Magento 2

Desarrollo de Extensiones en Magento

Escrito por
José Luis Rojo Sánchez

Índice de Contenidos

Introducción

Si estás familiarizado a trabajar con **Magento 1x** te habrás dado cuenta que hay muchos conceptos parecidos con los que ya trabajabas.

Sin embargo el código es un framework totalmente nuevo, hay muchos cambios en la estructura modular y otros ficheros relacionados.

Los mayores cambios con los que nos encontramos en Magento 2 son los siguientes:

- **Una nueva estructura modular**: La estructura de directorios ha cambiado, convirtiendo partes de una extensión que estaban ubicados en diferentes directorios en una solo por lo que mantener nuestra extensión se nos hará mucho más fácil.

- **Ficheros de configuración**: La configuración de una extensión en **Magento 2** es dividida en múltiples y pequeños ficheros que son validados por un XML Schema Definition (XSD).

  Estos ficheros de configuración pueden ser pertenecer a un área específica como (adminhtml, frontend, cron y api) que sobrescribirán las configuraciones principales.

- **Namespaces e inyecciones de dependencias**: En Magento 2, el código usa namespaces para identificar los nombres de las clases así evitamos la duplicidad. Las dependencias se inyectarán en la propia clase.

Declaración y registro de extensiones

Magento 2 es un sistema modular que nos permite la creación de extensiones. Estos extensiones se crean usando controladores, bloques, helpers, modelos, layouts, templates, etc ... Cada extensión realiza una acciones.

La forma de construir una extensión en Magento 2 es un poco diferente a la forma de construirla en Magento 1. El mayor cambio es que ahora todos los ficheros están incluidos en el directorio de la extensión, lo que nos permite de una forma sencilla encontrar y solucionar fallos dentro de ella.

La localización de la extensión también es diferente, ya que no hay código separado como en Magento 1 (core, community y local). Dependiendo de la forma en que instales la extensión esta se ejecutará desde un directorio vender/ cuando esta sea instalada desde Composer y para una extensión propia su ubicación será app/code/.

Cuando empecemos a programar una extensión es importante ejecutar Magento 2 en modo desarrollador que nos permitirá controlar de una forma más eficiente los errores y solucionarlos de forma más sencilla. Muy importante activar **display_errors** en app/bootstrap.php y controlar errores en los ficheros exception.log, system.log ubicados en la carpeta /var/log

```
$ php ./bin/magento deploy:mode:set developer
```

Por lo que si queremos programar una nueva extensión la ruta será la siguiente: /app/code/<vendor>/<module>/

Vamos a explicar paso a paso como realizarlo paso a paso. Vamos a crear un gestor de noticias para nuestra página por lo que empezaremos creando su estructura principal de directorios.

```
/app/code/Ilovemarketing/News/
```

Declaración de Módulo

Procedemos a declarar nuestro módulo y es aquí donde indicaremos el nombre de nuestro módulo, sus versiones de instalación y sus dependencias con las que podremos manipular el orden de carga.

Por lo que module.xml se encargará de registrar en módulo en Magento 2 .

- **name** es el nombre de nuestra extensión
- **setup_version** es la versión usada para hacer scripts de Setups y Upgrades creando los schemas de nuestras tablas.
- **sequence** nos permite definir las dependencias de nuestra extensión.

Crearemos en /app/code/Ilovemarketing/News/etc/ el fichero **module.xml**

```
<?xml version="1.0" encoding="UTF-8"?>
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:Module/etc/module.xsd">
   <module name="Ilovemarketing_News" setup_version="0.0.1" data_version="0.0.1">
      <sequence>
         <module name="Magento_Store"/>
      </sequence>
   </module>
</config>
```

Registro del Módulo

Con el uso de Composer los ficheros pueden ser instalados en dos lugares diferentes. Para que el auto cargador de composer sepa que fichero cargar cuando una clase es instanciada el PATH debe ser registrado.

Para ello tendremos que crear un fichero llamado registration.php en la raíz de nuestro módulo. (__DIR__) es la localización donde será instalado. El registro puede ser realizado por los siguientes tipos de componentes:

- **MODULE**: para extensiones y modulos
- **LIBRARY**: para extensiones que son usadas como una librería.
- **THEME**: para temas.
- **LANGUAGE**: para paquetes de idiomas.

```
<?php
\Magento\Framework\Component\ComponentRegistrar::register(
   \Magento\Framework\Component\ComponentRegistrar::MODULE,
   'Ilovemarketing_News',
   __DIR__
);
```

Composer

Por último creamos un fichero composer.json en el directorio raíz de nuestro módulo y en el que incluimos diferentes configuraciones de nuestra extensión y que nos permitirán usar Composer para instalar nuestra extensión.

Lo elementos más importantes a tener en cuenta en este fichero son:

- **name**: El nombre de la extensión siguiendo el formato <vendor>/<extension> y en minúsculas. Se usará para instalar la extensión via Composer.

- **type**: Definirá el tipo de extensión. Los tipos son los siguientes:

    - **magento2-module**: para extensiones y módulos
    - **magento2-theme**: para temas.
    - **magento2-language**: paquetes de idiomas.

- **require**: define que paquetes necesitan ser instalados para que la extensión funcione correctamente. Durante la extensión via Composer este comprobará sus requerimientos y los instalará en caso de no ser encontrados. En caso de que no sea posible su instalación esta fallará.

- **autoload**: Indica la información que necesita ser cargada a través del cargador de Composer.

```
{
   "name": "ilovemarketing/news",
   "description": "I love Marketing - News - extension",
   "keywords": ["magento2", "ilovemarketing", "news"],
   "type": "magento2-module",
   "license": "OSL-3.0",
   "require": {
      "php": "~5.5.0|~5.6.0|~7.0.0"
   },
   "autoload": {
      "files": [ "registration.php" ],
      "psr-4": {
         "Ilovemarketing\\News\\": ""
      }
   }
}
```

Comprobamos el estado de nuestro módulo para ver su estado actual y tendrá que aparecer como deshabilitado.

```
$ php -dmemory_limit=1G ./bin/magento module:status
```

Procedemos con la activación del módulo.

```
$ php ./bin/magento module:enable Ilovemarketing_News
```

Y upgradeamos la base de datos para que confirmar los cambios.

```
$ php -dmemory_limit=1G ./bin/magento setup:upgrade
```

Además de poder instalar paquetes via Composer, también hay otras opciones que podremos utilizar:

- A través del **MarketPlace** de magento.
- Vía repositorio de control de versiones (**Github**). Para ello tendremos que incluir las siguientes líneas a nuestro composer .json

```
{
   "repositories": [ {
      "type": "vcs",
      "url": "https://github.com/[github account]/[package]"
   } ]
}
```

- **Packagist**: que es el repositorio oficial de Composer. Para añadir una extensión a Packagist necesitas grabarla antes en un repositorio público como puede ser Github o BitBucket para luego subirla a Packagist vía URL.

Cada paquete podremos instalarlo ejecutando el siguiente comando en el directorio raíz de tu Magento 2.

```
composer require <vendor-name>/<module-name>
```

Rutas y Controladores

Para poder visualizar datos de nuestra nueva extensión en el frontend (la parte pública de nuestra web) necesitamos lo siguiente:

- Configurar unas rutas.
- Crear un controlador que maneje las peticiones de dicha ruta.
- Un layout que especificará que se necesita mostrar.
- Un bloque o bloques declarados en el layout.
- Un template que recibirá los datos del bloque (este es opcional)

Nos centraremos inicialmente en la configuración de rutas y controladores por lo que los ficheros que vamos a usar en este tema son:

/app/code/Ilovemarketing/News/etc/frontend/routes.xml
/app/code/Ilovemarketing/News/Controller/Index/Index.php

Declaración de Rutas

Vamos a definir la rutas de nuestro módulo y que nos permitirán accederlo vía URL a través de nuestro navegador por lo que para ello creamos un fichero llamado **routes.xml** en el directorio **/etc/frontend/** de nuestro módulo. El atributo **frontName** define nuestra ruta.

```
<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:App/etc/routes.xsd">
   <router id="standard">
      <route id="news" frontName="news">
         <module name="Ilovemarketing_News" />
      </route>
   </router>
</config>
```

Magento reconoce estas rutas de la siguiente forma:

- http://tudominio.com/<ruta>/<controlador>/<accion>

Por lo que una vez definida la ruta y sus controladores la url de será accesible desde

- http://tudominio.com/news/*
- http://tudominio.com/news/index/*
- http://tudominio.com/news/index/index/
- http://tudominio.com/news/index/index/[parametros/valores]

Controladores

Ahora necesitamos crear un controlador y sus acciones para que así funcione nuestra nueva ruta para así poder visualizar el contenido de nuestro nuevo módulo.

En **Magento2** cada controlador debe contener una función **execute**() y deberá ser única para cada controlador que creemos. Creamos nuestro controlador en el directorio **Controller/Index/** y lo llamaremos **Index.php**

```
<?php
namespace Ilovemarketing\News\Controller\Index;

use Magento\Framework\App\Action\Action;
use Magento\Framework\App\Action\Context;

class Index extends Action {

   public function __construct(
   Context $context) {

      parent::__construct($context);

   }

   public function execute() {

      die('Mi pagina de noticias');

   }
}
```

Esto muestra una página en blanco en la siguiente url:

- http://tudominio.com/news/*

Es una página en blanco con el texto que le hemos indicado. En el siguiente tema le daremos forma donde trataremos los blocks, layouts y templates.

Blocks, Layouts y Templates

Ya hemos visto cómo se declara un módulo y como se definen sus rutas y controladores.

En este paso vamos a tratar con la siguiente estructura para seguir mejorando nuestro módulo y poder aplicar una vista a nuestras noticias.

Para ello usaremos los bloques, layouts y templates. Estos son los ficheros que vamos a utilizar en este tema:

/app/code/Ilovemarketing/News/Controler/Index/Index.php
/app/code/Ilovemarketing/News/view/frontend/layout/news_index_index.xml
/app/code/Ilovemarketing/News/Block/Index.xml
/app/code/Ilovemarketing/News/view/fontend/template/newsList.phtml

Para renderizar los elementos de una página magento usa bloques, layouts y templates. Continuando el ejercicio anterior, vamos a modificar un poco nuestro controlador para poder usarlos.

En el controlador vamos a incluir la llamada a una clase nueva llamada PageFactory que es la que fabrica las páginas en magento, para luego llamar a su método create que será el encargado de renderizar la página. El namespace es la ubicación de nuestro controlador.

```
<?php
namespace Ilovemarketing\News\Controller\Index;

use Magento\Framework\App\Action\Action;
use Magento\Framework\App\Action\Context;
use Magento\Framework\View\Result\PageFactory;

class Index extends Action {

   protected $_pageFactory;

   public function __construct(
      Context $context,
      PageFactory $pageFactory) {

      $this->_pageFactory = $pageFactory;
      parent::__construct($context);

   }
```

```
    public function execute() {

        return $this->_pageFactory->create();

    }

}
```

Layouts

El siguiente paso es crear el **layout**, que será la estructura de la página del módulo.

El layout lo crearemos en **view/frontend/layout/** y lo llamaremos **news_index_index.xml**. Como ya comentamos llamara a <modulo> / <controlador> / <acción>.

```
<?xml version="1.0" encoding="UTF-8"?>
<page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" layout="1column"
xsi:noNamespaceSchemaLocation="urn:magento:framework:View/Layout/etc/page_con
figuration.xsd">
    <head>
        <title>Mi módulo de noticias</title>
    </head>
    <body>
        <referenceContainer name="main">
            <block class="Ilovemarketing\News\Block\NewsList" name="NewsList"
template="Ilovemarketing_News::newsList.phtml" />
        </referenceContainer>
    </body>
</page>
```

Una novedad que trae Magento2 es que usa **schemas** de xml que llaman a un fichero **XSD** (xml schema definiton) el cual nos indica cómo será la estructura correcta del fichero xml. Sino lo declaramos, este no funcionará correctamente.

No vamos a entrar ahora en instrucciones para crear layouts porque ese tema lo trataremos en la parte de creación de temas con Magento2. Básicamente le estamos diciendo que en el content de nuestra página web, cree un nuevo bloque y que representara la información en el template indicado.

Bloques

Vamos a crear un bloque que es el que enviar la información a nuestro template en **Block/** y lo llamaremos **NewsList.php**

Todos los bloques en magento extienden la clase **Template**.

```
<?php
namespace Ilovemarketing\News\Block;

use Magento\Framework\View\Element\Template;

class NewsList extends Template {

    public function getNews() {

        return __('Estas son mis noticias, mas adelante aprenderé como consultarlas de la base de datos!');

    }

}
```

Templates

Por último creamos el fichero de template (con extensión .phtml) en **view/fontend/template/** y lo llamaremos **newsList.phtml**

Podemos usar la variable $**block** o $**this** para llamar al método de la clase del bloque que creamos anteriormente.

```
<h1><?php echo $this->getNews(); ?></h1>
```

Si vamos a http://dominio.com/news/ podremos ver nuestra pagina:

En la siguiente lección vamos a ir creando un Setup para cargar nuestras tablas a la base de datos y poder continuar con los modelos.

Modelos de la Base de Datos

El proceso de grabado de información en un nuestras tablas de la base de datos es manejado por Modelos.

Estos Modelos almacenan los datos. Mientras estos son salvados un **ResourceModel** es llamado y que hará de enlace entre nuestro Modelo y las tablas de la Base de Datos.

Todas las operaciones **CRUD** (create, read, update y delete) pasarán a través de este ResourceModel. Cuando carguemos un número x de registros una **Colección** será usada a la cual se le podrán aplicar filtros.

Usar modelos de bases de datos requiere que la configuración sea correcta. De otra forma el auto cargador no será capaz de encontrar los ficheros.

Continuando con el tema anterior vamos a incluir una tabla de noticias a nuestra base de datos y unos datos con los que trabajar. Vamos a crear la siguiente estructura de ficheros para que nuestros modelos trabajen correctamente:

/app/code/Ilovemarketing/News/Model/Model.php
/app/code/Ilovemarketing/News/ResourceModel/News.php
/app/code/Ilovemarketing/News/Model/ResourceModel/News/News.php

Modelo

Lo primero que vamos a crear es nuestro modelo en **Model/** y lo llamaremos **News.php.** Esta clase especifica el **ResourceModel** que vamos a utilizar usando la función **_init()**.

Cuando la función save() es llamada por el modelo, este llamará a la función save() en el ResourceModel. Esta función save() es un método de la clase **AbstractModel** que está extendiendo nuestro Modelo.

```
<?php
namespace Ilovemarketing\News\Model;

use Magento\Framework\Model\AbstractModel;

class News extends AbstractModel {

   protected function _construct() {

      $this->_init('Ilovemarketing\News\Model\ResourceModel\News');

   }

}
```

Resource Model

Crearemos la clase ResourceModel en **Model/ResourceModel/** y la llamaremos **News.php.** Esta será nuestra conexión hacia las tabla de la base de datos.

Todos los ResourceModel deben extender la clase abstracta de **Magento AbstractDb**. El constructor inicializa nuestra tabla de la base de datos y tendremos que indicar su identificador con clave primaria.

Este será usado para crear las consultas necesarias para cargar o salvar datos. Dependiendo de la consulta que se realice (inserción, actualización o borrado), una consulta SQL es generada automáticamente (INSERT, UPDATE o DELETE). Esta será manejada por el framework y construida por la clase **Zend_Db_Adapter_Pdo_Mysql**.

```
<?php
namespace Ilovemarketing\News\Model\ResourceModel;

use Magento\Framework\Model\ResourceModel\Db\AbstractDb;

class News extends AbstractDb {

   protected function _construct() {

      $this->_init('ilovemarketing_news', 'news_id');

   }
}
```

Colecciones

Por último crearemos una colección en **Model/ResourceModel/News/** y la llamaremos **News.php** que inicializará tanto el **Model** como el **ResourceModel** creados anteriormente.

ResourceModel es necesario para cargar los datos provenientes de nuestras tablas de la base de datos y permitiéndonos filtrar los registros si esto fuera necesario.

La colección será representada como un array de Modelos que nos permitirá controlar todas sus funcionalidades (métodos mágicos para tratar datos, inclusión, salvado y borrado de datos).

```
<?php
namespace Ilovemarketing\News\Model\ResourceModel\News;

use Magento\Framework\Model\ResourceModel\Db\Collection\AbstractCollection;

class News extends AbstractCollection {

   protected $_idFieldName = 'news_id';

   protected function _construct() {

      $this->_init(
           'Ilovemarketing\News\Model\News',
           'Ilovemarketing\News\Model\ResourceModel\News'
      );
   }
}
```

Creando tablas usando Setups

Cuando estamos usando modelos de base de datos en nuestros módulos, sus tablas necesitan ser creadas de alguna forma y para ello usaremos setups.

Todas los scripts de creación de tablas son incluidos en estos setups y se ejecutarán durante la instalación de una extensión.

Hay cuatro ficheros incluidos para crear Schemas e inserción de datos y que se ejecutarán por orden durante la activación de un módulo.

- Setup/**InstallSchema.php**
- Setup/**UpgradeSchema.php**

La instalación se ejecutará siempre y cuando no existan registros en la tabla **setup_module** que contiene la información de las versiones de nuestro módulo.

Los Upgrades solo son ejecutados cuando la versión **schema_version** incluida en nuestro **setup_module** es inferior a la versión configurada en nuestro **/etc/module.xml**

Cuando es necesario insertar datos o incluir nuevos atributos tenemos que hacerlo en los siguientes ficheros:

- Setup/**InstallData.php**
- Setup/**UpgradeData.php**

InstallShema

Al igual que los anteriores su **data_version** es almacenada en la tabla **setup_module.** Vamos a crear un **InstallSchema** para nuestro módulo en el que crearemos una tabla de noticias, una tabla de categorías y otra tabla de tipos de noticias:

```
<?php
namespace Ilovemarketing\News\Setup;

use Magento\Framework\Setup\InstallSchemaInterface;
use Magento\Framework\Setup\ModuleContextInterface;
use Magento\Framework\Setup\SchemaSetupInterface;
use Magento\Framework\DB\Adapter\AdapterInterface;
use Magento\Framework\DB\Ddl\Table;
```

```
class InstallSchema implements InstallSchemaInterface {

    private $_table = [
        'news' => 'ilovemarketing_news',
        'news_types' => 'ilovemarketing_news_types',
        'news_categories' => 'ilovemarketing_news_categories'
    ];

    public function install(
            SchemaSetupInterface $setup,
            ModuleContextInterface $context) {

        $installer = $setup;
        $installer->startSetup();

        if
(!$installer->getConnection()->isTableExists($setup->getTable($this->_table['news']))) {

            $table = $installer->getConnection()
                ->newTable($installer->getTable($this->_table['news']))
                ->addColumn('news_id', Table::TYPE_SMALLINT, null, ['identity' => true,
'nullable' => false, 'primary' => true, 'unsigned' => true], 'News ID')
                ->addColumn('news_type_id', Table::TYPE_SMALLINT, null, ['nullable' => false,
'unsigned' => true], 'News Type ID')
                ->addColumn('news_title', Table::TYPE_TEXT, 255, ['nullable' => false], 'News
Title')
                ->addColumn('news_content', Table::TYPE_TEXT, '2M', [], 'News Content')
                ->addColumn('news_date', Table::TYPE_TIMESTAMP, null, ['nullable' => false,
'default' => Table::TIMESTAMP_INIT], 'News Creation Time')
                ->addIndex($setup->getIdxName($installer->getTable($this->_table['news']),
['news_title'], AdapterInterface::INDEX_TYPE_FULLTEXT), ['news_title'], ['type' =>
AdapterInterface::INDEX_TYPE_FULLTEXT])
                ->setComment('ilovemarketing_news - Table')
                ->setOption('type', 'InnoDB')
                ->setOption('charset', 'utf8');

            $installer->getConnection()->createTable($table);

        }

        if
(!$installer->getConnection()->isTableExists($setup->getTable($this->_table['news_types']
))) {

            $table = $installer->getConnection()
                ->newTable($installer->getTable($this->_table['news_types']))
```

```
            ->addColumn('news_type_id', Table::TYPE_SMALLINT, null, ['identity' => true,
'nullable' => false, 'primary' => true, 'unsigned' => true], 'News Type ID')
            ->addColumn('news_type_name', Table::TYPE_TEXT, 255, ['nullable' => false],
'News Type Name')
            ->setComment('ilovemarketing_news_types - Table')
            ->setOption('type', 'InnoDB')
            ->setOption('charset', 'utf8');

        $installer->getConnection()->createTable($table);

    }

    if
(!$installer->getConnection()->isTableExists($setup->getTable($this->_table['news_categ
ories']))) {

        $table = $installer->getConnection()
            ->newTable($installer->getTable($this->_table['news_categories']))
            ->addColumn('news_categorie_id', Table::TYPE_SMALLINT, null, ['identity' =>
true, 'nullable' => false, 'primary' => true, 'unsigned' => true], 'News Categorie ID')
            ->addColumn('news_categorie_name', Table::TYPE_TEXT, 255, ['nullable' =>
false], 'News Categorie Name')
            ->setComment('ilovemarketing_news_categories - Table')
            ->setOption('type', 'InnoDB')
            ->setOption('charset', 'utf8');

        $installer->getConnection()->createTable($table);

    }

    $installer->endSetup();
  }
}
```

Cuando ejecutamos un upgrade desde línea de comandos todos nuestros módulos son evaluados para comprobar su actual versión en sus respectivos ficheros de configuración.

Primero se ejecutarán los Install y Upgrades de nuestros Schemas, para finalizar con las Install y Upgrades de datos.

Para poder ejecutar y testar los ficheros de instalación durante una fase de pruebas podemos eliminar la línea de la tabla **setup_module** correspondiente a nuestro módulo.

```
php -dmemory_limit=1G ./bin/magento setup:upgrade
```

Todos los métodos para la creación de una tabla son declarados en la clase Magento\Framework\DB\ Adapter\AdapterInterface**Table**

- **addColumn**(): Este método añade nuevas columnas a la tabla pudiendo enviar los siguientes parámetros:
    - name: nombre del campo en la tabla
    - type: Este será el tipo de campo y pueden ser: TYPE_BOOLEAN TYPE_SMALLINT TYPE_INTEGER TYPE_BIGINT TYPE_FLOAT TYPE_NUMERIC TYPE_DECIMAL TYPE_DATE TYPE_TIMESTAMP TYPE_DATETIME TYPE_TEXT TYPE_BLOB TYPE_VARBINARY
    - size: el tamaño del campo
    - options: opciones sobre el campo:
        - unsigned: solo para campos numericos (true/false)
        - precision: Para campos numericos decimales.
        - scale: Para campos numéricos y decimales
        - default: El valor por defecto cuando creamos un nuevo registro.
        - nullable: en caso de que el campo sea NULL (por defecto es true)
        - primary: Establecer la columna como clave primaria.
        - primary_position: Para campos con claves primarias y establecerá su orden.
        - identity/auto_increment: Hacer auto incremental una columna.
    - comment: descripción de la columna.
- **addForeignKey**: Añade una clave foranea para relacionarla con otra tabla.
    - fkName: El nombre de la clave foranea.
    - column: Columna para la clave foranea.
    - refTable: Tabla donde estarán sus referencias.
    - refColumn: El nombre de la columna en la tabla refererenciada.
    - onDelete: Podemos establecer una acción en caso de ser borrado un registro.
- **addIndex**: Añade a una columna un indice para búsquedas (lo necesitaremos para los campos de tipo búsqueda funcionen correctamente en el backend).
    - indexName: Nombre del índice.
    - fields: Columnas usadas para nuestro indice. Puede ser una columna o un array de columnas.
    - options: un array con configuraciones adicionales.

Cuando necesitemos cambiar la forma de una tabla, podemos usar los siguientes métodos. Estos métodos pueden ser usados directamente sobre la clase $installer->**getConnection**().

- **dropTable**: elimina una tabla de la base de datos.
    - tableName: nombre de la tabla.
    - schemaName: Nombre Schema de la tabla (opcional)
- **renameTable**: renombra una tabla de la base de datos.
    - oldTableName: El nombre actual de la tabla.
    - newTableName: El nuevo nombre de la tabla.
    - schemaName: El nombre Schema (opcional)
- **addColumn**: Añade una columna extra a la tabla actual.
    - tableName: nombre de la tabla que vamos a modificar.
    - columnName: nombre de la nueva columna.
    - definition: un array de parámetros.
        - Type: tipo de columna
        - Length: Tamaño de la columna.
        - Default: Valor por defecto.
        - Nullable: Si la columna puede ser NULL
        - Identify/Auto_Increment: hacer única la columna.
        - Comment: descripcion
        - After: El lugar donde se incluirá la columna.
    - schemaName: El nombre Schema (opcional)
- **changeColumn**: cambia el nombre de una columna
    - tableName: el nombre de la tabla a cambiar.
    - oldColumnName: el nombre de la columna que deseamos cambiar.
    - newColumnName: El nuevo nombre para la columna.
    - definition: array de opciones (ver addColumn)
    - flushData: Hace un flush de la cache de la tabla.
    - schemaName: Nombre Schema (opcional)
- **modifyColumn**: Cambia la definición de una columna.
    - tableName: el nombre de la tabla
    - columnName: nombre de la columna
    - definition: array de opciones (ver addColumn)
    - flushData: flush de la cache de la tabla.

    - schemaName: Nombre Schema.

- **dropColumn**: elimina una columna de la tabla.

    - tableName: nombre de la tabla
    - columnName: nombre de la columna que deseamos eliminar.
    - schemaName: Nombre schema.

- **addIndex**: Añade un índice.

    - tableName: Nombre de la tabla.
    - indexName: nombre del indice a incluir.
    - fields: columnas que seran usadas como indice.
    - indexType: tipo de indice. INDEX_TYPE_PRIMARY INDEX_TYPE_UNIQUE INDEX_TYPE_INDEX INDEX_TYPE_FULLTEXT
    - schemaName: Nombre Schema.

- **dropIndex**: Elimina un índice de una tabla.

    - tableName: nombre de la tabla.
    - indexName: Nombre del índice.
    - schemaName: Nombre Schema.

- **addForeignKey**: Añade una nueva clave foránea.

    - fkName: nombre de la clave foránea.
    - tableName: Nombre de la tabla.
    - columnName: Nombre de la columna para usar como clave foránea.
    - refTableName: Nombre de la tabla referenciada.
    - refColumnName: Nombre de la columna de la tabla referenciada.
    - onDelete: ejecutará una acción en caso de borrado.
    - purge: remueve datos incorrectos (por defecto false)
    - schemaName: nombre Schema.
    - refSchemaName: Nombre del Schema referenciado.

Upgrades

Vamos a crear un fichero de Upgrade para incluir unas nueva columna a la tabla indicada.

Este fichero se ejecutará si la versión de nuestro módulo es diferente al instalado por lo que es mejor comprobar antes la versión instalada antes de ejecutar las actualizaciones necesarias.

```
if (version_compare($context->getVersion(), '0.0.1', '<')) {
```

En este caso el UpgradeSchema se ejecutará solo si la versión actual de nuestro module.xml es inferior a la 0.0.1.

```
<?php
namespace Ilovemarketing\News\Setup;

use Magento\Framework\Setup\UpgradeSchemaInterface;
use Magento\Framework\Setup\ModuleContextInterface;
use Magento\Framework\Setup\SchemaSetupInterface;

class UpgradeSchema implements UpgradeSchemaInterface {

  private $_table = [
     'news' => 'ilovemarketing_news',
     'news_types' => 'ilovemarketing_news_types',
     'news_categories' => 'ilovemarketing_news_categories'
  ];

   public function upgrade(
         SchemaSetupInterface $setup,
         ModuleContextInterface $context) {

      $installer = $setup;
      $installer->startSetup();

      // update version
      // en este upgrade vamos a incluir un nuevo campo
      if (version_compare($context->getVersion(), '0.0.1', '<')) {

         $tableName = $installer->getTable($this->_table['news']);
         $installer->getConnection()->addColumn($tableName, 'news_comment', [
            'type' => \Magento\Framework\DB\Ddl\Table::TYPE_TEXT,
            'length'    => 255,
```

```
            'unsigned' => true,
            'nullable' => false,
            'default' => '0',
            'comment' => 'Comment'
        ]);

        // update de nuestra version
        // en este upgrade vamos acrear unos indexers para poder realizar busquedas

        // otro forma de incluir una columna
        $column = [
          'type' => \Magento\Framework\DB\Ddl\Table::TYPE_SMALLINT,
          'length' => 1,
          'nullable' => false,
          'comment' => 'Es visible',
          'default' => '1'
        ];

        $installer->getConnection()->addColumn($this->_table['news'], 'news_visible',
$column);

      }

      $installer->endSetup();
   }
}
```

InstallData

Y por último creamos un **InstallData.php** para insertar datos a nuestras tablas.

```
<?php
namespace Ilovemarketing\News\Setup;

use Magento\Framework\Module\Setup\Migration;
use Magento\Framework\Setup\InstallDataInterface;
use Magento\Framework\Setup\ModuleContextInterface;
use Magento\Framework\Setup\ModuleDataSetupInterface;

class InstallData implements InstallDataInterface {

  private $_table = [
      'news' => 'ilovemarketing_news',
      'news_types' => 'ilovemarketing_news_types',
```

```
    'news_categories' => 'ilovemarketing_news_categories'
];


public function install(
        ModuleDataSetupInterface $installer,
        ModuleContextInterface $context) {

    $installer->startSetup();
    if (version_compare($context->getVersion(), '0.0.1', '<')) {

        $data = [
            ['Noticia 1', 'Noticia de prueba 1'],
            ['Noticia 2', 'Noticia de prueba 2'],
            ['Noticia 3', 'Noticia de prueba 3'],
            ['Noticia 4', 'Noticia de prueba 4'],
            ['Noticia 5', 'Noticia de prueba 5'],
            ['Noticia 6', 'Noticia de prueba 6'],
            ['Noticia 7', 'Noticia de prueba 7'],
            ['Noticia 8', 'Noticia de prueba 8'],
            ['Noticia 9', 'Noticia de prueba 9'],
            ['Noticia 10', 'Noticia de prueba 10']
            ];

        foreach ($data as $key => $dataField) {

            $bind = [
                'news_title' => $dataField[0],
                'news_content' => $dataField[1]
            ];
            $installer->getConnection()->insert(
                $installer->getTable($this->_table['news']),
                $bind
            );
        }

        // content news_types
        $data = [
            ['Borrrador'],
            ['Publicada'],
            ['Papelera']
        ];

        foreach ($data as $key => $dataField) {

            $bind = [
```

```
                'news_type_name' => $dataField[0]
            ];
            $installer->getConnection()->insert(
                $installer->getTable($this->_table['news_types']),
                $bind
            );
        }

        // content news_categories
        $data = [
            ['Noticias']
        ];

        foreach ($data as $key => $dataField) {

            $bind = [
                'news_categorie_name' => $dataField[0]
            ];
            $installer->getConnection()->insert(
                $installer->getTable($this->_table['news_categories']),
                $bind
            );
        }
    }
    $installer->endSetup();
  }
}
```

Visualización de datos

En pasos anteriores vimos cómo trabajar con modelos para trabajar con nuestras tablas de la base de datos y la creación de setups para poder crear dichas tablas y sus datos.

El siguiente paso es modificar nuestra ruta y controlador que creamos inicialmente para configurar una ruta y controlador para poder consultarlos desde nuestro frontend. Vamos a trabajar con los siguientes ficheros:

/app/code/llovemarketing/News/etc/frontend/routes.xml
/app/code/llovemarketing/News/Controller/Index/Index.php
/app/code/llovemarketing/News/view/frontend/layout/news_index_index.xml
/app/code/llovemarketing/News/Block/NewsList.php
/app/code/llovemarketing/News/view/frontend/template/news.phtml

Rutas

El fichero de rutas que creamos inicialmente en el curso lo vamos a dejar tal cual lo teníamos.

```
<?xml version="1.0" encoding="UTF-8"?>

<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:App/etc/routes.xsd">
   <router id="standard">
      <route id="news" frontName="news">
         <module name="llovemarketing_News" />
      </route>
   </router>
</config>
```

Controlador

El controlador que creamos inicialmente vamos a modificarlo para que muestre el theme correctamente:

```
<?php
namespace llovemarketing\News\Controller\Index;

use Magento\Framework\App\Action\Action;
```

```
use Magento\Framework\App\Action\Context;
use Magento\Framework\View\Result\PageFactory;

class Index extends Action {

   protected $_pageFactory;

   public function __construct(
         Context $context,
         PageFactory $pageFactory) {

      $this->_pageFactory = $pageFactory;
      parent::__construct($context);

   }

   public function execute() {

      $page = $this->_pageFactory->create();
      return $page;

   }
}
```

Layout

Trabajaremos con el fichero de layout que es el encargado de preparar lo que se va a mostrar.

En este caso creará un nuevo bloque dentro del contenedor "content" y enviará los datos al template que le indiquemos. Nuestro layout tiene que estar ubicado en la carpeta **view/frontend/layout/** y lo llamaremos **news_index_index.xml** que corresponde a [ruta]/[controlador]/[acción].

- Ruta corresponde al **frontname** que declaramos en nuestro fichero de rutas.
- El controlador es el **path** donde está ubicado (Controller/Index/)
- Y la acción es la **clase PHP** de nuestro controlador que está ubicada en dicho path.

En este caso cuando ejecutamos http://nuestrodominio.com/news/ este estará enviando una petición a http://nuestrodominio.com/news/**index/index**

```
<?xml version="1.0" encoding="UTF-8"?>

<page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" layout="1column"
xsi:noNamespaceSchemaLocation="urn:magento:framework:View/Layout/etc/page_con
figuration.xsd">
   <body>
      <referenceContainer name="content">
         <block class="Ilovemarketing\News\Block\NewsList" name="NewsList"
template="Ilovemarketing_News::newsList.phtml" />
      </referenceContainer>
   </body>
</page>
```

Bloque

Ahora procedemos a crear el nuevo bloque que hemos declarado en el layout. Los bloques los crearemos en la carpeta Block/.

```
<?php
namespace Ilovemarketing\News\Block;

use Magento\Framework\View\Element\Template;
use Magento\Framework\View\Element\Template\Context;
use Ilovemarketing\News\Model\ResourceModel\News\NewsFactory;

class NewsList extends Template {

   public function __construct( Context $context,
                        NewsFactory $newsFactory ) {

      $this->_newsFactory = $newsFactory;
      parent::__construct($context);

   }

   public function getNews()  {

      $news = $this->_newsFactory->create();
      $news->addFieldToFilter('news_visible', 1);
      $news->setOrder('news_date','desc');

      return $news;
   }
```

```
}
```

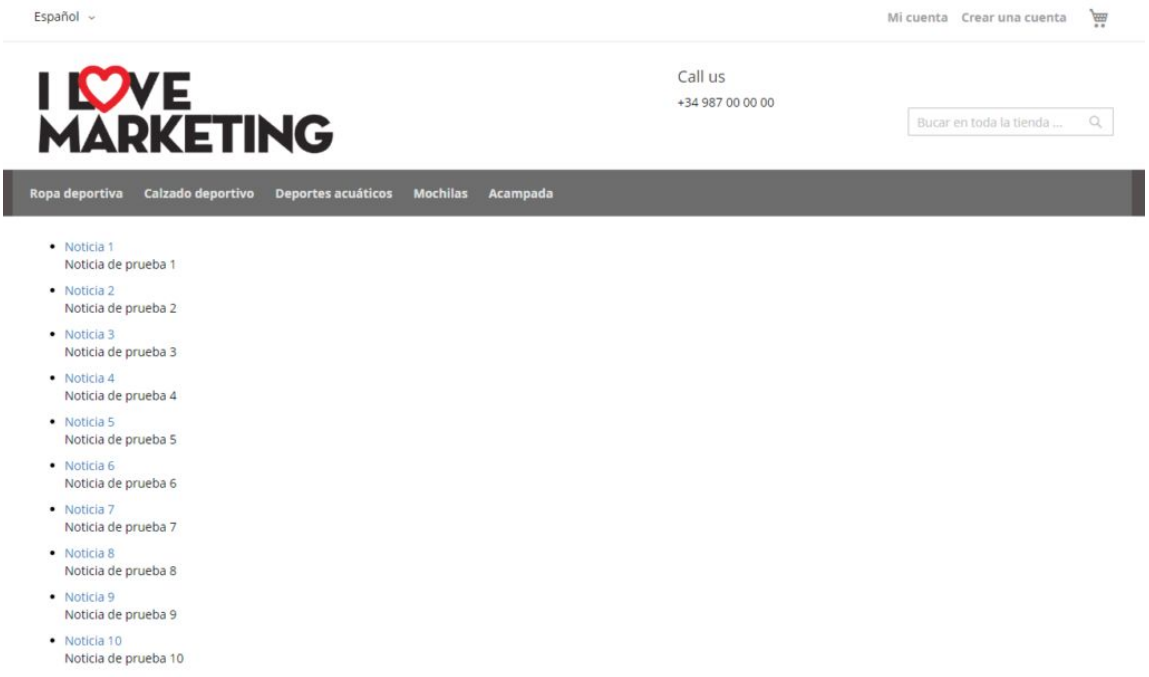
Template

El paso final es crear nuestro template en la carpeta de vistas /view/frontend/templates/ y al que llamaremos newsList.phtml

```
<?php
if (!emtpy($block->getNews())) { ?><ul><?php
   foreach($block->getNews() as $item) { ?>
      <li>
         <a href="index/view/id/<?php echo $item["news_id"]; ?>"><?php echo
$item["news_title"]; ?></a>
         <p><?php echo $item["news_content"]; ?></p>
      </li>
   <?php
   } ?></ul><?php
}
```

El el siguiente tema vamos a ir viendo cómo incluir paginadores en el caso de que tengamos muchos registros y cómo podemos entrar al detalle de cada registro enviando parámetros.

Si visitamos http://nuestrodominio.com/news/ nuestra página de noticias renderizará así:



Paginación de registros

Ya hemos visto anteriormente cómo listar registros provenientes de las tablas de nuestra base de datos usando controladores, bloques, layouts y templates.

Cuando la cantidad de datos es superior a la que queremos mostrar los filtraremos usando paginadores. Para poder realizar dicha tarea modificaremos los siguientes ficheros que corresponden a nuestro Bloque y Template:

/app/code/Ilovemarketing/News/Block/NewsList.php
/app/code/Ilovemarketing/News/view/frontend/newsList.php

Bloque

Empezaremos modificando nuestro bloque **NewsList.php**

```
<?php
namespace Ilovemarketing\News\Block;

use Magento\Framework\View\Element\Template;
use Magento\Framework\View\Element\Template\Context;
use Ilovemarketing\News\Model\ResourceModel\News\NewsFactory;

class NewsList extends Template {

   public function __construct( Context $context,
                        NewsFactory $newsFactory ) {

      $this->_newsFactory = $newsFactory;
      parent::__construct($context);

      // title, metas
      $this->pageConfig->getTitle()->set(__('Listado de Noticias'));
      $this->pageConfig->setDescription($this->getConfig('Listado de Noticias'));
      $this->pageConfig->setKeywords($this->getConfig('Listado de Noticias'));

   }

   public function getNews()  {

      $this->_limit = 5;
```

```
        // Sino hay parametros recibido de pagina, defecto 1
        $page=($this->getRequest()->getParam('p'))? $this->getRequest()->getParam('p') : 1;

        // limite de registros
        $pageSize = ($this->getRequest()->getParam('limit'))?
$this->getRequest()->getParam('limit') : $this->_limit;

        $news = $this->_newsFactory->create();
        $news->addFieldToFilter('news_visible',1)
            ->setOrder('news_date','desc')
            ->setPageSize($pageSize)
            ->setCurPage($page);

        return $news;
    }

    /*
     * El metodo callBack _prepareLayout()
     * Se lanza despues de que un bloque ha sido añadido al layout
     */
    protected function _prepareLayout() {

        parent::_prepareLayout();

        if ($this->getNews()) {

            $pager = $this->getLayout()->createBlock(
                'Magento\Theme\Block\Html\Pager',
                'news.news.pager'
            )->setAvailableLimit(
                    [
                        5=>5,
                        10=>10,
                        15=>15
                    ]
             )
             ->setShowPerPage(true)
             ->setLimit($this->_limit)
             ->setCollection($this->getNews());

            $this->setChild('pager', $pager);

        }
    }

    /*
```

```
     * método callbBack getPagerHtml()
     * Será llamado antes de que el bloque html sea renderizado
     */
    public function getPagerHtml() {

        return $this->getChildHtml('pager');

    }
}
```

Template

Y por último incluimos la llamada a nuestra función **getPagetHtml()** en nuestro template:

```
<?php
if ($block->getPagerHtml()) { ?>
   <div class="toolbar top"><?php echo $block->getPagerHtml(); ?></div><?php
}

if (!empty($block->getNews())) { ?><ul><?php
    foreach($block->getNews() as $item) { ?>
        <li>
            <a href="index/view/id/<?php echo $item["news_id"]; ?>"><?php echo
$item["news_title"]; ?></a>
            <p><?php echo $item["news_content"]; ?></p>
        </li>
    <?php
    } ?></ul><?php
}
```

Y nuestro paginador pinta de la siguiente forma:

Acceso a enlaces con parámetros.

Ahora que tenemos nuestros listados querremos acceder al detalle de cada noticia o registro.

Para ello tenemos que crear un nuevo controlador, bloque, layout y template para visualizar la vista de detalle de nuestra noticia.

Vamos a trabajar con los siguientes ficheros:

/app/code/Ilovemarketing/News/Controller/Index/View.php
/app/code/Ilovemarketing/News/view/frontend/layout/view_index_index.xml
/app/code/Ilovemarketing/News/view/frontend/templates/newsView.phtml
/app/code/Ilovemarketing/News/Block/NewsView.php

Controlador

El primer paso es crear un nuevo controlador para nuestra vista de detalle de noticia. La nueva clase la crearemos en **Controller/Index** y la llamaremos **View.php**.

Esta va a extender al controlador principal que creamos anteriormente para listar las noticias. De esta manera nos evitamos repetir las declaraciones modelos u otras clases y métodos.

```
<?php
namespace Ilovemarketing\News\Controller\Index;

use Ilovemarketing\News\Controller\Index\Index;

class View extends Index {

    public function execute() {

        $page = $this->_pageFactory->create();
        return $page;

    }
}
```

Layout

Preparamos nuestro nuevo layout en **view/frontend/layout/** y lo llamaremos **news_index_view.xml** el que definimos la creación de un nuevo bloque que renderizará los datos capturados en la vista de template indicada.

```
<?xml version="1.0" encoding="UTF-8"?>

<page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" layout="1column"
xsi:noNamespaceSchemaLocation="urn:magento:framework:View/Layout/etc/page_con
figuration.xsd">
   <body>
      <referenceContainer name="content">
         <block class="Ilovemarketing\News\Block\NewsView" name="NewsView"
template="Ilovemarketing_News::newsView.phtml" />
      </referenceContainer>
   </body>
</page>
```

Bloque

Creamos el nuevo Bloque en **/Block** llamado **NewsView.php** que se encargará de hacer la petición a nuestra tabla para así poder consultar los datos.

```
<?php
namespace Ilovemarketing\News\Block;

use Magento\Framework\View\Element\Template;
use Magento\Framework\View\Element\Template\Context;
use Ilovemarketing\News\Model\NewsFactory;

class NewsView extends Template {

   protected $_newsFactory;

   public function __construct( Context $context,
                        NewsFactory $newsFactory ) {

      $this->_newsFactory = $newsFactory;
      parent::__construct($context);

   }
```

```
    public function getNewsId(){

        $news_id = $this->getRequest()->getParam('id');
        $news = $this->_newsFactory->create();
        $news->load($news_id);

        return $news;

    }

    protected function _prepareLayout() {

        return parent::_prepareLayout();

    }
}
```

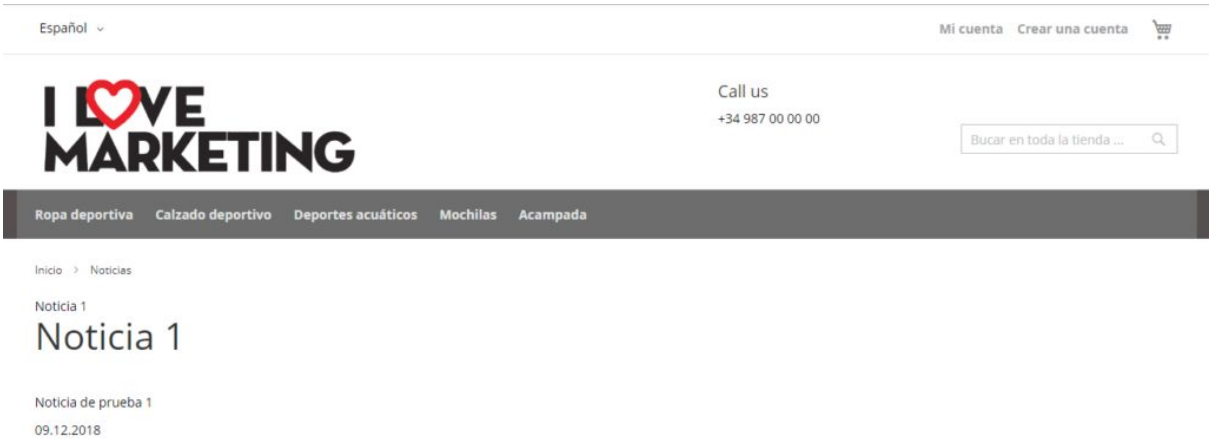
Template

Y por último creamos el template en **/view/frontend/templates** y al que llamaremos **newsView.phtml** y que mostrará nuestro detalle de noticia .

```
<?php
$news = $this->getNewsId();
$fecha = \DateTime::createFromFormat('Y-m-d H:i:s', $news["news_date"]);
$block->setTitle(__('Some page title')); ?>

<div class="news">
    <h1><?php echo $news["news_title"]; ?></h1>
    <p><?php echo $news["news_content"]; ?></p>
    <p><?php echo $fecha->format('d.m.Y'); ?></p>
</div>
```

Nuestra vista de noticia se visualiza de la siguiente forma en la url:

- http://tudominio.com/**news/index/view/id/1**
-



Menú de migas de pan

Vamos a incluir un menú de migas de pan a nuestro módulo. Lo podemos hacer desde el controlador de noticias o desde el mismo bloque.

Desde un Bloque

Declaramos la clase **UrlInterface** y que nos permitirá usar los métodos **getBaseUrl**(), **getUrl**(), **getCurrentUrl**(), **getUrl**('test/test2') , etc ... ya que necesitaremos usarlos en nuestro menú de migas de pan.

En un controlador no necesitamos hacer esto ya que **ObjectManagerInterface** es llamado por la factoría de páginas **PageFactory**.

```
<?php
use Magento\Framework\UrlInterface;
```

En el constructor de nuestro bloque inyectamos la dependencia **UrlInterface** para poder utilizarla.

```
public function __construct( Context $context
                    UrlInterface $urlInterface) {

   $this->_urlInterface = $urlInterface;
   parent::__construct($context);

}
```

Creamos el método **addbreadcrumb**() en nuestro bloque y que se encargará de generar el menú de migas de pan personalizado (**breadcrumb**) en nuestra página. Por lo que nuestro bloque newsList.php quedaría de la siguiente forma.

```
<?php
namespace Ilovemarketing\News\Block;

use Magento\Framework\View\Element\Template;
use Magento\Framework\View\Element\Template\Context;
use Magento\Framework\UrlInterface;
use Ilovemarketing\News\Model\ResourceModel\News\NewsFactory;

class NewsList extends Template {
```

```
    public function __construct( Context $context,
                          NewsFactory $newsFactory,
                          UrlInterface $urlInterface) {

        $this->_newsFactory = $newsFactory;
        $this->_urlInterface = $urlInterface;

        parent::__construct($context);

    }

    public function addbreadcrumb() {

        // @var \Magento\Theme\Block\Html\Breadcrumbs
        $breadcrumbs = $this->getLayout()->getBlock('breadcrumbs');
        $breadcrumbs->addCrumb('home',
            [
                'label' => __('Inicio'),
                'title' => __('Inicio'),
                'link' => $this->_urlInterface->getUrl()
            ]
        );
        $breadcrumbs->addCrumb('news',
            [
                'label' => __('Noticias'),
                'title' => __('Noticias'),
                'link' => $this->_urlInterface->getCurrentUrl()
            ]
        );

        return $this->getLayout()->getBlock('breadcrumbs')->toHtml();
    }

    public function getNews()  {

        // get limit fields from configuration
        $this->_limit = 5;

        // Sino ha parametro recibido de pagina, defecto 1
        $page=($this->getRequest()->getParam('p'))? $this->getRequest()->getParam('p') : 1;

        // limite de registros
        $pageSize = ($this->getRequest()->getParam('limit'))?
$this->getRequest()->getParam('limit') : $this->_limit;

        $news = $this->_newsFactory->create();
```

```
        $news->addFieldToFilter('news_visible',1);
        $news->setOrder('news_date','desc');
        $news->setPageSize($pageSize);
        $news->setCurPage($page);
        return $news;
    }

    /*
     * El metodo callBack _prepareLayout()
     * Se lanza despues de que un bloque ha sido añadido al layout
     */
    protected function _prepareLayout() {

        parent::_prepareLayout();

        // get the title from configuration file
        $this->pageConfig->getTitle()->set(__('Mi Módulo de noticias'));
        // meta description from configuration file
        $this->pageConfig->setDescription('Mi Módulo de noticias');
        // meta keywords from configuration file
        $this->pageConfig->setKeywords('noticias');

        if ($this->getNews()) {

            $pager = $this->getLayout()->createBlock(
                'Magento\Theme\Block\Html\Pager',
                'test.news.pager'
            )->setAvailableLimit(array(5=>5,10=>10,15=>15))
             ->setShowPerPage(true)
             ->setLimit($this->_limit)
             ->setCollection($this->getNews());

            $this->setChild('pager', $pager);
            $this->getNews()->load();
        }
        return $this;
    }

    /*
     * método callbBack getPagerHtml()
     * Será llamado antes de que el bloque html sea renderizado
     */
    public function getPagerHtml() {
        return $this->getChildHtml('pager');
    }
```

```
}
```

Y por último llamamos al método **addbreadcrumb**() que acabamos de crear desde el **template** que muestra nuestras noticias:

```
<?php
if ($block->addbreadcrumb()) {
   echo $block->addbreadcrumb();
}
//echo $block->getUrl();

if ($block->getPagerHtml()) { ?>
  <div class="toolbar top"><?php echo $block->getPagerHtml(); ?></div><?php
}

if (!empty($block->getNews())) { ?><ul><?php
   foreach($block->getNews() as $item) { ?>
      <li>
         <a href="index/view/id/<?php echo $item["news_id"]; ?>"><?php echo
$item["news_title"]; ?></a>
         <p><?php echo $item["news_content"]; ?></p>
      </li>
   <?php
   } ?></ul><?php
}
```

Desde un controlador

Desde el controlador es más sencillo, tan solo tenemos que crear nuestro método encargado de generar las migas de pan y llamarlo después de crear la página.

```
<?php
namespace Ilovemarketing\News\Controller\Index;

use Magento\Framework\App\Action\Action;
use Magento\Framework\App\Action\Context;
use Magento\Framework\View\Result\PageFactory;

class Index extends Action {

   protected $_pageFactory;

   public function __construct(
```

```
        Context $context,
        PageFactory $pageFactory) {

    $this->_pageFactory = $pageFactory;
    parent::__construct($context);

  }

  public function addbreadcrumb() {

    $pageFactory = $this->_pageFactory->create();

    // @var \Magento\Theme\Block\Html\Breadcrumbs
    $breadcrumbs = $pageFactory->getLayout()->getBlock('breadcrumbs');
    $breadcrumbs->addCrumb('home',
      [
        'label' => __('Inicio'),
        'title' => __('Inicio'),
        'link' => $this->_url->getUrl('/')
      ]
    );
    $breadcrumbs->addCrumb('news',
      [
        'label' => __('Noticias'),
        'title' => __('Noticias'),
        'link' => $this->_url->getUrl('news')
      ]
    );
  }

  public function execute() {

    $page = $this->_pageFactory->create();
    $this->addbreadcrumb();
    return $page;

  }
}
```

Interfaz de menús del Backend

Ya tenemos parte de nuestro módulo visible para el usuario pero ahora necesitamos crear una zona donde gestionar nuestro módulo. Para ello necesitamos crear una interfaz en el backend de magento y que nos permitirá administrar nuestro módulo.

Vamos a trabajar con los siguientes ficheros para poder crear dicha interfaz:

/app/code/Ilovemarketing/News/etc/adminhtml/routes.xml
/app/code/Ilovemarketing/News/etc/adminhtml/menu.xml
/app/code/Ilovemarketing/News/Controller/Adminhtml/News/Index.php

Rutas

El primer fichero que vamos a crear es **routes.xml**. Al igual que en el frontend necesitamos un fichero de rutas para indicarle como acceder a nuestro módulo.

```
<?xml version="1.0" encoding="UTF-8"?>

<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:App/etc/routes.xsd">
   <router id="admin">
      <route id="news" frontName="news">
         <module name="Ilovemarketing_News" />
      </route>
   </router>
</config>
```

Layout

Una vez que tenemos la ruta el siguiente paso es crear la interfaz para poder administrarlo. Para ello tenemos que crear el fichero **menu.xml** que contendrá los menús de nuestro módulo.

```
<?xml version="1.0" encoding="UTF-8"?>

<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Backend:etc/menu.
xsd">
   <menu>
```

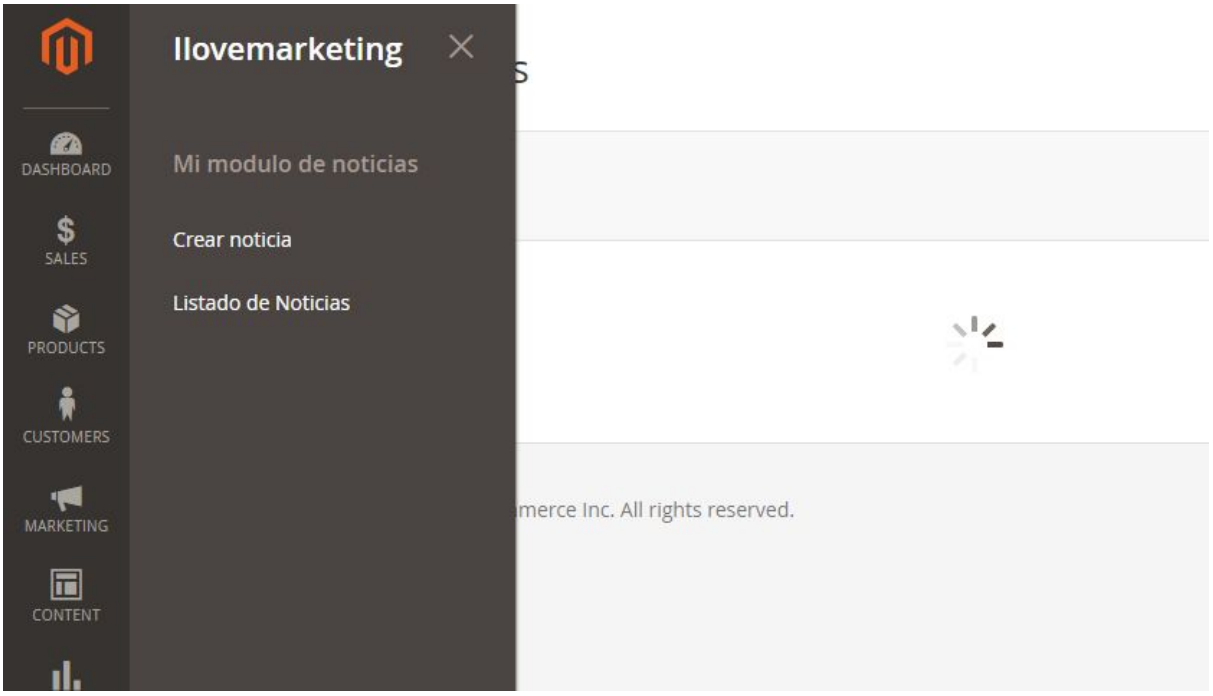
```
        <add id="llovemarketing_News::vendor" title="llovemarketing"
module="llovemarketing_News" sortOrder="250"
resource="llovemarketing_News::vendor" />
        <add id="llovemarketing_News::news" title="Mi modulo de noticias"
module="llovemarketing_News" sortOrder="1" parent="llovemarketing_News::vendor"
resource="llovemarketting_News::news" />
        <add id="llovemarketing_News::newsAdd" title="Crear noticia"
module="llovemarketing_News" sortOrder="2" parent="llovemarketing_News::news"
resource="llovemarketing_News::newsAdd" action="news/news/add" />
        <add id="llovemarketing_News::newsList" title="Listado de Noticias"
module="llovemarketing_News" sortOrder="3" parent="llovemarketing_News::news"
resource="lllovemarketing_News::newsList" action="news/news/index" />
    </menu>
</config>
```

Los atributos que tenemos que tener en cuenta de este fichero son:

- **id**: identificador único para la nueva entrada del menú.
- **title** : es el titulo del menú y que será visible en nuestro backoffice.
- **module**: el modulo al que pertenece el menú.
- **action**: acción realizará el módulo. Dónde sera redireccionado al hacer click en el.
- **parent**: es el identificador padre si este fuera hijo de otro.
- **sortOrder**: prioridad en el menú.
- **dependsOnModule**: Visualizara la pestaña siempre y cuando el módulo esté presente.

Limpiamos la caché y comprobamos que nuestro interfaz se muestra correctamente.

Nuestro listado de noticias se muestra vacío pero en el siguiente tema veremos cómo crear una grilla para listar los registros.



CRUD (grilla de datos)

En este paso vamos a crear una grilla al estilo **magento2** para listar las noticias de nuestra tabla.

Para ello vamos a utilizar esta estructura de ficheros e iremos explicando paso a paso el funcionamiento de cada uno de ellos:

/app/code/Ilovemarketing/News/Controller/Adminhtml/News/Index.php
/app/code/Ilovemarketing/News/view/adminhtml/layout/news_news_index.xml
/app/code/Ilovemarketing/News/view/adminhtml/ui_component/test_test_listing
/app/code/Ilovemarketing/News/Ui/Component/Listing/Column/TestActions.php
/app/code/Ilovemarketing/News/etc/di.xml

Controlador

Empezaremos creando el Controlador **Controller/Adminhtml/News/** y al que llamaremos **Index.php**

```
<?php
namespace Ilovemarketing\News\Controller\Adminhtml\News;

use Magento\Backend\App\Action;
use Magento\Backend\App\Action\Context;
use Magento\Framework\View\Result\PageFactory;

class Index  extends Action  {

   protected $_pageFactory;

   public function __construct( Context $context,
                        PageFactory $pageFactory  ) {

      parent::__construct($context);
      $this->_pageFactory = $pageFactory;

   }

   public function execute() {

      $_page = $this->_pageFactory->create();
      $_page->getConfig()->getTitle()->prepend(__('Noticias'));
      return $_page;
```

```
    }

}
```

Layout

A continuación crearemos el layout que va a contener nuestra grilla. Para ello tenemos que crear una nueva carpeta **view/adminhtml/layout/** y crearemos el nuevo fichero de layout llamado **news_news_index.xml**.

- **news** es el frontname definido en nuestro fichero de rutas en etc/adminhtml/routes.xml
- **news** es el path donde esta nuestro controlador (en este caso Adminhtml/News)
- **index** es la clase que hemos creado para dicho controlador.

Por lo que la ruta accedida desde el navegador sera http://dominio.com/admin/news/news/index.

UiComponent

En el layout hacemos referencia al contenedor "**content**" y este contendrá nuestro **uiComponent listing** que se encargará de mostrar la grilla con nuestros datos.

```
<?xml version="1.0" encoding="UTF-8"?>

<page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:View/Layout/etc/page_con
figuration.xsd">
    <head>
        <title>Gestión de noticias</title>
    </head>
    <body>
        <referenceContainer name="content">
            <uiComponent name="news_news_listing"/>
        </referenceContainer>
    </body>
</page>
```

Vamos a crear el **componente UI listing** e iremos explicando todo su contenido. Para ello crearemos una nueva carpeta en **view/ui_component/** y el layout llamado **news_news_listing.xml.**

Vamos a intentar resumirlo lo más que se pueda y en los siguientes temas iremos incluyéndole funcionalidades.

```
<?xml version="1.0" encoding="UTF-8"?>

<listing xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Ui:etc/ui_configurati
on.xsd">

    <!-- argument
    data_source -> declaramos los data_sources en uso que conectan los links de tu grid
con la base de datos usando js_config.
    spinner -> es el nombre de las etiquetas de las columnas y que usara nuestro grid.
    buttons -> botones de nuestra grilla
    -->
    <argument name="data" xsi:type="array">

        <!-- definimos donde se encontrará el data source -->
        <item name="js_config" xsi:type="array">
            <item name="provider"
xsi:type="string">news_news_listing.news_news_listing_data_source</item>
            <item name="deps"
xsi:type="string">news_news_listing.news_news_listing_data_source</item>
        </item>

        <!-- definimos donde encontrar las columnas -->
        <item name="spinner" xsi:type="string">news_news_columns</item>

        <!-- definimos los botones -->
        <item name="buttons" xsi:type="array">
            <item name="add" xsi:type="array">
                <item name="name" xsi:type="string">add</item>
                <item name="label" xsi:type="string" translate="true">Add</item>
                <item name="class" xsi:type="string">primary</item>
                <item name="url" xsi:type="string">*/*/add</item>
            </item>
        </item>

    </argument>

    <!-- configuracion del DataSource
    Es un componente UI que nos proporciona las datos en un formato especifico
    Listing necesita el DataSource correctamente configurado para funcionar  -->
    <dataSource name="news_news_data_source">
        <argument name="dataProvider" xsi:type="configurableObject">
```

```
        <!-- unique name for the grid -->
        <argument name="class" xsi:type="string">NewsGridDataProvider</argument>
        <!-- name of the data source same as in argument/js_config/provider -->
        <argument name="name"
xsi:type="string">news_news_listing_data_source</argument>
        <argument name="primaryFieldName" xsi:type="string">news_id</argument>
        <argument name="requestFieldName" xsi:type="string">news_id</argument>
        <argument name="data" xsi:type="array">
          <item name="config" xsi:type="array">
            <item name="component"
xsi:type="string">Magento_Ui/js/grid/provider</item>
            <item name="update_url" xsi:type="url" path="mui/index/render"/>
            <item name="storageConfig" xsi:type="array">
              <item name="indexField" xsi:type="string">news_id</item>
            </item>
          </item>
        </argument>
      </argument>
    </dataSource>

    <!-- definimos la grilla
    definir el numero de columnas de la grilla
    el nombre de las columnas lo definimos anteriormente en el spinner -->
    <columns name="news_news_columns">

      <argument name="data" xsi:type="array">
        <item name="config" xsi:type="array">
          <item name="storageConfig" xsi:type="array">
            <item name="provider"
xsi:type="string">news_news_listing.news_news_listing.listing_top.bookmarks</item>
            <item name="namespace" xsi:type="string">current</item>
          </item>
          <item name="childDefaults" xsi:type="array">
            <item name="controlVisibility" xsi:type="boolean">true</item>
            <item name="storageConfig" xsi:type="array">
              <item name="provider"
xsi:type="string">news_news_listing.news_news_listing.listing_top.bookmarks</item>
              <item name="root" xsi:type="string">columns.${ $.index }</item>
              <item name="namespace" xsi:type="string">current.${
$.storageConfig.root}</item>
            </item>
          </item>
        </item>
      </argument>

      <!--
```

```
    incluye un checkbox delante de cada fila para permitir operaciones masivas tales
como el borrado -->
    <selectionsColumn name="ids">
      <argument name="data" xsi:type="array">
        <item name="config" xsi:type="array">
          <item name="indexField" xsi:type="string">news_id</item>
        </item>
      </argument>
    </selectionsColumn>

    <!-- Columna ID -->
    <column name="news_id">
      <argument name="data" xsi:type="array">
        <item name="config" xsi:type="array">
          <item name="filter" xsi:type="string">textRange</item>
          <item name="sorting" xsi:type="string">desc</item>
          <item name="label" xsi:type="string" translate="true">ID</item>
        </item>
      </argument>
    </column>

    <!-- columna news_date -->
    <column name="news_date">
      <argument name="data" xsi:type="array">
        <item name="config" xsi:type="array">
          <item name="visible" xsi:type="boolean">true</item>
          <item name="dataType" xsi:type="string">date</item>
          <item name="formElement" xsi:type="string">input</item>
          <item name="source" xsi:type="string">news_date</item>
          <item name="dataScope" xsi:type="string">news_date</item>
          <item name="label" xsi:type="string" translate="true">Date</item>
          <item name="filter" xsi:type="string">text</item>
          <item name="validation" xsi:type="array">
            <item name="required-entry" xsi:type="boolean">true</item>
          </item>
        </item>
      </argument>
    </column>

    <!-- conlumna news_title -->
    <column name="news_title">
      <argument name="data" xsi:type="array">
        <item name="config" xsi:type="array">
          <item name="visible" xsi:type="boolean">true</item>
          <item name="dataType" xsi:type="string">text</item>
          <item name="formElement" xsi:type="string">input</item>
```

```
            <item name="source" xsi:type="string">news</item>
            <item name="dataScope" xsi:type="string">news</item>
            <item name="label" xsi:type="string" translate="true">Title</item>
            <item name="filter" xsi:type="string">text</item>
            <item name="validation" xsi:type="array">
              <item name="required-entry" xsi:type="boolean">true</item>
            </item>
          </item>
        </argument>
      </column>

      <!-- columna news_content -->
      <column name="news_content">
        <argument name="data" xsi:type="array">
          <item name="config" xsi:type="array">
            <item name="visible" xsi:type="boolean">true</item>
            <item name="dataType" xsi:type="string">text</item>
            <item name="formElement" xsi:type="string">input</item>
            <item name="source" xsi:type="string">news</item>
            <item name="dataScope" xsi:type="string">news</item>
            <item name="label" xsi:type="string" translate="true">content</item>
            <item name="filter" xsi:type="string">text</item>
            <item name="validation" xsi:type="array">
              <item name="required-entry" xsi:type="boolean">true</item>
            </item>
          </item>
        </argument>
      </column>

      <!-- acciones
      add, delete, ...
      -->
      <actionsColumn name="actions"
class="Ilovemarketing\News\Ui\Component\Listing\Column\NewsActions">
        <argument name="data" xsi:type="array">
          <item name="config" xsi:type="array">
            <item name="resizeEnabled" xsi:type="boolean">false</item>
            <item name="resizeDefaultWidth" xsi:type="string">107</item>
            <item name="indexField" xsi:type="string">news_id</item>
          </item>
        </argument>
      </actionsColumn>
    </columns> <!-- fin de la grilla -->

</listing>
```

El componente **listing** se divide en 4 partes principales:

- **argument**: Donde declaramos los data_sources que vamos a usar (que conectan nuestra grilla con la base de datos) con el atributo js_config. También declaramos el **spinner** que es el nombre de la etiqueta que agrupa las columnas y que sera usado en nuestra grilla. Al final podremos declarar nuestros botones (sus etiquetas, acciones y clases).
- **dataSource**: En este área es donde definimos el dataProvider (que es el objeto que consulta los datos en la base de datos). En el atributo class definimos el objeto que será usado y que más adelante tendremos que definir en el fichero di.xml (que es el fichero de dependencias). Usando el atributo name le asignamos un nombre a nuestro dataSource en el que indicaremos el campo primario de nuestra tabla a consultar. Definimos el componente que tiene que usar en la configuracion (Magento_Ui/js/grid/provider) y el identificador de nuestra tabla.
- **columns**: fueron definidas en la sección de spinner de argument en el que le asignamos el nombre news_news_columns. En el definimos las columnas que se mostraran en nuestra grilla, el tipo, ordenación y nombres.

Actions

<actionsColumn> define las acciones a realizar en nuestro componente por lo que vamos a crearlo en Ilovemarketing\News**Ui\Component\Listing\Column** y lo llamaremos **NewsActions.php.** Todas las acciones extienden la clase **Column.**

```
<?php
namespace Ilovemarketing\News\Ui\Component\Listing\Column;

use Magento\Framework\View\Element\UiComponent\ContextInterface;
use Magento\Framework\View\Element\UiComponentFactory;
use Magento\Ui\Component\Listing\Columns\Column;
use Magento\Framework\UrlInterface;

class NewsActions extends Column {

    protected $_urlBuilder;

    public function __construct(
        ContextInterface $context,
        UiComponentFactory $uiComponentFactory,
        UrlInterface $urlBuilder,
        array $components = [],
        array $data = []
    ) {
```

```
        $this->_urlBuilder = $urlBuilder;
        parent::__construct($context, $uiComponentFactory, $components, $data);

    }

    public function prepareDataSource( array $dataSource ) {

        if (isset($dataSource['data']['items'])) {
            foreach ($dataSource['data']['items'] as &$item) {
                $item[$this->getData('name')]['edit'] = [
                    'href' => $this->_urlBuilder->getUrl(
                        'news/news/edit',
                        ['id' => $item['news_id']]
                    ),
                    'label' => __('Edit'),
                    'hidden' => false,
                ];
                $item[$this->getData('name')]['delete'] = [
                    'href' => $this->_urlBuilder->getUrl(
                        'news/news/delete',
                        ['id' => $item['news_id']]
                    ),
                    'label' => __('Delete'),
                    'hidden' => false,
                ];
            }
        }

        return $dataSource;
    }
}
```

Inyección de dependencias

Si echamos un vistazo a nuestro fichero listing comprobamos que tenemos que definir un objeto llamado **NewsGridDataProvider.**

<argument name="class" xsi:type="string">NewsGridDataProvider</argument>
Para poder usarlo tendremos que en un nuevo fichero de dependencias llamado **di.xml** y ubicado en la carpeta **etc/**

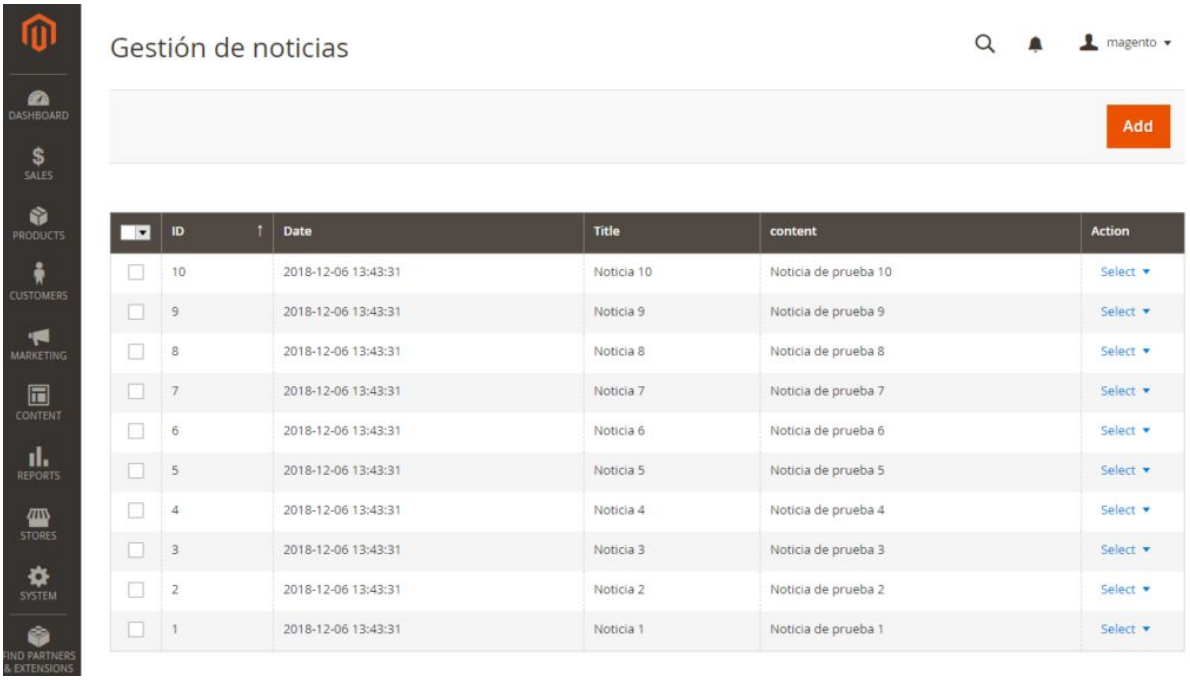
```
<?xml version="1.0" encoding="UTF-8"?>

<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:ObjectManager/etc/config.
xsd">

   <virtualType name="NewsGridDataProvider"
type="Magento\Framework\View\Element\UiComponent\DataProvider\DataProvider">
      <arguments>
         <argument name="collection" xsi:type="object"
shared="false">Ilovemarketing\News\Model\ResourceModel\News\News</argument>
         <argument name="filterPool" xsi:type="object"
shared="false">NewsGridFilterPool</argument>
      </arguments>
   </virtualType>

   <virtualType name="NewsGridFilterPool"
type="Magento\Framework\View\Element\UiComponent\DataProvider\FilterPool">
      <arguments>
         <argument name="appliers" xsi:type="array">
            <item name="regular"
xsi:type="object">Magento\Framework\View\Element\UiComponent\DataProvider\Regu
larFilter</item>
            <item name="fulltext"
xsi:type="object">Magento\Framework\View\Element\UiComponent\DataProvider\Fullt
extFilter</item>
         </argument>
      </arguments>
   </virtualType>

   <type
name="Magento\Framework\View\Element\UiComponent\DataProvider\CollectionFacto
ry">
      <arguments>
         <argument name="collections" xsi:type="array">
            <item name="news_news_listing_data_source"
xsi:type="string">Ilovemarketing\News\Model\ResourceModel\News\News</item>
         </argument>
      </arguments>
   </type>

   <virtualType name="Ilovemarketing\News\Model\ResourceModel\News\News"
type="Magento\Framework\View\Element\UiComponent\DataProvider\SearchResult">
      <arguments>
         <argument name="mainTable"
xsi:type="string">ilovemarketing_news</argument>
```

```
        <argument name="resourceModel"
xsi:type="string">Ilovemarketing\News\Model\ResourceModel\News</argument>
      </arguments>
   </virtualType>

</config>
```

En este fichero declaramos las dependencias.

- Definimos un VirtualType **NewsGridDataProvider** y le pasamos la clase de nuestra colección.
- A continuación le indicamos a Magento que use filtros al que llamaremos NewsGridFilterPool .
- Creamos un nuevo tipo para nuestra colección al que llamaremos **news_news_listing_data_source** que es el mismo nombre que definimos en nuestro dataSource del fichero de layout news_news_listing.xml
- Creamos el tipo SearchResult pasandole nuestra colección indicándole la tabla que tiene que consultar y el resourceModel de nuestro modelo.

Y con esto ya preparado podremos ver nuestra grilla funcionando y listando datos de nuestra base de datos. En el siguientes temas iremos incluyéndole nuevas funcionalidades.



Componentes adicionales de la Grilla (paginador, buscador, exportación de datos, filtrados, etc)

Anteriormente definimos varios elementos en nuestro <listing> que se encargaba de crear nuestra grilla. Ademas de definir sus <**arguments**>, <**dataSource**> y <**columns**> podremos usar una nueva etiqueta llamada <**container**> al la cual le podremos incluir nuevas funcionalidades.

Incluiremos dentro de nuestro fichero **news_news_listing.xml** dentro de la etiqueta <listing> este cuarto componente.

```
<container name="listing_top">
   <argument name="data" xsi:type="array">
      <item name="config" xsi:type="array">
         <item name="template" xsi:type="string">ui/grid/toolbar</item>
      </item>
   </argument>

   <!-- paginador -->
   <!-- buscador -->
   <!-- exportación de datos -->
   <!-- Bookmarks -->
   <!-- Filtrado de datos -->
</container>
```

Vamos a incluir nuevas funcionalidades a nuestra grilla:

- Un **Paginador** en el que podremos indicar el numero de registros por página.
- Un **Buscador** de datos, para ello hay que configurar indices en los campos de la tabla que estamos buscando.
- Opción de **exportación** de datos (XML o CSV).
- Incluir **marcadores** del estado actual de nuestra grilla.
- Opciones de **filtrado** de datos.

Paginador

Empezaremos incluyendo un paginador. Tendremos que tener en cuenta en selectProvider el nombre del campos unico que vamos a consultar.

```
<paging name="listing_paging">
        <argument name="data" xsi:type="array">
          <item name="config" xsi:type="array">
            <item name="storageConfig" xsi:type="array">
              <item name="provider"
xsi:type="string">news_news_listing.news_news_listing.listing_top.bookmarks</item>
              <item name="namespace" xsi:type="string">current.paging</item>
            </item>
            <item name="selectProvider"
xsi:type="string">news_news_listing.news_news_listing.news_news_columns.news_id</it
em>
            <item name="displayArea" xsi:type="string">bottom</item>
            <item name="sizesConfig" xsi:type="array">
              <item name="value" xsi:type="number">5</item>
            </item>
          </item>
        </argument>
     </paging>
```

Buscador

Si quisiéramos incluir un buscador tan solo tendremos que incluir:

```
    <filterSearch name="fulltext">
        <argument name="data" xsi:type="array">
          <item name="config" xsi:type="array">
            <item name="provider"
xsi:type="string">news_news_listing.news_news_listing_data_source</item>
            <item name="chipsProvider"
xsi:type="string">news_news_listing.news_news_listing.listing_top.listing_filters_chips</it
em>
            <item name="storageConfig" xsi:type="array">
              <item name="provider"
xsi:type="string">news_news_listing.news_news_listing.listing_top.bookmarks</item>
              <item name="namespace" xsi:type="string">current.search</item>
            </item>
          </item>
```

```
        </argument>
    </filterSearch>
```

Exportación de datos

y en caso de querer un botón para exportar nuestros datos:

```
<exportButton name="export_button"/>
```

Incluir un marcador para salvar nuestra vista de página:

```
<bookmark name="bookmarks">
        <argument name="data" xsi:type="array">
            <item name="config" xsi:type="array">
                <item name="component"
xsi:type="string">Magento_Ui/js/grid/controls/bookmarks/bookmarks</item>
                <item name="displayArea" xsi:type="string">dataGridActions</item>
                <item name="storageConfig" xsi:type="array">
                    <item name="saveUrl" xsi:type="url" path="*/*/save"/>
                    <item name="deleteUrl" xsi:type="url" path="*/*/delete"/>
                    <item name="namespace" xsi:type="string">news_news_listing</item>
                </item>
            </item>
        </argument>
    </bookmark>
```

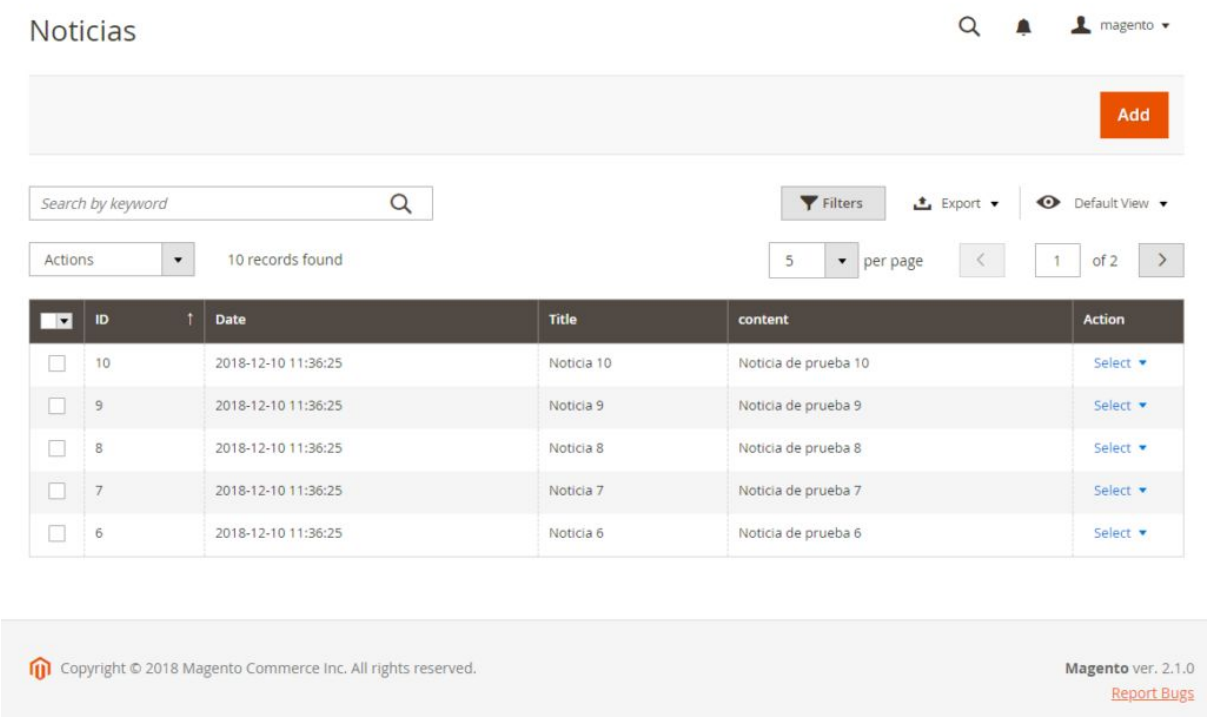
Filtrado de datos

Opciones de filtrado de datos

```
<filters name="listing_filters">
        <argument name="data" xsi:type="array">
            <item name="config" xsi:type="array">
                <item name="storageConfig" xsi:type="array">
                    <item name="provider"
xsi:type="string">news_news_listing.news_news_listing.listing_top.bookmarks</item>
                    <item name="namespace" xsi:type="string">curren.filters</item>
                </item>
                <item name="childDefaults" xsi:type="array">
                    <item name="provider"
xsi:type="string">news_news_listing.news_news_listing.listing_top.listing_filters</item>
                    <item name="imports" xsi:type="array">
                        <item name="visible"
```

```
xsi:type="string">news_news_listing.news_news_listing.listing_top.bookmarks:current.columns.${ $.index }.visible</item>
                    </item>
                </item>
            </item>
        </argument>
    </filters>
```

Por lo que nuestra grilla y sus nuevos componentes se vería asi:



Edición de Registros

Vamos a empezar con la edición de registros ya que para incluir registros simplificaremos el código.

Para que la edición de registros funcione correctamente vamos a trabajar con los siguientes ficheros:

Ilovemarketing/News/Controller/Adminhtml/News/Edit.php
Ilovemarketing/News/view/adminhtml/layout/news_news_edit.xml
Ilovemarketing/News/view/adminhtml/ui_component/news_news_form.xml
Ilovemarketing/News/Model/News/DataProvider.php

Botones
Ilovemarketing/News/Block/Adminhtml/News/Edit/GenericButton.php
Ilovemarketing/News/Block/Adminhtml/News/Edit/SaveButton.php
Ilovemarketing/News/Block/Adminhtml/News/Edit/DeleteButton.php
Ilovemarketing/News/Block/Adminhtml/News/Edit/ResetButton.php
Ilovemarketing/News/Block/Adminhtml/News/Edit/BackButton.php

Controlador

Empezaremos con el controlador que se encargará de editar nuestros registros, por lo que crearemos en **Controller/Adminhtml/News/** el fichero **Edit.php** que contendrá la clase de nuestro controlador.

```
<?php
namespace Ilovemarketing\News\Controller\Adminhtml\News;

use Magento\Backend\App\Action;
use Magento\Backend\App\Action\Context;
use Magento\Framework\View\Result\PageFactory;
use Ilovemarketing\News\Model\News;

class Edit extends Action {

   protected $_pageFactory;

   public function __construct( Context $context,
                        PageFactory $pageFactory  ) {
```

```
        parent::__construct($context);
        $this->_pageFactory = $pageFactory;

    }

    public function execute() {

        $_page = $this->_pageFactory->create();
        $newsArray = $this->getRequest()->getParam('news');

        if (is_array( $newsArray )) {

            $_news = $this->_objectManager->create(News::class);
            $_news->setData($newsArray)->save();

            $resultRedirect = $this->resultRedirectFactory->create();
            return $resultRedirect->setPath('*/*/index');

        }

        return $_page;
    }
}
```

Layout

Lo segundo que vamos a crear es el layout que se encargara de crear el formulario.
Para ello creamos en /view/adminhtml/layout/ el fichero news_news_edit.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<page xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:framework:View/Layout/etc/page_con
figuration.xsd">

    <head>
        <title>Noticia</title>
    </head>

    <update handle="styles"/>
    <update handle="editor"/>
    <body>
        <referenceContainer name="content">
            <uiComponent name="news_news_form"/>
```

```
    </referenceContainer>
  </body>

</page>
```

UiComponent

Y al igual que en nuestra grilla del tema anterior , para poder crear el formulario necesitamos crear un componente UI. Crearemos el componente en el mismo directorio **/view/adminhtml/ui_component/** y lo llamaremos **news_news_form.xml**

```
<?xml version="1.0" encoding="UTF-8"?>

<form xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Ui:etc/ui_configurati
on.xsd">
  <argument name="data" xsi:type="array">
    <item name="js_config" xsi:type="array">
      <item name="provider"
xsi:type="string">news_news_form.news_news_form_data_source</item>
      <item name="deps"
xsi:type="string">news_news_form.news_news_form_data_source</item>
    </item>
    <item name="label" xsi:type="string" translate="true">Sample Form</item>
    <item name="layout" xsi:type="array">
      <item name="type" xsi:type="string">tabs</item>
    </item>

    <item name="buttons" xsi:type="array">
      <item name="back"
xsi:type="string">Ilovemarketing\News\Block\Adminhtml\News\Edit\BackButton</item>
      <item name="delete"
xsi:type="string">Ilovemarketing\News\Block\Adminhtml\News\Edit\DeleteButton</ite
m>
      <item name="reset"
xsi:type="string">Ilovemarketing\News\Block\Adminhtml\News\Edit\ResetButton</item
>
      <item name="save"
xsi:type="string">Ilovemarketing\News\Block\Adminhtml\News\Edit\SaveButton</item>
    </item>
  </argument>

  <dataSource name="news_news_form_data_source">
    <argument name="dataProvider" xsi:type="configurableObject">
```

```
        <argument name="class"
xsi:type="string">Ilovemarketing\News\Model\News\DataProvider</argument>
        <argument name="name"
xsi:type="string">news_news_form_data_source</argument>
        <argument name="primaryFieldName" xsi:type="string">news_id</argument>
        <argument name="requestFieldName" xsi:type="string">id</argument>
      </argument>
      <argument name="data" xsi:type="array">
        <item name="js_config" xsi:type="array">
          <item name="component"
xsi:type="string">Magento_Ui/js/form/provider</item>
        </item>
      </argument>
    </dataSource>

    <fieldset name="news">
      <argument name="data" xsi:type="array">
        <item name="config" xsi:type="array">
          <item name="label" xsi:type="string" translate="true">Noticia</item>
        </item>
      </argument>

      <!-- This field represents form id and is hidden -->
      <field name="news_id">
        <argument name="data" xsi:type="array">
          <item name="config" xsi:type="array">
            <item name="visible" xsi:type="boolean">false</item>
            <item name="dataType" xsi:type="string">text</item>
            <item name="formElement" xsi:type="string">input</item>
            <item name="source" xsi:type="string">news</item>
          </item>
        </argument>
      </field>

      <field name="news_type_id">
        <argument name="data" xsi:type="array">
          <item name="options"
xsi:type="object">Ilovemarketing\News\Model\Source\NewsTypeOptions</item>
          <item name="config" xsi:type="array">
            <item name="label" xsi:type="string">tipo</item>
            <item name="visible" xsi:type="boolean">true</item>
            <item name="dataType" xsi:type="string">text</item>
            <item name="wysiwyg" xsi:type="boolean">true</item>
            <!-- textarea, wysiwyg multiline, select, multiselect, ,radio, checkbox,
,password,time,note,label,link,image,file,date,submit -->
            <item name="formElement" xsi:type="string">select</item>
```

```
            <item name="source" xsi:type="string">news</item>
          </item>
        </argument>
      </field>

      <field name="news_title">
        <argument name="data" xsi:type="array">
          <item name="config" xsi:type="array">
            <item name="label" xsi:type="string">Title</item>
            <item name="visible" xsi:type="boolean">true</item>
            <item name="dataType" xsi:type="string">text</item>
            <item name="formElement" xsi:type="string">input</item>
            <item name="source" xsi:type="string">news</item>
            <item name="validation" xsi:type="array">
              <item name="required-entry" xsi:type="boolean">true</item>
            </item>
          </item>
        </argument>
      </field>

      <field name="news_content">
        <argument name="data" xsi:type="array">
          <item name="config" xsi:type="array">
            <item name="label" xsi:type="string">content</item>
            <item name="visible" xsi:type="boolean">true</item>
            <item name="dataType" xsi:type="string">text</item>
            <item name="wysiwyg" xsi:type="boolean">true</item>
            <item name="sortOrder" xsi:type="number">130</item>
            <item name="rows" xsi:type="number">12</item>
            <!-- textarea, wysiwyg multiline, select, multiselect, ,radio, checkbox,
,password,time,note,label,link,image,file,date,submit -->
            <item name="formElement" xsi:type="string">wysiwyg</item>
            <item name="source" xsi:type="string">news</item>
          </item>
        </argument>
      </field>
    </fieldset>
</form>
```

DataProvider

Si nos fijamos bien el dataSource necesita un nuevo objeto que nos proporcionará los datos para rellenar el formulario. Por ello crearemos un nuevo modelo en **Model/News/** y lo llamaremos **DataProvider.php**. Este clase extenderá la clase Magento\Ui\DataProvider**AbstractDataProvider**

```
<?php
namespace Ilovemarketing\News\Model\News;

use Ilovemarketing\News\Model\ResourceModel\News\NewsFactory;

class DataProvider extends \Magento\Ui\DataProvider\AbstractDataProvider {

    public function __construct(
        $name,
        $primaryFieldName,
        $requestFieldName,
        NewsFactory $newsFactory,
        array $meta = [],
        array $data = []
    ) {
        $this->collection = $newsFactory->create();
        parent::__construct($name, $primaryFieldName, $requestFieldName, $meta,
$data);
    }

    public function getData(){
        if (isset($this->loadedData)) {
            return $this->loadedData;
        }

        $items = $this->collection->getItems();
        $this->loadedData = array();
        foreach ($items as $news) {
            $this->loadedData[$news->getId()]['news'] = $news->getData();
        }


        return $this->loadedData;

    }
}
```

En uno de los campos hacemos referencia a un campo select que carga datos de otra tabla de nuestra base de datos. En concreto es el campo news_type_id y que hace referencia a un nuevo modelo que tenemos que crear para los tipos de noticias.

Ya sabemos como crear nuevos modelos por lo que no repetiremos ese tema, pero si explicaremos como cargar datos en nuestro selector usando dicho modelo.

Para ello crearemos en **Model\Source** el fichero **NewsTypeOptions.php** que mostrara los datos provenientes de nuestra tabla. El método que usaremos para completar el selector es toOptionArray()

```
<?php
namespace Ilovemarketing\News\Model\Source;

use Magento\Framework\Data\OptionSourceInterface;
use Magento\Framework\View\Element\Template;
use Ilovemarketing\News\Model\ResourceModel\News\NewsTypesFactory;
use Magento\Framework\View\Element\Template\Context;

class NewsTypeOptions extends Template implements OptionSourceInterface {

   protected $_newsTypesFactory;

   public function __construct(
         Context $context,
         NewsTypesFactory $NewsTypesFactory ) {

      $this->_newsTypesFactory = $NewsTypesFactory;
      parent::__construct($context);

   }

   public function getNewsTypes() {

      return $this->_newsTypesFactory->create();

   }

   public function toOptionArray() {

      foreach ($this->getNewsTypes() as $value) {
         $newsTypesArray[] = [
            'label' => $value["news_type_name"],
            'value' => $value['news_type_id']
         ];
      }
```

```
        return $newsTypesArray;

    }

}
```

Botones

Ya por último podremos incluir las acciones que realizarán los botones del formulario. Si nos fijamos en nuestro componente ui, este contenía los bloques con sus acciones correspondientes

```
<item name="buttons" xsi:type="array">
    <item name="back"
xsi:type="string">Ilovemarketing\News\Block\Adminhtml\News\Edit\BackButton</item>
    <item name="delete"
xsi:type="string">Ilovemarketing\News\Block\Adminhtml\News\Edit\DeleteButton</ite
m>
    <item name="reset"
xsi:type="string">Ilovemarketing\News\Block\Adminhtml\News\Edit\ResetButton</item
>
    <item name="save"
xsi:type="string">Ilovemarketing\News\Block\Adminhtml\News\Edit\SaveButton</item>
</item>
```

Empezaremos creando el botón genérico, que es el elemento que permite a otros botones existir. Es la base que extiende al otros botones. Las clases las crearemos en **Block/Adminhtml/News/Edit**

```
<?php
namespace Ilovemarketing\News\Block\Adminhtml\News\Edit;

use Magento\Backend\Block\Widget\Context;
use Magento\Framework\Registry;

/**
 * Class GenericButton
 */
class GenericButton {

    /**
     * Url Builder
```

```
     *
     * @var \Magento\Framework\UrlInterface
     */
    protected $urlBuilder;

    /**
     * Registry
     *
     * @var \Magento\Framework\Registry
     */
    protected $registry;

    /**
     * Constructor
     *
     * @param \Magento\Backend\Block\Widget\Context $context
     * @param \Magento\Framework\Registry $registry
     */
    public function __construct(

        Context $context,
        Registry $registry ) {

        $this->urlBuilder = $context->getUrlBuilder();
        $this->registry = $registry;

    }

    /**
     * Return the synonyms group Id.
     *
     * @return int|null
     */
    public function getId() {

        $contact = $this->registry->registry('news');
        return $contact ? $contact->getId() : null;

    }

    /**
     * Generate url by route and parameters
     *
     * @param   string $route
     * @param   array $params
     * @return  string
```

```
     */
    public function getUrl($route = '', $params = []) {

        return $this->urlBuilder->getUrl($route, $params);

    }
}
```

Block para el salvado de datos

```
<?php
namespace Ilovemarketing\News\Block\Adminhtml\News\Edit;

use
Magento\Framework\View\Element\UiComponent\Control\ButtonProviderInterface;

class SaveButton extends GenericButton implements ButtonProviderInterface {

    public function getButtonData() {

        return [
            'label' => __('Save'),
            'class' => 'save primary',
            'data_attribute' => [
                'mage-init' => ['button' => ['event' => 'save']],
                'form-role' => 'save',
            ],
            'sort_order' => 90,
        ];
    }
}
```

Bloque de borrado

```
<?php
namespace Ilovemarketing\News\Block\Adminhtml\News\Edit;

use
Magento\Framework\View\Element\UiComponent\Control\ButtonProviderInterface;

class DeleteButton extends GenericButton implements ButtonProviderInterface {

    public function getButtonData() {

        $data = [];
        if ($this->getId()) {
            $data = [
                'label' => __('Borrar Noticia'),
```

```
            'class' => 'delete',
            'on_click' => 'deleteConfirm(\''
                . __('Estás seguro que deseas borrarlo ?')
                . '\', \'' . $this->getDeleteUrl() . '\')',
            'sort_order' => 20,
        ];
    }
    return $data;
  }

  public function getDeleteUrl() {

    return $this->getUrl('*/*/delete', ['news_id' => $this->getId()]);

  }
}
```

Bloque reset

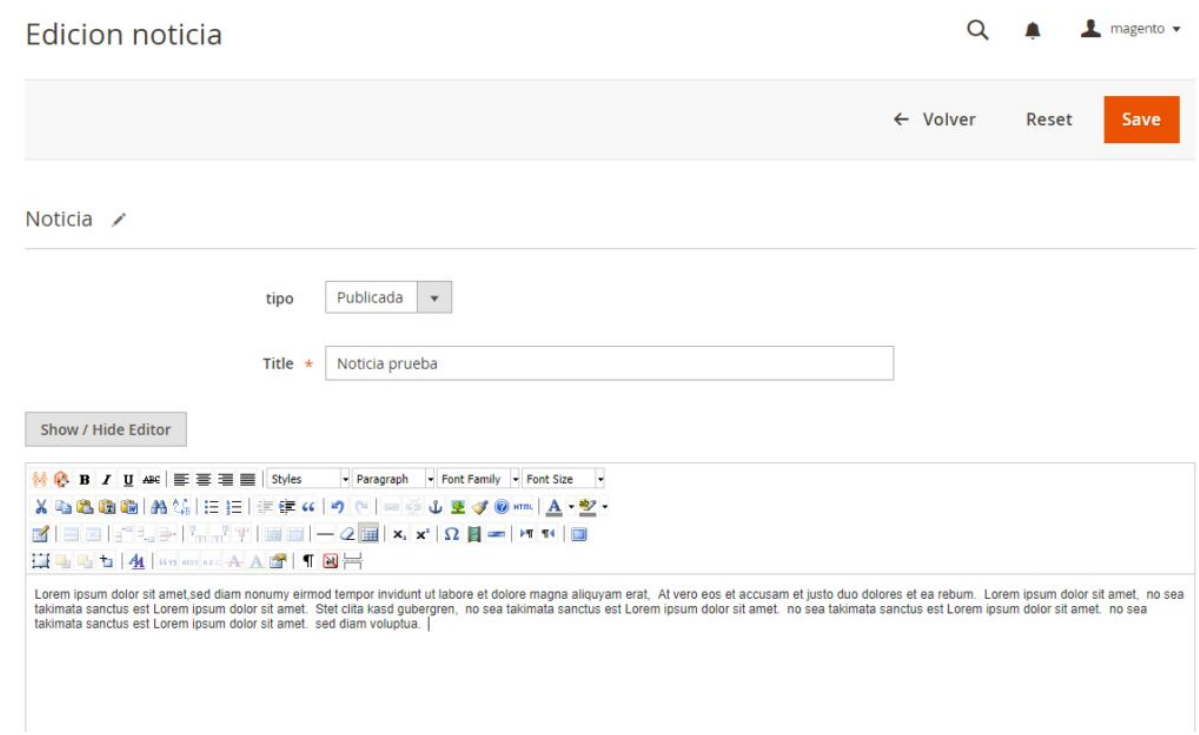
```
<?php
namespace Ilovemarketing\News\Block\Adminhtml\News\Edit;

use
Magento\Framework\View\Element\UiComponent\Control\ButtonProviderInterface;

class ResetButton implements ButtonProviderInterface {

  public function getButtonData() {

    return [
      'label' => __('Reset'),
      'class' => 'reset',
      'on_click' => 'location.reload();',
      'sort_order' => 30
    ];

  }
}
```

Bloque Volver

```
<?php
namespace Ilovemarketing\News\Block\Adminhtml\News\Edit;

use
Magento\Framework\View\Element\UiComponent\Control\ButtonProviderInterface;
```

```
class BackButton extends GenericButton implements ButtonProviderInterface {

    public function getButtonData() {

        return [
            'label' => __('Volver'),
            'on_click' => sprintf("location.href = '%s';", $this->getBackUrl()),
            'class' => 'back',
            'sort_order' => 10
        ];

    }

    public function getBackUrl() {

        return $this->getUrl('*/*/');

    }
}
```

Y así sería el resultado de nuestro formulario de edición es el siguiente



Creación de Registros

Ya hemos visto como editar registros desde nuestra grilla. Para incluir la funcionalidad de incluir nuevos registros incluimos en nuestro news_news_listing.xml el siguiente código que nos permitía incluir un botón de añadir nuevos registros desde nuestra grilla.

```
<item name="buttons" xsi:type="array">
   <item name="add" xsi:type="array">
      <item name="name" xsi:type="string">add</item>
      <item name="label" xsi:type="string" translate="true">Add</item>
      <item name="class" xsi:type="string">primary</item>
      <item name="url" xsi:type="string">*/*/add</item>
   </item>
</item>
```

Controlador

Lo que si vamos a crear es un nuevo controlador y que redireccionará al controlador de edicion para así ahorrar código y tener que crear un nuevo layout para este. Por lo que crearemos el conrolador en **Controller/Adminhtml/News** y lo llamaremos **Add.php**

```
<?php
namespace Ilovemarketing\News\Controller\Adminhtml\News;

use Magento\Backend\App\Action;
use Magento\Backend\App\Action\Context;
use Magento\Backend\Model\View\Result\ForwardFactory;

class Add extends Action {

   protected $_forwardFactory;

   public function __construct(
         Context $context,
         ForwardFactory $forwardFactory ) {

      $this->_forwardFactory = $forwardFactory;
      parent::__construct($context);

   }
```

```
    public function execute() {

        $resultForward = $this->_forwardFactory->create();
        return $resultForward->forward('edit');

    }
}
```

Operaciones de Borrado de Datos

Para que nuestro CRUD sea de todo funcional necesitamos poder borrar datos. Por ello vamos a crear un controlador más que realizará dicha acción.

Lo situaremos en la misma ubicación que los anteriores, por lo que lo vamos a crear en **Controller/Adminhtml/News/** con el nombre de fichero **Delete.php**.

```
<?php
namespace llovemarketing\News\Controller\Adminhtml\News;

use llovemarketing\News\Model\News as News;
use Magento\Backend\App\Action;

class Delete extends Action {

  public function execute() {

      // Parámetro recibido por URL
      $id = $this->getRequest()->getParam('id');

      if (!($news = $this->_objectManager->create(News::class)->load($id))) {

          $this->messageManager->addError(__('No se puede realizar la acción de borrado. Inténtelo de nuevo.'));
          $resultRedirect = $this->resultRedirectFactory->create();
          return $resultRedirect->setPath('*/*/index', array('_current' => true));

      } try {

          $news->delete();
          $this->messageManager->addSuccess(__('Tu registro ha sido borrado correctamente!'));

      } catch (Exception $e) {

          $this->messageManager->addError(__('Un error se ha producido mientras se borraba el registro: '));
          $resultRedirect = $this->resultRedirectFactory->create();
          return $resultRedirect->setPath('*/*/index', array('_current' => true));

      }

      $resultRedirect = $this->resultRedirectFactory->create();
```

```
        return $resultRedirect->setPath('*/*/index', array('_current' => true));
    }
}
```

Borrado Masivo

Recuerda que las acciones del desplegable de la grilla las podemos controlar desde el componente **/Ui/Component/Listing/Column/NewsActions.php** y en el que podremos incluir nuevas acciones a realizar sobre la fila seleccionada.

Hasta aquí todo correcto y ya podremos realizar operaciones de borrado.
Vamos a ir un paso más allá en las operaciones de borrado ya que podría darse el caso de querer borrar varios registros a la vez.

Para ello nos vamos a fijar en nuestro componente news_news_listing.xml en la siguiente sección:

```
<columns name="news_news_columns">

        <selectionsColumn name="ids">
            <argument name="data" xsi:type="array">
                <item name="config" xsi:type="array">
                    <item name="indexField" xsi:type="string">news_id</item>
                </item>
            </argument>
        </selectionsColumn>
```

Este nos habilitará un checkbox para seleccionar múltiples registros de nuestra grilla e indicando que su indice será el campo news_id que es la clave primaria que lo identificará como campo único en nuestra tabla. Como ya lo teníamos incluido anteriormente no vamos a tocar nada.

El siguiente paso es incluir en nuestro <container> de news_news_lisitng.xml el siguiente código que se encargará del borrado de filas.

```
<massaction name="listing_massaction">
    <argument name="data" xsi:type="array">
        <item name="config" xsi:type="array">
            <item name="selectProvider"
xsi:type="string">news_news_listing.news_news_listing.news_news_columns.ids</item>
            <item name="displayArea" xsi:type="string">bottom</item>
            <item name="indexField" xsi:type="string">news_id</item>
        </item>
    </argument>
```

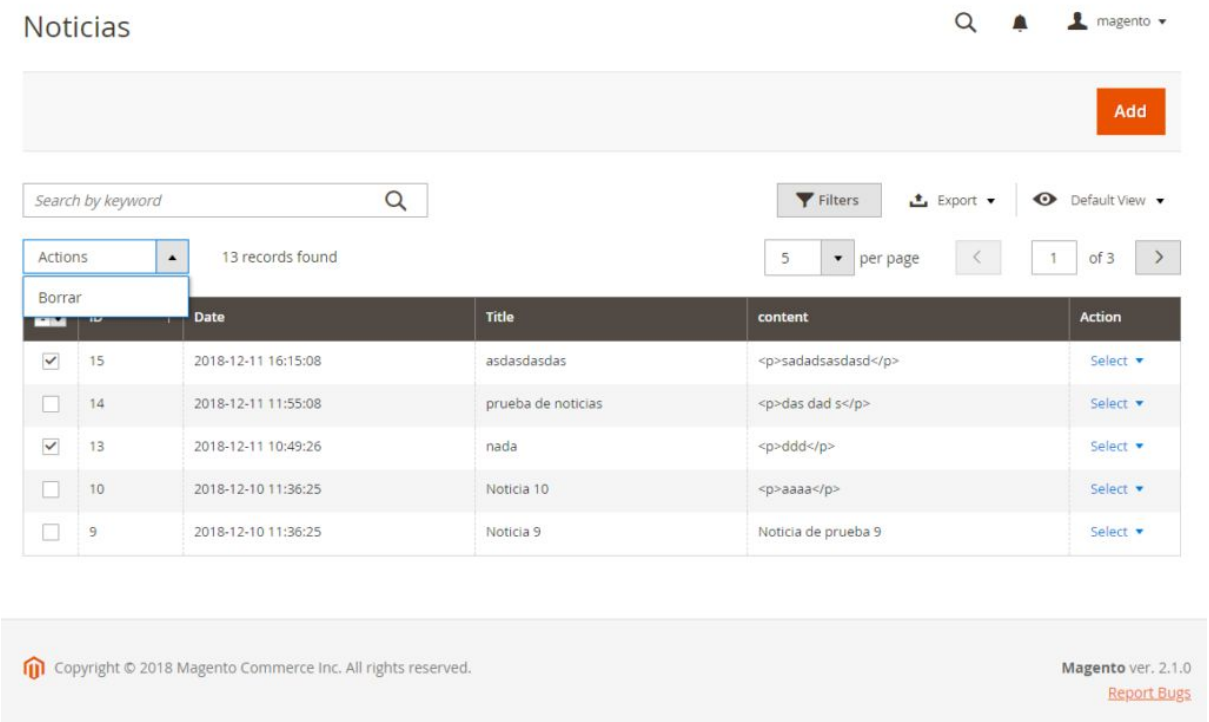
```
    <action name="delete">
        <argument name="data" xsi:type="array">
            <item name="config" xsi:type="array">
                <item name="type" xsi:type="string">delete</item>
                <item name="label" xsi:type="string" translate="true">Borrar
Seleccionado</item>
                <item name="url" xsi:type="url" path="*/*/massDelete"/>
                <item name="confirm" xsi:type="array">
                    <item name="title" xsi:type="string" translate="true">Borrar todas las
noticias seleccionadas</item>
                    <item name="message" xsi:type="string" translate="true">Estás seguro que
quieres borrar las noticias seleccionadas?</item>
                </item>
            </item>
        </argument>
    </action>
</massaction>
```

Por último creamos el controlador que será el encargado de borrar los datos. Al igual que el resto lo creamos en **Controller/Adminhtml/News/** con el nombre de **massDelete.php**.

```
<?php
namespace Ilovemarketing\News\Controller\Adminhtml\News;

use Magento\Backend\App\Action;
use Ilovemarketing\News\Model\News;

class MassDelete extends Action {

    public function execute() {

        $ids = $this->getRequest()->getParam('selected', []);
        if (!is_array($ids) || !count($ids)) {
            $resultRedirect = $this->resultRedirectFactory->create();
            return $resultRedirect->setPath('*/*/index', array('_current' => true));
        }
        foreach ($ids as $id) {
            if ($news_id = $this->_objectManager->create(News::class)->load($id)) {
                $news_id->delete();
            }
        }
        $this->messageManager->addSuccess(__('%1 registro(s) han sido eliminados.',
count($ids)));

        $resultRedirect = $this->resultRedirectFactory->create();
```

```
        return $resultRedirect->setPath('*/*/index', array('_current' => true));
    }

}
```

Nuestra grilla y borrado queda de la siguiente forma:



Configuraciones de Módulo

En magento es posible guardar los valores de configuración para una web, tienda, módulo a través del backend.

Esos valores pueden ser usados para guardar configuraciones de nuestros módulos, opciones de configuración cualquier otro dato requerido de nuestros módulos.

Todos esos datos son grabados en la tabla **core_config_data_table** y sólo serán accesibles a través del backend de magento. Desde **store** -> **configuration**

System

Para crear nuestro fichero de tendremos que crear el directorio **etc/adminhtml/** y ahí crearemos el fichero **system.xml**

Vamos a explicar paso a paso para que sirve cada elemento de nuestro fichero de configuración.

```
<?xml version="1.0" encoding="UTF-8"?>

<config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="urn:magento:module:Magento_Config:etc/system_file.xsd">
   <system>

      <tab id="ilovemarketing" translate="label" sortOrder="1000"
class="Ilovemarketing_News">
         <label>I love Marketing</label>
      </tab>

      <section id="ilovemarketing" translate="label" type="text" sortOrder="1"
showInDefault="1" showInWebsite="1" showInStore="1">

         <label>Noticias</label>
         <tab>ilovemarketing</tab>
         <resource>Ilovemarketing_News::config_news</resource>

         <group id="news" translate="label" type="text" sortOrder="1" showInDefault="1"
showInWebsite="1" showInStore="1">

            <label>Módulo de Noticias</label>
```

```
            <comment>Configuración del Módulo de Noticias</comment>

            <!-- Campo para habilitar el modulo -->
            <field id="enabled" translate="label" type="select" sortOrder="1"
showInDefault="1" showInWebsite="0" showInStore="0">
               <label>Habilitar Módulo</label>

<source_model>Magento\Config\Model\Config\Source\Yesno</source_model>
            </field>

            <!-- Campo texto title -->
            <field id="newsTitle" translate="label" type="text" sortOrder="2"
showInDefault="1" showInWebsite="1" showInStore="1">
               <label>Título</label>
            </field>

            <!-- Campo texto control registros -->
            <field id="newsLimit" translate="label" type="select" sortOrder="3"
showInDefault="1" showInWebsite="1" showInStore="1">
               <label>Registros por página</label>

<source_model>Ilovemarketing\News\Model\Config\Source\NewsLimit</source_model
>
            </field>

            <!-- campo select personalizado -->
            <field id="newsFeatured" translate="label" type="select" sortOrder="4"
showInDefault="1" showInWebsite="1" showInStore="1">
               <label>Noticias a destacar</label>

<source_model>Ilovemarketing\News\Model\Config\Source\NewsFeatured</source_mo
del>
            </field>

            <!-- campo que dependera de newsFeatured -->
            <field id="newsFeaturedDependable" translate="label" type="select"
sortOrder="5" showInDefault="1" showInWebsite="1" showInStore="1">
               <label>Cantidad de noticias a destacar</label>
               <depends>
                  <field id="*/*/newsFeatured">2</field>
               </depends>

<source_model>Ilovemarketing\News\Model\Config\Source\NewsFeaturedLimit</sourc
e_model>
            </field>
```

```
            <!-- campo para un grupo de opciones -->
            <field id="newsTypes" translate="label" type="multiselect" sortOrder="6"
showInDefault="1" showInWebsite="0" showInStore="0" canRestore="1">
                <label>Tipo de noticias que se mostrarán</label>

<source_model>Ilovemarketing\News\Model\Config\Source\NewsTypes</source_model
>
            </field>

           <!-- campo para un grupo de opciones -->
            <field id="newsVisible" translate="label" type="multiselect" sortOrder="6"
showInDefault="1" showInWebsite="0" showInStore="0" canRestore="1">
                <label>Visibilidad</label>

<source_model>Ilovemarketing\News\Model\Config\Source\NewsVisible</source_mode
l>
            </field>

        </group>

        <group id="newsMeta" translate="label" type="text" sortOrder="2"
showInDefault="1" showInWebsite="1" showInStore="1">
            <label>Meta Tags</label>

            <!-- Campo texto meta title -->
            <field id="newsMetaTitle" translate="label" type="text" sortOrder="1"
showInDefault="1" showInWebsite="1" showInStore="1">
                <label>Meta Titulo</label>
            </field>

            <!-- Campo texto meta description -->
            <field id="newsMetaDescription" translate="label" type="text" sortOrder="2"
showInDefault="1" showInWebsite="1" showInStore="1">
                <label>Meta Descripción</label>
            </field>

            <!-- Campo texto meta keywords -->
            <field id="newsMetaKeywords" translate="label" type="text" sortOrder="3"
showInDefault="1" showInWebsite="1" showInStore="1">
                <label>Meta Keywords</label>
            </field>

        </group>
    </section>
</system>
```

</config>

Magento agrupa su configuraciones en secciones <**section>** las cuales contienen múltiples grupos **<group>** con una serie de campos de configuración. Cada sección es asignada a una pestaña **<tab>** donde se visualizan sus secciones.

Creando nuevas pestañas

Cuando nuestras opciones de configuración no encajen en cualquiera de las ya existentes podremos crear nuestra propia **<tab>** en nuestro fichero **system.xml**

```
<tab id="ilovemarketing" translate="label" sortOrder="1000"
class="llovemarketing_News">
   <label>I love Marketing</label>
</tab>
```

- **id**: identificador único para nuestra <tab> y que sera usada posteriormente en sus secciones.
- **translate**: especifica que elementos están disponibles para ser traducidos en las etiquetas <label>.
- **sortOrder**: nos permitirá ubicar el <tab> en relación a los demás tabs.
- **class**: nombre de nuestra clase (es opcional).

El label de nuestra tab debe ser lo más descriptivo posible para que los clientes entiendan para que sirve.

Creando secciones

Las secciones se muestran dentro de la <tab> a la que hagan referencia. En Cada sección podemos configurar múltiples grupos y estos pueden agrupar una serie de campos.

```
<section id="ilovemarketing" translate="label" type="text" sortOrder="1"
showInDefault="1" showInWebsite="1" showInStore="1">

   <label>Noticias</label>
   <tab>ilovemarketing</tab>
   <resource>llovemarketing_News::config_news</resource>

    <!-- groups -->

</section>
```

Atributos de <section>

- **id:** identificador único para la sección.
- **translate**: elementos que se van a traducir (suelen ser etiquetas <label>).
- **sortOrder**: ordenacion del elemento.
- **showInDefault**: Muestra esta sección en la configuración por defecto de tienda.
- **showinWebsite**: Visualiza esta sección en la configuración de website.
- **showInStore**: Visualiza esta sección en la configuración de tienda.

Elementos de <section>

- **class**: Clase que usaremos para esta sección.
- **header_css**: Clase css para usar en el header de la sección.
- **label**: Nombre de nuestra sección.
- **tab**: La Id de la tab en la que va a mostrar esta sección
- **resource**: ACL (access control list) Que usuarios tendrán acceso a esta sección.

Creando Grupos

Podremos crear múltiples grupos que serán separados por <fieldsets>. Estos pueden expandirse o contraerse y podrán agrupar uno o más campos.

```
<group id="news" translate="label" type="text" sortOrder="1" showInDefault="1" showInWebsite="1" showInStore="1">

<label>Módulo de Noticias</label>
<comment>Configuración del Módulo de Noticias</comment>

.....
</group>
```

Los atributos de grupo son los siguientes:

- **id**: identificador único de grupo.
- **translate**: los campos que necesitan traducción (En este caso el label).
- **type**: el tipo de campo.
- **sortOrder**: Orden del campo en el grupo.
- **showInDefault**: Muestra esta sección en la configuración por defecto de tienda.
- **showinWebsite**: Visualiza esta sección en la configuración de website.
- **showInStore**: Visualiza esta sección en la configuración de tienda.

EL label será mostrado en el encabezado del grupo.

Creando nuevos campos

Como hemos dicho anteriormente cada grupo puede contener múltiples campos. En la base de datos, la configuración es grabada como si de un **path** se tratara **<section>/<group>/<field>.**

```
<field id="newsTitle" translate="label" type="text" sortOrder="2" showInDefault="1"
showInWebsite="1" showInStore="1">
   <label>Título</label>
</field>
```

Los atributos que podemos usar en un campo de configuración son:

- **id**: identificador único del campo.
- **translate**: Los campos que necesitan ser traducidos (puede ser un label, placeholder, comment, tooltips u otros elementos del campo).
- **type**: El tipo de campo que puede ser:
    - **text**/**textarea**
    - **obscure** para campos de tipo password
    - **select** que necesitaran de un un elemento **source_model** para consultar sus opciones.
    - **multiselect** que sera un selector de múltiples opciones.
    - **image**: para cargar una imagen. En este caso necesitaremos un **backend_model** para grabar el fichero, un **uploar_dir** donde sera grabada la imagen y una **base_url** que será el path donde se construirá la url desde la que se accederá desde el backend.
    - **sortOrder**: Ordenación del campo respecto al resto de campos.
    - **showInDefault**: Muestra esta sección en la configuración por defecto de tienda.
    - **showinWebsite**: Visualiza esta sección en la configuración de website.
    - **showInStore**: Visualiza esta sección en la configuración de tienda.

Existen otra seria de elementos que podemos configurar en un campo:

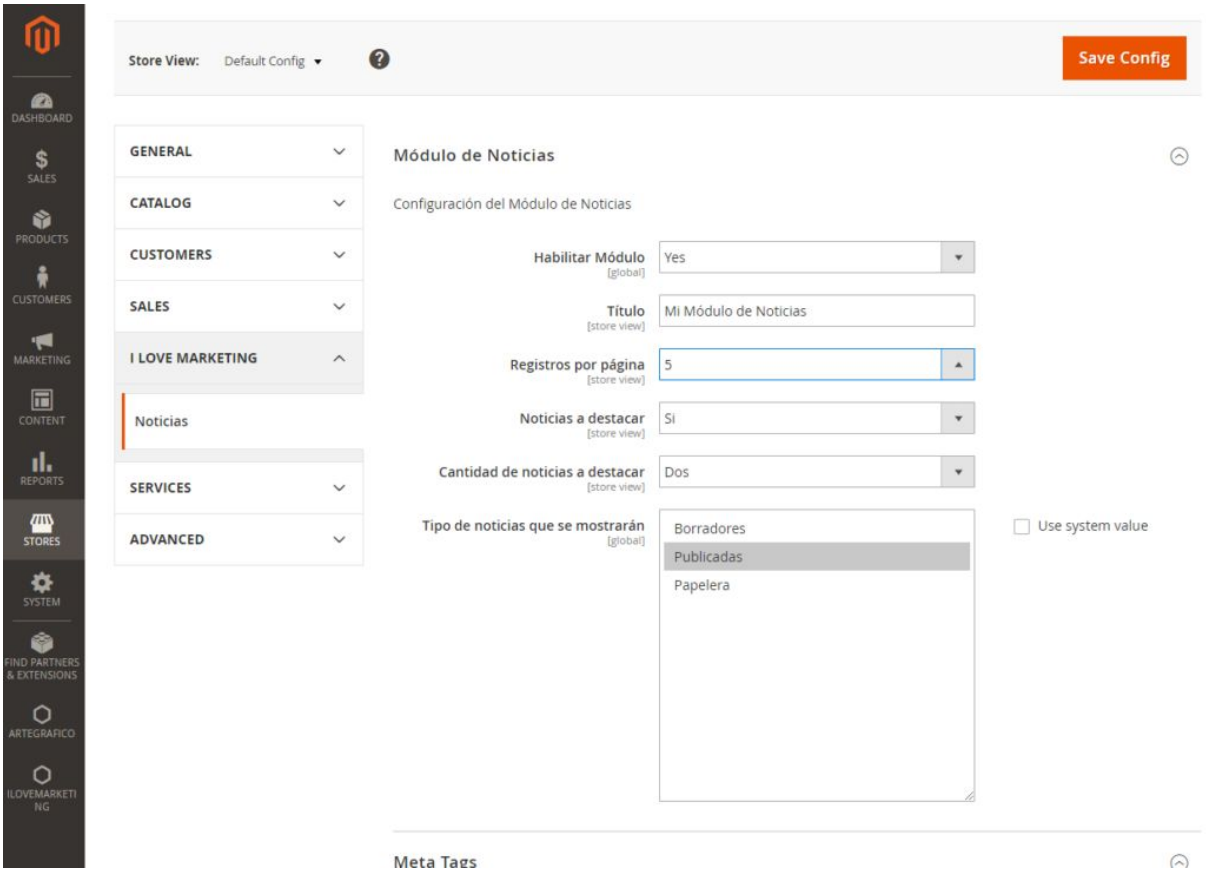
- **label** : una descripción o etiqueta para describir el campo.
- **comment**: Un comentario que será mostrado debajo del campo.
- **tooltip**: Un texto que se mostrará cuando situemos el ratón encima del campo.
- **frontend_class**: Una clase usada para el campo.
- **frontend_model**: Un modelo personalizado (block) que usaremos para renderizar el campo.

- **backend_model**: Un modelo de backend que sera usado cuando grabemos datos.
- **source_model**: La ubicación del modelo de datos usado en un campo multiselect.
- **depends**: Es una opción que usaremos para mostrar u ocultar campos dependiendo del valor seleccionado de otro campo.

```
<depends>
   <field id="[campo_a_comprobar]">[valor]</field>
</depends>
```

Si el campo tiene el valor que le hemos indicado, este campo sera visible. En caso contrario se ocultara al usuario.

Continuaremos en el siguiente tema porque mostrará errores ya que aún no hemos creado los **source_model.**



Funcionamiento de los campos de configuración

Ya hemos visto como preparar nuestros campos de configuración. Ahora veremos como funcionan muchos de ellos a la hora de seleccionar diferentes opciones o consultar datos en otras tablas.

Source Models

Hay bastantes **source_model** creados por programadores de Magento. En nuestro primer campo de configuración hemos usado uno de ellos y que nos permite seleccionar Si/No de un selector:

```
<field id="enabled" translate="label" type="select" sortOrder="1" showInDefault="1" showInWebsite="0" showInStore="0">
   <label>Habilitar Módulo</label>
   <source_model>Magento\Config\Model\Config\Source\Yesno</source_model>
</field>
```

Otros modelos bastante útiles son los siguientes:

- Magento\Config\Model\Config\Source**Yesnocustom** (campo Si/No personalizado)
- Magento\Config\Model\Config\Source**Enabledisable** (guarda 0/1 en nuestra tabla de configuración)
- Magento\Catalog\Model\Config\Source**TimeFormat** (formato de fcha 12 h/24 h)
- Magento\AdminNotification\Model\Config\Source**Frequency** (selector de frecuencias 1, 2, 6, 12, 24 horas)
- Magento\Cron\Model\Config\Source**Frequency** (Frecuencias *Daily/Weekly/Monthly* y songuardas como D/W/M)
- Magento\GoogleAdwords\Model\Config\Source**Language** (idioma en formato ISO 639-1, porm ejemplo - es)
- Magento\Config\Model\Config\Source**Locale** (idioma locale - en_US)

Pero no siempre querremos un modelo específico de Magento. En estos casos necesitaremos crear el nuestro propio. Echemos un vistazo al campo que controla la cantidad de registros por pagina de nuestro módulo de noticias.

```
<field id="newsLimit" translate="label" type="select" sortOrder="3" showInDefault="1" showInWebsite="1" showInStore="1">
   <label>Registros por página</label>
```

```
    <source_model>Ilovemarketing\News\Model\Config\NewsLimit</source_model>
</field>
```

El elemento **source_model** define la clase que vamos a crear y que se encargará de rellenerar nuestro select. Por lo que vamos a crear un modelo en **Model/Config/** y le llamaremos **NewsLimit.php**.

Nuestra clase implementa Magento\Framework\Option**ArrayInterface** y retornará un array de valores usando el método de **toOptionArray().**

```
<?php
namespace Ilovemarketing\News\Model\Config\Source;

use Magento\Framework\Option\ArrayInterface;

class NewsLimit implements ArrayInterface {

    public function toOptionArray() {

        return [
            ['value' => 5, 'label' => '5'],
            ['value' => 10, 'label' => '10'],
            ['value' => 15, 'label' => '15'],
            ['value' => 20, 'label' => '20'],
            ['value' => 25, 'label' => '25']
        ];
    }
}
```

Ahora que ya sabemos como podemos completar selectores con arrays de valores, aprenderemos como funciona un selector dependiente de otro. Para ello nos centraremos en el selector de noticias a destacar.

Usaremos el elemento **<depends>** que lo que hace es comprobar si el campo **newsFeatured** tiene el valor **2**. En caso de que el usuario seleccione dicho valor el campo se activará.

```
<field id="newsFeatured" translate="label" type="select" sortOrder="4"
showInDefault="1" showInWebsite="1" showInStore="1">
    <label>Noticias a destacar</label>

<source_model>Ilovemarketing\News\Model\Config\Source\NewsFeatured</source_mo
del>
</field>

<!-- campo que dependera de newsFeatured -->
```

```
<field id="newsFfeaturedDependable" translate="label" type="select" sortOrder="5"
showInDefault="1" showInWebsite="1" showInStore="1">
   <label>Cantidad de noticias a destacar</label>
   <depends>
      <field id="*/*/newsFeatured">2</field>
   </depends>

<source_model>Ilovemarketing\News\Model\Config\Source\NewsFeaturedLimit</sourc
e_model>
</field>
```

El elemento **source_model** llamando a la clase **NewsFeatured.php** lo preparamos de la misma forma que hicimos con el selector de paginas.

```
<?php
namespace Ilovemarketing\News\Model\Config;

class NewsFeatured implements \Magento\Framework\Option\ArrayInterface {

   public function toOptionArray() {

      return [
         ['value' => 1, 'label' => __('No')],
         ['value' => 2, 'label' => __('Si')]
      ];

   }
}
```

Y el **source_model** de la clase **NewsFeaturedLimit.php** funcionará de la misma forma.

```
<?php
namespace Ilovemarketing\News\Model\Config\Source;

class NewsFeaturedLimit {

   public function toOptionArray () {

      return [
         ['value' => 1, 'label' => __('Una')],
         ['value' => 2, 'label' => __('Dos')],
         ['value' => 3, 'label' => __('Tres')],
         ['value' => 4, 'label' => __('Cuatro')]
      ];
```

```
    }
}
```

El funcionamiento de un campo select multiple también usará un **source_model.** En este caso el selector múltiple nos permite seleccionar multiples opciones.

```
<!-- campo para un grupo de opciones -->
<field id="newsTypes" translate="label" type="multiselect" sortOrder="6"
showInDefault="1" showInWebsite="0" showInStore="0" canRestore="1">
    <label>Tipo de noticias que se mostrarán</label>

<source_model>Ilovemarketing\News\Model\Config\Source\NewsTypes</source_model
>
</field>
```

Vamos a preparar el **source_model** al que llamaremos **NewsTypes.php** y que se va a encargar de consultar los diferentes tipos de la tabla de **ilovemarketing_news_types** que creamos en nuestro Setup. El selector se rellena con los campos de la tabla de tipos de noticias.

Necesitamos crear un modelo para nuestros tipos de noticias, pero eso ya explicamos como hacerlo en temas anteriores.

```
<?php
namespace Ilovemarketing\News\Model\Config\Source;

use Magento\Framework\View\Element\Template;
use Ilovemarketing\News\Model\ResourceModel\News\NewsTypesFactory;
use Magento\Framework\View\Element\Template\Context;

class NewsTypes extends Template {

    protected $_newsTypesFactory;

    public function __construct(
            Context $context,
            NewsTypesFactory $NewsTypesFactory ) {

        $this->_newsTypesFactory = $NewsTypesFactory;
        parent::__construct($context);

    }

    public function getNewsTypes() {

        return $this->_newsTypesFactory->create();
```

```
    }

    public function toOptionArray() {

        $newsTypesArray = [];
        foreach ($this->getNewsTypes() as $value) {
            $newsTypesArray[] = [
                'label' => $value["news_type_name"],
                'value' => $value['news_type_id']
            ];
        }
        return $newsTypesArray;

    }
}
```

También puede darse el caso en el que querramos consultar opciones de un fichero XML. El campo se configura de la misma forma que lo hicimos con las consultas a la base de datos.

```
<!-- campo para un grupo de opciones -->
<field id="newsTypes" translate="label" type="multiselect" sortOrder="6"
showInDefault="1" showInWebsite="0" showInStore="0" canRestore="1">
    <label>Tipo de noticias que se mostrarán</label>

<source_model>Ilovemarketing\News\Model\Config\Source\NewsTypes</source_model
>
</field>
```

Tendremos que crear su fichero de configuración en **etc/** y lo llamaremos **config.xml**

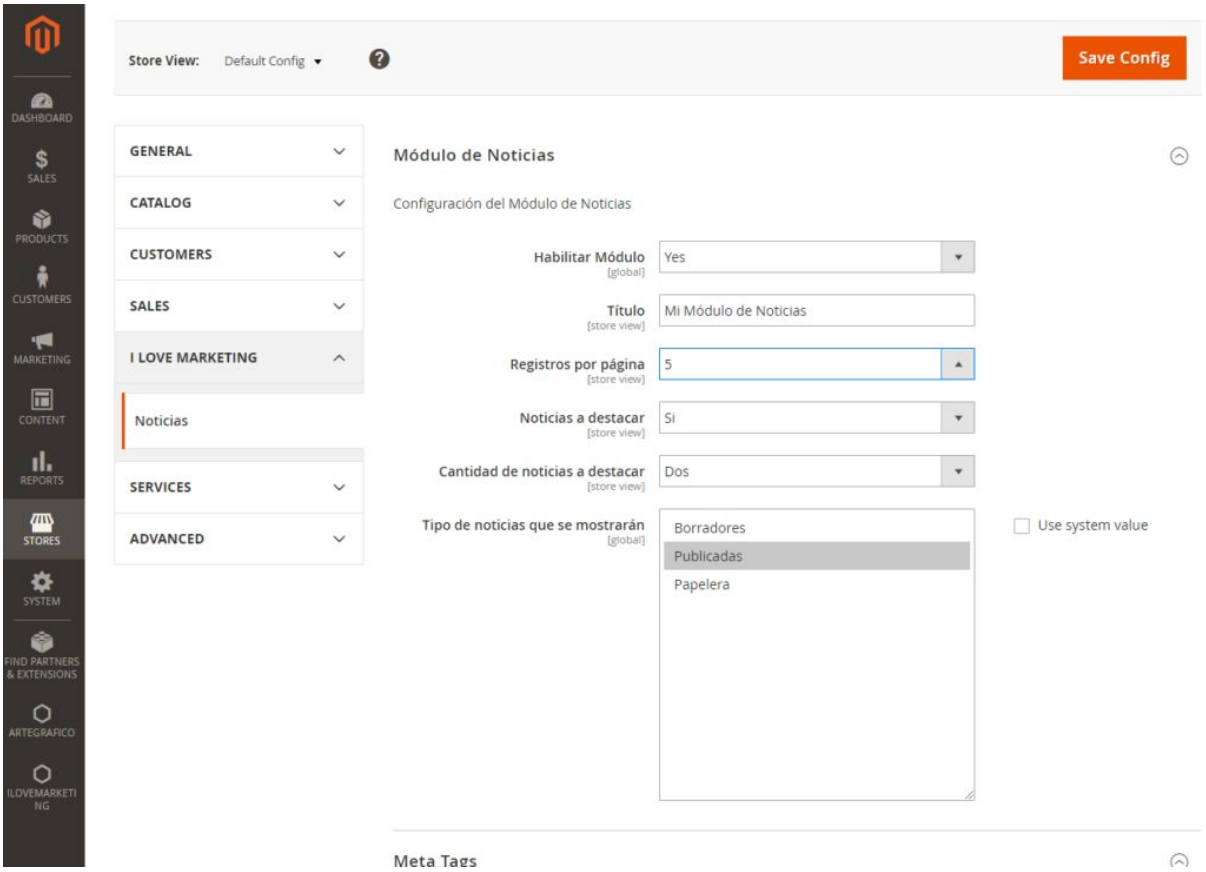
```
<?xml version="1.0" encoding="UTF-8"?>
<config>
    <default>
        <news>
            <visibility>
                <first>
                    <label>Private</label>
                </first>
                <second>
                    <label>Public</label>
                </second>
            </visibility>
        </news>
```

```
    </default>
</config>
```

Necesitaremos crear el **source_model** que va a consultar el nuevo campo en **Model/Config/** y lo llamaremos **NewsVisible.php**

```
<?php
namespace Ilovemarketing\News\Model\Config\Source;

use Magento\Captcha\Model\Config\Form\AbstractForm;

class NewsVisible extends AbstractForm {

    protected $_configPath = 'news/visibility';

    public function toOptionArray() {

        $optionArray = [];
        $backendConfig = $this->_config->getValue($this->_configPath, 'default');
        if ($backendConfig) {
            foreach ($backendConfig as $formName => $formConfig) {
                if (!empty($formConfig['label'])) {
                    $optionArray[] = [
                        'label' => $formConfig['label'],
                        'value' => $formName
                    ];
                }
            }
        }
        return $optionArray;

    }
}
```

Nuestras configuraciones en el backend se mostrarán de la siguiente forma:



Acceso a Valores de Configuración

Helpers

Para recuperar los datos provenientes de nuestros campos de configuración, vamos a crear un **helper** que nos ayudará con dicha tarea y evitando repetir código.

Empezaremos creando nuestro helper en **Helper/** y lo llamaremos **NewsConfig.php**. Nuestro helper extenderá la clase **AbstractHelper**.

Crearemos una método llamado **getConfig()** que recibira el path del campo que queremos recuperar. Recordar que el path se consigue con los identificadores únicos de **<section> / <group> / <field>**

```
<?php
namespace llovemarketing\News\Helper;

use Magento\Framework\App\Helper\AbstractHelper;
use Magento\Store\Model\ScopeInterface;

class NewsConfig extends AbstractHelper {

   public function getConfig($path) {

      return $this->scopeConfig->getValue(
         $path,
         ScopeInterface::SCOPE_STORE
      );

   }
}
```

Lo siguiente que vamos a hacer es que nuestro paginador cargue el valor de **newsLimit** que configuramos en nuestro fichero de configuración **system.xml**

Para ello vamos a editar el bloque Block/**NewsList.php** que se encarga de listar las noticias.

- Incluiremos la nueva dependencia a nuestro helper.
- Modificaremos el método **getNews()** y **_prepareLayout()** para indicarle el límite de registros que se consultarán por página.

- También podremos consultar otros valores como los meta tags.

```
<?php
namespace Ilovemarketing\News\Block;

use Magento\Framework\View\Element\Template;
use Magento\Framework\View\Element\Template\Context;
use Magento\Framework\UrlInterface; // urls
use Ilovemarketing\News\Model\ResourceModel\News\NewsFactory;
use Ilovemarketing\News\Helper\NewsConfig;


class NewsList extends Template {

   public function __construct( Context $context,
                        NewsFactory $newsFactory,
                        NewsConfig $newsConfig,
                        UrlInterface $urlInterface) {

      $this->_newsFactory = $newsFactory;
      $this->_newsConfig = $newsConfig;
      $this->_urlInterface = $urlInterface;
      // podemos usar ahora sus metodos getBaseUrl(), getUrl(), getCurrentUrl(),
getUrl('test/test2') etc ..

      parent::__construct($context);

      // get the title, description and keywords from conf

$this->pageConfig->getTitle()->set(__($this->getConfig('ilovemarketing/news/newsTitle')))
;

$this->pageConfig->setDescription($this->getConfig('ilovemarketing/newsMeta/newsMe
taDescription'));

$this->pageConfig->setKeywords($this->getConfig('ilovemarketing/newsMeta/newsMeta
Keywords'));

   }

  public function getNews()  {

      // get limit fields from configuration
      $this->_limit = $this->getConfig('ilovemarketing/news/newsLimit');

      // Sino ha parametro recibido de pagina, defecto 1
```

```
        $page=($this->getRequest()->getParam('p'))? $this->getRequest()->getParam('p') : 1;

        // limite de registros
        $pageSize = ($this->getRequest()->getParam('limit'))?
$this->getRequest()->getParam('limit') : $this->_limit;

        $news = $this->_newsFactory->create();
        $news->addFieldToFilter('news_visible',1)
            ->setOrder('news_date','desc')
            ->setPageSize($pageSize)
            ->setCurPage($page);

        return $news;

    }

    /*
     * El metodo callBack _prepareLayout()
     * Se lanza despues de que un bloque ha sido añadido al layout
     */
    protected function _prepareLayout() {

        parent::_prepareLayout();

        if ($this->getNews()) {

            $pager = $this->getLayout()->createBlock(
                'Magento\Theme\Block\Html\Pager',
                'news.news.pager'
            )->setAvailableLimit(
                    [
                        5=>5,
                        10=>10,
                        15=>15,
                        25=>25
                    ]
             )
             ->setShowPerPage(true)
             ->setLimit($this->_limit)
             ->setCollection($this->getNews());

            $this->setChild('pager', $pager);

        }

    }
```

```
    /*
     * método callbBack getPagerHtml()
     * Será llamado antes de que el bloque html sea renderizado
     */
    public function getPagerHtml() {

        return $this->getChildHtml('pager');

    }

    /*
     * helper/NewsConfig.php
     */
    public function getConfig($path) {
        return $this->_newsConfig->getConfig($path);
    }


}
```

Nuestra página de noticias presenta el siguiente aspecto: