EBook Gratis

APRENDIZAJE silverstripe

Free unaffiliated eBook created from
Stack Overflow contributors.

#silverstripe

Tabla de contenido

Acerca de

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: silverstripe

It is an unofficial and free silverstripe ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official silverstripe.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Capítulo 1: Empezando con silverstripe

Observaciones

Silverstripe es un sistema de gestión de contenido PHP de código abierto. Un desarrollador puede querer usarlo porque

- Licencia BSD - lo que significa que puede ser renombrado como su propia aplicación
- Código limpio orientado a objetos muy fácil de entender y usar, junto con extender y personalizar
- Sencillo y potente motor de plantillas que hace que los temas sean muy fáciles de crear.

Puede usar la mayoría de las bases de datos, principalmente MySQL

Versiones

Versión	Fecha de lanzamiento
3.4.0	2016-06-03

Examples

Instalación

SilverStripe se puede instalar a través del compositor o mediante la extracción del archivo zip descargado.

Para instalar a través del compositor ejecutamos el siguiente comando

```
composer create-project silverstripe/installer /path/to/project 3.4.0
```

Puede encontrar un archivo zip de descarga en la página de descarga del sitio web de SilverStripe. Una vez descargado, este archivo debe extraerse en el directorio raíz del proyecto deseado.

Al visitar el sitio web por primera vez, se presentará un asistente de instalación para configurar y configurar la instalación de SilverStripe.

Personalización del CMS / White Labeling

SilverStripe CMS puede personalizarse para cambiar el logotipo de CMS, el enlace y el nombre de la aplicación.

Esto se puede lograr con la siguiente configuración de `config.yml`

```
LeftAndMain:
  application_name: 'My Application'
  application_link: 'http://www.example.com/'
  extra_requirements_css:
    - mysite/css/cms.css
```

mysite / css / cms.css

```
.ss-loading-screen {
    background: #fff;
}
.ss-loading-screen .loading-logo {
    background: transparent url('../images/my-logo-loading.png') no-repeat 50% 50%;
}
.cms-logo a {
    background: transparent url('../images/my-logo-small.png') no-repeat left center;
}
```

Lea Empezando con silverstripe en línea:
https://riptutorial.com/es/silverstripe/topic/1771/empezando-con-silverstripe

Capítulo 2: Añadir Ons y Módulos

Observaciones

Se recomienda que los complementos y módulos se registren con Packagist, lo que significa que se encuentran y se registran con el repositorio de complementos SilverStripe

Se recomienda la instalación de módulos a través del uso de Composer.

Examples

Módulo de extensiones de campo de cuadrícula SilverStripe

El módulo SilverStripe Grid Field Extensions tiene algunas características muy buenas para mejorar el `GridField` básico ...

- `GridFieldAddExistingSearchButton` : un formulario de búsqueda más avanzado para agregar elementos
- `GridFieldAddNewInlineButton` : se basa en `GridFieldEditableColumns` para permitir la creación de registros en línea.
- `GridFieldAddNewMultiClass` : permite al usuario seleccionar de una lista de clases para crear un nuevo registro desde
- `GridFieldEditableColumns` : permite la edición en línea de registros
- `GridFieldOrderableRows` - arrastra y suelta reordenar filas
- `GridFieldRequestHandler` : una clase de utilidad básica que se puede usar para crear vistas de detalle de campos de cuadrícula personalizadas que incluyen pestañas, `GridFieldRequestHandler` y otras características de CMS
- `GridFieldTitleHeader` : un encabezado simple que muestra los títulos de las columnas

Más documentación se encuentra dentro del módulo aquí .

Mejores botones para GridField

El módulo Better Buttons for GridField agrega nuevas acciones de formulario y botones al formulario de detalle de GridField.

- Guarde y agregue otro: cree un registro y vaya directamente a agregar otro, sin tener que hacer clic en el botón Atrás, y luego vuelva a agregar
- Guardar y cerrar: guarda el registro y vuelve a la vista de lista
- Eliminación fácil de usar: se extrae de la bandeja de acciones constructivas y se aleja, por lo que es menos probable que se haga clic accidentalmente. Incluye confirmación de acción en línea en lugar de un cuadro de alerta del navegador
- Cancelar: igual que el botón Atrás, pero en una ubicación más conveniente
- Registro anterior / siguiente: navega al registro anterior o siguiente en la lista sin volver a la vista de lista

- y muchos más...

Más documentación (e imágenes) en la documentación del módulo.

UserForms

El módulo UserForms permite a los usuarios de CMS crear formularios dinámicos a través de una interfaz de arrastrar y soltar y sin involucrarse en ningún código PHP.

Principales características

- Construya un formulario utilizando todos los campos de formulario principales (texto, correo electrónico, menú desplegable, radio, casilla de verificación ...)
- Capacidad para ampliar las formas de usuario de otros módulos para proporcionar campos adicionales.
- Posibilidad de enviar por correo electrónico a varias personas el envío del formulario
- Ver los envíos enviados y exportarlos a CSV
- Definir mensajes de error personalizados y configuraciones de validación.
- Opcionalmente, muestre y oculte campos usando javascript basado en la entrada de los usuarios
- Muestra un mensaje de confirmación cuando navega lejos de un formulario parcialmente completado

Más enlaces de documentación se pueden encontrar aquí en el repositorio de github

Mostrar la lógica

El módulo Display Logic le permite agregar condiciones para mostrar u ocultar ciertos campos de formulario según el comportamiento del lado del cliente. Este módulo es increíblemente útil para hacer que los formularios sean mucho más profesionales al mostrar solo los campos apropiados y sin agregar mucho JavaScript personalizado.

Ejemplo de uso ...

```
$products->displayIf("HasProducts")->isChecked();

$sizes->hideUnless("ProductType")->isEqualTo("t-shirt")
      ->andIf("Price")->isGreaterThan(10);

$payment->hideIf("Price")->isEqualTo(0);

$shipping->displayIf("ProductType")->isEqualTo("furniture")
           ->andIf()
              ->group()
                ->orIf("RushShipping")->isChecked()
                ->orIf("ShippingAddress")->isNotEmpty()
              ->end();
```

Hay muchos más ejemplos en el módulo readme.md

Menú de CMS agrupado

El Módulo de menú de CMS agrupado le permite agrupar los elementos del menú de CMS en listas anidadas que se expanden cuando se desplaza el mouse sobre. Esto es útil cuando hay tantos elementos del menú del CMS que el espacio en la pantalla se convierte en un problema.

Tablero

El módulo Dashboard proporciona una página de inicio para el CMS en SilverStripe 3 con widgets configurables que muestran información relevante. Los paneles se pueden crear y ampliar fácilmente. El objetivo del módulo Dashboard es proporcionar a los usuarios un launchpad para acciones comunes de CMS, como crear tipos de página específicos o explorar contenido nuevo.

Hay imágenes y videos sobre este módulo que se pueden encontrar en esta publicación del blog .

Hay algunos paneles incluidos por defecto ...

- Páginas recientemente editadas
- Archivos subidos recientemente
- RSS Feed
- Enlaces rápidos
- Editor de secciones
- Google analitico
- Clima

Cuando tiene este módulo instalado, crea un panel de control por miembro, por lo que si tiene una gran cantidad de miembros que nunca usarán el administrador y el rendimiento se convierte en un problema, le recomiendo crear los miembros con estas configuraciones adicionales antes de escribirlo ...

```
Member::create(array(
   'HasConfiguredDashboard' => 1
));
```

Hay mucha más documentación en los módulos readme.md.

Lea Añadir Ons y Módulos en línea: https://riptutorial.com/es/silverstripe/topic/4339/anadir-ons-y-modulos

Capítulo 3: El autoloader

Observaciones

Cuando realice cambios en las clases, deberá ejecutar dev / build? Flush = 1 para reconstruir el manifiesto.

Examples

MyClass.php

```
<?php

class MyClass {
    ...
}

class OtherClass {
    ...
}

?>
```

Silverstripe cargará automáticamente cualquier clase que tenga el mismo nombre que su nombre de archivo.

OtherClass también se cargará porque está en un archivo que se está leyendo.

MyPage.php

```
<?php

class MyPage_Controller extends BookingPage_Controller {
    ...
}

?>
```

Para las funciones del controlador, puede omitir la parte "_Controller" en el nombre del archivo.

Si se debe ignorar un directorio, incluya un archivo llamado "_manifest_exclude"

Lea El autoloader en línea: https://riptutorial.com/es/silverstripe/topic/3817/el-autoloader

Capítulo 4: El sistema de configuración

Observaciones

¿Qué es el sistema de configuración?

SilverStripe utiliza un sistema de configuración global para almacenar configuraciones para las clases y la aplicación. Estas variables de configuración se pueden usar para definir la estructura de los Modelos, la configuración de seguridad en los Controladores o las claves API para servicios de terceros.

Cómo funciona

`Config` valores de `Config` son `SS_ConfigStaticManifest` por el `SS_ConfigStaticManifest` durante un `dev/build` y el `SS_ConfigStaticManifest` caché (añadiendo `?flush` a cualquier URL`) o en la primera ejecución del código de la aplicación.

El `SS_ConfigStaticManifest` escaneará todas las clases de PHP y los archivos de configuración YAML en busca de cualquier valor de configuración y creará un caché de estos valores.

> Cuando realice cambios en la `Config` Configuración a través de YAML o variables `private static` , deberá vaciar la memoria caché para que estos cambios surtan efecto.

Examples

Configuración de valores de configuración

`Config` valores de `Config` se pueden establecer de tres maneras:

1. A través de variables `private static` en cualquier clase dentro de un proyecto SilverStripe
2. A través de los archivos de configuración yaml (almacenados en la carpeta del módulo / _config / [archivo] .yml)
3. A través de PHP en tiempo de ejecución (`Config::inst()->update('Director', 'environment_type', 'dev')`

   En general, es mejor establecer los valores de configuración a través de los primeros 2 métodos, ya que estos se almacenan en caché estáticamente al vaciar el caché.

Entorno con estadísticas privadas.

```
class MyDataObject extends DataObject {
```

```
    private static $db = array(
        'Title' => 'Varchar',
    );

}
```

Todas `private static` variables de clase `private static` en el código de un proyecto SilverStripe (incluidos los módulos, pero no los paquetes en el directorio `vendor/`) se cargarán en la `Config` .

Ajuste con YAML

Puede agregar esto a `mysite/_config/config.yml` (o cualquier otro archivo YAML en esa ruta).

```
Director:
  environment_type: dev
```

El uso de archivos YAML es una excelente manera de anular los valores de `Config` predeterminados para las clases o módulos principales

Ajuste en tiempo de ejecución

Esto se haría normalmente en `mysite/_config.php`

```
Config::inst()->update('Director', 'environment_type', 'dev');
```

La actualización de la `Config` en PHP debe evitarse siempre que sea posible, ya que es más lento que usar los valores almacenados en caché

Lea El sistema de configuración en línea: https://riptutorial.com/es/silverstripe/topic/4699/el-sistema-de-configuracion

Capítulo 5: Extensiones de datos

Examples

Agregando campos a un objeto de datos

Puede usar el mecanismo `DataExtension` para agregar campos de base de datos adicionales a un DataObject existente:

```
class MyMemberExtension extends DataExtension
{
    private static $db = [
        'HairColour' => 'Varchar'
    ];
}
```

Y aplicar la extensión:

```
# File: mysite/_config/app.yml
Member:
  extensions:
    - MyMemberExtension
```

Esto agregará `HairColour` como un campo a los objetos `Member` .

Añadiendo métodos a un objeto de datos

Puede agregar métodos públicos a un objeto de datos usando el mecanismo de extensión, por ejemplo:

```
class MyMemberExtension extends DataExtension
{
    public function getHashId()
    {
        return sha1($this->owner->ID);
    }
}
```

Cuando se aplica a la clase de `Member` , el ejemplo anterior devolvería el hash `sha1` de la ID de `Member` al acceder al `Member` través de la propiedad protegida `$this->owner` . P.ej:

```
$member = Member::get()->byId(123);
var_dump($member->getHashId()); // string(40) "40bd001563085fc35165329ea1ff5c5ecbdbbeef"
```

Aplicando una DataExtension a una clase

La forma más común es aplicar la extensión a través de Config. Ejemplo:

```
# File: mysite/_config/config.yml
Member:
  extensions:
    - MyMemberExtension
```

La variable de configuración de las `extensions` es de tipo "array", por lo que puede agregar varias extensiones como esta:

```
# File: mysite/_config/config.yml
Member:
  extensions:
    - MyMemberExtension
    - MyOtherMemberExtension
```

Si escribió la clase que se ampliará, puede definir la (s) extensión (es) como variable estática:

```
<?php
class MyClass extends DataObject
{
    private static $extensions = ['MyCustomExtension'];
}
```

Lea Extensiones de datos en línea: https://riptutorial.com/es/silverstripe/topic/3519/extensiones-de-datos

Capítulo 6: Formas

Sintaxis

- Form :: create ($ this, __FUNCTION__, $ fields, $ actions, $ validator) // creación de formularios estándar
- Form :: create (...) -> addExtraClass ('my-css-class another-class') // agrega clases CSS a tu formulario
- Form :: create (...) -> loadDataFrom (Member :: get () -> byID (1)); // rellenar un formulario con los detalles de un objeto

Examples

Creando un formulario

Aquí hay un formulario de ejemplo básico con un campo de texto requerido y un botón de envío, que se envía a una función personalizada:

```
class Page_Controller extends ContentController {

    private static $allowed_actions = array(
        'ExampleForm'
    );

    public function ExampleForm() {
        $fields = FieldList::create(
            TextField::create('Name', 'Your Name')
        );

        $actions = FieldList::create(
            FormAction::create('doExampleFormAction', 'Go')
        );

        $requiredFields = RequiredFields::create('Name');

        $form = Form::create(
            $this,
            'ExampleForm',
            $fields,
            $actions,
            $requiredFields
        );

        return $form;
    }

    public function doExampleFormAction($data, Form $form) {
        $form->sessionMessage('Hello '. $data['Name'], 'success');

        return $this->redirectBack();
    }
}
```

Para mostrar este formulario, agregamos `$ExampleForm` a nuestra plantilla de página:

```
$ExampleForm
```

Creando un simple formulario AJAX

SilverStripe tiene un soporte razonablemente bueno para enviar datos de formularios utilizando solicitudes AJAX. A continuación se muestra un código de ejemplo de cómo configurar un formulario básico que acepta envíos tanto de AJAX como del comportamiento predeterminado tradicional del navegador (como es una buena práctica).

Añadiendo el formulario a nuestro controlador.

Primero necesitamos definir nuestra forma; su `Page_Controller` debe verse algo como esto:

```
class Page_Controller extends ContentController {

    /**
     * A list of "actions" (functions) that are allowed to be called from a URL
     *
     * @var array
     * @config
     */
    private static $allowed_actions = array(
        'Form',
        'complete',
    );

    /**
     * A method to return a Form object to display in a template and to accept form
submissions
     *
     * @param $request SS_HTTPRequest
     * @return Form
     */
    public function Form($request) {
        // include our javascript in the page to enable our AJAX behaviour
        Requirements::javascript('framework/thirdparty/jquery/jquery.js');
        Requirements::javascript('mysite/javascript/ajaxforms.js');
        //create the fields we want
        $fields = FieldList::create(
            TextField::create('Name'),
            EmailField::create('Email'),
            TextareaField::create('Message')
        );
        //create the button(s) we want
        $buttons = FieldList::create(
            FormAction::create('doForm', 'Send')
        );
        //add a validator to make sure the fields are submitted with values
        $validator = RequiredFields::create(array(
            'Name',
            'Email',
            'Message',
        ));
        //construct the Form
```

```
        $form = Form::create(
            $this,
            __FUNCTION__,
            $fields,
            $buttons,
            $validator
        );

        return $form;
    }

    /**
     * The form handler, this runs after a form submission has been successfully validated
     *
     * @param $data array RAW form submission data - don't use
     * @param $form Form The form object, populated with data
     * @param $request SS_HTTPRequest The current request object
     */
    public function doForm($data, $form, $request) {
        // discard the default $data because it is raw submitted data
        $data = $form->getData();

        // Do something with the data (eg: email it, save it to the DB, etc

        // send the user back to the "complete" action
        return $this->redirect($this->Link('complete'));
    }

    /**
     * The "complete" action to send users to upon successful submission of the Form.
     *
     * @param $request SS_HTTPRequest The current request object
     * @return string The rendered response
     */
    public function complete($request) {
        //if the request is an ajax request, then only render the include
        if ($request->isAjax()) {
            return $this->renderWith('Form_complete');
        }
        //otherwise, render the full HTML response
        return $this->renderWith(array(
            'Page_complete',
            'Page',
        ));
    }

}
```

Si agrega estas funciones a `Page_Controller` estarán disponibles en **todos los** tipos de página; es posible que esto no sea conveniente y debería considerar si sería más apropiado crear un nuevo tipo de página (como ContactPage) para tener este formulario en

Aquí hemos definido métodos para:

- Crear el `Form`
- Un manejador de formularios (para guardar o enviar los envíos a algún lugar, esto se ejecuta después de que el `Form` haya validado con éxito sus datos)

- Una acción `complete` , a la que se enviará el usuario después de completar con éxito el envío del formulario.

Personalizando plantillas para un fácil reemplazo de contenido

A continuación, debemos configurar nuestras plantillas: modifique su archivo Layout / Page.ss:

```
<% include SideBar %>
<div class="content-container unit size3of4 lastUnit">
    <article>
        <h1>$Title</h1>
        <div class="content">$Content</div>
    </article>
    <div class="form-holder">
        $Form
    </div>
        $CommentsForm
</div>
```

Esto se toma del tema simple predeterminado, con una pequeña adición de que el formulario ahora está envuelto en un `<div class="form-holder">` para que podamos reemplazar fácilmente el formulario con un mensaje de éxito.

También necesitamos crear una plantilla `Layout/Page_complete.ss` - será la misma que la anterior, excepto que el `div form-holder` del `form-holder` será:

```
<div class="form-holder">
    <% include Form_complete %>
</div>
```

A continuación, cree el `Includes/Form_complete` include - es importante usar un include para que podamos representar **solo** esta sección de la página para nuestras respuestas a las solicitudes de AJAX:

```
<h2>Thanks, we've received your form submission!</h2>
<p>We'll be in touch as soon as we can.</p>
```

Creando el oyente de formulario javascript

Finalmente, debemos escribir nuestro javascript para enviar el formulario por AJAX en lugar del comportamiento predeterminado del navegador (colóquelo en mysite / javascript / ajaxform.js):

```
(function($) {
    $(window).on('submit', '.js-ajax-form', function(e) {
        var $form = $(this);
        var formData = $form.serialize();
        var formAction = $form.prop('action');
        var formMethod = $form.prop('method');
```

```
        var encType = $form.prop('enctype');

        $.ajax({
            beforeSend: function(jqXHR,settings) {
                if ($form.prop('isSending')) {
                    return false;
                }
                $form.prop('isSending',true);
            },
            complete: function(jqXHR,textStatus) {
                $form.prop('isSending',false);
            },
            contentType: encType,
            data: formData,
            error: function(jqXHR, textStatus, errorThrown) {
                window.location = window.location;
            },
            success: function(data, textStatus, jqXHR) {
                var $holder = $form.parent();
                $holder.fadeOut('normal',function() {
                    $holder.html(data).fadeIn();
                });
            },
            type: formMethod,
            url: formAction
        });
        e.preventDefault();
    });
})(jQuery);
```

Este javascript enviará el formulario utilizando AJAX y, al completarlo, desvanecerá el formulario, lo reemplazará con la respuesta y volverá a desvanecerse.

Para usuarios avanzados:

Con este ejemplo, todas las formas en su sitio serán "anexadas", esto puede ser aceptable, pero a veces necesita cierto control sobre esto (por ejemplo, las formas de búsqueda no funcionarían así). En su lugar, puede modificar el código ligeramente para buscar solo formularios con una determinada clase.

`Page_Controller` el método de `Form` en `Page_Controller` manera:

```
public function Form() {
    ...
    $form->addExtraClass('js-ajax-form');
    return $form;
}
```

Modifique el javascript como tal:

```
$(window).on('submit', '.js-ajax-form', function(e) {
    ...
})(jQuery);
```

Solo las formas con la clase `js-ajax-form` ahora actuarán de esta manera.

Lea Formas en línea: https://riptutorial.com/es/silverstripe/topic/4126/formas

Capítulo 7: LeftAndMain

Introducción

`LeftAndMain` es más una API de nivel inferior y no suele ser necesaria debido a la existencia de `ModelAdmin` . Sin embargo, si desea crear una interfaz de usuario personalizada que no requiera necesariamente la funcionalidad de `ModelAdmin` en el panel de administración de su módulo, entonces `LeftAndMain` es el lugar donde desea comenzar.

Examples

1. Empezando

El objetivo de esta guía es comenzar a crear su propia Interfaz de usuario subclasificando la clase `LeftAndMain` .

Al final de esta guía, habrá creado su primera interfaz `Hello World` en el Panel de administración.

Requerimientos

Esta guía requiere que tenga al menos la versión `3.*` del marco Y CMS, pero menor que la versión `4.*` .

Si desea utilizar esta guía, deberá intercambiar las referencias de clase con el Nombre de clase de calidad total (FQCN) tal como se define en la guía de actualización de SS4.

Preparación

> **tl; dr** Ignora los siguientes pasos y simplemente crea la estructura debajo de ellos

1. Cree una carpeta con el nombre de cualquier cosa que elija en el directorio raíz para su proyecto SilverStripe, para este ejemplo `/helloworld/` y crearemos un archivo vacío dentro de esa carpeta llamada `_config.php` . Se `_config.php` un `_config.php` como mínimo en cada directorio de módulos para que SilverStripe detecte su existencia.

2. Dentro de su nueva carpeta, cree una subcarpeta llamada exactamente `/code/` y dentro de esa carpeta, para propósitos de organización; crear otra carpeta llamada `/admin/`

3. Cree `/helloworld/code/admin/HelloWorldLeftAndMain.php` y coloque el siguiente código en él por ahora.

```
class HelloWorldLeftAndMain extends LeftAndMain {
```

```
}
```

4. Cree el archivo de plantilla que se utilizará con esta clase llamada `/helloworld/templates/Includes/HelloWorldLeftAndMain.ss`

Estructura

```
/framework/
/cms/
/helloworld/
    + _config.php
    + /code/
        + /admin/
            + /HelloWorldLeftAndMain.php
    + /templates/
        + /Includes/
            + /HelloWorldLeftAndMain_Content.ss
```

2. Configurando HelloWorldLeftAndMain.php

Si aún no lo has hecho, simplemente inicia este archivo con:

```
class HelloWorldLeftAndMain extends LeftAndMain {

}
```

Configurar

Lo primero que debe hacer es definir el `$url_segment` que se usará para acceder a la interfaz y el título (`$menu_title`) que aparecerá en el menú de navegación lateral del panel de administración:

```
private static $url_segment = 'helloworld';
private static $menu_title  = 'Hello World';
```

Las siguientes variables de configuración son opcionales y no se utilizan en esta guía:

```
private static $menu_icon   = 'helloworld/path/to/my/icon.png';
private static $url_rule    = '/$Action/$ID/$OtherID';
```

Añadiendo hojas de estilo y Javascript

`LeftAndMain` permite anular el método `init` en su padre, podemos usar esto para requerir archivos específicos para nuestra interfaz. Sin lugar a dudas, siempre debe necesitar una hoja de estilo CSS que estilice los elementos para su interfaz de usuario.

Como sugerencia, se recomienda no confiar nunca en las clases de CSS proporcionadas por el

CMS, ya que están sujetas a cambios sin previo aviso y, posteriormente, destruirán el Look & Feel de su UI (por ejemplo, `3.*` a `4.*` ha visto un cambio completo de la interfaz, por lo tanto, cualquier clase de CSS en la que confíe en `3.*` debe ser rediseñado para su conversión a `4.*`)

Así que vamos a agregar nuestro archivo `helloworld/css/styles.css` :

```
public function init() {
    parent::init();

    Requirements::css('helloworld/css/styles.css');
    //Requirements::javascript('helloworld/javascript/script.min.js');
}
```

No necesitamos ninguna funcionalidad de Javascript para este ejemplo, pero en lo anterior he incluido cómo se lograría agregar un archivo a Javascript usando la clase de Requisitos .

Después de lo cual puede adoptar lo que ha estado acostumbrado cuando trata con `Page_Controller` , como `$allowed_actions` etc.

> **NO SE PUEDE** anular el `index()` .

En su lugar, el `index()` se asume como `HelloWorldLeftAndMain_Content.ss` y, a partir de ahí, es necesario tratar con la visualización de los índices a través de las funciones de la plantilla (consulte el ejemplo a continuación)

Código Completo

```
class HelloWorldLeftAndMain extends LeftAndMain {
    private static $url_segment = 'helloworld';
    private static $menu_title  = 'Hello World';
    private static $allowed_actions = array(
        'some_action'
    );

    public function init() {
        parent::init();

        Requirements::css('helloworld/css/styles.css');
        //Requirements::javascript('helloworld/javascript/script.min.js');
    }

    public function Hello($who=null) {
        if (!$who) {
            $who = 'World';
        }

        return "Hello " . htmlentities($who);
    }
}
```

3. La plantilla (HelloWorldLeftAndMain_Content.ss)

La estructura esperada de esta plantilla puede ser un poco complicada, pero todo se reduce a esto:

1. **Hay 3 secciones que vale la pena destacar para esta guía:**

    - `.north`
    - `.center`
    - `.south`

2. Debe estar completamente envuelto dentro de un elemento que tenga el `data-pjax-fragment="Content"` . Esto es para que las llamadas AJAX generadas desde el sidemenu, sepan dónde está el "Contenido" para que pueda mostrarlo adecuadamente:

```
<div class="cms-content center $BaseCSSClasses" data-layout-type="border" data-pjax-fragment="Content">

</div>
```

No voy a entrar en detalles sobre la funcionalidad de la plantilla. He incluido comentarios donde es relevante, pero no debería leer esta guía si no comprende la sintaxis de la plantilla para SilverStripe.

Código Completo

Lo único desde abajo; de lo que debería esperar que `<% include CMSBreadcrumbs %>` ya con estilo es `<% include CMSBreadcrumbs %>` todo lo demás que debe abastecerse en el archivo CSS que se incluyó anteriormente

```
<div class="cms-content center $BaseCSSClasses" data-layout-type="border" data-pjax-fragment="Content">
    <%-- This will add the breadcrumb that you see on every other menu item --%>
    <div class="cms-content-header north">
        <div class="cms-content-header-info">
            <% include CMSBreadcrumbs %>
        </div>
    </div>

    <div class="center">
        <%-- Our function in HelloWorldLeftAndMain.php --%>
        $Hello('USER');
        <%-- ^ outputs "Hello USER" --%>
    </div>

    <div class='south'>
        Some footer-worthy content
    </div>
</div>
```

Ahora, todo lo que queda por hacer es para usted en `/dev/build` y `?flush=1` entonces puede revisar

nuestro pequeño módulo inútil en el Panel de Administración!

Lea LeftAndMain en línea: https://riptutorial.com/es/silverstripe/topic/8300/leftandmain

Capítulo 8: ModelAdmin

Examples

Ejemplo simple

Dada una sencilla `DataObject` como esto:

```
class MyDataObject extends DataObject {
    private static $db = array(
        'Name' => 'Varchar(255)'
    );
}
```

Para proporcionar un completo Crear-Leer-Actualizar-Eliminar para los objetos, este es el código `ModelAdmin` requerido:

```
class MyModelAdmin extends ModelAdmin {
    private static $mangaged_models = array(
        'MyDataObject'
    );
    private static $url_segment = 'my-model-admin';
    private static $menu_title = 'My Model Admin';
    private static $menu_icon = 'mysite/images/treeicons/my-model-admin.png';
    private static $menu_priority = 9;
}
```

Controla el nombre de objeto de datos que se muestra en la interfaz de usuario

```
class MyDataObject extends DataObject {

    private static $singular_name = 'My Object';
    private static $plural_name = 'My Objects';

    ...
}
```

Los DataObjects se pueden ordenar por defecto

```
class SortDataObject extends DataObject {

    private static $db = array(
        'Name' => 'Varchar',
        'SortOrder' => 'Int'
    );

    private static $default_sort = 'SortOrder DESC';
}
```

Columnas de control mostradas para el objeto de datos

```
class MyDataObject extends DataObject {

    private static $db = array(
        'Name' => 'Varchar'
    );

    private static $has_one = array(
        'OtherDataObject' => 'OtherDataObject'
    );

    private static $summary_fields = array(
        'Name',
        'OtherDataObject.Name'
    );

    private static $field_labels = array(
        'OtherDataObject.Name' => 'Other Data Object'
    );
}
```

ModelAdmin utiliza el campo `summary_fields` para generar las columnas que muestra. Para especificar el nombre de la columna, use `field_labels` como se muestra.

Uso de searchable_fields para controlar los filtros para ese objeto en ModelAdmin

```
class MyDataObject extends DataObject {

    private static $db = array(
        'Name' => 'Varchar'
    );

    private static $has_one = array(
        'OtherDataObject' => 'OtherDataObject'
    );

    private static $summary_fields = array(
        'Name',
        'OtherDataObject.Name'
    );

    private static $searchable_fields = array(
        'Name',
        'OtherDataObjectID' => array(
            'title' => 'Other Data Object'
        )
    );

}
```

Tenga en cuenta el `OtherDataObjectID` que convierte un campo de texto en un menú desplegable del objeto relacionado para filtrar.

Eliminar GridField scaffolded para las relaciones

```
class MyDataObject extends DataObject {

    ...

    private static $has_many = array(
        'OtherDataObjects' => 'OtherDataObject'
    );

    function getCMSFields() {
        $fields = parent::getCMSFields();

        if ($gridField = $fields->dataFieldByName('OtherDataObjects')) {
            $gridField->getConfig()
                ->removeComponentsByType('GridFieldExportButton');
        }

        return $fields;
    }
}
```

Para eliminar el botón de exportación de ModelAdmin

```
class MyAdmin extends ModelAdmin {

    ...

    function getEditForm($id = null, $fields = null) {
        $form = parent::getEditForm($id, $fields);

        if ($this->modelClass == 'MyDataObjectName') {
            $form->Fields()
                ->fieldByName($this->sanitiseClassName($this->modelClass))
                ->getConfig()
                ->removeComponentsByType('GridFieldExportButton');
        }
        return $form;
    }
}
```

Lea ModelAdmin en línea: https://riptutorial.com/es/silverstripe/topic/3836/modeladmin

Capítulo 9: Usando el ORM

Examples

Leer y escribir DataObjects

Los DataObjects en SilverStripe representan una fila de tabla de base de datos. Los campos en el modelo tienen métodos mágicos que manejan la obtención y configuración de datos a través de sus nombres de propiedades.

Dado que tenemos un simple objeto de datos como ejemplo:

```
class Fruit extends DataObject
{
    private static $db = ['Name' => 'Varchar'];
}
```

Puede crear, configurar datos y escribir una `Fruit` siguiente manera:

```
$apple = Fruit::create();
$apple->Name = 'Apple';
$apple->write();
```

De manera similar, puede recuperar el objeto `Fruit` siguiente manera:

```
$apple = Fruit::get()->filter('Name', 'Apple')->first();
var_dump($apple->Name); // string(5) "Apple"
```

Lea Usando el ORM en línea: https://riptutorial.com/es/silverstripe/topic/3463/usando-el-orm

Creditos

S. No	Capítulos	Contributors
1	Empezando con silverstripe	3dgoo, Barry, Community, zanderwar
2	Añadir Ons y Módulos	Barry
3	El autoloader	Barry
4	El sistema de configuración	Dan Hensby
5	Extensiones de datos	bummzack, Dan Hensby, Robbie Averill
6	Formas	3dgoo, Dan Hensby
7	LeftAndMain	zanderwar
8	ModelAdmin	3dgoo, Barry, Turnerj
9	Usando el ORM	3dgoo, bummzack, Robbie Averill