Root's Sage 9

Técnicas actuales para crear temas

Técnicas actuales para crear temas

Hola, soy Paulo Carvajal

- 20+ años de experiencia creando webs.
- 12+ años trabajando con WordPress.
- Soy encargado del área web en vudumedia.com.
- Desarrollador freelance en Toptal y en Codeable.
- Licenciado en Bellas artes por la UPV-EHU y técnico especialista en Medios audiovisuales.

¿Qué es Sage?

¿Qué es Sage?

Un starter theme para WordPress con un flujo de trabajo actual.

- Es un starter theme.
- Desarrollo moderno y fácil de mantener.
- Tiene una gran comunidad.
- Experiencia usando Underscore o Genesis.
- Obliga a escribir mejor código: OOP y ES6.

Hay otros temas de inicio:
Underscores, Bones, FoundationPress, etc.

¿Qué es Sage?

Herramientas, tecnologías y ventajas

- Mejor estructura y organización de ficheros.
- Usa Composer para la gestión de paquetes PHP.
- Usa npm/yarn para la gestión de paquetes front-end.
- Escribe las hojas de estilo más fácilmente con SASS.
- Compila y optimiza los recursos (CSS, imágenes, fuentes, etc.).
- Utiliza el motor de plantilla Blade: Template inheritance.

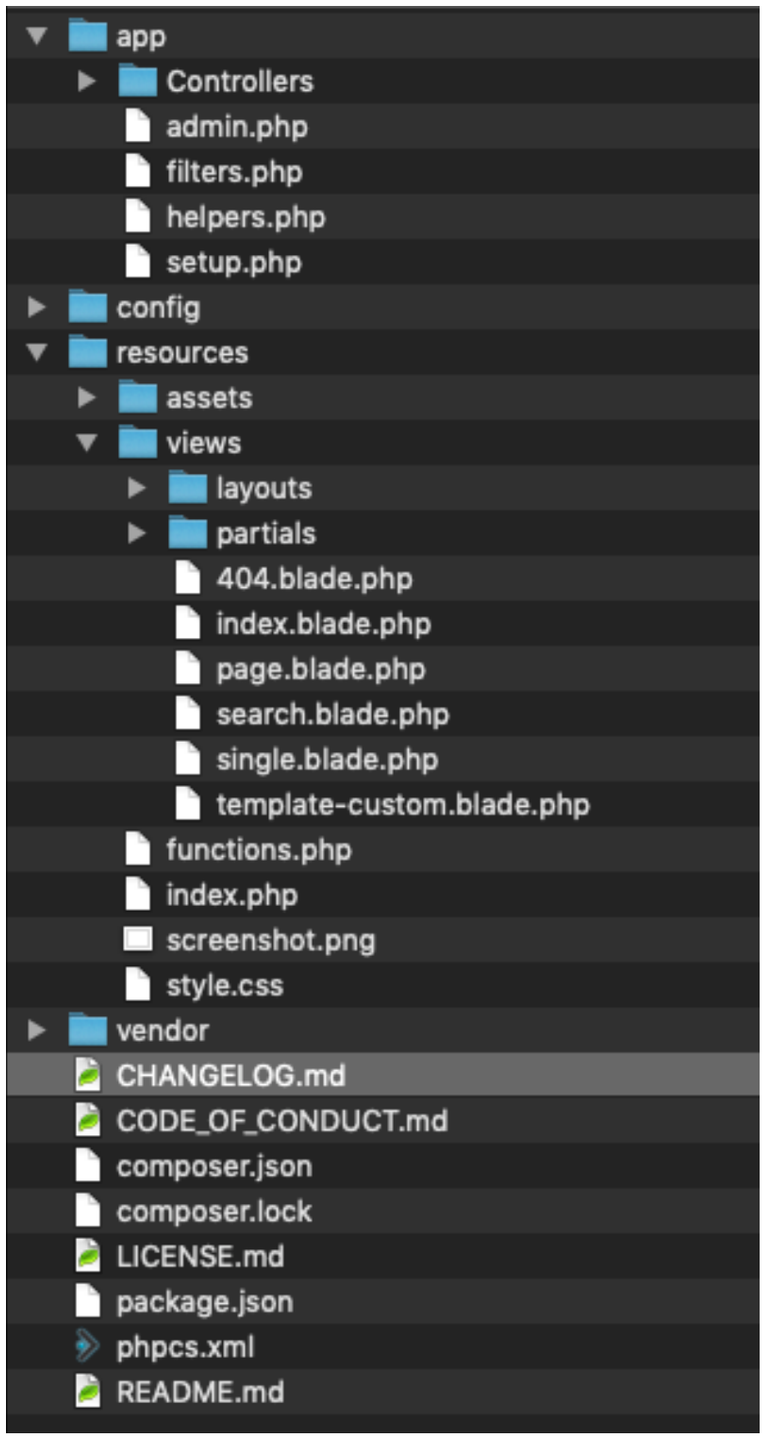
¿Qué es Sage?

Requerimientos

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

Estructura de ficheros

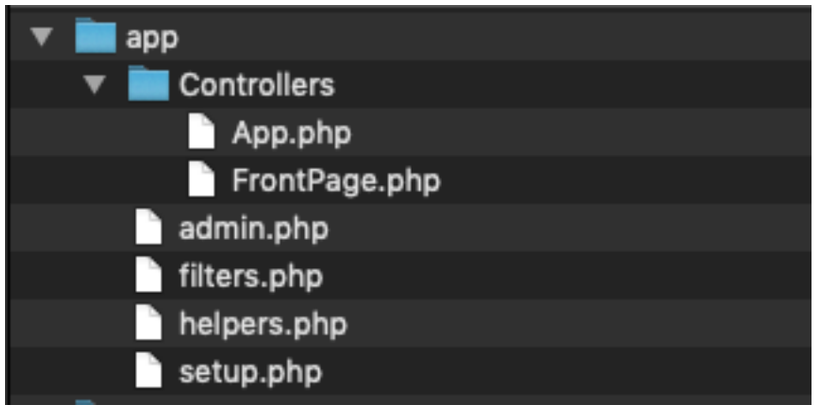
Estructura de ficheros



Acercamiento a MCV

- app: Controladores, filtros, setup.
- resources: assets y views

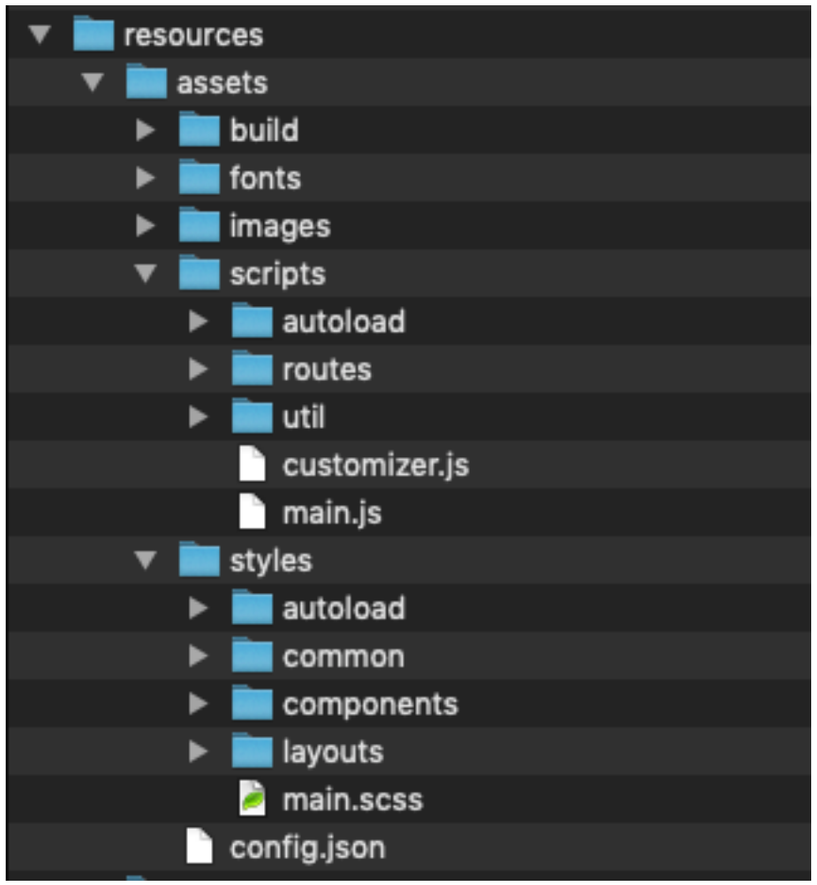
Estructura de ficheros



Functions

- Filtros.
- Controladores.
- setup.php

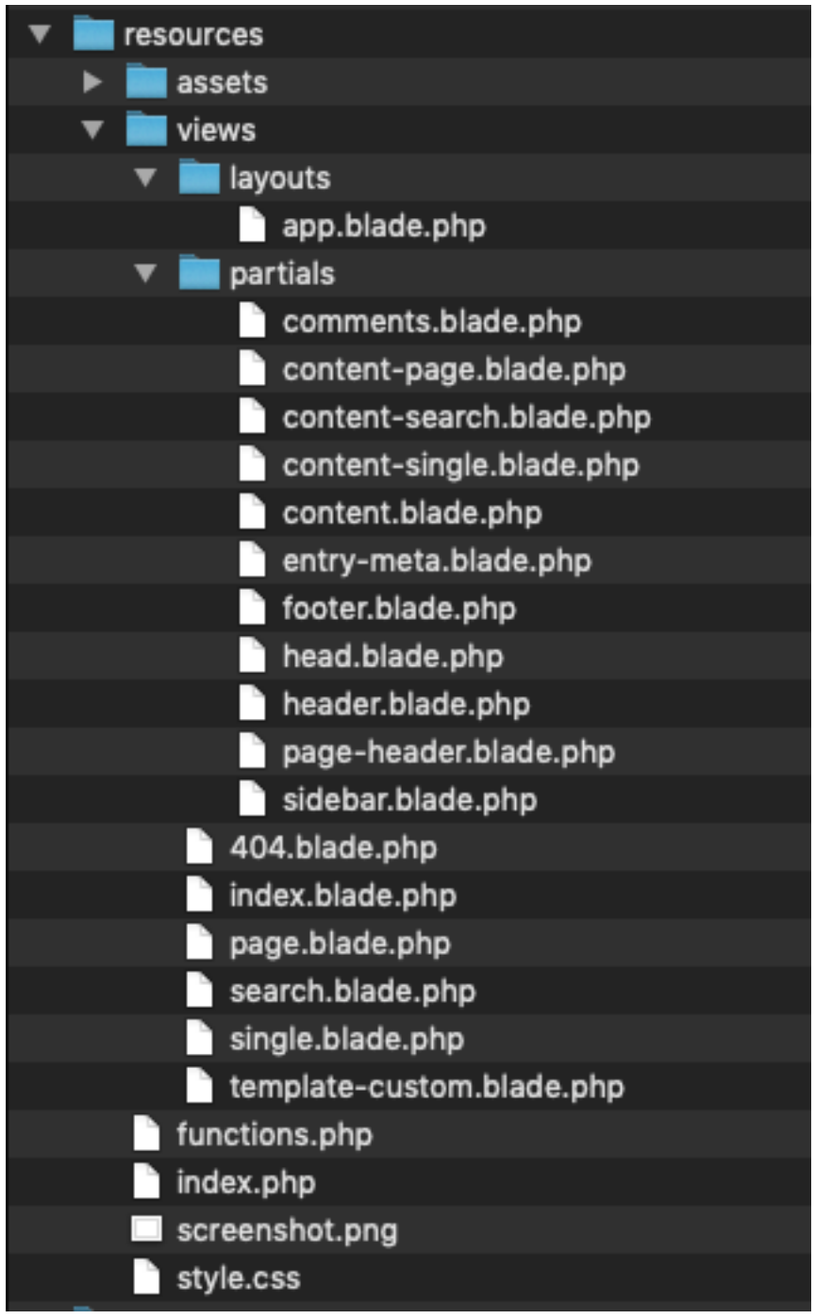
Estructura de ficheros



Assets

- Uso de SCSS.
- Módulos JS ES6.
- Router Javascript basado en el DOM.
- Generación y minificación con nmp.

Estructura de ficheros



Vistas

- Jerarquía de WordPress.
- Uso de Blade.
- Uso de parciales.

Gestión de assets

Gestión de assets

Compilación en un único JS y en un único CSS.
Comprime las imágenes.
Enrutado de Javascript basado en el DOM.
ESLint y stylelint.

Gestión de assets

Gestiona los scripts desde un fichero de configuración

El archivo config.json en el directorio **assets** controla los diferentes scripts que se añaden al tema.

De forma predeterminada, Sage compila dos archivos JS y un archivo CSS:

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

Gestión de assets

Gestiona los scripts desde un fichero de configuración

Se pueden añadir nuevos archivos para ser generados fácilmente, esto es útil para ficheros que sólo deban cargar en determinadas páginas.

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

Y no olvidar encolarlos adecuadamente:

```
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);
```

Gestión de assets

Los tres comandos de Sage

yarn build — Compila y optimiza los ficheros bajo assets.
yarn build:production — Compila los assets para producción.
yarn start — Compila los assets cuando se modifica cualquier fichero e inicia una sesión Browersync.

Gestión de assets

```
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
```

Gestión de assets

Enrutado de Javascript basado en el DOM.

Por ejemplo, para la plantilla:
views/mi-template.blade.php

Crearíamos el fichero:
assets/scripts/routes/miTemplate.js

Gestión de assets

Enrutado de Javascript basado en el DOM.

E importamos el fichero miTemplate.js en 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,
});
```

PHP moderno

PHP standards

Sage usa la especificación PSR-2, la de uso más extendido, en vez de la especificación propia de WordPress.

Usa notación corta para los arrays: [].
Permite funciones anónimas.
Sintaxis corta de echo.
Usa Namespaces para evitar colisiones de nombres.
Funciona bajo el namespace App.

PHP standards

Usa notación corta para los arrays

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

PHP standards

Permite funciones anónimas

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

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

PHP standards

Sintaxis corta de echo

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

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

PHP standards

Namespaces

El espacio de nombres es simplemente una forma de mantener las funciones y clases organizadas y evitar colisiones de nombres.

Hay una nueva palabra clave que acompaña a los espacios de nombres, llamada **use**. Esta palabra clave nos permite decirle a PHP que busque un espacio de nombres para cada llamada a una función.

PHP standards

Namespaces

Llamadas a funciones usando namespace:

```
<?php

namespace Branch\Titles;

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

O en el namespace global:

```
namespace Branch\Titles;

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

Plantillas Blade

Plantillas Blade

- Plantillas más legibles y estructuradas.
- Herencia de plantillas para crear diferentes layouts.
- Separación de datos de las vistas.
- Escapado automático de datos.
- Sigue la jerarquía de plantillas de WordPress.
- Pueden recibir datos del padre, por filtros o de un controlador.

Plantillas Blade

Plantillas más legibles y estructuradas

```
<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

Plantillas Blade

Herencia de plantillas para crear diferentes 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

Plantillas Blade

Mostrar datos en las plantillas

Las declaraciones de {{}} de Blade se envían automáticamente a través de la función **htmlspecialchars** de PHP para evitar ataques XSS.

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

Datos sin escapar:

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

Ejecutar PHP:

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

Referenciar assets en el front:

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

Plantillas Blade

Directivas y estructuras de control

```
{{ $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
```

Plantillas Blade

Pasar datos a las plantillas

Datos globales:
Sage incluye la función sage('blade')->share() que permite pasar datos a **todas** vistas:

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

A partir de esto se podría usar esto en cualquier plantilla:

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

Plantillas Blade

Pasar datos a las plantillas

Datos parciales:
Las directivas de Blade permiten pasar datos a las vistas parciales:

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

Podríamos decir que es un get_template_part avanzado.

Plantillas Blade

Pasar datos a las plantillas

Sage incluye un filtro sage/template/{$class}/data que se puede usar para pasar datos a las plantillas.

```
Por ejemplo:
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;
});”


Y en la plantilla: {{ $data[‘header_content'] }}
```

El Controller

El Controller

- Los nombres de las clases siguen la misma jerarquía que WordPress.
- El nombre de la clase debe coincidir con el nombre de archivo.
- Usar funciones públicas para devolver datos a las vistas.
- Usar funciones estáticas públicas para ejecutar el método que devuelve datos a la plantilla.

El Controller

Los nombres de las clases

Siguen la jerarquía de WordPress. Las clases deben llamarse igual cambiando a 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
```

El Controller

Uso de métodos para generar 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'));
    }
}
```

La variable $event-duration ahora es accesible en la vista single-event.blade.php:
{{ $event_duration }}

Atención siempre a los “cases”:
Fichero - ClassName: single-event -> SingleEvent
Método- Variable: eventDuration -> $event_duration

El Controller

Uso de métodos estáticos

```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
```

El Controller

Creando componentes

También puede crear componentes reutilizables e incluirlos en cualquier clase de Controlador utilizando traits de PHP.

```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;
}
```

El Controller

Herencia usando Tree

Por defecto, cada controlador pisa su jerarquía de plantillas dependiendo de la especificidad del controlador (igual que funcionan las plantillas de WordPress).

Se pueden heredar datos de Controladores menos específicos en la jerarquía implementando la Interfaz Tree.

Por ejemplo, el controlador app/Controllers/Single.php heredará los métodos de app/Controllers/Singular.php si el primero implementa Tree.

El Controller

Advanced Custom Fields

El controlador tiene un útil módulo auxiliar para automatizar la transferencia de campos de ACF.

Los campos automatizados usan los nombres de las variables de ACF y los pasan a la vista. El controlador también pasa los valores de las opciones de forma predeterminada.

Los valores se devuelven como objetos, pero se puede modificar para para mantenerlos como arrays.

El 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;
});
```

En resumen

En resumen

- Hay que tener la mente abierta nuevas maneras de trabajar.
- Aunque parece muy distinto a la manera de hacer habitual son todo ventajas.
- La barrera de entrada es mucho menor de lo que parece.
- No hay que usarlo todo todo el tiempo.
- No hace falta saber cómo funcionan las tripas para usarlo.

Enlaces de interés

Enlaces de interés

- 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/