EBook Gratuito

APPRENDIMENTO silverstripe

Free unaffiliated eBook created from
Stack Overflow contributors.

#silverstripe

Sommario

Di

Capitolo 1: Iniziare con Silverstripe

Osservazioni

Silverstripe è un sistema di gestione dei contenuti PHP open source. Uno sviluppatore potrebbe volerlo usare perché

- Licenza BSD - significa che può essere rinominato come la propria applicazione
- Pulisci il codice orientato agli oggetti molto facile da capire e da usare, oltre a estendere e personalizzare
- Motore di template semplice e potente che crea temi molto facili da creare

Può utilizzare la maggior parte dei database, principalmente MySQL

Versioni

Versione	Data di rilascio
3.4.0	2016/06/03

Examples

Installazione

SilverStripe può essere installato tramite compositore o tramite l'estrazione del file zip scaricato.

Per installare attraverso il compositore, eseguiamo il seguente comando

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

Un file zip di download può essere trovato nella pagina di download del sito Web SilverStripe. Una volta scaricato, questo file deve essere estratto nella directory principale del progetto desiderato.

Visitando il sito Web per la prima volta verrà presentata una procedura guidata di installazione per configurare e configurare l'installazione di SilverStripe.

Personalizzazione del CMS / White Labelling

SilverStripe CMS può essere personalizzato per modificare il logo CMS, il link e il nome dell'applicazione.

Questo può essere ottenuto con le seguenti impostazioni di `config.yml`

```
LeftAndMain:
  application_name: 'My Application'
```

```
  application_link: 'http://www.example.com/'
  extra_requirements_css:
    - mysite/css/cms.css
```

miosito / 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;
}
```

Leggi Iniziare con Silverstripe online: https://riptutorial.com/it/silverstripe/topic/1771/iniziare-con-silverstripe

Capitolo 2: Aggiungi on e moduli

Osservazioni

I componenti aggiuntivi e i moduli sono incoraggiati a essere registrati con Packagist, il che significa che sono stati trovati e registrati con il repository aggiuntivo SilverStripe

L'installazione dei moduli è consigliata tramite l' uso di Composer

Examples

Modulo Estensioni Campo Griglia SilverStripe

Il modulo SilverStripe Grid Field Extensions ha alcune funzioni molto carine per migliorare il `GridField` base ...

- `GridFieldAddExistingSearchButton` - un modulo di ricerca più avanzato per aggiungere elementi
- `GridFieldAddNewInlineButton` - si basa su `GridFieldEditableColumns` per consentire la creazione in linea di record.
- `GridFieldAddNewMultiClass` : consente all'utente di selezionare da un elenco di classi per creare un nuovo record da
- `GridFieldEditableColumns` : consente la modifica in linea dei record
- `GridFieldOrderableRows` : trascina e riordina le righe
- `GridFieldRequestHandler` - una classe di utilità di base che può essere utilizzata per creare viste dettagliate del campo della griglia personalizzate, tra cui schede, breadcrumb e altre funzionalità CMS
- `GridFieldTitleHeader` - una semplice intestazione che mostra i titoli delle colonne

Più documentazione si trova all'interno del modulo qui .

Migliori pulsanti per GridField

Il modulo Better Buttons for GridField aggiunge nuove azioni modulo e pulsanti al modulo di dettaglio GridField.

- Salva e aggiungi un altro: crea un record e vai direttamente ad aggiungerne un altro, senza dover fare clic sul pulsante Indietro, quindi aggiungilo di nuovo
- Salva e chiudi: salva il record e torna alla visualizzazione elenco
- Eliminazione intuitiva: estratta dal vassoio di azioni costruttive e spostata, quindi è meno probabile che venga cliccato accidentalmente. Include la conferma dell'azione in linea anziché la casella di avviso del browser
- Annulla: come il pulsante Indietro, ma in una posizione più comoda
- Record precedente / successivo: passa al record precedente o successivo nell'elenco senza tornare alla visualizzazione elenco

- e tanti altri...

Più documentazione (e immagini) sulla documentazione per il modulo

form

Il modulo UserForms consente agli utenti CMS di creare moduli dinamici tramite un'interfaccia drag and drop e senza essere coinvolti in alcun codice PHP.

Caratteristiche principali

- Costruisci un modulo utilizzando tutti i principali campi del modulo (testo, email, menu a discesa, radio, casella di controllo ..)
- Possibilità di estendere le forme utente da altri moduli per fornire campi aggiuntivi.
- Possibilità di inviare tramite email a più persone l'invio del modulo
- Visualizza le richieste inviate ed esportale in CSV
- Definisci i messaggi di errore personalizzati e le impostazioni di convalida
- Opzionalmente mostra e nasconde i campi usando javascript in base all'input degli utenti
- Visualizza un messaggio di conferma quando si naviga lontano da un modulo parzialmente completato

Ulteriori collegamenti alla documentazione possono essere trovati qui nel repository github

Visualizza logica

Il modulo Logica display consente di aggiungere condizioni per visualizzare o nascondere determinati campi modulo in base al comportamento lato client. Questo modulo è incredibilmente utile per rendere le forme molto più professionali mostrando solo i campi appropriati e senza aggiungere molti JavaScript personalizzati.

Esempio di utilizzo ...

```
$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();
```

Ci sono molti altri esempi sul modulo readme.md

Menu CMS raggruppato

Il modulo menu CMS raggruppato consente di raggruppare gli elementi del menu CMS in elenchi

annidati che si espandono al passaggio del mouse. Questo è utile quando ci sono così tante voci del menu CMS che lo spazio sullo schermo diventa un problema.

Cruscotto

Il modulo Dashboard fornisce una pagina iniziale per il CMS in SilverStripe 3 con widget configurabili che visualizzano informazioni pertinenti. I pannelli possono essere creati ed estesi facilmente. L'obiettivo del modulo Dashboard è fornire agli utenti un launchpad per azioni CMS comuni, come la creazione di tipi di pagine specifici o la navigazione di nuovi contenuti.

Ci sono immagini e video su questo modulo possono essere trovati in questo post del blog .

Ci sono alcuni pannelli inclusi di default ...

- Pagine modificate di recente
- File caricati di recente
- RSS Feed
- Link veloci
- Editor di sezione
- statistiche di Google
- Tempo metereologico

Quando hai installato questo modulo, crea un dashboard per membro, quindi se hai una grande quantità di membri che non useranno mai l'amministratore e le prestazioni diventano un problema, ti consiglio di creare i membri con queste impostazioni extra prima di scriverlo ...

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

C'è molta più documentazione nei moduli readme.md

Leggi Aggiungi on e moduli online: https://riptutorial.com/it/silverstripe/topic/4339/aggiungi-on-e-moduli

Capitolo 3: DataExtensions

Examples

Aggiunta di campi a un oggetto DataObject

È possibile utilizzare il meccanismo `DataExtension` per aggiungere campi di database aggiuntivi a un DataObject esistente:

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

E applica l'estensione:

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

Questo aggiungerà `HairColour` come un campo `Member` oggetti `Member` .

Aggiunta di metodi a un oggetto DataObject

È possibile aggiungere metodi pubblici a un DataObject utilizzando il meccanismo di estensione, ad esempio:

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

Se applicato alla classe `Member` , l'esempio precedente restituirebbe l'hash `sha1` dell'ID `Member` accedendo al `Member` tramite la proprietà protetta `$this->owner` . Per esempio:

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

Applicazione di un DataExtension a una classe

Il modo più comune è di applicare l'estensione tramite Config. Esempio:

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

La variabile di configurazione delle `extensions` è di tipo "array", quindi puoi aggiungere più estensioni come questa:

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

Se hai scritto la classe che deve essere estesa, puoi definire l'estensione (s) come variabile statica:

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

Leggi DataExtensions online: https://riptutorial.com/it/silverstripe/topic/3519/dataextensions

Capitolo 4: Il caricatore automatico

Osservazioni

Quando si apportano modifiche alle classi, è necessario eseguire un dev / build? Flush = 1 per ricostruire il manifest.

Examples

MyClass.php

```
<?php

class MyClass {
    ...
}

class OtherClass {
    ...
}

?>
```

Qualsiasi classe che abbia lo stesso nome del suo nome verrà caricata automaticamente da Silverstripe.

Anche OtherClass verrà caricato perché si trova in un file che viene letto.

MyPage.php

```
<?php

class MyPage_Controller extends BookingPage_Controller {
    ...
}

?>
```

Per le funzioni del controller è possibile omettere la parte "_Controller" nel nome del file.

Se una directory deve essere ignorata, includere un file denominato "_manifest_exclude"

Leggi Il caricatore automatico online: https://riptutorial.com/it/silverstripe/topic/3817/il-caricatore-automatico

Capitolo 5: Il sistema di configurazione

Osservazioni

Qual è il sistema di configurazione

SilverStripe utilizza un sistema di configurazione globale per memorizzare le impostazioni per le classi e l'applicazione. Queste variabili di configurazione possono essere utilizzate per definire la struttura di Modelli, le impostazioni di sicurezza sui controller o le chiavi API per i servizi di terze parti.

Come funziona

`Config` valori di `Config` sono popolati da `SS_ConfigStaticManifest` durante un `dev/build` e cache flush (aggiungendo `?flush` a qualsiasi URL") o alla prima esecuzione del codice dell'applicazione.

`SS_ConfigStaticManifest` esegue la scansione di tutte le classi PHP e dei file di configurazione YAML per tutti i valori di configurazione e crea una cache di questi valori.

> Quando si apportano modifiche alle impostazioni di `Config` tramite YAML o variabili `private static` , è necessario svuotare la cache affinché queste modifiche abbiano effetto.

Examples

Impostazione dei valori di configurazione

`Config` valori di `Config` possono essere impostati in tre modi:

1. Tramite variabili `private static` su qualsiasi classe all'interno di un progetto SilverStripe
2. Tramite i file di configurazione yaml (memorizzati in module-folder / _config / [file] .yml)
3. Via PHP in fase di esecuzione (`Config::inst()->update('Director', 'environment_type', 'dev')`

   Generalmente è meglio impostare i valori di configurazione tramite i primi 2 metodi poiché questi vengono staticamente memorizzati nella cache durante il lavaggio della cache.

Impostazione con statica privata

```
class MyDataObject extends DataObject {
```

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

}
```

Tutte `private static` variabili di classe `private static` codice di un progetto SilverStripe (inclusi i moduli, ma non i pacchetti nella directory del `vendor/`) verranno caricati nella `Config` .

Impostazione con YAML

Puoi aggiungerlo a `mysite/_config/config.yml` (o qualsiasi altro file YAML in quel percorso).

```
Director:
  environment_type: dev
```

L'utilizzo dei file YAML è un ottimo modo per sostituire i valori di `Config` predefiniti per le classi o i moduli principali

Impostazione in fase di esecuzione

Questo in genere dovrebbe essere fatto in `mysite/_config.php`

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

L'aggiornamento della `Config` in PHP dovrebbe essere evitato laddove possibile poiché è più lento rispetto all'utilizzo dei valori memorizzati nella cache

Leggi Il sistema di configurazione online: https://riptutorial.com/it/silverstripe/topic/4699/il-sistema-di-configurazione

Capitolo 6: Le forme

Sintassi

- Form :: create ($ this, __FUNCTION__, $ fields, $ actions, $ validator) // creazione di moduli standard
- Form :: create (...) -> addExtraClass ('my-css-class another-class') // aggiungi classi CSS al tuo Form
- Modulo :: create (...) -> loadDataFrom (Stati :: get () -> byID (1)); // compila un modulo con i dettagli di un oggetto

Examples

Creare un modulo

Ecco un modulo di esempio di base con un campo di testo richiesto e un pulsante di invio, che viene inviato a una funzione personalizzata:

```
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();
    }
}
```

Per visualizzare questo modulo aggiungiamo `$ExampleForm` al nostro modello di pagina:

```
$ExampleForm
```

Creare un semplice modulo AJAX

SilverStripe ha un discreto supporto per l'invio dei dati dei moduli utilizzando le richieste AJAX. Di seguito è riportato un codice di esempio su come impostare un modulo di base che accetti gli invii sia da AJAX sia dal comportamento predefinito del browser predefinito (come è buona pratica).

Aggiunta del modulo al nostro controller

Per prima cosa dobbiamo definire la nostra forma; il tuo `Page_Controller` dovrebbe assomigliare a questo:

```
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',
        ));
    }

}
```

L'aggiunta di queste funzioni a `Page_Controller` le renderà disponibili su **tutti i** tipi di pagina - questo potrebbe non essere desiderato e dovresti considerare se sarebbe più appropriato creare un nuovo tipo di pagina (come ContactPage) per avere questo modulo su

Qui abbiamo definito i metodi per:

- Crea il `Form`
- Un gestore di moduli (per salvare o inviare gli invii da qualche parte, questo viene eseguito dopo che il `Form` ha convalidato con successo i suoi dati)

- Un'azione `complete` , a cui l'utente verrà inviato dopo aver completato con successo l'invio del modulo.

Personalizzazione dei modelli per una facile sostituzione dei contenuti

Quindi abbiamo bisogno di impostare i nostri modelli - modificare il file 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>
```

Questo è preso dal tema semplice predefinito, con un'aggiunta minore che il modulo è ora racchiuso in un `<div class="form-holder">` modo che possiamo facilmente sostituire il modulo con un messaggio di successo.

Abbiamo anche bisogno di creare un modello `Layout/Page_complete.ss` - questo sarà lo stesso di sopra tranne che il `div` del `form-holder` sarà:

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

Quindi crea l' `Includes/Form_complete` : è importante utilizzare un include in modo che possiamo eseguire il rendering **solo di** questa sezione della pagina per le nostre risposte alle richieste AJAX:

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

Creazione del listener di moduli javascript

Infine, dobbiamo scrivere il nostro javascript per inviare il modulo da AJAX al posto del comportamento predefinito del browser (posizionalo in 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);
```

Questo javascript invierà il modulo utilizzando AJAX e al completamento svanirà il modulo e lo sostituirà con la risposta e lo dissolverà nuovamente.

Per utenti esperti:

Con questo esempio tutti i moduli sul tuo sito saranno "ajaxificati", questo può essere accettabile, ma a volte hai bisogno di un controllo su questo (per esempio, i moduli di ricerca non funzionerebbero bene come questo). Invece, è possibile modificare leggermente il codice per cercare solo i moduli con una determinata classe.

Modificare il metodo `Form` su `Page_Controller` in `Page_Controller` modo:

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

Modificare il javascript in questo modo:

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

Solo i moduli con la classe `js-ajax-form` ora agiranno in questo modo.

Leggi Le forme online: https://riptutorial.com/it/silverstripe/topic/4126/le-forme

Capitolo 7: LeftAndMain

introduzione

`LeftAndMain` è più di un'API di livello inferiore e non è spesso necessaria a causa dell'esistenza di `ModelAdmin` . Tuttavia, se si desidera creare un'interfaccia utente personalizzata che non richieda necessariamente la funzionalità di `ModelAdmin` nel pannello di amministrazione del modulo, `LeftAndMain` è il punto da cui si desidera iniziare.

Examples

1. Per iniziare

Questa guida ha lo scopo di iniziare a creare la tua interfaccia utente sottoclassi della classe `LeftAndMain` .

Alla fine di questa guida, avrai creato la tua prima interfaccia `Hello World` nel pannello di amministrazione.

Requisiti

Questa guida richiede di avere almeno la versione `3.*` del framework AND CMS ma inferiore alla versione `4.*` .

Se si desidera utilizzare questa guida, sarà necessario sostituire tutti i riferimenti di classe con il Nome classe di qualità completa (FQCN) come definito nella guida all'upgrade di SS4.

Preparazione

> **tl; dr** Ignora i seguenti passaggi e crea semplicemente la struttura sotto di loro

1. Crea una cartella con il nome di qualsiasi cosa tu scelga nella directory radice per il tuo progetto SilverStripe, per questo esempio useremo `/helloworld/` e creeremo un file vuoto all'interno di quella cartella chiamata `_config.php` . Un `_config.php` al minimo è richiesto in ogni directory del modulo per SilverStripe per rilevarne l'esistenza.
2. All'interno della tua nuova cartella, crea una sottocartella chiamata esattamente `/code/` e all'interno di quella cartella, per scopi organizzativi; crea un'altra cartella chiamata `/admin/`
3. Crea `/helloworld/code/admin/HelloWorldLeftAndMain.php` e inserisci il seguente codice per ora.

```
class HelloWorldLeftAndMain extends LeftAndMain {
```

```
}
```

4. Crea il file modello che verrà utilizzato con questa classe chiamata `/helloworld/templates/Includes/HelloWorldLeftAndMain.ss`

Struttura

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

2. Configurazione di HelloWorldLeftAndMain.php

Se non lo hai già fatto, puoi semplicemente avviare questo file con:

```
class HelloWorldLeftAndMain extends LeftAndMain {

}
```

Configurazione

La prima cosa che dovresti fare è definire `$url_segment` che sarà usato per accedere all'interfaccia e il titolo (`$menu_title`) che apparirà nel menu di navigazione laterale del pannello di amministrazione:

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

Le seguenti variabili di configurazione sono facoltative e non utilizzate in questa guida:

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

Aggiunta di fogli di stile e Javascript

`LeftAndMain` ti permette di sovrascrivere il metodo `init` nel suo genitore, possiamo usarlo per richiedere file specifici per la nostra interfaccia. Indubbiamente dovresti sempre aver bisogno di un foglio di stile CSS che modifichi gli elementi per la tua interfaccia utente.

Come consiglio, è consigliabile non fare affidamento sulle classi CSS fornite dal CMS in quanto queste sono soggette a modifiche senza preavviso e distruggeranno successivamente il look & feel della tua interfaccia utente (ad esempio, `3.*` a `4.*` ha visto un completa restituzione dell'interfaccia quindi tutte le classi CSS utilizzate in `3.*` devono essere sottoposte a restyling per la conversione in `4.*`)

Quindi aggiungiamo il nostro file `helloworld/css/styles.css` :

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

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

Non abbiamo bisogno di alcuna funzionalità Javascript per questo esempio ma in quanto sopra ho incluso come si potrebbe ottenere l'aggiunta di un file Javascript usando la classe Requirements .

Dopo di che puoi adottare ciò a cui sei abituato quando si tratta di `Page_Controller` come `$allowed_actions` ecc con una differenza notevole, tuttavia

> **NON PUO '** sovrascrivere `index()` .

Invece `index()` è assunto come `HelloWorldLeftAndMain_Content.ss` e da lì, è necessario occuparsi della visualizzazione degli indici tramite le funzioni del modello (vedi esempio sotto)

Codice 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. Il modello (HelloWorldLeftAndMain_Content.ss)

La struttura prevista di questo modello può essere un po 'complicata, ma tutto si riduce a questo:

1. **Ci sono 3 sezioni che vale la pena notare per questa guida:**

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

2. Deve essere racchiuso interamente all'interno di un elemento che ha l' `data-pjax-fragment="Content"` . Questo è così le chiamate AJAX generate dal sidemenu, sapere dove è il "Contenuto" in modo che possa visualizzarlo in modo appropriato:

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

</div>
```

Non entrerò nei dettagli sulla funzionalità dei template, ho incluso commenti dove rilevanti ma non dovresti leggere questa guida se non capisci la sintassi dei template per SilverStripe

Codice completo

L'unica cosa da sotto; che dovresti aspettarti di uscire già in stile è il `<% include CMSBreadcrumbs %>` tutto il resto devi provvedere da solo nel file CSS che è stato incluso in precedenza

```
<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>
```

Ora tutto ciò che resta da fare è per voi di `/dev/build` e `?flush=1` quindi potete controllare il nostro piccolo modulo inutile nel pannello di amministrazione!

Leggi LeftAndMain online: https://riptutorial.com/it/silverstripe/topic/8300/leftandmain

Capitolo 8: ModelAdmin

Examples

Semplice esempio

Dato un semplice oggetto `DataObject` come questo:

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

Per fornire una `ModelAdmin` completa di Read-Update-Delete per gli oggetti, questo è il codice `ModelAdmin` richiesto:

```
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;
}
```

Controlla il nome DataObject visualizzato nell'interfaccia utente

```
class MyDataObject extends DataObject {

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

    ...
}
```

DataObjects può essere ordinato per impostazione predefinita

```
class SortDataObject extends DataObject {

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

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

Colonne di controllo visualizzate per DataObject

```
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 utilizza `summary_fields` per generare le colonne visualizzate. Per specificare il nome della colonna, utilizzare `field_labels` come mostrato.

Usando searchable_fields per controllare i filtri per quell'oggetto in 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'
        )
    );

}
```

Notare `OtherDataObjectID` che converte un campo di testo in un `OtherDataObjectID` a tendina dell'oggetto correlato con cui filtrare.

Rimuovi il GridField impalcato per le relazioni

```
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;
    }
}
```

Per rimuovere il pulsante di esportazione da 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;
    }
}
```

Leggi ModelAdmin online: https://riptutorial.com/it/silverstripe/topic/3836/modeladmin

Capitolo 9: Utilizzando l'ORM

Examples

Lettura e scrittura di DataObjects

DataObjects in SilverStripe rappresentano una riga della tabella del database. I campi nel modello hanno metodi magici che gestiscono il recupero e l'impostazione dei dati tramite i loro nomi di proprietà.

Dato che abbiamo un semplice DataObject come esempio:

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

Puoi creare, impostare i dati e scrivere un `Fruit` come segue:

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

È possibile recuperare l'oggetto `Fruit` modo simile come segue:

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

Leggi Utilizzando l'ORM online: https://riptutorial.com/it/silverstripe/topic/3463/utilizzando-l-orm

Titoli di coda

S. No	Capitoli	Contributors
1	Iniziare con Silverstripe	3dgoo, Barry, Community, zanderwar
2	Aggiungi on e moduli	Barry
3	DataExtensions	bummzack, Dan Hensby, Robbie Averill
4	Il caricatore automatico	Barry
5	Il sistema di configurazione	Dan Hensby
6	Le forme	3dgoo, Dan Hensby
7	LeftAndMain	zanderwar
8	ModelAdmin	3dgoo, Barry, Turnerj
9	Utilizzando l'ORM	3dgoo, bummzack, Robbie Averill