Root's Sage 9

Current techniques to create themes

Current techniques to create themes

Hi, I’m Paulo Carvajal

- more than 20 years of experience creating websites.
- more than 12 years working with WordPress.
- I'm in charge of the Web Area in vudumedia.com.
- Freelance developer at Toptal and Codeable.
- Bachelor of Fine Arts, Basque Country University - UPV-EHU - and technical specialist in Audiovisual Media.

¿What is Sage?

¿What is Sage?

A starter theme for WordPress with a current workflow.

- It is a starter theme.
- Modern development and easy to maintain.
- It has a great community.
- Experience using Underscore or Genesis.
- It forces to write better code: OOP and ES6.

There are other start themes:
Underscores, Bones, FoundationPress, etc.

¿What is Sage?

Tools, technologies and advantages

Optimized structure and files organization.

It uses Composer to manage PHP packages.

It uses npm/yarn for front-end package management.

It writes style sheets with SASS easily.

It compiles and optimizes resources (CSS, images, fonts, etc.).

It uses the template engine Blade: Template inheritance.

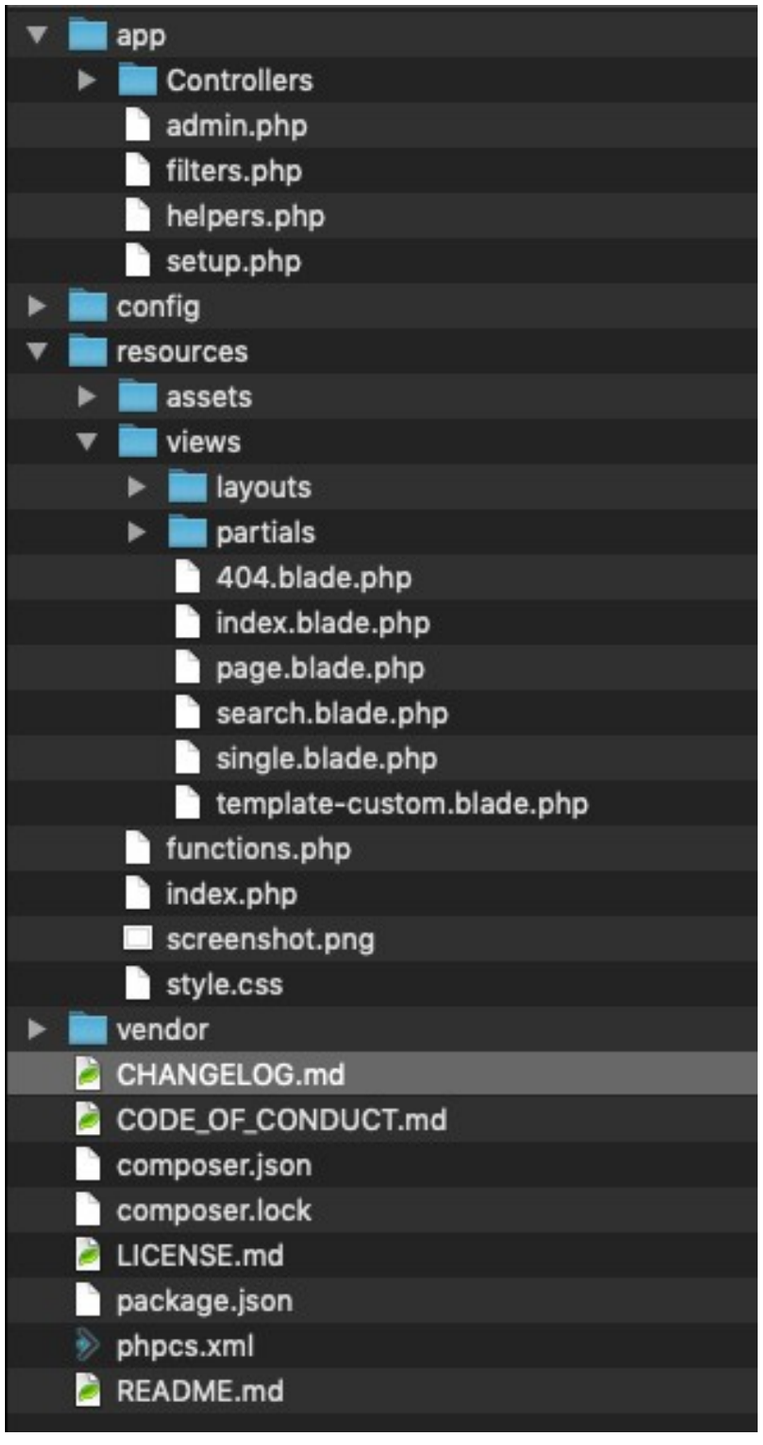
¿What is Sage?

Requirements

- WordPress >= 4.7
- PHP >= 7.1.3 (with php-mbstring enabled)
- Composer
- Node.js >= 8.0.0
- Yarn

Files structure

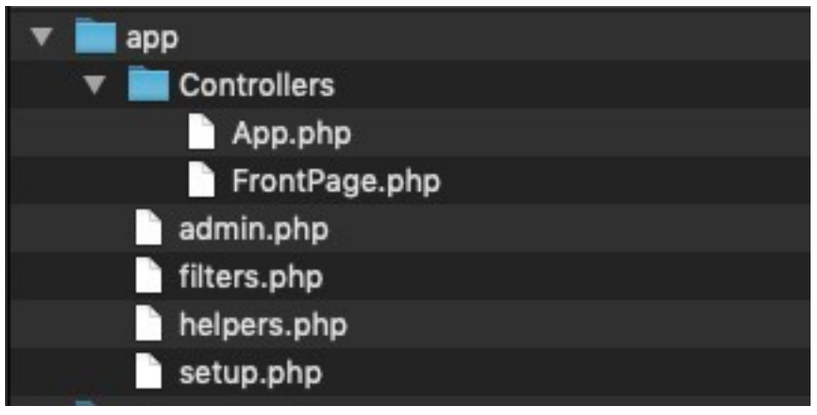
Files structure



Approach to MCV

- app: controllers, filters, setup.
- resources: assets and views.

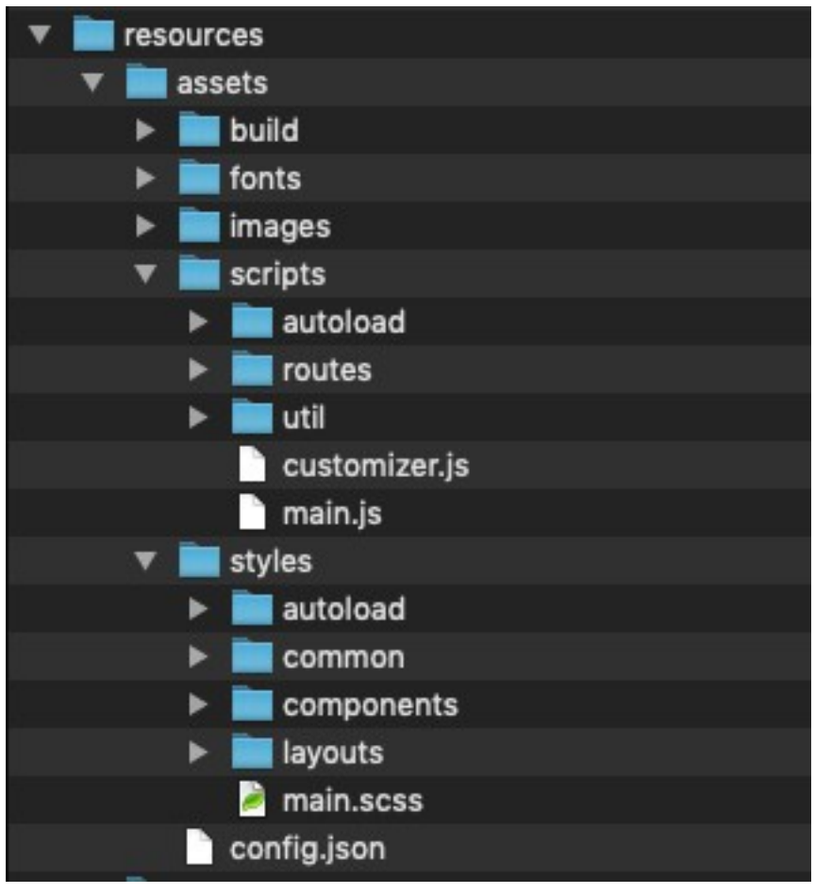
Files structure



Functions

- Filters.
- Controllers.
- setup.php

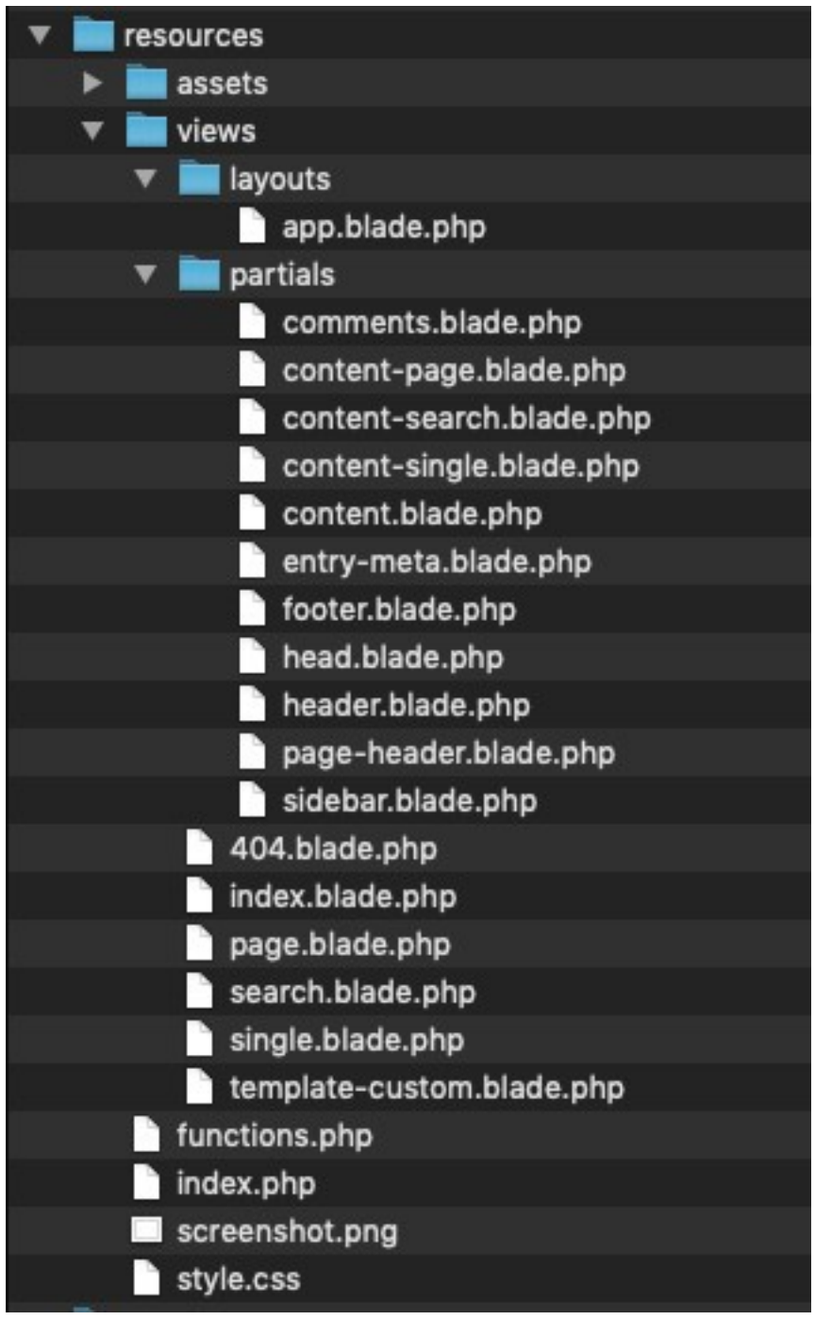
Files structure



Assets

- Use of SCSS.
- Modules JS ES6.
- Router Javascript base on DOM.
- Generation and minification with nmp.

Files structure



Views

- WordPress hierarchy.
- Use of Blade.
- Use of partials.

Assets management

Assets management

- Compilation in a single JS and in a single CSS. Compress the images.
- Javascript routing based on DOM. ESLint and stylelint.

Assets management

Manage scripts from a configuration file.

The fij.json file in the assets directory controls different scripts that are added to the theme.

By default, Sage compiles two JS files and one CSS file:

```
"entry": {
  "main": [
    "./scripts/main.js",
    "./styles/main.scss"
  ],
  "customizer": [
    "./scripts/customizer.js"
  ]
}
```

Assets management

Manage scripts from a configuration file.

New files can be added to be generated easily, this is useful for files that only need to load on certain pages.

```
"entry": {
  "main": [
    "./scripts/main.js",
    "./styles/main.scss"
  ],
  "customizer": [
    "./scripts/customizer.js"
  ],
  "checkout": [
    "./scripts/checkout.js"
  ]
}
```

And do not forget to place them properly:

```
add_action('wp_enqueue_scripts', function () {
    if (is_page('checkout')) {
        wp_enqueue_script('sage/checkout.js', asset_path('scripts/checkout.js'),
['jquery'], null, true);
        }
}, 100);
```

Assets management

The three Sage commands

yarn build — Compiles and optimizes files under assets.
yarn build:production — Compiles assets for production.
yarn start — Compiles assets when any file is modified and starts a Browersync session.

Assets management

```
yarn start
yarn run v1.10.1
$ webpack --hide-modules --watch --config resources/assets/build/webpack.config.js

Webpack is watching the files…



 DONE  Compiled successfully in 3487ms

          13:33:25

[BS] [HTML Injector] Running...
[Browsersync] Proxying: http://example.test
[Browsersync] Access URLs:
 ------------------------------------
       Local: http://localhost:3000
    External: http://192.168.0.50:3000
 ------------------------------------
         UI: http://localhost:3001
 UI External: http://192.168.0.50:3001
 ------------------------------------
[Browsersync] Watching files...


 DONE  Compiled successfully in 949ms
```

Assets management

Javascript routing based on DOM.

For instance, the template:
views/mi-template.blade.php

We would create the file:
assets/scripts/routes/miTemplate.js

Assets management

Javascript routing based on DOM.

And we import the file myTemplate.js into main.js:

```
import miTemplate from './routes/miTemplate';
```

```
// import external dependencies
import 'jquery';

// import local dependencies
import Router from './util/Router';
import common from './routes/common';
import home from './routes/home';
import aboutUs from './routes/about';
import miTemplate from './routes/miTemplate';

/** Populate Router with DOM routes */
const routes = new Router({
  common, // All pages
  home,
  aboutUs, // change from about-us to aboutUs
  miTemplate,
});
```

Modern PHP

PHP standards

Sage uses the PSR-2 specification, the most widely used, instead of the WordPress own specification.

Uses short notation for arrays: [].
It allows anonymous functions.
Short syntax of echo.
Uses Namespaces to avoid name collisions. It works under the namespace App.

PHP standards

Use short notation for arrays

```
$new_query = new WP_Query([
  'post_type'      => 'peliculas',
  'posts_per_page' => 5,
  'tax_query'      => [
    [
      'taxonomy' => 'generos',
      'field'    => 'slug',
      'terms'    => ‘drama'
] ]
]);
```

PHP standards

It allows anonymous functions.

```
// Función anónima
add_filter('excerpt_length',function() {
  return 20;
});
```

```
// Función declarada como variable
$length = function() {
  return 20;
};
add_filter('excerpt_length', $length);
```

PHP standards

Echo' short syntax

```
// echo  estándar
<?php echo $category->name;  ?>
```

```
// Sintaxis  corta
<?= $category->name; ?>
```

PHP standards

Namespaces

The namespace is simply a way to keep the functions and classes organized to avoid names collisions.

There is a new keyword that accompanies namespaces, it is called **use**. This keyword allows us to tell PHP to look up a namespace for each function call.

PHP standards

Namespaces

Functions calls using namespace:

```
<?php

namespace Branch\Titles; function title() {
  if (is_archive()) {
    returnget_the_archive_title();
  } else {
    return get_the_title();
  }
}
add_filter('the_title', __NAMESPACE__ . '\\title');
```

or into the global namespace:

```
namespace Branch\Titles;

$query = new \WP_Query();
```

Blade templates

Blade templates

- More readable and structured templates.
- Inheritance of templates to create different layouts.
- Separation of data from the views.
- Data automatic escaping.
- Follows the hierarchy of WordPress templates.
- They can receive data from the father, from filters or a controller.

Blade templates

More readable and structured templates.

```
<article @php(post_class())>
  <header>
    <h2 class="entry-title">
      <a href="{{ get_permalink() }}">
        {{ get_the_title() }}
      </a>
    </h2>
    @include('partials/entry-meta')
  </header>
  <div class="entry-summary">
    @php(the_excerpt())
  </div>
</article>
```

views/partials/content-single.blade.php

Blade templates

Inheritance of templates to create different layouts.

```blade
<!doctype html>
<html {!! get_language_attributes() !!}>
  @include('partials.head')
  <body @php body_class() @endphp>
    @php do_action('get_header') @endphp
    @include('partials.header')
    <div class="wrap container" role="document">
      <div class="content">
        <main class="main">
          @yield('content')
        </main>
        @if (App\display_sidebar())
          <aside class="sidebar">
            @include('partials.sidebar')
          </aside>
        @endif
      </div>
    </div>
    @php do_action('get_footer') @endphp
    @include('partials.footer')
    @php wp_footer() @endphp
  </body>
</html>
```

layouts/app.blade.php

```blade
@extends('layouts.app')

@section('content')
  @include('partials.page-header')

  @if (!have_posts())
    @include('partials.not-found')
    {!! get_search_form(false) !!}
  @endif

  @while (have_posts()) @php the_post() @endphp
    @include('partials.content-'.get_post_type())
  @endwhile

  {!! get_the_posts_navigation() !!}
@endsection
```

views/index.blade.php

Blade templates

Show data in templates

The Blade's {{}} statements are sent automatically through the PHP **htmlspecialchars** function to avoid XSS attacks.

```
Hello, {{ $name }}.
```

Data without escaping:

```
Hello, {!! $name !!}.
```

Run PHP:

```
@php the_post() @endphp
```

Index assets in the front:

```
<imgsrc="@asset('images/estrella.jpg')" alt="Estrella">
```

Blade templates

Directives and control structures

{{ $var }} - Echo content
{{ $var or 'default' }} - Echo content with a default value
{!! $var !!} - Echo unescaped content
{{-- Comment --}} - A Blade comment
@extends('layout') - Extends a template with a layout
@if(condition) - Starts an if block
@else - Starts an else block
@elseif(condition) - Start a elseif block
@endif - Ends a if block
@foreach($list as $key => $val) - Starts a foreach block
@endforeach - Ends a foreach block
@for($i = 0; $i < 10; $i++) - Starts a for block
@endfor - Ends a for block
@while(condition) - Starts a while block
@endwhile - Ends a while block
@unless(condition) - Starts an unless block
@endunless - Ends an unless block
@include(file) - Includes another template
@include(file, ['var' => $val,...]) - Includes a template, passing new variables.
@yield('section') - Yields content of a section.
@show - Ends section and yields its content
@section('name') - Starts a section
@stop - Ends section
@endsection - Ends section

Blade templates

Data moving to templates

Global data:
Sage includes the function sage('blade')->share() which allows to move data into all views:

```
add_action('the_post', function() {
    sage('blade')->share('links',
    [
    'facebook' => 'https://facebook.com/rootswp',
        'twitter' => 'https://twitter.com/rootswp'
]);
});
```

Now, it could be used in any template:

```
{{ $links['facebook'] }} o {{ $links['twitter'] }}
```

Blade templates

Move data into templates

Partial data:
Blade directives allow you to move data into partial views:

```
@include('partials.content-page', ['var' => $val,...])
```

We could say that it is an advanced get_template_part.

Blade templates

Move data into templates

Sage includes a sage/template/{$ class}data filter that can be used to move data into templates.

```
For example: app/controllers/front-
page.php app/controllers/archive-
event.php app/controllers/single-
event.php
app/controllers/template-about.php


add_filter('sage/template/page/data',
function (array $data)  {
    $data['header_image'] =
    get_field('header_image');
    $data['header_content'] = get_field('header_content'); return
    $data;
});”

And into the template: {{ $data[‘header_content']  }}
```

The Controller

The Controller

- The names of the classes follow the same hierarchy as WordPress.
- The name of the class must match the file name.
- Use public functions to return data into views.
- Use public static functions to execute the method that returns data into the template.

The Controller

The names of the classes

Follow the WordPress hierarchy. Classes should be called alike, just switching to CamelCase.

```php
<?php // @ app/controllers/single-event.php

namespace App;

use Sober\Controller\Controller;

class SingleEvent extends Controller {

    public function eventStartingLocationDescription() {
        return get_field('event_starting_location_description');
    }

    public function eventDuration() {
        return wpautop(get_field('event_duration'));
    }
}
```

app/controllers/front-page.php
app/controllers/archive-event.php
app/controllers/single-project.php
app/controllers/template-about.php

The Controller

Use of methods to generate variables

```php
<?php // @ app/controllers/single-event.php

namespace App;

use Sober\Controller\Controller;

class SingleEvent extends Controller
{

    public function eventDuration()
    {
        return wpautop(get_field('event_duration'));
    }
}
```

Now, the variable $event-duration is approachable into the view single-event.blade.php:
{{ $event_duration }}

Always pay attention to “cases”:
File - ClassName: single-event -> SingleEvent
Method- Variable: eventDuration -> $event_duration

The Controller

Use of static methods

```php
<?php // @ app/Controllers/Archive.php

namespace App\Controllers;

use Sober\Controller\Controller;

class Archive extends Controller
{
    public static function title()
    {
        return get_post()->post_title;
    }
}
```

```php
<?php // @ resources/views/archive.php

@extends('layouts.app')

@section('content')

  @while (have_posts()) @php(the_post())
    {{ Archive::title() }}
  @endwhile

@endsection
```

The Controller

Creating components

It also can create reusable components and include them in any class of controller using PHP traits.

```php
<?php // @ app/Controllers/Partials/Images.php

namespace App\Controllers\Partials;

trait Images
{
    public function images()
    {
        return get_field('images');
    }
}
```

```php
<?php // @ app/Controllers/Single.php

namespace App\Controllers;

use Sober\Controller\Controller;

class Single extends Controller
{
    use Partials\Images;
}
```

The Controller

Inheritance using Tree

By default, each controller steps its template hierarchy depending on the controller's specificity (just as WordPress templates work).

Data from less specific controllers in the hierarchy can be inherited implementing the Tree Interface.

For example, the app/Controllers/Single.php controller will inherit the app/Controllers/Singular.php methods when the first implements Tree.

The Controller

Advanced Custom Fields

The controller has a useful auxiliary module to automate the transfer from ACF fields.

The automated fields use the names of the ACF variables and move them into the view. The controller also moves the options values by default.

The values are returned as objects, but they can be modified to keep them as arrays.

The Controller

Advanced Custom Fields

```php
<?php // @ app/Controllers/Single.php

namespace App\Controllers;

use Sober\Controller\Controller;

class Single extends Controller
{
    // Pass on all fields from Advanced Custom Fields to the view
    protected $acf = true;

    // Pass on only field_1 from Advanced Custom Fields to the view
    protected $acf = 'field_1';

    // Pass on multiple fields from Advanced Custom Fields to the view
    protected $acf = ['field_1', 'field_2'];
}
```

```
add_filter('sober/controller/acf/array', function () {
    return true;
});
```

Briefly

Briefly

- You must be open-minded to new ways of working.
- Although it seems a very different way of doing, all you will find are advantages.
- The access barrier is smaller than it seems.
- You don't need to use everything all the time.
- It is not necessary to be an expert to use it.

Links of interest

Links of interest

- https://github.com/roots/sage
- https://discourse.roots.io
- https://laravel.com/docs/5.7/blade
- https://scotch.io/tutorials/simple-laravel-layouts-using-blade
- https://github.com/soberwp/controller
- https://github.com/Log1x/sage-directives
- https://www.browsersync.io/