascripts/
       |- jquery.js
       |- app.js
  |- stylesheets/
       |- jqueryui.css
       |- reset.css
       |- foundation.sass
       |- app.sass
myapp.rb
```

The `serve` method can be specified so that the folder from which the assets gets served from may be explained to the gem.

```
assets do

    serve '/js', :from => 'assets/javascripts'
    js :application, [
      '/js/jquery.js',
      '/js/app.js'
      # You can also do this: 'js/*.js'
    ]

    serve '/css', :from => 'assets/stylesheets'
    css :application, [
      '/css/jqueryui.css',
      '/css/reset.css',
      '/css/foundation.sass',
      '/css/app.sass'
     ]

    js_compression :jsmin
    css_compression :sass
  end
```

And everything else remains the same.

Foundation framework + Compass

Level Awesome

The previous sections dealt with limited number of assets. What if you have vendor assets that need to be served in a particular order?

This section deals with using the foundation framework with `sinatra-assetpack`. The example also uses the `compass` gem to start a project with the `zurb-foundation` framework.

```
gem install zurb-foundation
gem install compass
```

Complete instructions are not provided here. Follow the Zurb-foundation link provided in the links section for detailed instructions

The created project has a structure like this:

```
├── javascripts
|   ├── foundation
|   |   ├── foundation.alerts.js
|   |   ├── foundation.clearing.js
|   |   ├── foundation.cookie.js
|   |   ├── foundation.dropdown.js
|   |   ├── foundation.forms.js
|   |   ├── foundation.joyride.js
|   |   ├── foundation.js
|   |   ├── foundation.magellan.js
|   |   ├── foundation.orbit.js
|   |   ├── foundation.placeholder.js
|   |   ├── foundation.reveal.js
|   |   ├── foundation.section.js
|   |   ├── foundation.tooltips.js
|   |   └── foundation.topbar.js
|   └── vendor
|       ├── custom.modernizr.js
|       ├── jquery.js
|       └── zepto.js
|   |__ app.js
├── sass
|   ├── _settings.scss
|   ├── app.scss
|   └── normalize.scss
└── stylesheets
|   ├── app.css
|   └── normalize.css
myapp.rb
```

In this app, the `.sass` to `.css` conversion is handled by running `compass watch` which compiles the `.scss` files whenever it detects a change. So, we'll ignore the conversion for now. The concentration would be on how to configure the app to get the files in a particular order.

In this case, the order of the JS files that need to be loaded/merged is:

1. Vendor JS files like jQuery, Modernizr and Zepto.

2. The "foundation.js" file which defines the `Foundation` prototype that gets used in the rest of the `foundation.*.js` files.

3. The `foundation.*.js` files.

Any change in the load order and you might see some of the plugins failing.

```
assets do
  serve '/js', :from => 'javascripts'

  js :foundation, [
    '/js/foundation/foundation.js',
    '/js/foundation/foundation.*.js'
  ]

  js :application, [
    '/js/vendor/*.js',
    '/js/app.js'
  ]

  serve '/css', :from => 'stylesheets'
  css :application, [
    '/css/normalize.css',
    '/css/app.css'
   ]

  js_compression :jsmin
  css_compression :sass
end
```

Inside the view:

```
<%= css :application %>
<%= js :application %>
<%= js :foundation %>
```

Do this, and you'll see that the files are loaded properly and there will be no JS errors in the browser's console.

Merging ¶↑

If you have tried the code samples, you might've observed that the files are not getting merged before they are sent to the browser. This is true and happens only in development mode. Change it to production:

```
export RACK_ENV="production"
```

Voila! Now you should see only three asset files being loaded viz., "application.css", "application.js" and "foundation.js".

Pre-compilation ¶↑

Up until now, the concatenation, compression are done after a request reaches the server for the first time. This step is done only once for the first request. However, if you need to pre-generate the compressed and concatenated assets, you can use the rake task provided in the gem:

Create a `Rakefile` in your app directory with the following contents:

```
APP_FILE  = 'app.rb'
APP_CLASS = 'Sinatra::Application'

require 'sinatra/assetpack/rake'
```

And run `rake assetpack:build` to generate the assets which automatically get stored in a `public` directory.

Resources ¶↑

- Sinatra-Assetpack
- Foundation Framework
- Compass Framework

Did we miss something?

It's very possible we've left something out, that's why we need your help!This is a community driven project after all. Feel free to fork the project and send us a pull request to get your recipe or tutorial included in the book.

See the README for more details.

Top