Esaizu!

Fecha: 07/06/11	Manual de administrador y desarrollador	Estado	Confidencialidad
Autor: Mikel Madariaga	Versión 1.0 Revisado por:	Borrador [X] Revisable [ ] Final [ ]	Público [X] Interno [ ] Clientes [ ]

ÍNDICE DE CONTENIDOS

Introducción

Ámbito y naturaleza del documento

El presente documento es un manual de administrador que describe los requisitos del sistema y pasos de instalación de ***Esaizu!***, así como las posibilidades de personalización y ampliación que éste ofrece.

El proceso de instalación se divide en dos partes diferenciadas:

- **Configuración de servidor y aplicación**: Requisitos mínimos del servidor, software compatible e instalación base de la aplicación.
- **Integración con APIs de terceros**: Buena parte de las funcionalidades ofrecidas por ***Esaizu!*** requieren del registro de la aplicación en webs de terceros (Facebook, Twitter, etc).

El segundo apartado del documento, enfocado a desarrolladores, cubre los siguientes aspectos:

- **Bibliotecas utilizadas**: En el documento de hace mención a bibliotecas y frameworks PHP utilizados, facilitando enlaces a la documentación oficial y guías de introducción de estos.
- **Añadir funcionalidades**: Información detallada acerca de las posibilidades que ***Esaizu!*** ofrece a desarrolladores relacionada con la creación de nuevas funcionalidades y plugins.

INSTALACIÓN Y CONFIGURACIÓN DE SERVIDOR Y APLICACIÓN

Requisitos del servidor

Esaizu! Ha sido desarrollado, testado y optimizado para servidores con las siguientes características:

- Sistema operativo GNU/Linux
- Base de datos MySQL 5.1
- PHP 5.3 con php-cli
- Servidor web Apache

Aún así, no debería haber problemas para ser ejecutada en servidores Windows, bases de datos distintas a MySQL (Cuenta con una capa de abstracción compatible con mysql, mysqli, postgre, odbc, mssql, sqlite y oci8), u otras versiones de PHP ligeramente inferiores.

Así mismo se recomienda que el servidor cuente con algún **opcode cache** instalado (http://php.net/manual/en/book.apc.php, por ejemplo), especialmente en casos en los que un amplio número de usuarios vaya hacer uso intensivo de la aplicación.

Instalación

Breve descripción de ficheros y directorios destacados

Una vez descargada y descomprimida la aplicación nos encontraremos con la siguiente estructura de directorios y ficheros:

- application
  - cache
  - config
    - ***apikeys.php***
    - ***config.php***
    - ***database.php***
    - ***email.php***
  - logs
- doc

- public
  - *.htaccess*
  - *index.php*
- system
- *cli.php*
- *database.sql*

En el directorio raiz de ***Esaizu!*** Encontramos los fichero ***database.sql y cli.php***.

El primero contiene las tablas de base de datos que requiere la aplicación y **cli.php** es el script encargado de lanzar todos los procesos que corren en segundo plano (Actualizar el contenido de los usuarios activos entre otros).

Inmediatamente después encontramos el directorio **public**. Este es el único directorio de la aplicación accesible para el usuario final. Aquí se alojan las hojas de estilos, imágenes, scripts del lado del cliente, así como el **index.php** de ***Esaizu!*** y cualquier otro fichero que por la razón que fuera ha de ser accesible públicamente.

Dependiendo del fichero descargado (***Esaizu!*** + documentación para desarrolladores), encontraremos el directorio **doc** que contiene la documentación de las principales clases del core de la aplicación.

Por último, veremos el directorio application donde se alojan los ficheros de configuración, logs del sistema y la propia aplicación.

Base de datos

Como es habitual en la mayoría de aplicaciones web, el primer paso de instalación será crear la base de datos e importar el fichero database.sql, para de esta manera, crear las tablas.

NOTA: El fichero *database.sql* sigue la sintaxys de bases de datos MySQL, si necesitamos utilizar otra base de datos es posible que tengamos que aplicar ciertas modificaciones.

Una vez tengamos lista la base de datos, el próximo paso es crear un usuario administrador del sistema en la tabla *users*. ***Esaizu!*** admite un único administrador de sistema y éste corresponde al **id de usuario *"1"*.** Por ejemplo:

- id : 1
- userName : "miNombreDeUsuario"

- password : "miPassword" cifrado mediante la funcion crypt de PHP con salt
- email : "email@example.com"
- activated : 1

El administrador del sistema es en la práctica un usuario más, con la excepción de tener la posibilidad de validar usuarios de la aplicación y activar/desactivar plugins. Así mismo, una vez logueado el administrador será redirigido al panel de gestión de usuarios en lugar de a la propia aplicación (A la que posteriormente podrá acceder si así lo desea).

Ya tenemos la base de datos de ***Esaizu!*** lista

Servidor Web

A continuación necesitamos mover la aplicación a nuestro servidor web. Es importante recordar que el único directorio público, accesible desde el navegador, debe ser la carpeta ***public***. Si bien es cierto que éste no es un requisito para que la aplicación funcione, por motivos de seguridad, es recomendable que así sea. A continuación tenemos un ejemplo de una configuración válida para un servidor web apache:

```
<VirtualHost *:80>
    ServerAdmin webmaster@localhost
    ServerName www.example.com
    DocumentRoot /var/www/esaizu/public
    <Directory />
        Options FollowSymLinks
        AllowOverride None
    </Directory>
```

```
    <Directory /var/www/esaizu/public/>
        Options -Indexes FollowSymLinks MultiViews
        AllowOverride All
        Order allow,deny
        allow from all
    </Directory>
    ErrorLog ${APACHE_LOG_DIR}/error.log
    LogLevel warn
    CustomLog ${APACHE_LOG_DIR}/access.log combined
</VirtualHost>
```

Configurar *Esaizu!*

A continuación editaremos algunos ficheros de configuración en el directorio application/config.

- *database.php*

  Definimos el host, nombre de la base de datos, usuario ,password y driver (mysql, oracle, etc).

- *config.php*

  Es aquí donde se define la configuración básica de *Esaizu!*

  - base_url : La URL pública de la aplicación
  - appName : Nombre de la aplicación
  - public : Acepta los valores “0” o “1”.
    - Con un valor igual a 0, las cuentas de usuario han de ser validadas por el administrador del sistema.
    - Con un valor igual a 1, cualquier usuario puede registrarse y usar la aplicación sin necesidad de mediación por parte del administrador.
  - index_page : Si nuestro servidor web acepta reglas modrewrite podemos dejar este campo vacío. En caso contrario asignamos el valor “index.php”. En el proyecto se incluye un .htaccess con todas las reglas necesarias, junto al index.php dentro del directorio public. En sites con alto tráfico de datos es recomendable mover estas reglas

a la propia configuración de apache para aumentar el rendimiento del servidor web.

- encryption_key : Clave privada para cifrar datos sensibles y cookies de la aplicación.

- *email.php*

  La configuración definida en este fichero es utilizada por el sistema a la hora de enviar emails (Confirmación de usuarios por ejemplo). En mayoría de casos bastará con modificar las dos últimas líneas:

  - from : Email del remitente
  - from_name : Nombre de la aplicación

  Esaizu! Es compatible con los protocolos mail, sendmail, y smtp.

Configurar plugins

En la instalación por defecto están activados los plugins para feeds RSS/ATOM y wordpress. El resto de plugins requieren el alta en los servicios de terceros a los que se van a conectar. Para el uso de los plugins de twitter, facebook y otras plataformas de terceros se necesita registrar la aplicación para obtener las claves de conexión. A continuación se detalla el proceso de obtención de las mismas.

Facebook

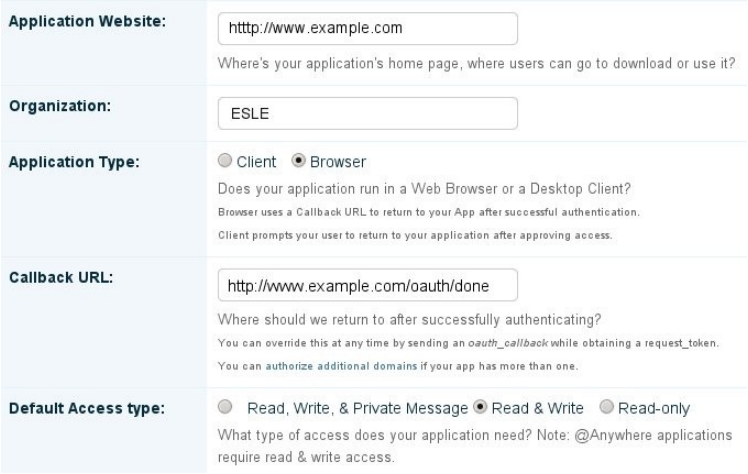
Para registrar la aplicación en la red social facebook es necesaria una cuenta de usuario en este site. Una vez logeados accedemos a la dirección http://www.facebook.com/developers/ y pulsamos sobre "*crear una nueva aplicación*".

Una vez finalizado el registro, facebook nos facilita tres códigos que debemos copiar en el fichero de configuración ***apikeys.php***.

```
$config["facebook"] = array(
    'appId' => '***************', //Identificación de aplicación
    'apiKey' => '********************************', //Clave API
    'secret' => '********************************' //Código secreto
);
```

Twitter

El registro de aplicación en twitter, red social basada en el microblogging , se realiza desde https://dev.twitter.com/apps/new. Aquí tenemos un ejemplo válido:



En el campo "Application Website" escribimos la url de la aplicación. En "Callback URL" copiamos el valor de "Application Website" y añadimos ***oauth/done***.

NOTA: Si nuestro servidor web no acepta reglas modrewrite es necesario añadir "index.php/" al final de "Application website".

En cuanto tengamos las claves "Consumer key" y "Consumer secret" las copiamos en **apikeys.php**

```
$config["twitter"] = array(
    'requestTokenUrl' => 'http://twitter.com/oauth/request_token',
    'authorizeUrl' => 'http://twitter.com/oauth/authorize',
    'accessTokenUrl' => 'http://twitter.com/oauth/access_token',
    'consumerKey' => "********************",
    'consumerSecret' => "****************************************"
);
```

Linkedin

Esaizu! también dispone de un plugin para Linkedin, una red social orientada a negocios. Para activar este plugin debemos acceder a la dirección https://www.linkedin.com/secure/developer y registrar nuestra aplicación. El formulario de registro es muy similar a otros vistos anteriormente, donde nos piden el nombre del programa, una breve descripción, url de la aplicación, etc.

NOTA: A diferencia de twitter, en "OAuth Redirect URL" debemos dejar el valor en blanco.

Una vez finalizado el registro, procedemos a copiar las claves en **apikeys.php**

```
$config["linkedin"] = array(
    'key' => '**********************************************************',
    'secret' => '**********************************************************'
);
```

Flickr

Flickr es una web para compartir o vender fotos y vídeos de manera pública o privada.

Para activar este servicio dentro de ***Esaizu!*** accedemos a http://www.flickr.com/services/apps/create/, pulsamos sobre obtener clave API y seguimos los pasos indicados en la web.

Al igual que el resto de plugins, una vez dispongas de las claves de acceso a la API, las copiamos en **apikeys.php**

```
$config["flickr"] = array(
    'key' => '********************************',
    'secret' => '****************'
);
```

Bit.ly

Bit.ly es un servicio de reducción de URLs que, de manera opcional, podemos utilizar en ***Esaizu!*** Podemos pedir las claves de bit.ly para que el enlace a los mensajes publicados desde ***Esaizu!*** y a su vez referenciados/mencionados se acorte automáticamente.

Para obtener estas claves creamos una cuenta en http://bit.ly, pulsamos sobre nuestro avatar y entramos en la sección settings donde se nos mostrará la API key.

Escribimos el nombre de usuario y contraseña en **apikeys.php** para finalizar.

```
$config["bitly"] = array(
    "user" => "nombreDeUsuarioEnBitly",
    "key" => "**********************************",
);
```

Activar plugins

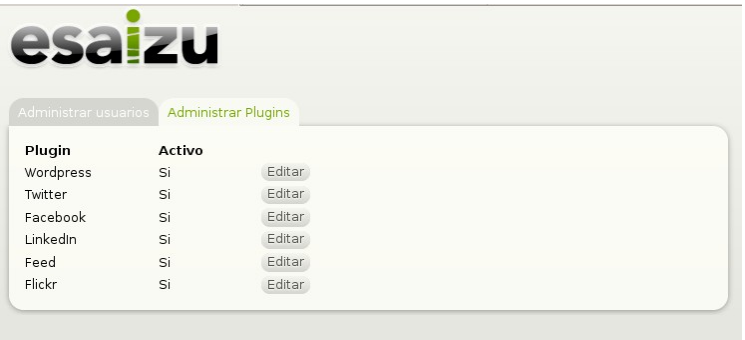
Activar nuevos plugins en ***Esaizu!*** requiere, además de las claves API de cada uno de los servicios, que estos sean marcados como activos por el administrador. Esta tarea se realiza desde el panel de administrador de la propia aplicación o bien modificando directamente valores en base de datos.

1. *Panel de administrador:*

   Tras Logearnos con el usuario administrador que creamos al principio de este documento y nos dirigimos a la sección "Gestionar plugins". Esta sección también permite modificar la frecuencia con la que ***Esaizu!*** Actualiza el contenido de los usuarios activos, si bien, los valores por defecto deberían ser correctos en la mayoría de casos.

   NOTA: Algunos servicios como flickr o twitter tienen un limite de peticiones por hora/día, infórmate antes de modificar la frecuencia de actualización.

2. *Base de datos:*



Accedemos a la tabla plugins y modificamos los valores de la columna "activated". El valor "1" corresponde a plugins activos y 0 a inactivos.

Permisos de escritura

A continuación tenemos que asegurarnos que los siguientes directorios cuentan con permisos de escritura:

- application
  - cache
  - logs
- public
  - tmp

Activar procesos en segundo plano

Para finalizar, solo nos queda añadir al gestor de tareas programas de nuestro sistema operativo una tarea que ejecute el script encargado de actualizar el contenido de ***Esaizu!*** Y enviar mensajes programados. La frecuencia de ejecución recomendada es de 1 minuto o menos.

NOTA: ***Esaizu!*** Cuenta con un sistema de control que impide que dos procesos Cron se ejecuten en paralelo.

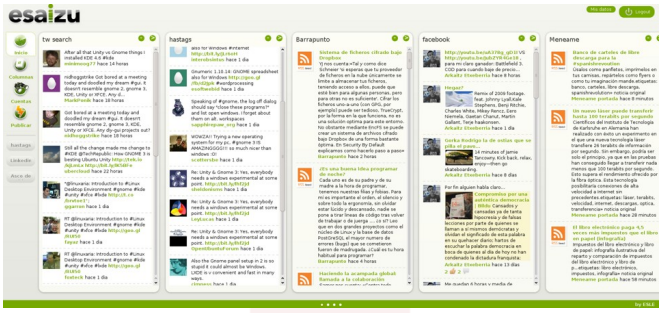
En sistemas operativos Linux, ejecutamos ***contrab -e*** desde consola y añadimos la siguiente línea (reemplazando la ruta):

* * * * * cd /ruta/a/esaizu && /usr/bin/php cli.php "cron" >> ./application/logs/cron.log

En el fichero.log indicado se recogen todas las iteraciones y posibles errores que pudieran surgir durante la ejecución. Por lo general solamente veremos mensajes como el siguiente:

```
Cron task finished at 2011-05-24 15:24:05. Elapsed time: 4 seconds.
```

Llegados a este punto podemos dar por finalizada la instalación y configuración de *Esaizu!*.



GUÍA PARA DESARROLLADORES

Bibliotecas utilizadas

Esaizu! ha sido desarrollado sobre el framework de código abierto ***codeigniter 2*** *haciendo uso puntual de la librería de* ***Zend framework 1.11.***

- *Codeigniter:* Sencillo framework PHP orientado a objetos que sigue el patrón MVC (Modelo, Vista, Controlador). Destaca por su breve curva de aprendizaje, consumo de memoria y rendimiento.
  - Web oficial : http://codeigniter.com/
  - Guia de usuario: http://codeigniter.com/user_guide/
  - Vídeo tutoriales: http://codeigniter.com/tutorials/
- *Zend framework: Framework creado por los desarrolladores de PHP, que debido a su arquitectura, permite hacer uso de su librería de componentes de manera independiente.*
  - *Web oficial: http://framework.zend.com/*
  - *Documentación: http://framework.zend.com/manual/en/*

El desarrollo de nuevos plugins y, en menor medida, modificar cualquier otro apartado de ***Esaizu!*** requerirá un dominio básico-medio de codeigniter por parte del desarrollador.

La arquitectura escogida permite que nuevos desarrolladores extiendan ***Esaizu!*** de forma sencilla sin renunciar a muchos de los servicios ofrecidos por Zend framework.

Introducción al código de E*saizu!*

Tal y cómo se detalla anteriormente, la aplicación sigue un patrón MVC que resumimos a continuación:

Controladores:

Se trata de las clases encargadas de gestionar las peticiones enviadas desde el navegador. El nombre del controlador, así como el método a ejecutar, se recogen desde la propia URL. Veamos un ejemplo:

http://www.example.com/columns/edit/90 hace una petición al controlador columns (Alojado en application/controllers/columns.php) y ejecuta el método edit que tenemos a continuación :

```
public function edit($id)
    {
        //El valor de la variable id se recoge automáticamente desde la url
        //en este ejemplo, es 90

        //Cargamos el objeto encargado de validar la sesión de usuario
        $auth = Auth::get_instance();
        $myAccounts = $auth->getEnabledAccounts();

        /*
        * La mayoría de modelos de Esaizu! Siguen el patrón Data/Mapper
        * Los objetos Data gestionan y validan los datos, mientras que, los
        * mappers se encargan de interactuar con la base de datos
        */

        //Creamos un nuevo objeto de datos del tipo Columna
        $column = new Columns_Data();

        //Asignamos el id y dueño de la columna
        $column->setId($id);
        $column->setIdU($auth->getUserId());

        //Cargamos el mapper de columnas para ejecutar una búsqueda
        $columnMapper = new Columns_Mapper();

        //El método find devuelve un array de objetos con los resultados
        //que coinciden con los valores definidos en $column
        //o un array vacío en caso de no encontrar coincidencias
        $myColumn = array_shift($columnMapper->find($column));

        if (!$myColumn instanceof Columns_Data) {

            //En caso de no encontrar la columna a modificar
            //mostramos un error en pantalla
            throw new Exception("Column not found");
        }

        $activeUserPlugins = array();
        foreach ($myColumn->getPlugins() as $plugin) {
```

```
            //Recogemos los ids de los plugins asignados a la columna
            $activeUserPlugins[] = $plugin->getId();
        }

        //Agrupamos todos los datos que tienen que llegar a la vista en un array
        $data = array(
            "obj" => $myColumn,
            "accounts" => $myAccounts,
            "activeUserPlugins" => $activeUserPlugins,
             /*Las variables "js" y "css" aceptan, de manera excepcional,
             * añadir o restar valores a la configuración predefinida
             * en application/conf/marterview.php mediante los modificadores
             * "+js", "-js", "+css" y "-css"
             */
            "+js" => array("application.columns.js")
        );

        //Cargamos la vista alojada en application/views/columns/editar.php
        $this->load->masterview("columns/editar", $data);
    }
```

Modelos:

Esaizu! cuenta con varias clases abstractas que definen los métodos públicos que los modelos deben de implementar, o simplifican el desarrollo de estos, en función del tipo de modelo. Podemos encontrarlos en el directorio application/models/Common. A continuación tenemos un breve resumen de cada uno de ellos:

- **Auth** : Clase abstracta que han de implementar todos de los plugins encargados de validar si los datos de conexión facilitados por el usuario, respecto a un servicio de terceros, son correctos. Además de definir si el plugin requiere de validación remota (oauth) o no.

- **Cron :** Clase abstracta que han de implementar todos de los plugins que necesiten realizar tareas programadas en background (Cron).

- **Data : Clase que implementan los** modelos de datos. Incluyen reglas de validación y gestión de errores para cada uno de los atributos. Suelen ser reflejo de la estructura de una tabla en base de datos, así como de sus relaciones, pero no incluyen ningún método que posibilite la interacción con esta.

- **DB :** Clase encargada de trabajar con los mensajes en bruto, recogidos desde servicios de terceros, transformarlos y guardarlos en base de datos de manera legible para el resto de la aplicación.

- **Mapper** : Los mappers son los modelos encargados de tratar con la base de datos. Por lo general tienen un objeto Data asociado que utilizan para devolver datos a la aplicación.

- **Public :** Clase abstracta que han de implementar todos de los plugins que requieran tener métodos accesibles directamente desde la aplicación. Estos métodos se ejecutan desde el **controlador invoke**, que simplemente hace de pasarela. Se utiliza, por ejemplo, para aplicar la acción "me gusta" a mensajes de facebook (Funcionalidad exclusiva para los mensajes de este site que no esta cubierta por la propia aplicación).

- **Render :** Clase abstracta que implementan los objetos Render de cada plugin. Estos modelos se encargan de dibujar los mensajes y definir las rutas de los formularios de publicación y nueva cuenta.

Debido a lo peculiar del sistema base de ***Esaizu!*** (Zend library + Codeigniter), se ha procurado unificar la nomenclatura de ambos frameworks en la medida de lo posible. De este modo, los nombres de modelos en la aplicación reflejan la ruta en la que se encuentran y son cargados al vuelo a medida que se inician (No es necesario hacer ningún require) tal y como se hace en Zend Framework.

Por ejemplo, el nombre de la clase del modelo que se encuentra en application/models/Plugins/Facebook/Cron.php será Plugins_Facebook_Cron.

En cuanto a los componentes de Zend, ejecutar "*new Zend_Oauth_Consumer()*" hace un require automático de la clase alojada en system/Zend/Oauth/Consumer.php.

Vistas

Las vistas son plantillas responsables de dibujar el HTML final a partir de los datos recibidos desde el controlador. ***Esaizu!*** cuenta con dos métodos para cargar las vistas.

- View : Método del objeto load. Se trata de la forma más sencilla de invocar una vista. Acepta tres parámetros:

  - Nombre de la vista: Nombre del fichero alojado en el directorio views o cualquiera de sus subdirectorios.

  - Datos : Array asociativo (Nombre => Valor) mediante el cual enviar los datos a la vista.

  - Dibujar : Parámetro booleano que indica a la vista si el resultado debe ser dibujado en pantalla o devuelto al controlador. False por defecto (Dibujar en pantalla).

- Masterview : Este método dibuja, además del contenido, la cabecera, ficheros javascript, hojas de estilo y pie de página de ***Esaizu!***, tal y como fuera definido en el fichero de configuración

masterview.php . Acepta cuatro parámetros:

- Nombre de la vista: ruta a la vista encargada de dibujar el contenido.
- Datos : Array asociativo (Nombre => Valor) mediante el cual enviar los datos a la vista. Los datos son mezclados con aquellos predefinidos, teniendo estos prioridad sobre los segundos.
- Configuración predefinida: Indica que configuración predefinida se ha de cargar ("default" por defecto).
- Dibujar: Parámetro booleano que indica a la vista si el resultado debe ser dibujado en pantalla o devuelto al controlador.

Desarrollo de plugins

Todos los plugins de ***Esaizu!*** siguen una estructura muy definida que solo varía en función de si se trata de plugins de solo lectura (RSS) o lectura y escritura (Twitter y Facebook entre otros).

Directorios

El primer paso para crear un nuevo plugin en ***Esaizu!*** será crear los directorios que alojan modelos y vistas. Accedemos a la carpeta application/models/Plugins y creamos un nuevo directorio cuyo nombre debe ir capitalizado y coincidir con el nombre de nuestro plugin. A continuación nos dirigimos a /application/views/plugins y repetimos el proceso.

Render

Los renders son modelos que indican a ***Esaizu!*** que formularios debe mostrar para añadir una nueva cuenta y publicar contenido con este plugin además de dibujar mensajes en la applicación. Estos modelos extienden la clase ***Common_Render*** que tenemos a continuación:

```
<?php
abstract class Common_Render
{
    protected $_ci;
    function __construct()
    {
        // Get framework instance
        $this->_ci =& get_instance();
```

```
    }
     /**
     * Devuelve la ruta al formulario para la creación de una nueva identidad.
     * False en caso de no existir
     *
     * @return string | boolean
     */
    abstract public function getNewAccountFormView();

    /**
     * Devuelve la ruta al formulario de publicación.
     * También puede añadir las dependencias javascript/css a el html final
mediante
     * los métodos injectJs y injectCss. Por ejemplo:
     * $this->_ci->output->injectJs(array(
     *      "application.publish.twitter.js")
     * );
     * En caso de que el plugin no incluya la funcionalidad de publicar,
devolver false
     * @return string | boolean
     */
    abstract public function getPublishFormView();

    /**
     * Indica si el plugin en cuestión es capaz de
     * referenciar mensajes publicados por otros plugins
     * @return boolean
     */
    abstract public function showReferenceButton();

    /**
     * Devuelve un objetivo Message_Data y lo dibuja en pantalla
     * @param Message_Data $data
     * @param boolean $return
     * @return string
     */
    abstract public function messageBox(Message_Data $data, $return = false);
}
```

Creamos un nuevo fichero en models/Plugins/Miplugin/ con el nombre Render.php, copiamos la estructura base del modelo a partir de *Common_Render* y la modificamos para que se adapte a nuestras necesidades. Nos quedará algo similar a el siguiente código:

```
<?php
Plugins_Miplugin_Render extends Common_Render
{
    function __construct()
    {
        parent::__construct();
    }

    public function getNewAccountFormView()
    {
        //return false;
        return "plugins/miplugin/new_account";
    }

    public function getPublishFormView()
    {
        //return false;
        return "plugins/miplugin/publish";
    }

    public function showReferenceButton()
    {
        //return false;
        return true;
    }

    public function messageBox(Message_Data $obj, $return = false)
    {
        return $this->_ci->load->view(
            "plugins/miplugin/message",
            array("message" => $obj),
            $return
```

```
        );
    }
}
```

Vistas

Llegados a este punto nos disponemos a crear las vistas mencionadas en nuestro Render. Dependiendo del plugin, crearemos una vista de nueva identidad (Wordpress por ejemplo muestra un formulario donde introducir dominio, nombre de usuario y contraseña), un formulario de publicación y una plantilla para dibujar los mensajes. Puedes basarte en cualquiera de los plugins existentes para completar este paso.

Auth

Todo plugin de ***Esaizu!*** Cuenta con un modelo encargado de validar los datos introducidos por el usuario al crear una nueva identidad. Dependiendo de los requisitos del plugin esta validación puede ser desde una simple comprobación de la url facilitada (Feeds) hasta peticiones oauth.

Empezaremos por crear el esqueleto del modelo a partir de *Common_Auth*:

```
class Plugins_Twitter_Auth extends Common_Auth
{
    private $_data;
    private $_config;

    function __construct()
    {
        // Call the Model constructor
        parent::__construct();

        // Cargamos la configuración del plugin si es necesario
        $this->_ci->config->load('apikeys', TRUE);
        $this->_config = $this->_ci->config->item('miplugin', 'apikeys');
    }

    /**
     * Define si la validación es remota (oAuth por ejemplo) o local
     * @return boolean
     */
```

```
    public function hasRemoteAuth()
    {
        //return false;
        return true;
    }
    public function remoteAuth()
    {
      /**
       * En caso de que el plugin haga validaciones remotas
       * escribir el código necesario aquí (Incluidos redirects a urls
       * externas si es necesario). En caso contrario dejamos el método vacío
      */
    }

    public function setData($data)
    {
         $this->_data = $data;
    }

    public function getData()
    {
        return $this->_data;
    }

    public function validate()
    {
      /**
       * Este es el método encargado de dar el visto buena a la nueva
       * identidad. Si el plugin dispone  de autentificación remota
       * se invoca después de que el usuario haya aceptado o rechazado
       * los términos en la web que ofrece el servicio (Facebook por ejemp.)
       */
       //return false;
       return true;
    }
}
```

Cron

A continuación crearemos la clase encargada de recoger el contenido de la identidad. Una vez más nos dirigimos al directorio application/models/Plugins/Miplugin para crear un nuevo script llamado Cron.php .

Los modelos Cron extienden de la clase abstracta Common_Cron y cuentan con dos métodos públicos:

- Connect : Método encargado de abrir la conexión con el servicio remoto.
- Update : Método encargado de recoger el nuevo contenido y opcionalmente guardarlo en base de datos (Por lo general los modelos Cron delegan la comunicación con la base de datos en modelos DB, si bien es cierto que éste no es un requisito de la aplicación).

El desarrollo de los modelos Cron cuenta con la dificultad añadida de ser clases que se ejecutan en segundo plano. Si bien es cierto que los errores y mensajes quedan registrados en los ficheros de log, es posible que este feedback por parte de la aplicación nos resulte insuficiente en las primeras etapas de desarrollo.

Podemos simular la ejecución de una tarea programada del siguiente modo:

- Abrimos el controlador app.php y nos dirigimos al final del mismo. Encontraremos un método privado de nombre *testing* que contiene un ejemplo. Reemplazamos el valor de `$accountId` y cambiamos "`Plugins_Facebook_Cron`" por el nombre correspondiente nuestro plugin.
- Declaramos el método testing público (public).
- Accedemos a la url "www.miesaizu.com/app/testing" para ejecutarlo.

NOTA : Para ejecutar el método testing del controlador App es necesario tener previamente una cuenta valida con el plugin a testear. Para esto es necesario registrar el plugin en base de datos, proceso que se detalla en el próximo punto de este documento.

Activar plugin

Una vez tengamos una versión preliminar de las clases mencionadas anteriormente nos dispondremos a activar el plugin en base de datos. Para ello, nos dirigimos a la tabla plugins y añadimos un nuevo registro. Aquí tenemos una breve descripción de los distintos campos de la tabla.

- name: Alias del plugin. Se trata del valor que se mostrará a los usuarios en la aplicación.
- Activated : Acepta los valores 0 y 1. Ponemos "1".
- className: Nombre del directorio padre de los modelos de nuestro plugin capitalizado.

Solamente debería contener caracteres alfabéticos.

- UpdateFrecuency: Frecuencia de actualización del contenido en segundos.

Publish

En plugins que permitan publicar contenido en servicios de terceros, aún nos queda pendiente programar la clase encargada de tal funcionalidad. Todos los publicadores de ***Esaizu!*** Extienden de la clase abstracta `Common_Publish` que define los siguientes métodos:

- connect : Método encargado de establecer requisitos y configuraciones necesarias para publicar contenido. Recibe como parámetro el token de conexión.
- validatePost : Recibe el post del formulario enviado por el usuario, así como la cuenta destino. Se encarga de validar que el contenido a publicar es correcto y de no ser así, asigna los mensajes de error. Devuelve un objeto del tipo Message_Data.
- publish : Método que se ejecuta inmediatamente después de `validatePost` siempre y cuando no se devuelvan errores. Es el encargado de publicar el mensaje.
- Reference : Este método nos permite publicar breves referencias a mensajes publicados en otros servicios habilitados en ***Esaizu!***. Recibe un objeto del tipo `Message_Data` y la cuenta en la que se ha de refenciar.