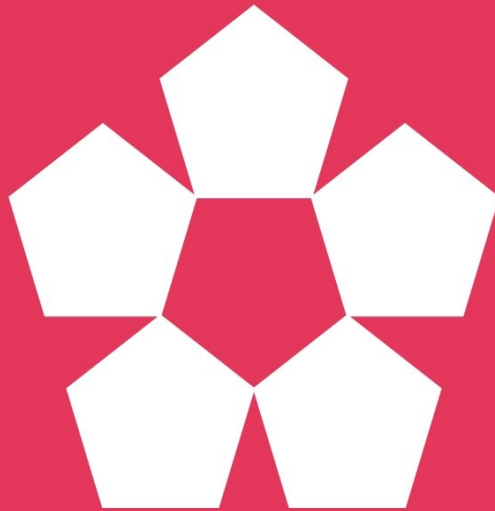


Theme Development with Sage



**Build well organized & easily maintained
WordPress themes using a modern web
development workflow**

by Ben Word

Third edition

Sage 9.0.0

February 2018

<https://roots.io>

© 2018 Ben Word

Summary

Introduction	1
What is Sage?	2
Why Sage?	3
Starting a project	4
Installing the Sage starter theme	4.1
Theme setup	4.2
Planning out the theme	4.3
Customizing templates	5
Blade templates	5.1
Template files	5.2
Building template partials	5.3
Responsive images & post thumbnails	5.4
Displaying the sidebar	5.5
Customizing the front-end	6
CSS setup and organization	6.1
Default CSS and JS	6.2
Bootstrap	6.3
Browsersync	6.4
JavaScript and DOM-based routing	6.5
3rd party packages	6.6
Asset paths in CSS and templates	6.7
Theme functionality	7
Adding additional files	7.1

Namespaces	7.2
Take advantage of newer PHP features	7.3
Theme customizations	8
WordPress Customizer	8.1
Advanced Custom Fields	8.2
Theme deployment	9
Theme troubleshooting	10
Theme updates and maintenance	11
Resources	12
Wrapping up	13
CHANGELOG	14

In your typical WordPress theme, every page template will look something like the following:

```

get_header(); ?>

<div id="primary" class="content-area">
  <main id="main" class="site-main" role="main">

    <?php
    while ( have_posts() ) : the_post();
      get_template_part( 'template-parts/content', 'page' );
      // If comments are open or we have at least one comment, load up the comment template.
      if ( comments_open() || get_comments_number() ) :
        comments_template();
      endif;
    endwhile; // End of the loop.
    ?>

  </main><!-- #main -->
</div><!-- #primary -->

<?php
get_sidebar();
get_footer();

```

Even though we know that every template will take this base format and render the header, footer, sidebar calls each time, we still need to continuously repeat the code.

One of the biggest benefits of using Blade templates is the ability to remove any repeated markup from individual templates and put it into a single file. This file, [resources/views/layouts/app.blade.php](#), becomes the base layout file. By doing this we can put the focus entirely on the page specific markup and loop, simplifying our templates to look like this:

```

@extends( 'layouts.app' )

@section( 'content' )
  @while( have_posts() ) @php( the_post() )
    @include( 'partials.page-header' )
    @include( 'partials.content-page' )
  @endwhile
@endsection

```

It's neat. It's tidy. You never need to make calls to `get_header()`, `get_footer()`, or `get_sidebar()` again. You can also refactor the base layout of your site by editing [app.blade.php](#).

Passing data to templates

Sage includes a `sage/template/{class}/data` filter that can be used to pass data to templates. This is the most simple way to pass data. The filter is based of body classes and can be used to target specific templates, for example:

- `sage/template/home/data` — Home page
- `sage/template/about/data` — About page
- `sage/template/page/data` — All pages
- `sage/template/post-type-archive-event/data` — `event` post type archive
- `sage/template/single-event/data` — `event` single post template

Note: Sage comes ready for you to modify the `body_class` by editing the filter at the top of [app/filters.php](#).

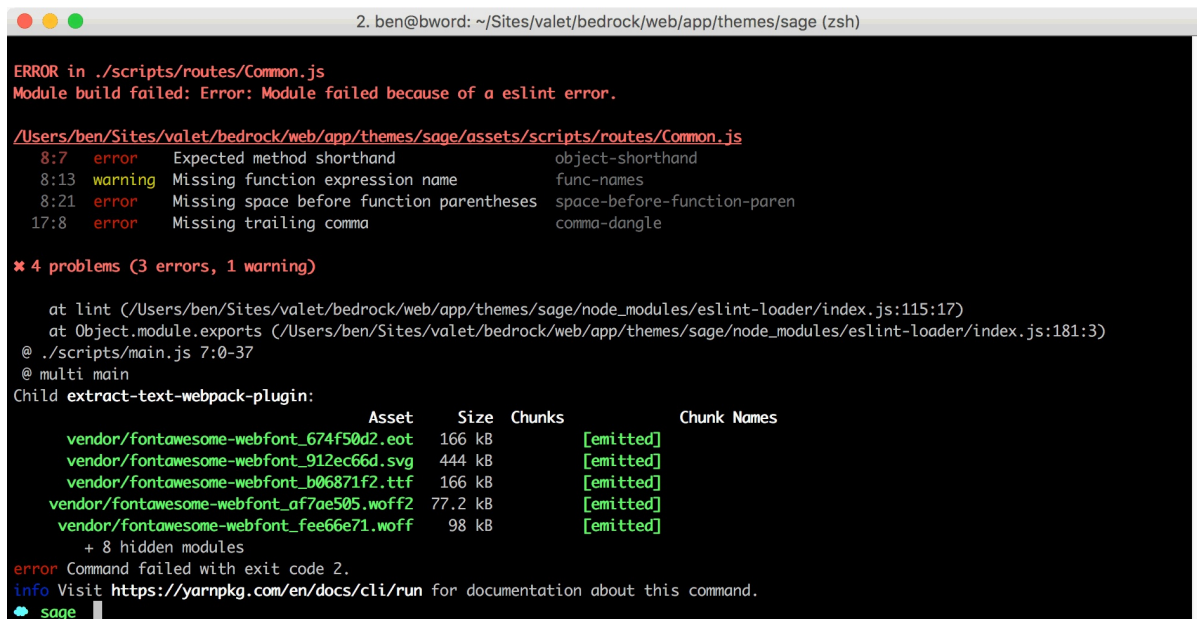
In the example below we're passing the the values of two ACF fields, `header_image` and `header_content`, to all pages:

JavaScript and DOM-based routing

Sage uses ES6 for the JavaScript that's included in the theme. Since Sage strives to use the current best practices for modern front-end development, the theme allows the usage of ES6 and also enforces coding standards with ESLint.

🔗 See the resources chapter for lots of recommended content on learning more about ES6!

Since Sage doesn't ship with any JavaScript out of the box, except for the DOM-based router, you might not notice this until you modify the JS and attempt to build.



```

2. ben@bword: ~/Sites/valet/bedrock/web/app/themes/sage (zsh)
ERROR in ./scripts/routes/Common.js
Module build failed: Error: Module failed because of a eslint error.

/Users/ben/Sites/valet/bedrock/web/app/themes/sage/assets/scripts/routes/Common.js
  8:7   error   Expected method shorthand          object-shorthand
  8:13  warning  Missing function expression name    func-names
  8:21  error   Missing space before function parentheses  space-before-function-paren
 17:8   error   Missing trailing comma              comma-dangle

* 4 problems (3 errors, 1 warning)

at lint (/Users/ben/Sites/valet/bedrock/web/app/themes/sage/node_modules/eslint-loader/index.js:115:17)
at Object.module.exports (/Users/ben/Sites/valet/bedrock/web/app/themes/sage/node_modules/eslint-loader/index.js:181:3)
@ ./scripts/main.js 7:0-37
@ multi main
Child extract-text-webpack-plugin:
      Asset      Size  Chunks             Chunk Names
  vendor/fontawesome-webfont_674f50d2.eot  166 kB  [emitted]
  vendor/fontawesome-webfont_912ec66d.svg  444 kB  [emitted]
  vendor/fontawesome-webfont_b06871f2.ttf  166 kB  [emitted]
  vendor/fontawesome-webfont_af7ae505.woff2  77.2 kB  [emitted]
  vendor/fontawesome-webfont_fee66e71.woff   98 kB  [emitted]
+ 8 hidden modules
error Command failed with exit code 2.
info Visit https://yarnpkg.com/en/docs/cli/run for documentation about this command.
sage

```

The [ESLint configuration](#) is located at `assets/build/.eslintrc`, where you might want to make changes that fit your own coding styles. We've picked [Airbnb's JavaScript standard](#) as the basis for ours.

To disable the linting in a specific file or within a file, use [inline comments](#):

```

/* eslint-disable */
alert('foo');
/* eslint-enable */

```

The primary theme JavaScript file located at `resources/assets/scripts/main.js` is used to import dependencies from vendored packages (Bootstrap) as well as local dependencies. Out of the box, Sage comes with the following local dependencies:

- `common` (`resources/assets/scripts/routes/common.js`) — JavaScript fired on all pages
- `home` (`resources/assets/scripts/routes/home.js`) — JavaScript fired on a page with a body class that contains `home`
- `aboutUs` (`resources/assets/scripts/routes/about.js`) — JavaScript fired on a page with a body class that contains `about-us`

The local dependencies are based on DOM-based routing, which lets you conditionally execute JS on certain pages based on the page's body classes. To add a new route, you'll need to create the file, import it, and add it to the `routes` variable in `main.js`.

When you're working with a body class that contains a dash, such as `contact-us`, you'll need to replace the dash with a

Asset paths in CSS and templates

The folder structure of `resources/assets/` and `dist/` are the same. When you run the build script, the assets from the `resources/assets/` directory get compiled into the same location within the `dist/directory/`.

Images that are in `assets/images/` get optimized and copied over to the `dist/images/` folder. Subdirectories are kept intact, too. For instance:

```
resources/assets/images/logo.svg      -> dist/images/logo.svg
resources/assets/fonts/opensans.woff2  -> dist/fonts/opensans.woff2
resources/assets/images/home/product-01.jpg -> dist/images/home/product-01.jpg
```

If you're in a stylesheet and you'd like to reference an image, `resources/assets/images/logo.svg`, you would write:

```
.brand {
  background: url(../images/logo.svg);
}
```

If you're in a template and you'd like to reference the same image, you would use the `@asset` directive:

```

```

If you're in a stylesheet and you'd like to reference a font, you would write:

```
@font-face {
  font-family: 'opensans';
  src: url('../fonts/opensans.woff2') format('woff2');
  font-weight: normal;
  font-style: normal;
}
```

- [Vue.js and Sage 9 example](#)
- [Move required theme files to theme root folder](#)
- [WooCommerce and Sage 9](#)
- [Sage 9 child theme](#)
- [Router custom functions in Sage 9](#)

People to follow

WordPress

- [Alain Schlessler](#)
- [Danny van Kooten](#)
- [Giuseppe Mazzapica](#)
- [Iain Poulson](#)
- [K. Adam White](#)
- [Tom J Nowell](#)

Trac tickets worth following

If you login to your WordPress.org account on Trac you can star tickets to get notifications of any new activity. These tickets are some of the ones that I'm following:

- [#3833](#) — Extra `</p>` inside blockquote
- [#4298](#) — wpautop bugs
- [#5250](#) — wpautop() issue with lists
- [#7795](#) — Activate and Deactivate Theme hooks
- [#12877](#) — Modular themes: Apply template hierarchy to folders within a theme
- [#13239](#) — Filter `locate_template_names` variable
- [#14502](#) — Enable `/post-type/taxonomy/term/` permalinks
- [#15086](#) — `get_template_part()` should let you specify a directory
- [#21022](#) — Allow `bcrypt` to be enabled via filter for pass hashing
 - Use [wp-password-bcrypt](#) in the meantime (included with Bedrock)
- [#21790](#) — WP main query isn't set correctly for static front page
- [#22316](#) — Plugin Dependencies
- [#23221](#) — Multisite in subdirectory with root site address
 - Use [multisite-url-fixer](#) in the meantime
- [#23912](#) — Add Composer package description
- [#24044](#) — Add `index` to `wp_options` to aid/improve performance
- [#24152](#) — Use JSON as alternative to headers 🍕
- [#31475](#) — Add ability to change the folder location for templates
- [#31258](#) — SVG replaced by default image in media library
- [#33381](#) — Strategize the updating of minimum PHP version
- [#33472](#) — Templating Engine
- [#33473](#) — Shortcodes + Widgets + Nav Menus. Unified “component” API (aka Content Blocks)
- [#34136](#) — Allow `register_post_type`'s rewrite to remove CPT slug
- [#35669](#) — Store widgets in a custom post type instead of options
- [#36292](#) — Rewrites: Next Generation
- [#36335](#) — Core autoloader proposal
- [#39309](#) — Secure WordPress Against Infrastructure Attacks