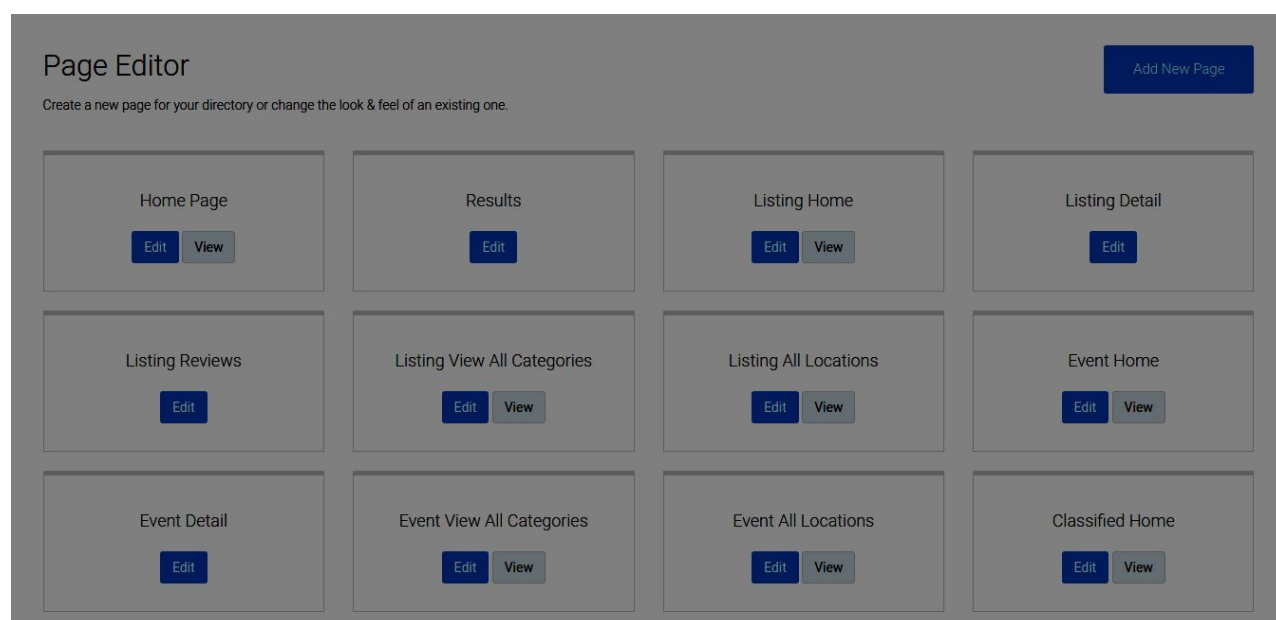


eDirectory 11.2
Editor de Paginas

Introducción

Editor de páginas es la nueva herramienta disponible para NDS administrador del sitio para controlar la apariencia de las páginas públicas, crear nuevas páginas personalizadas, añadir un contenido personalizado a una página existente o editar la información de la página, como el título, URL y el contenido de SEO .

Con el editor de página se puede ordenar los widgets en una página como mejor le parezca. Puede añadir un widget de página, eliminar o editar su contenido .



Página inicial del editor.

¿Qué es un widget?

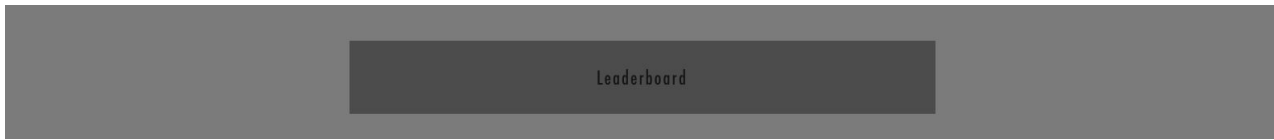
Widget no es nada más que un contenido de bloque horizontal de una página principal. Todavía confundido? Vamos a ver algunos ejemplos :



- *Banner Leaderboard (728 x 90)*

Esta es la presentación de un widget de Banner Leaderboard en el administrador del sitio.

Este consiste solamente de un bloque con un banner de tipo Leaderboard, es el widget más simple para utilizar como ejemplo, no hay contenido interno en él para personalizar desde el administrador del sitio.




Así se ve en el frontend.

- *Regístrate para recibir nuestro boletín de noticias*

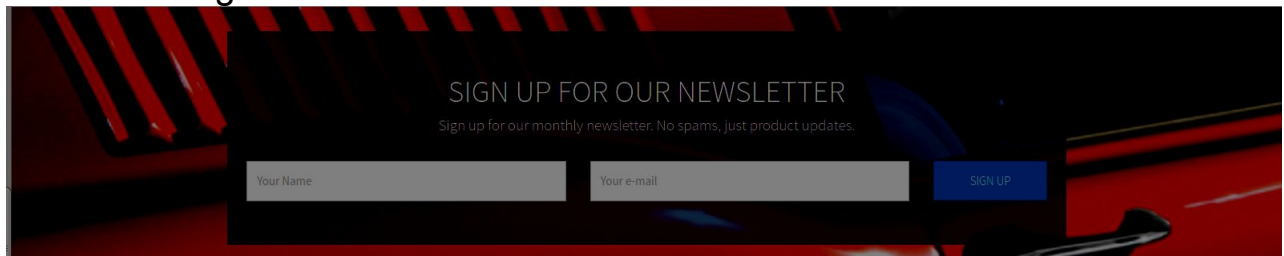


Esta es la representación del gestor de widgets de noticias en el sitio.

El widget Boletín Informativo tiene contenido que puede ser cambiado por el administrador del sitio, como el título, la descripción o la imagen de fondo.

Observe que en la parte derecha de la imagen hay un icono de edición (), diferente al del widget anterior.

Asi
se
ve
en
el



frontend.

Existen los widgets más complejos como el de "cabecera". En este, el administrador del sitio puede editar todas las etiquetas de la barra de inicio de sesión, así como los elementos de la navegación del sitio, las etiquetas y enlaces. Y también se puede cambiar el logotipo

*** Nota Importante:**

Cuando el Administrador del sitio cambia el contenido de cualquier tipo de widget "cabecera" o "Pie de página" que está cambiando ese widget para todas las páginas. A diferencia de otros widgets que su contenido es diferente para cada página. *

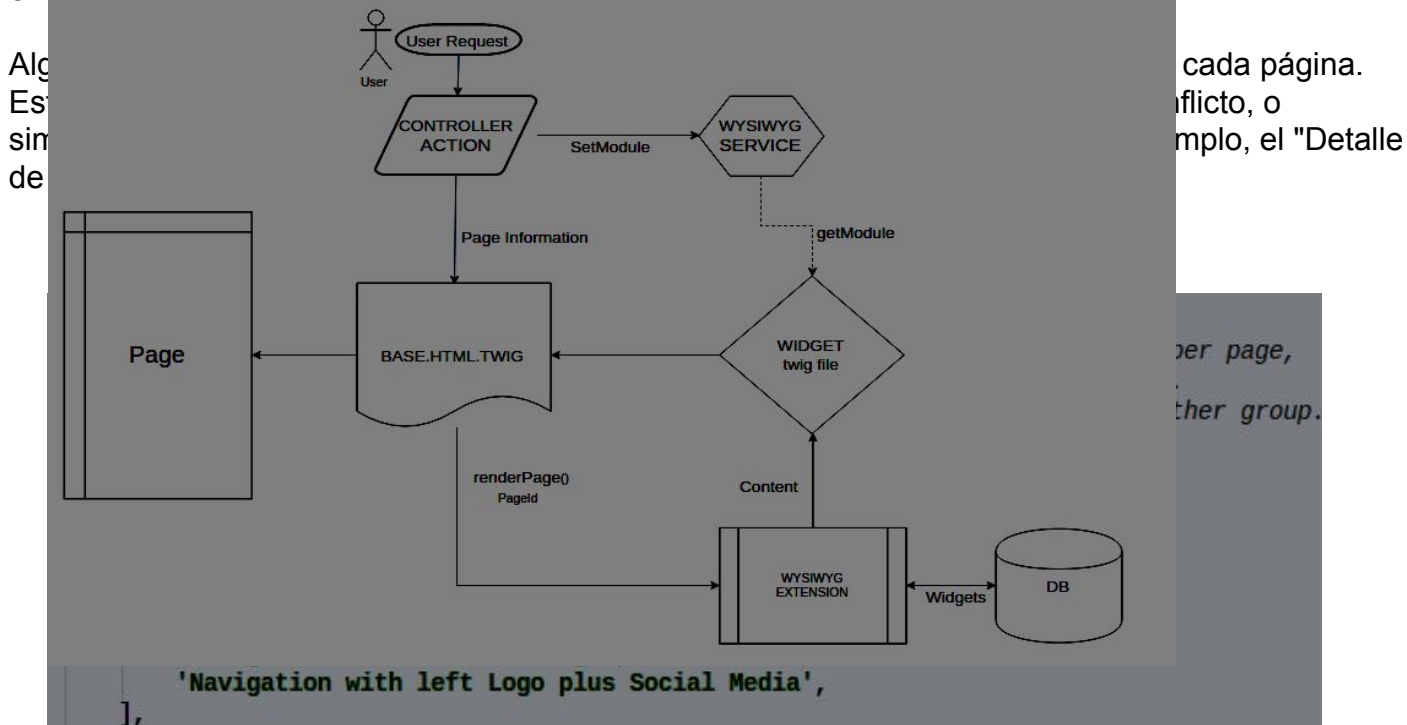
*** Nota Importante – El império contra-ataca:**

Los widgets que sólo disponen de campos de texto editables proporcionan una casilla de verificación en caso de que el administrador del sitio replicará los cambios realizados en ese widget en cada página. El widget de "Descarga nuestras aplicaciones de barras" también encaja en esta regla.

Cada widget tiene un archivo de plantilla que contiene el código HTML, y estos archivos de plantilla, la carpeta de widgets para cada sujeto. Por ejemplo, en el tema del restaurante es el camino '/ Recursos / temas / restaurante / widgets'.

Cada widget puede ser re-ubicado por el administrador del sitio. Sólo tiene que arrastrar el widget a la posición que desee y guardar. Por ejemplo, podemos mover el widget "Header" por debajo "del pie de página" .

El administrador del sitio puede agregar o eliminar cualquier widget a tu página. Con algunas excepciones en el elemento que desee agregar, ya que algunos widgets sólo están disponibles para ciertas páginas. Por ejemplo, el widget de " resultado de contenido con filtros izquierdo" sólo está disponible para la página de "Resultados". La lista completa de las excepciones se puede encontrar en el archivo "Widget_Templates.php".



Una lista completa de grupos de widgets que no pueden ser duplicados puede ser encontrada en las propiedades privadas `$widgetNonDuplicate` del servicio de `Wysiwyg` (`Wysiwyg.php`).

No puedo esperar a ver lo que nuestros clientes creativos van a hacer con esta nueva funcionalidad de Editor de Páginas.

La magia detrás del Editor de Páginas

Ahora vamos a hablar algo bueno, ya sabes el nuevo tekpix?
Es decir, vamos a profundizar en la estructura del Editor de Páginas?

- Flujo de trabajo en el Frontend

Con la llegada del Editor de Páginas, el Flujo de trabajo ha sufrido algunos cambios.

Ahora, cuando el usuario hace una petición al sistema, se cae en el controlador de acción responsable de la ruta solicitada y que informó al Servicio de Wysiwyg a qué módulo le pertenece la acción, ya que algunos reproductores necesitan saber a qué módulo pertenecen para ver su contenido correctamente. Además de todos los widgets de tipo banner o que contengan un banner, existen más widgets que dependen de un módulo, he aquí algunos ejemplos:

- **Explorar por bloque de Categorías con Imágenes**
- **Bloque de Comentarios**
- **Todas las locaciones**

Básicamente los widgets que tienen contenido que puede variar según el módulo, como categorías, ubicaciones y comentarios.

Cuando se trata de una página que no pertenece a la configuración de módulo de servicio WYSIWYG para módulo de listado.

Al final de toda Acción regresamos a Twig esa página y pasarlo a la información de la página (id, título, SEO).

Con los cambios del Editor de Páginas, Twig será devuelto en la mayoría de los casos a `base.html.twig`.

* **Nota Importante:** Solamente las acciones de los Resultados y el módulo de Detalle permanecen usando su respectivo Twig y no utilizan `base.html.twig` como las demás .

En la **`base.html.twig`** se llama la clase `WysiwygExtension` método `renderPage`, pasando el identificador de página. El método `renderPage` a su vez recupera de la base de datos todos los widgets que pertenecen ordenaron esa página, así como el contenido que el administrador del sitio ha configurado para ese widget

El `renderPage` sigue el orden de los campos de cada widget configurado por el Administrador de sitio y va montando la página final adicionando los archivos `twig` del al ruta en el campo **`twig_file`** de cada widget y su contenido(campo **`contenido`**).

* **Nota Importante:** el **`contenido`** que es pasado a el archivo `twig` es el de la entidad de `PageWidget`. El **`contenido`** de la entidad del `Widget` siempre será el default de ese widget y nunca el configurado por el administrador del sitio.

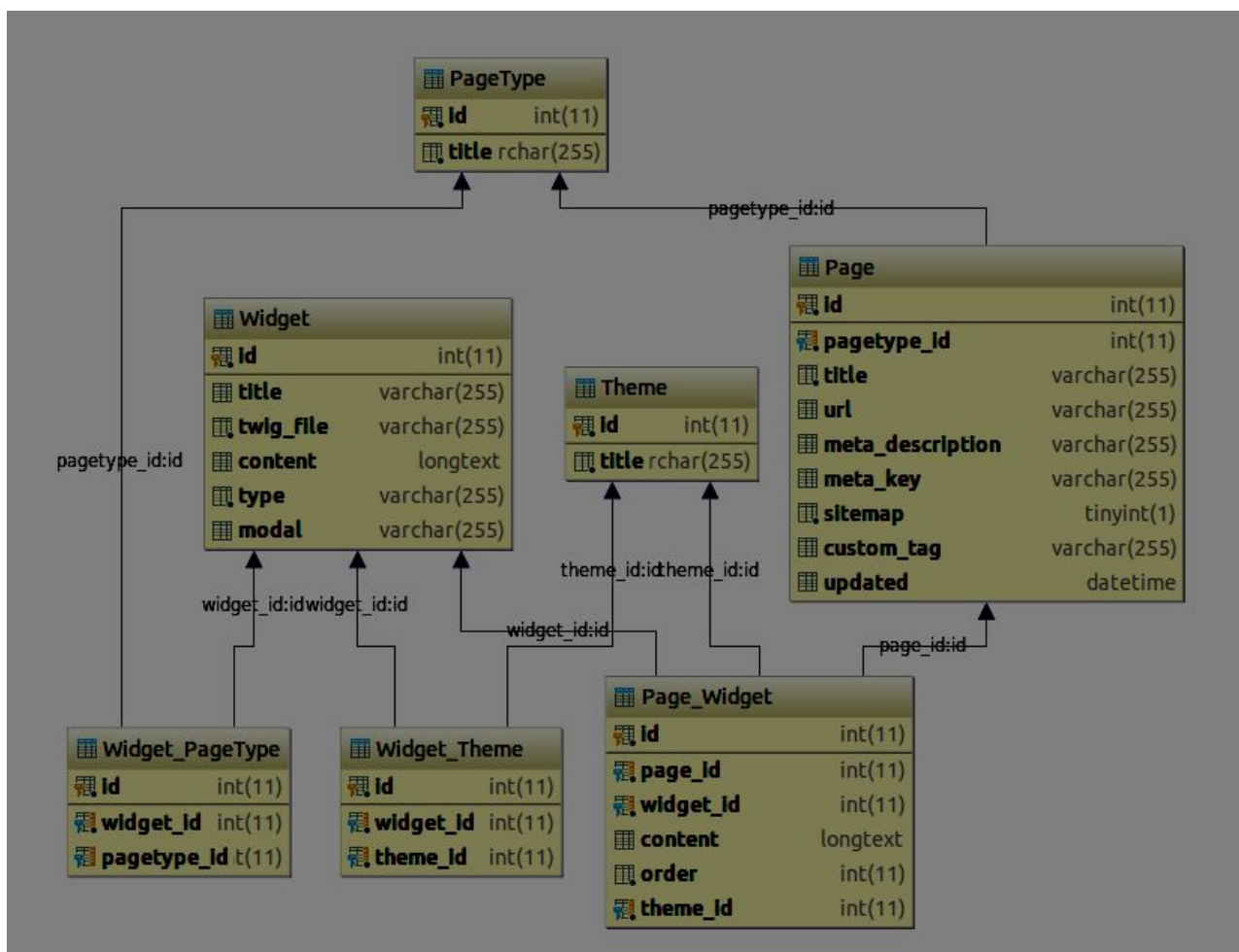
**** Nota Importante – Ofensiva Global:** Las viejas variables que fueron envidadas por el *Controlador* al twig de cada *Acción* ahora son las variables globales de **twig**, porque como ahora para renderizar pasamos por varios **twigs** los necesitamos como referencias globales. Asegúrese de no utilizar variables con el mismo nombre. Ejemplo de cómo agregar una variable global:

```
$twig = $this->container->get("twig");
$twig->addGlobal('item', $item);
```

Solamente cuando *renderPage* termina de montar todos los widgets es que regresamos a la página para el usuario determinando el final del ciclo

- La Base de Datos

Una



parte importante de toda la funcionalidad es entender la Base de Datos, y con el Editor de Páginas nada cambia.

El primer punto más importante con relación a la Base de Datos del Editor de Páginas es entender que hay dos entidades principales, la entidad *Widget* y la *Página*. La idea de funcionalidad gira en torno esas dos entidades.

La tabla de páginas sustituido a la antigua tabla de contenido, y tenía sus campos relacionados con el título y SEO incorporado por página. La columna de tipo se ha convertido en una tabla nueva, el tipo de página, que registra todos los tipos de

página. Estos tipos se encuentran al comienzo del servicio Wysiwyg como constantes .

Para el contenido del campo de la tabla de contenido se creó un widget '*Custom Content*' en el que el administrador del sitio puede agregar el contenido HTML que desee. Así que ahora el administrador del sitio puede agregar código HTML en cualquier lugar de la página a través del widget de 'contenido personalizado' .

La tabla *Widget* concentra las principales informaciones relacionadas con los widgets, entre ellas: ¿cuál es el archivo Twig (campo *twig_file*) Que se representará Cuando el widget se incluye en la página, el contenido es modificable por el administrador del sitio, lo que es el modal Identificación (modal) que permite la edición del widget a el Administrador del sitio y tipo (*type*) de *Widget* que define en que grupo se integrara en el modal de añadir widgets.

En la Tabla *Widget_Theme* esta cual widget está disponible para cual tema. Esto porque no todos los widgets están disponibles para todos los temas.

La tabla *Widget_PageType* registra los widgets que son únicos a ciertas páginas. Es decir, si un widget es único, habrá un registro en esta tabla entre el widget y el tipo de página. De lo contrario habrá un solo registro de widget sin página (*null*).

La tabla de *Theme* contiene los temas de eDirectory existentes .

Por último, la tabla ***Page_Widget*** establece la relación entre las páginas y los widgets, es decir, cada fila de esta tabla contiene un registro que indica que página (*page_id*) tiene un widget (*widget_id*) en un orden determinado y también el contenido) configurado por el administrador del sitio, para el tema indicado (*theme_id*).

Es en esta tabla donde puede averiguar qué widgets componen la Página de Inicio en el *Tema Predeterminado*, cuál es el orden de ellos y cuál es el contenido de cada widget.

*** Nota Importante:** Las imágenes de widgets no se guardan en la columna de contenido. Obedecen a la estructura de la versión anterior. Como por ejemplo el "deslizador" que tenía una tabla específica para esto.

- *La Carga de Datos*

La versión 11.2 de eDirectory se añadió una nueva e importante paquete al Editor página llamada ***DataFixtures***, que se encarga de hacer la carga por defecto de todos los registros de bases de datos relacionadas con el editor de página.

Al igual que casi toda la información sobre el editor de página se almacena en la base de datos, añadimos este paquete para que conste el contenido predeterminado de las páginas de eDirectory .

El paquete hace que los objetos creados se inserten en la base de datos. Se define un orden de inserción de las clases de obedecer las instalaciones del banco como las relaciones .

```
/**
 * the order in which fixtures will be loaded
 * the lower the number, the sooner that this fixture is loaded
 *
 * @return int
 */
public function getOrder()
{
    return 2;
}
```

Para crear relaciones a través de LoadData es necesario crear una referencia para el objeto que desea relacionar con otro, de modo que la siguiente clase que sigue el orden definido sepa cómo encontrar el objeto para crear la relación

```
$this->addReference($widget->getTitle(), $widget);
```

La documentación completa del paquete se puede encontrar aquí.

- *Administrador del Sitio*

El editor de páginas está disponible en el Administrador del Sitio en el área de diseño y personalización. En esta área de Administrador del Sitio se puede administrar las páginas de eDirectory y sus widgets.

Algunos puntos importantes sobre el área del Editor de Páginas:

- La funcionalidad javascript de la interfaz del Editor de Páginas concentra en los archivos 'web/scripts/widgets.js' e 'sitemgr/assets/custom-js/widget.php'.

- Todo widget editable llama un modal cuyo *id* corresponde a un valor del campo **modal**. Los widgets que solo contienen campos de texto para ser editados utilizan el mismo modal ID 'edit-generic-modal'. Y esos que no tienen contenido editable tendrán un campo **modal** vacío.

- Todo widget tiene una imagen de representación para cada tema. Las imágenes se encuentran en : 'web/sitemgr/assets/img/widget-placeholder'. Es importante resaltar que el nombre de cada imagen se refiere al título del widget pasando por la función 'system_generateFriendlyURL', si no hay imagen para ese widget, el sistema utiliza la imagen en el "Custom Content" widget.

Otro punto interesante en la funcionalidad de redefinir una página, es que le permite al Administrador del Sitio restaurar los widgets de una página a su formato original. Para eso fue adicionado el *Wysiwyg Service* (Wysiwyg.php) la configuración original de las páginas de eDirectory. Para cada página existe una función que retorna los widgets originales de dicha página para cada tema.

```
/**
 * Returns the widgets that compose the Listing Home
 * Used for load data and reset feature at sitemgr
 *
 * @return mixed
 */
public function getListingHomeDefaultWidgets()
{
    $pageWidgetsTheme = [
        Theme::DEFAULT_THEME => [
            'Header',
            'Search Bar',
            'Leaderboard ad bar (728x90)',
            '3 Featured Listings',
            'Browse by category block with images',
            'Best Of Listings',
            '3 rectangle ad bar',
            'Browse by Location with Right Banner (160x600)',
            'Banner Large Mobile, one banner Sponsored Links and one Google Ads',
            'Download our apps bar',
            'Footer',
        ],
    ],
}
```

Este es un extracto de la función que devuelve los patrones de los widgets de la página "Listing Home".

La función completa de cada página puede ser encontrada en *Wysiwyg Service* (Wysiwyg.php).

```
/**
 * Returns all the pages that have some widget that has any content different from its default
 * USED IN LOAD DATA
 *
 * @return array
 */
public function getDefaultSpecificWidgetContents()
{
    $translator = $this->container->get('translator');
    $language = substr($this->container->get("multi_domain.information")->getLocale(), 0, 2);

    // Set specific contents
    $contents = [];
    $contents[Wysiwyg::EVENT_HOME_PAGE]['Search Bar'] = json_encode(['labelExploreAndFind' => 'Explore and find Events']);
    $contents[Wysiwyg::CLASSIFIED_HOME_PAGE]['Search Bar'] = json_encode(['labelExploreAndFind' => 'Explore and find Classifieds']);
}
```

* **Nota Importante:** Algunas páginas poseen widgets con contenido diferente al original del widget. Una lista completa se encuentra en la función 'getDefaultSpecificWidgetContents' en *Wysiwyg Service*

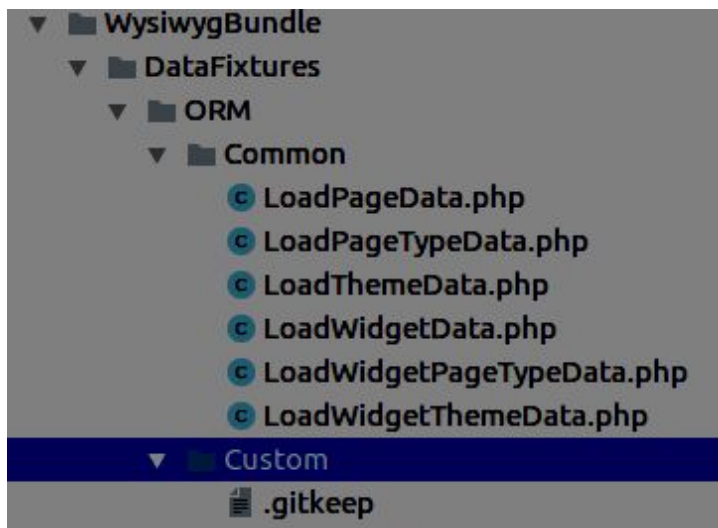
(Wysiwyg.php).

Costumizando el Editor de Paginas

Para costumizar el Editor de Paginas es preciso saber que la mayoría de la información esta almacenada en la base de datos, por lo tanto es posible crear widgets solamente a partir de la inserción de la información en la base de datos y agregar su twig, por ejemplo. Pero esa no es la forma más adecuada y tampoco es el foco de esta guía.

Las principales adaptaciones que involucran al **Editor de Páginas** crearían nuevos widgets, páginas o temas. La mejor manera de hacerlo es a través de LoadData, de esta forma, todo está registrado en el código y no sólo en la base de datos .

Por lo tanto, para cualquiera de las customizaciones siguientes, agregar un widget, página o tema, es preciso copiar los archivos del folder **Common** de *LoadData* dentro del folder **Custom**.



* **Nota Importante:** No se olvide de corregir el espacio de nombres de los archivos copiados.

- Agregar un Widget

Para esta guía vamos a utilizar como ejemplo la creación del widget '**Popular Listings**' .

Para empezar es necesario crear el archivo twig '**popular-listings.html.twig**' en la sub carpeta(sub-folder) "listing" contenida en la carpeta "widgets" de cada tema, por ejemplo para el tema **default** estaria en '/Resources/themes/default/widgets/listing', a menos que el nuevo widget tenga exclusividad de temas, entonces solo es preciso para los temas en los que aparece.

Entonces tenemos que colocar la información del widget en el bundle de LoadData.

En el archivo que acaba de copiar 'LoadWidgetData.php', al final del array `$standardWidgets` existe un comentario de ejemplo de como agregar un widget.

En caso de dudas, se debe de crear un nuevo nodo de array con la información del nuevo widget y su contenido por defecto. Ejemplo:

Con esto, definimos el título, archivo *twig*, el **Tipo** para saber en cual tab será habilitado, el contenido con un único *label*, y como la única cosa editable será el *label(s)*, el **modal** genérico del nuevo widget '**Popular Listings**'.

*** Nota Importante:** Si el widget es más complejo y el Administrador del Sitio puede editar más cosas además del *label(s)* es necesario crear un modal específico para este nuevo widget en la carpeta de modales de widgets en el area de Administrador del sitio: `"/web/includes/modals/widget"`.

El archivo recién copiado `'LoadWidgetPageTypeData.php'` , vamos a añadir, si es necesario, un nudo en el extremo del *array* `$exceptionsWidgets` con las páginas a las que este nuevo widget puede ser agregado, si el nuevo widget está disponible para todas las páginas omitir este paso

```
'Popular Listings' => [
    $this->getReference("TYPE_".Wysiwyg::HOME_PAGE),
    $this->getReference("TYPE_".Wysiwyg::LISTING_HOME_PAGE),
],
```

En nuestro ejemplo, se define el nuevo widget '**Popular Listings**'. Será exclusivo del **"Home Page"** y el **"Listing Home"**.

Ahora para definir en qué temas el nuevo widget está disponible, es necesario agregar un nuevo nodo con el título del nuevo widget en la matriz de retorno en las funciones que definen lo que los reproductores de cada sujeto en *Wysiwyg Service* (*Wysiwyg.php*).

Cada tema tiene en su función la lista de sus widgets, y también hay uno para los widgets que son iguales en todos los temas.

```
/**
 * Returns the commons and the Default Theme widgets
 *
 * @return array
 */
public function getDefaultThemeWidgets()
{
    $trans = $this->container->get('translator');

    return array_merge($this->getCommonThemeWidgets(), [
        $trans->trans('Header', [], 'widgets', 'en'),
        $trans->trans('Footer', [], 'widgets', 'en'),
        $trans->trans('Popular Listings', [], 'widgets', 'en'),
    ]);
    /*
     * CUSTOM ADDWIDGET
     * here are an example of how you add the widget 'Widget test' for Default theme
     * if you need that 'Widget test' to be available for all themes you have
     * to remove it from here and add at the right function above
     *
     * $trans->trans('Widget test', [], 'widgets', 'en'),*/
}
```

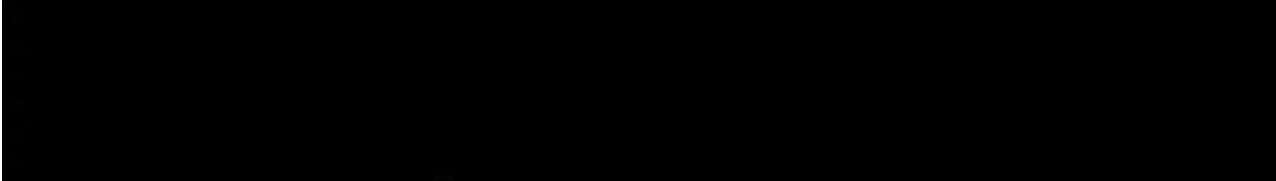
En este caso dejamos nuestro nuevo widget '**Popular Listings**' disponible solamente en el tema

predeterminado. Si tiene que ser común en todos los casos es solo de agregar al resultado de la función `'getCommonThemeWidgets'`.

Bien, ahora sólo tiene que ejecutar el comando en el terminal de LoadData insertar los nuevos datos, **asegúrese de poner el dominio correcto**.

```
php app/console doctrine:fixtures:load
--fixtures=src/ArcaSolutions/WysiwygBundle/DataFixtures/ORM/Custom --append
--domain=seu.dominio.com
```

Los siguientes mensajes deben ser devueltos si todos los pasos se han realizado correctamente:



Listo, el nuevo widget **'Popular Listings'** ha sido agregado.

- Agregar una Página

Para esta guía vamos a usar de ejemplo la creación de la página **"Categories Home"**, teniendo en cuenta que ya se añadió una ruta, se creó un controlador y la acción, y todo lo que era necesario para agregar una nueva página en eDirectory.

Añadir una nueva página es muy similar a agregar un widget. Así como al agregar un nuevo *widget*, es necesario copiar los archivos de la carpeta **Common** de *LoadData* a la carpeta **Custom**.

*** Nota Importante:** No se olvide de corregir el espacio de nombres de los archivos copiados.

Antes de adicionar algo a *LoadData* necesitamos crear una constante para el tipo de esta nueva página. Toda página tiene su tipo listado como constante en *Wysiwyg Service* (`Wysiwyg.php`). Agregue el nuevo tipo al final de la lista:

```
const BLOG_HOME_PAGE = "Blog Home";
const BLOG_DETAIL_PAGE = "Blog Detail";
const BLOG_CATEGORIES_PAGE = "Blog View All Categories";
/**
 * CUSTOM ADDPAGETYPE
 * here are an example of how you add a PageType constant to be used in the load data
 */
/*const TEST_PAGE = "Test Page";*/
const CATEGORIES_HOME = "Categories Home";
```

En el archivo recién copiado `'LoadPageTypeData.php'` adicione un nuevo tipo de página a un nuevo nodo al final del array `'$standardPageTypes'`.

En el archivo recién copiado 'LoadPageData.php' adicione la información de la página al nuevo nodo al final del *array* '\$standardPages'.

```
/**
 * CUSTOM ADDPAGE
 * Here are an example of how you add a page,
 * and you will need to create a PageType for this page
 *
 * Don't forget to create the alias for the url, if needed
 */
[
    'title' => $trans->trans('Categories Home', [], 'widgets', 'en'),
    'url' => $this->container->getParameter('alias_categories_home_url_divisor'),
    'metaDesc' => '',
    'metaKey' => '',
    'sitemap' => false,
    'customTag' => '',
    'pageType' => $this->getReference("TYPE_".Wysiwyg::CATEGORIES_HOME),
],
```

El próximo paso es crear la función que define los widgets de la Página por defecto para cada tema. En el *Wysiwyg Service* (*Wysiwyg.php*) cree una función utilizando la constante con el nuevo tipo de página que fue adicionado como parte del nombre de la función, porque la funcionalidad redefinir la página toma la función de acuerdo al tipo de página.

```
/**
 * Returns the widgets that compose the Categories Home
 * Used for load data and reset feature at sitemgr
 *
 * @return mixed
 */
public function getCategoriesHomeDefaultWidgets()
{
    $pageWidgetsTheme = [
        Theme::DEFAULT_THEME => [
            'Header',
            'Search Bar',
            'Leaderboard ad bar (728x90)',
            'Browse by category block with images',
            '3 rectangle ad bar',
            'Banner Large Mobile, one banner Sponsored Links and one Google Ads',
            'Download our apps bar',
            'Footer',
        ],
        Theme::DOCTOR_THEME => [...],
        Theme::RESTAURANT_THEME => [...],
        Theme::WEDDING_THEME => [...],
    ];

    return $pageWidgetsTheme[$this->theme];
}
```

Nota Importante: El nombre de esta función debe seguir la siguiente forma: 'get' + 'valor de la constante de tipo da página sin espacios' + 'DefaultWidgets'.

```
/* Get Default Widgets Method */
$method = 'get'.str_replace(' ', '', $pageType).'DefaultWidgets';
$pageWidgetsArr = $this->$method();
```

**** Nota Importante:** No olvide crear el *array* con los widgets de la página para cada tema, y que cada tema tenga su widget de *Header* y *Footer*.(encabezado y pie de página)

Una vez creada la función que define el default de los widgets de la página es

necesario enumerar ahí la lista para la función 'getAllPageDefaultWidgets' utilizada en *LoadData*.

```
/**
 * Returns an array with all the standard pages and its own array of default widgets
 * USED IN LOAD DATA
 *
 * @return array
 */
public function getAllPageDefaultWidgets()
{
    $pagesDefault = [];
    $pagesDefault[Wysiwyg::CATEGORIES_HOME] = $this->getHomePageDefaultWidgets();
    $pagesDefault[Wysiwyg::HOME_PAGE] = $this->getHomePageDefaultWidgets();
}
```

Si uno de los widgets de esta nueva página tiene un contenido diferente sólo para esta página es necesario definir su contenido diferenciado en función de 'getDefaultSpecificWidgetContents' .

```
/**
 * Returns all the pages that have some widget that has any content different from its default
 * USED IN LOAD DATA
 *
 * @return array
 */
public function getDefaultSpecificWidgetContents()
{
    $translator = $this->container->get('translator');
    $language = substr($this->container->get("multi_domain.information")->getLocale(), 0, 2);

    // Set specific contents
    $contents = [];
    $contents[Wysiwyg::CATEGORIES_HOME]['Search Bar'] = json_encode(['labelExploreAndFind' => 'Explore and find Categories']);
    $contents[Wysiwyg::EVENT_HOME_PAGE]['Search Bar'] = json_encode(['labelExploreAndFind' => 'Explore and find Events']);
}
```

Bien, ahora sólo tiene que ejecutar el comando en el terminal de LoadData insertar los nuevos datos, asegúrese de poner el dominio correcto.

```
php app/console doctrine:fixtures:load
--fixtures=src/ArcaSolutions/WysiwygBundle/DataFixtures/ORM/Custom --append
--domain=seu.dominio.com
```

Los siguientes mensajes deben ser devueltos si todos los pasos se han realizado



correctamente:

Ahí está la nueva página '**Categories Home**' fue agregada. Para finalizar, la edición de la nueva página dentro del Editor de Páginas, haga clic en reiniciar la página para que los widgets por defecto sean añadidos.

- *Agregar un Tema*

Para este guía vamos usar de ejemplo de como crear un nuevo tema '**School**,

considerando que todo el proceso antes de crear un nuevo tema fue seguido de acuerdo con este tutorial.

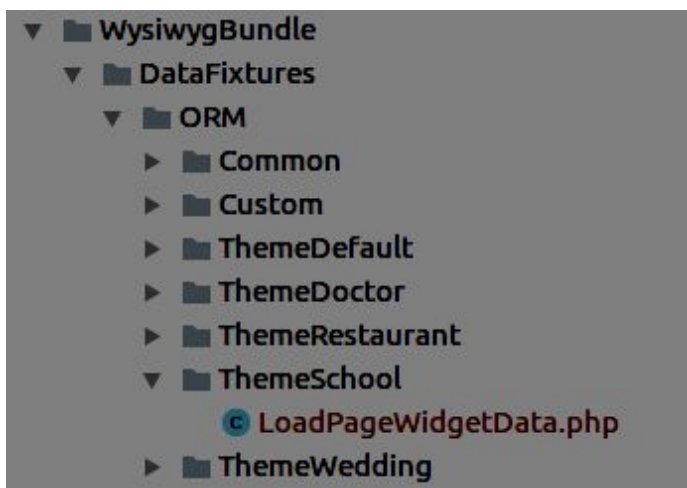
El primer paso es crear una constante con el título del nuevo tema de la entidad del Tema(Theme.php).

```
/**
 * Existing themes titles of eDirectory
 */
const DEFAULT_THEME = 'Default';
const DOCTOR_THEME = 'Doctor';
const RESTAURANT_THEME = 'Restaurant';
const WEDDING_THEME = 'Wedding';
const SCHOOL_THEME = 'School';
```

En el recién copiado 'LoadThemeData.php' agregue un nuevo nodo en el array '\$standardThemes' con la constante del título del nuevo tema.

```
/* Theme title is used as reference in LoadPageWidgetData,
 * so if you change here don't forget to change there
 */
$standardThemes = [
    Theme::DEFAULT_THEME,
    Theme::DOCTOR_THEME,
    Theme::RESTAURANT_THEME,
    Theme::WEDDING_THEME,
    Theme::SCHOOL_THEME,
    /**
     * CUSTOM ADDTHEME
     * here are an example of how you add the theme 'Test'
     *
     * Theme::TEST_THEME, */
];
```

A diferencia de los otros tabs, para agregar un Tema nuevo también es necesario crear una nueva carpeta en *LoadData* para el nuevo tema. La carpeta debe ser creada dentro del '**ORM**' con el nombre '**Tema**' además del título del nuevo tema, en nuestro ejemplo es '**ThemeSchool**'.



Dentro de la carpeta hay que crear una copia del archivo 'LoadPageWidgetData.php' en la carpeta **ThemeDefault**. En el cambiar todas las llamadas de constante del tema por **Defecto**, para el nombre del nuevo tema. **Ejemplo:**

Lo que queda ahora es hacer que una parte laboriosa en Servicio Wysiwyg (`Wysiwyg.php`).

La primera parte es sencilla, es necesario crear una función que devuelve una matriz de todos los widgets que están disponibles en este nuevo tema. Por ejemplo, digamos que nuestro nuevo tema 'School' es una copia del **tema por defecto**, por lo tanto, poseen todos los widgets de este tema común. La función se vería así:

```
/**
 * Returns the commons and the School Theme widgets
 *
 * @return array
 */
public function getSchoolThemeWidgets()
{
    $trans = $this->container->get('translator');

    return array_merge($this->getCommonThemeWidgets(), [
        $trans->trans('Header', [], 'widgets', 'en'),
        $trans->trans('Footer', [], 'widgets', 'en'),
    ]);
}
```

Lo importante de la función es retornar todos los widgets comunes a todos los temas mas exclusivos de este nuevo tema. El nombre de la función está formado por 'get' + 'valor de la constante del nuevo tema' + 'ThemeWidgets'. En nuestro ejemplo era: 'getSchoolThemeWidgets'.

La segunda parte es mas difícil, es necesario crear un nuevo *array* de cada función que lista los patrones de widgets de cada página del sistema. Este no debe listar los patrones de widgets de la página para ese nuevo tema. En el siguiente ejemplo hemos hecho esta adición en la función: "*Custom Page*".

Bien, ahora sólo tiene que ejecutar el comando en el terminal de *LoadData* insertar los nuevos datos, **asegúrese de poner el dominio correcto**.

Note que existe **um parametro mas en la ruta de la carpeta creada para este tema**, en nuestro ejemplo el tema **'School'** :

```
php app/console doctrine:fixtures:load  
--fixtures=src/ArcaSolutions/WysiwygBundle/DataFixtures/ORM/Custom  
--fixtures=src/ArcaSolutions/WysiwygBundle/DataFixtures/ORM/ThemeSchool --append  
--domain=seu.dominio.com
```

Los siguientes mensajes deben ser devueltos si todos los pasos se han realizado correctamente:



Y listo, se ha agregado el nuevo tema **'School'**.

