Sincronización de artículos en 2 sitios Web, usando la API REST de WordPress

Memoria de Proyecto Final de Máster

Máster Universitario en Desarrollo de Sitios y Aplicaciones Web

Autor: Pablo Blanco Berguño

Consultor: Carlos Caballero González

Profesor: César Pablo Córcoles Briongos

6 de enero de 2020

Créditos/Copyright

Textos

Licencia

Dedicatoria

Dedico el trabajo al motor de mi vida. Sin ella todo este esfuerzo no tendría sentido.

Cita

Uno de los factores con más influencia en la productividad de hoy es la calidad del código escrito ayer.

Anónimo.

Abstract

Este documento describe la aplicación web desarrollada para facilitar la sincronización de los artículos (posts) de un portal web, con los existentes en otro portal web. Ambos sitios están desarrollados con WordPress, y se pretende utilizar la API REST que incorpora el propio WordPress en su núcleo como herramienta para el acceso y modificación de los datos.

El supuesto práctico del que se parte es la existencia de un portal web principal, en producción, y un portal web en construcción, que pasará a sustituir al primero cuando finalice su desarrollo. La herramienta desarrollada permitirá al portal web en fase beta mantener una copia de los posts del portal web principal

Para la elaboración del front-end de la herramienta desarrollada se ha decidido incorporar React, como uno de las librerías javascript más importantes en la actualidad. Además se utilizará jQuery, ya incorporada por WordPress.

Palabras clave: Memoria, TFM, WordPress, React, API REST.

Abstract (english version)

This document describes the web application developed to facilitate the synchronization of posts from two web sites. Both sites are developed with WordPress.

I want to use the WordPress REST API as a tool for accessing and modifying data.

The practical case of which it is based, is the existence of a main web site, in production, and a web site under construction, which will replace the first one when its development ends. The tool developed will allow the beta web site to keep a copy of the main web site posts

React has been used to build the front-end, as one of the most important JavaScript libraries today. In addition, jQuery will be used, already incorporated by WordPress.

Keywords: Memory, TFM, WordPress, React, API REST.

Índice

Figuras y tablas

Índice de figuras

1. Introducción/Prefacio

1.1 Contexto y Justificación del Trabajo

El tema central de este trabajo es la API REST de WordPress y mi interés por profundizar en su conocimiento.

Figura 1: Logo de la API REST de WordPress

Automattic[1] ha puesto el acento en la API REST en los últimos años. Aunque hace ya bastante tiempo que WordPress es algo más que un simple CMS, la decisión de incluir esta API en su *core*[2], nos habla de un camino dirigido a extender las funciones de WordPress a una administración de los datos independiente de la aplicación.

La API REST aporta a cualquier sitio web creado con WordPress la capacidad de comunicarse e intercambiar datos con otro tipo de aplicaciones externas, no sólo aplicaciones web, con independencia del lenguaje usado por dichas aplicaciones. No hay porque limitarse a usar el bakc-end de WordPress, ni siquiera PHP como lenguaje de programación. Gracias a la API REST, se podrán desarrollar aplicaciones Web, aplicaciones móviles o aplicaciones de escritorio, que se relacionen con la API central de WordPress.

Es esta capacidad de comunicación la que me interesa conocer, desde un punto de vista técnico y funcional, con el objetivo de marcarme nuevas metas en un futuro basándome en WordPress, adaptando esta impresionante herramienta a nuevas formas de uso y de tipos de proyectos.

Para ello, el trabajo tratará de desarrollar un ejemplo de uso, inspirado en un artículo del portal web **Torque**[3]. La idea es la de mantener sincronizados dos portales web independientes, ambos creados con WordPress. Uno de ellos es el portal web principal y el otro un portal web beta que sustituirá en el futuro al portal web principal. El trabajo incluirá además una aplicación web, independiente de las anteriores, que

[1] Automattic Inc. es la empresa que tutela el desarrollo de WordPress y su fundación.

[2] La API REST se incorpora al núcleo de WordPress en la versión 4.7, en el año 2016. Hasta ese momento, dicha API se integraba en el sistema mediante la instalación de un plugin, no incluido en la instalación por defecto de WordPress.

[3] Portal web con artículos de desarrollo en WordPress. El artículo inspirador es: https://torquemag.io/2017/05/ways-start-using-wp-rest-api/

incorporará la parte front-end practicada durante el máster, y que será la encargada de configurar, establecer y monitorizar la comunicación entre los otros dos sitios web.

1.2 Objetivos del Trabajo

Los objetivos del trabajo, a un nivel general son:

- ***Creación de los portales web a sincronizar***

  Implantación de los dos portales web que se van a sincronizar. Implicará la configuración de servidores, así como la instalación y/o configuración de los recursos utilizados por los sitios web.

- ***Desarrollo de la aplicación que configura y establece la sincronización.***

  Se trata de la aplicación web, independiente de las otras 2, desde la que se establecerá la configuración, sincronización y monitorización de la sincronización.

- ***Desarrollo del backend de sincronización.***

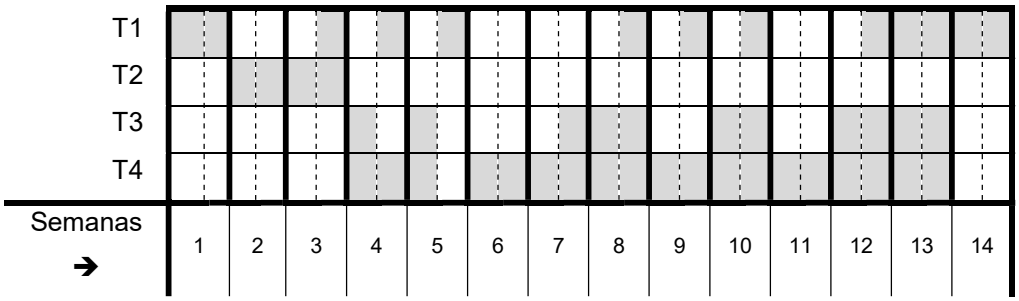
  Desarrollo de la funcionalidad que, basada en la API REST de Wordpress, implemente toda la sincronización.

1.3 Enfoque seguido

El **enfoque** del proyecto es “Profesionalizador”, es decir, se trata de buscar y desarrollar una solución a un supuesto práctico utilizando una tecnología concreta.

1.4 Planificación del Trabajo

El siguiente diagrama pretende ilustrar la distribución temporal del trabajo:



T1 ➔ Elaboración de la memoria y otra documentación.
T2 ➔ Desarrollo de los portales web a sincronizar.
T3 ➔ Desarrollo de la aplicación (front-end) que configura y monitoriza la sincronización.
T4 ➔ Desarrollo del back-end basada en el uso de la API REST.

2. Justificación de la herramienta

2.1 Una situación hipotética, pero muy real

Partimos de una situación hipotética, en la que una empresa u organización tiene un portal web en funcionamiento o producción, poblado de contenidos, desarrollado con WordPress.

Para nuestras explicaciones, denominemos a éste como el “Portal Web Principal”.

La misma empresa u organización está desarrollando otro portal Web, de forma paralela, con el objetivo de sustituir en un futuro al “Portal Web Principal”, en funcionamiento actualmente. Ese portal Web, en fase beta, puede que tenga una estructura o aspecto distinto, distintas funcionalidades, o ambas cosas. El portal Web nuevo estará desarrollado también con WordPress.

Éste es el “Portal Web Beta”.

2.2 Justificación de nuestra herramienta:

Los contenidos del portal web principal deberán seguir existiendo en el nuevo portal web cuando éste pase a ser el portal web principal. Además, el acceso a dichos contenidos tiene que seguir siendo fluidos en el nuevo portal, sin que las novedades funcionales, estéticas o estructurales que puedan existir afecten a dicha **accesibilidad**.

Para que los desarrolladores del portal web beta puedan probar su trabajo, se desea que dicho portal beta tenga una copia de los artículos (posts) actuales y de todos los nuevos que vayan surgiendo mientras dure el proceso de desarrollo.

Es aquí donde entra en juego nuestra herramienta de sincronización.

La herramienta desarrollada en este proyecto debe facilitar la sincronización entre ambos portales, de manera que el portal web beta tenga una copia de los contenidos del portal web principal, más concretamente de los artículos (posts).

Para ello, la herramienta debe ofrecer las siguientes **funcionalidades**:

1. Configurar los parámetros necesarios para la sincronización.
2. Ser capaz de analizar y mostrar el estado de la sincronización entre ambos portales en el momento actual.
3. Iniciar procesos de sincronización, cuando sea necesario, a petición del usuario que la utiliza.
4. Mantener un log en el que se pueda consultar cronológicamente los accesos a los datos de ambos portales, así como las modificaciones realizados como consecuencia de la sincronización.

3. Descripción

Se trata de un desarrollo web, independiente de los portales web principal y beta, descritos en el punto anterior, que ofrece al usuario un interfaz para el acceso a las 4 funcionalidades nombradas.

Para agilizar el desarrollo, se montará también sobre un sitio web basado en WordPress. La idea es aprovechar algunas de las funcionalidades que ofrece WordPress en aspectos como la gestión de usuarios, las clases para la gestión de la base de datos, tratamiento de las urls, filtros, etc., y poder centrarme así en los aspectos centrales del proyecto: La API REST de WordPress y el uso de un marco o una librería de última generación para el desarrollo del front-end.

3.1 Diagrama de nodos:

En este tercer portal web, independiente de los portales a sincronizar, se encontrarán los elementos siguientes:

- **Back-end**: Las clases y funciones que implementen el acceso a los contenidos de en ambos portales, principal y beta, así como la actualización del portal beta.
- **Front-end**: Interfaz de usuario para la configuración, monitorización y activación de la sincronización.
- **Base de datos**: Incorporaré a la base de datos de WordPress las tablas necesarias para almacenar información sobre la configuración de la sincronización (identificación de los sitios web y datos de autenticación), así como un log para almacenar los eventos producidos durante la sincronización.

Con lo anterior, más los dos portales a sincronizar, podemos describir el sistema con el siguiente diagrama:

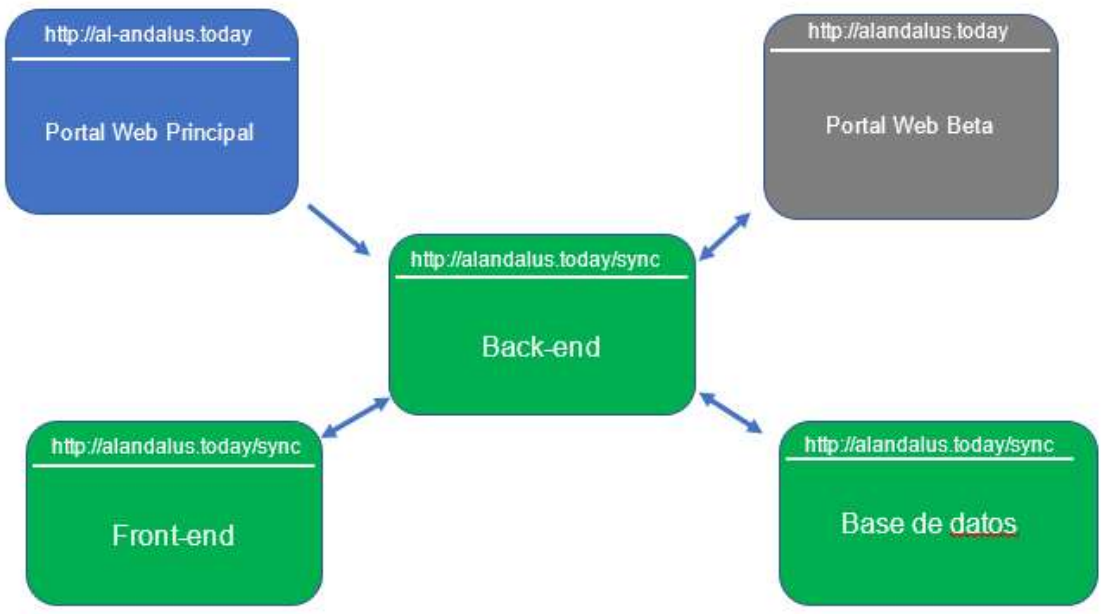


Figura 2: Diagrama de Nodos

3.2 Hostings para implantar el proyecto:

Todo el sistema se ha implantado en un servidor dedicado de uso personal, en el cual he creado los tres portales, con las siguientes urls de acceso:

- Sitio Web Principal:

  http://al-andalus.today

- Sitio Web Beta.

  http://alandalus.today

- Sitio Web para la gestión de la sincronización.

  http://alandalys.today/sync

Como se puede observar, sólo he creado dos hostings. Bajo el mismo dominio que el sitio web beta, he creado una instalación de WordPress, independiente de la otra, para la gestión de la sincronización.

3.3 Técnología en la que se basa el fron-end:

Para la implementación del fron-end, finalmente he decidido utilizar REACT[4], así como jQuery en alguno de los puntos del proyecto.

He utilizado REACT para desplegar el interfaz de usuario en las páginas de monitorización y sincronización, así como en la que muestra el log de eventos del back-end.

He utilizado jQuery en la página dedicada a la introducción de los datos de configuración.

3.4 Integración del backend y frontend en WordPress:

He creado un theme, hijo del theme Twenty Nineteen[5], para la creación de las plantillas que me faciliten la creación del front-end, la carga de las librerías javascript utilizadas, así como las clases PHP desarrolladas para el uso de la API REST en el back-end.

He denominado "**Sync Theme**" a este tema hijo. Lo podemos encontrar bajo el directorio **sync-theme**, dentro del directorio que WordPress dedica a los temas "*wp-content/themes*", dentro de la instalación de WordPress *http://alandalus.today/sync*.

Dentro del directorio sync-theme podemos encontrar todo el desarrollo creado para la sincronización de los portales web principal y beta.

---

[4] Se trata de una biblioteca Javascript de código abierto diseñada para crear interfaces de usuario, desarrollada en el seno de Facebook.

[5] Incluido en Wordpres por defecto en su versión 5.0.

Describimos a continuación los archivos y directorios contenidos en el tema "**Sync Theme**", que incorporan toda la funcionalidad de la aplicación desarrollada:

- 🗀 **sync-theme:**
  - **functions.php**

    Es el archivo que utiliza el theme para cargar sus funcionalidades, como por ejemplo quien es el theme padre. En nuestro caso lo usaremos para indicar las rutas de acceso a los archivos en los que hemos escrito el código fuente, así como para especificar alguna "action" de WordPress.
  - **sync-logs.php**

    Plantilla usada por el theme para cargar la página de carga de los logs.
  - **sync-post-state.php**

    Plantilla usada por el theme para cargar la página de monitorización de la sincronización.
  - **sync-config.php**

    Plantilla usada por el theme para cargar la página de configuración de la sincronización.
  - **style.css**

    Hoja de estilos usada por el theme.
  - **style_adm.css**

    Hoja de estilos cargada cuando se accede a alguna de las tres páginas con funcionalidad desarrolladas (configuración, logs y sincronización).
- 🗀 **includes**
  - **pb_constants.php**

    Definición de algunas constantes
  - **pbfn_general.php**

    Implementación de la carga de estilos y archivos javascripts, funciones que intervienen en el registro de los logs, así como defincón de endpoints de la API REST.
  - **pbclass_app.php**

    Clase que representa la aplicación, para el acceso a propiedades generales del portal: usuario que inicia sesión, urls y paths.
  - **pbclass_postinfo.php**

    Clases que apoyan a la clase TSync para la gestión de la sincronización.
  - **pbclass_sync.php**

Clase en la que se implementa la funcionalidad de sincronización.

- **pbclass_user.php**

  Clase para el acceso a los datos del usuario que accede a la aplicación.

🗀 **js**

- **pb_sync.js**

  Archivo con el código REACT/Javascript que facilita la creación del interfaz de sincronización.

- **pb_logs.js**

  Archivo con el código REACT/Javascript que facilita la creación del interfaz de acceso a los logs.

- **pb_config.js**

  Archivo con el código jQuery/Javascript que facilita la modificación de los datos de configuración.

3.5 Instalación de la herramienta:

Para instalar la herramienta simplemente hay que copiar el theme "**Sync Theme**" en el directorio de themes de WordPress y activarlo.

La activación del theme dispara la action "**load-themes.php**". Hemos implementado el hook "**pb_init_theme**" para que se dispare durante dicha action. Este detalle se puede encontrar en el archivo *functions.php* del theme y la línea en la que defino el hook es:

```
add_action('load-themes.php',  'pb_init_theme');
```

La función "**pb_init_theme**" comprueba que existan las tablas de configuración de la herramienta y de registros de los logs. Si no existen se crean. Además, después de la creación de la tabla de configuración, se inserta un registro que es el que almacenará la información de configuración.

La implementación de esta función se encuentra dentro del theme, en el archivo *pbfn-general.php*.

4. Arquitectura de la aplicación

A continuación, podemos observar un esquema general de los 3 portales web.

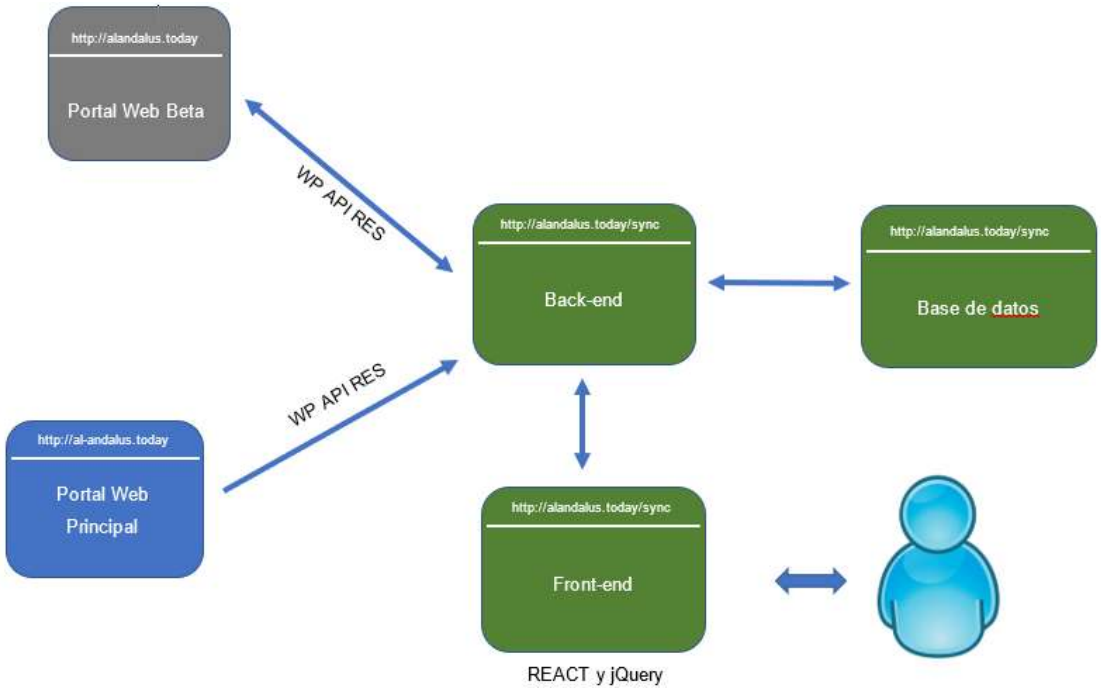


Figura 3: Esquema de la aplicación con las tecnologías usadas.

Las aplicaciones web son, en general, aplicaciones de 3 capas: una capa para el interfaz de usuario, una capa para la lógica de negocio en el backend, y una tercera capa para el sistema gestor de base de datos (incluso aunque se encuentre en el mismo servidor que el backend).

En nuestro esquema general, en el que incluimos los 3 portales web, podemos identificar una capa adicional que facilita la comunicación e interacción entre los portales web a través de la API REST.

La implementación de esa comunicación entre sitios remotos se encuentra en el backend de nuestra aplicación.

4.1 El Backend y la API REST de WordPress

El acceso a los artículos (posts) de los portales web principal y beta, así como la creación o eliminación de artículos en el portal beta, se realizan desde el backend de nuestra aplicación utilizando la API REST de WordPress.

La API REST proporciona una interfaz para que las aplicaciones interactúen con el sitio web creado con WordPress, enviando y recibiendo datos como objetos JSON (JavaScript Object Notation), y utilizando http como protocolo de comunicación.

La API REST proporciona puntos finales REST (URLs) que representan los artículos, páginas, taxonomías y otros tipos de datos integrados de WordPress. Nuestra aplicación puede enviar (y recibir) datos JSON a estos puntos finales para consultar los artículos en el sitio principal y beta, así como para crear y eliminar artículos en el sitio beta. Las respuestas también son devueltas codificadas en JSON.

El esquema siguiente representa con bastante fidelidad la arquitectura de los sistemas que usan una API REST.

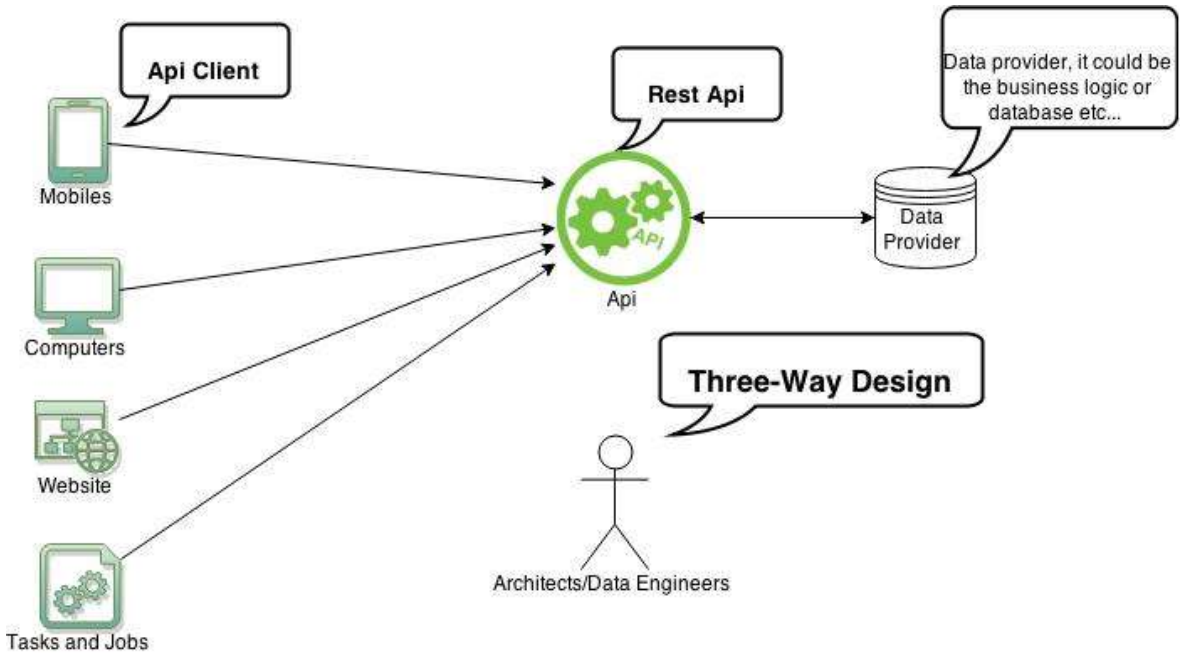


Figura 4: Arquitectura de un sistema que usa una API REST.

Los puntos finales de la API REST

Aunque el propio protocolo http define un pequeño conjunto de operaciones o métodos, como GET, PUT, POST o DELETE [6], son realmente lo endpoints los que definen los recursos disponibles a través de la API REST.

Un punto final especifica cual es la url que nos permite acceder a un recurso en el servidor gracias a la API REST, utilizando para ello un método HTTP. Por ejemplo, en WordPress podríamos realizar una petición HTTP usando el método GET a la siguiente url, para acceder a todos los posts del portal al-andalus.today:

```
http://al-andalus.today/wp-json/wp/v2/posts
```

En el anterior ejemplo, la base para acceder a la API REST en WordPress es *http://al-alandalus.today/wp-json/wp/v2*, es decir, la raíz del portal web (*http://al-alandalus.today*) seguido del fragmento */wp-json/wp/v2*.

[6] Para consultar todos los métodos http podemos consultar la siguiente página de la Wikipedia: https://es.wikipedia.org/wiki/Protocolo_de_transferencia_de_hipertexto#M%C3%A9todos_de_petici%C3%B3n

Después de dicha base, especificaremos los endpoints de acceso a cualquiera de los recursos que necesitemos. Podemos consultar los endpoints definidos en WordPress por defecto, en la versión actual de la API REST, en el siguiente enlace:

```
https://developer.wordpress.org/rest-api/reference/#rest-api-developer-endpoint-reference
```

Cuando se definen los endpoints también se especifica el **método HTTP que debería utilizarse**. Así, por ejemplo, en WordPress se utiliza GET para el acceso a los posts, POST para crear o modificar un post, y DELETE para eliminarlo.

Como ya hemos indicado, la API REST de WordPress utiliza JSON para codificar la información. Para poder procesar la información devuelta por un endpoint necesitamos conocer cuál es el **esquema que define la estructura del objeto JSON** retornado.

Así, por ejemplo, el esquema que contiene la información de un post es el siguiente:

```
https://developer.wordpress.org/rest-api/reference/posts/
```

Endpoints utilizados en la aplicación:

Para la consulta de los posts en el portal web principal utilizamos el siguiente endpoinnt usando el método HTTP GET:

```
http://al-andalus.today/wp-json/wp/v2/posts
```

Para la consulta de los posts en el portal web beta utilizamos el siguiente endpoinnt usando el método HTTP GET:

```
http://alandalus.today/wp-json/wp/v2/posts
```

Para el acceso a un post individual simplemente tendría que añadir al final de la url el identificador único del post. Por ejemplo, para acceder al post cuyo identificador es 6 en el portal beta, usaría la siguiente url:

```
http://alandalus.today/wp-json/wp/v2/posts/6
```

Para la creación de los de los posts en el portal web beta he utilizado el siguiente endpoint junto al método http POST:

```
http://alandalus.today/wp-json/wp/v2/posts
```

Junto a la petición POST, enviaremos en la petición http los datos del nuevo post, con los argumentos definidos en: *https://developer.wordpress.org/rest-api/reference/posts/#arguments-2*

Para la eliminación un post individual en el portal beta, usaría el mismo endpoint que en el acceso individual de un post, especificando el identificador único del post, pero en este caso usando el método http DELETE en lugar de GET. Por ejemplo, para eliminar el post cuyo identificador es 66, usaría la siguiente url:

```
http://alandalus.today/wp-json/wp/v2/posts/66
```

Para las peticiones en el backend utilizo la librería CURL de PHP. A continuación, muestro un ejemplo de uso, con las explicaciones de lo que hace el código fuente.

El siguiente es un método de la clase TSync (encargada de la sincronización), el cual realiza una petición HTTP GET a la url pasada como parámetro:

```php
protected function getRemoteData ($aUrl) {
    // Proceso en backgorund
    $mSh = curl_init($aUrl);
    if ($mSh === FALSE) {
        registrarLog(-1, 'Obteniendo datos remotos', 'No se pudo inicializar curl para la url: '.$aUrl);
        return NULL;
    }
    else {
        // Especifico el "content-type" para una codificación adecuada.
        curl_setopt($mSh, CURLOPT_HTTPHEADER, array('Content-Type: application/json; charset=utf-8'));
        // Si no especifico un "user agent" el servidor rechaza la petición.
        curl_setopt($mSh, CURLOPT_USERAGENT, 'Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:71.0)
            Gecko/20100101 Firefox/71.0');
        curl_setopt($mSh, CURLOPT_SSL_VERIFYPEER, FALSE);
        // Para que retorne el resultado como un string, en lugar de mostrarlo.
        curl_setopt($mSh, CURLOPT_RETURNTRANSFER, TRUE);
        // Realizo la petición
        $mData = curl_exec($mSh);
        // Cierro el recurso curl abierto
        curl_close($mSh);
        if ($mData === FALSE) {
            registrarLog(-1, 'Obteniendo datos remotos', 'No se obtuvieron los datos de la url: '.$aUrl);
            return NULL;  // Se produjo algún error
        }
        else {
            registrarLog(-1, 'Obteniendo datos remotos', 'La petición se ha atendido correctamente. Url: '. $
                aUrl);
            return $mData;
        }
    }
}
```

Definición de puntos finales personalizados en la API REST de WordPress

Aunque no era necesario, por pura experimentación, he creado 2 endpoints personalizados en la aplicación desarrollada, para el acceso desde el **front-end** a los datos de monitorización de la sincronización, así como a los logs registrados.

- Para el acceso a los datos de monitorización de sincronización, el endpoint es:
  `http://alandalus.today/sync/wp-json/sync/v1/sync-status/`

  Dicho endpoint debe invocarse utilizando el método http GET, y retorna, en formato json, una lista de objetos resultado de la comparación de los posts existentes en el sitio web remoto y beta. Cada elemento de la lista representa la información de la comparación de un post y contiene los siguientes campos:
  - **slug**: fragmento de la url correspondiente al acceso a
  - **where**: un identificador que nos indica si el post está sólo en uno de los portales o en ambos.
  - **mainPost**: datos descriptivos del post en el portal principal

- **betaPost**: datos de descriptivoss del post en portal beta.

Los campos “mainPost” y “betaPost” son a su vez objetos JSON con los siguientes campos:

- **id**: identificador único del post.
- **creacion**: fecha de creación del post.
- **modificacion**: fecha de la última modificación del post.
- **titulo**: título del post.
- **url**: url de acceso al post en el portal principal o beta, según el caso.

- Para el acceso a los logs registrados, el endpoint es:
  `http://alandalus.today/sync/wp-json/sync/v1/sync-logs/`

  Dicho endpoint debe invocarse utilizando el método http GET, y retorna, en formato json, una lista de objetos resultado de la consulta a la tabla de logs:
  - **fecha**: fecha en la que se registró el log.
  - **hora**: hora en la que se registró el log.
  - **proceso**: identifica el tipo de proceso que se estaba ejecutando durante el registro del log.
  - **mensaje**: texto con la información que se pretende registrar.

El código fuente para definir en WordPress dichos endpoints es el siguiente. En primer lugar indicamos que deseamos que se ejecute nuestra función “pb_rest_api_local_init” durante la action “rest_api_init”. Esta línea la incluimos en el archivo “functions.php” de nuestro theme hijo:

```php
// Inicializo la REST API Local
add_action( 'rest_api_init', 'pb_rest_api_local_init');
```

La implementación de la función “pb_rest_api_local_init” es la siguiente (la podemos encontrar en el archivo “includes/pbfn-general.php” de nuestro theme hijo):

```php
// Definción de endpoints personalizados en la API REST
function pb_rest_api_local_init () {
    // Definición del endpoint para el acceso al estaod de la sincronización
    register_rest_route( 'sync/v1', '/sync-status/', array(
        'methods' => WP_REST_Server::READABLE,
        'callback' => 'status_endpoint'
    ));

    // Definición del endpoint para el acceso a los logs
    register_rest_route( 'sync/v1', '/sync-logs/', array(
        'methods' => WP_REST_Server::READABLE,
        'callback' => 'logs_endpoint'
    ));
}
```

Como se puede observar en la función “pb_rest_api_local_init”, para la definición de cada endpoint tengo que especificar:

- La url de acceso al endpoint.
- Qué tipo de método http se puede utilizar para el acceso al endpoint.
- Qué función se ejecuta para procesar la petición y retornar el resultado.

Autenticación en la API REST de WordPress

Algunas acciones en la API REST de WordPress son públicas, como por ejemplo el acceso a los posts del portal principal o beta. Otras operaciones, en cambio, necesitan algún tipo de autenticación, como por ejemplo la modificación, creación o eliminación de posts.

El objetivo de la autenticación es garantizar que la aplicación cliente de la API REST (la que realiza la petición) se identifica como un actor autorizado para el acceso al recurso que requiera dicha identificación.

Hay varias formas de realizar dicha autenticación, entre otras:

- La soportada de forma nativa por WordPress es la **Cookie de Autenticación**. Se puede consultar todos los detalles en: https://developer.wordpress.org/rest-api/using-the-rest-api/authentication/
- **Básica**: especificando en cada petición un nombre de usuario y contraseña existente en el sitio remoto, con la capacidad de realizar las operaciones en cuestión (modificar, crear, eliminar, etc.,).
- Delegando en el protocolo **OAuth** la autorización. Recomendamos la lectura del siguiente artículo para más detalles: https://code.tutsplus.com/es/tutorials/wp-rest-api-setting-up-and-using-oauth-10a-authentication--cms-24797

Por simplicidad, hemos utilizado la segunda opción. Esta forma supone el envío del nombre del usuario y su contraseña en las cabeceras http de cada petición, razón por la cual no se considera segura y no se recomienda en un sitio en producción, sólo durante el desarrollo y testing.

Para dar soporte a la autenticación básica hemos instalado en el sitio web beta (el único sobre el que se van a realizar operaciones que requieran autenticación) el plugin “**Basic-Auth**”, creado por los desarrolladores de WordPress para su uso sólo durante el desarrollo o testing:

```
https://github.com/WP-API/Basic-Auth
```

Además, hemos necesitado añadir la siguiente línea al archivo .htaccess del sitio web beta:

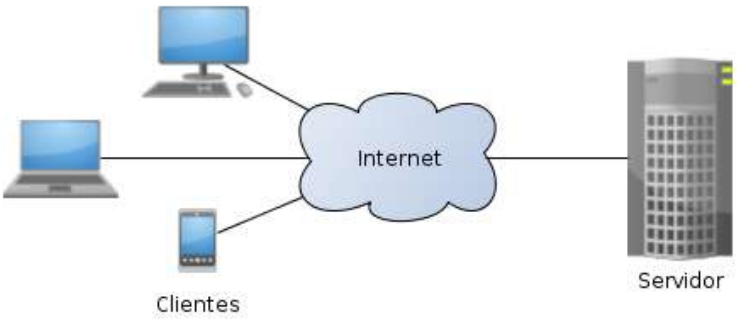
```
SetEnvIf Authorization "(.*)" HTTP_AUTHORIZATION=$1
```

Esta modificación en el archivo .htaccess tiene que ver con la configuración de Apache en nuestro servidor y no tiene por qué ser necesaria en otros casos. Lo que hace es definir la variable de entorno, relacionada con la autenticación, que debería enviarse en las cabeceras http con cada petición que lo requiera.

4.2 El Frontend: jQuery, React y … también la API REST.

Una aplicación web es un ejemplo clásico de software con arquitectura cliente-servidor. Tenemos por un lado una parte del sistema que funciona como proveedor de servicios o recursos, representado en nuestro caso por la funcionalidad implementada en código php en el lado del servidor, así como por la base de datos; y por otro lado, la parte del sistema que facilita al usuario el acceso a dichos servicios o recursos, representado por un interfaz ejecutado en el navegador web del usuario.

Figura 5: Arquitectura cliente-servidor en una aplicación web. [7]



Una aplicación web clásica escrita con PHP.

En una aplicación web clásica escrita con PHP, gran parte de la funcionalidad está implantada en las propias plantillas de html, incrustando en ellas el código PHP necesario. Ese código se ejecuta en lado del servidor y su resultado se incrusta en la plantilla que, parcial o totalmente, se utiliza para construir el documento html, que finalmente es devuelto al cliente que lo solicitó.

Siguiendo ese esquema clásico está realizada la página de configuración de nuestra herramienta, aquella en la que se guardan los datos de los dominios, así como los nombres y contraseñas de los usuarios administradores en los sitios principal y beta, para los accesos que requieran autenticación con la API REST.

Dicha página está creada con la plantilla "**sync-config.php**" de nuestro theme "**Sync Theme**" y es accesible a través de la url:

```
http://alandalus.today/sync/config/
```

Mostramos a continuación un fragmento de código de dicho archivo sync-config.php, concretamente aquel que despliega el formulario para la edición de los datos de configuración. Se puede observar como el código

[7] Fuente Wikipedia: https://es.wikipedia.org/wiki/Cliente-servidor#/media/Archivo:Cliente-Servidor.png

PHP ejecutado se utiliza para asignar el valor al atributo “value” de los elementos de tipo “input text”, es decir, aquellos utilizados para editar los datos. se ejecuta código php:

```
<form id="formconfig">
    <fieldset style="margin-top:0px;">
        <?php global $gSync; ?>
        <legend> Sitio Web Principal</legend>
        <p>
            <label for="url_main">Url:</label> <br />
            <input type="text" name="url_main" id="url_main" value="<?php echo($gSync->UrlMain); ?>" maxlength="250" />
        </p>
        <p>
            <label for="user_main">Administrador:</label><br />
            <input type="text" name="user_main" id="user_main" value="<?php echo($gSync->UserMain); ?>" maxlength="60" />
        </p>
        <p>
            <label for="passw_main">Contraseña:</label><br />
            <input type="password" name="passw_main" id="passw_main" value="<?php echo($gSync->PasswMain); ?>" maxlength="60
            " />
        </p>
    </fieldset>
    <fieldset style="margin-top:0px;">
        <legend>Sitio Web Beta</legend>
        <p>
            <label for="url_beta">Url:</label> <br />
            <input type="text" name="url_beta" id="url_beta" value="<?php echo($gSync->UrlBeta); ?>" maxlength="250" />
        </p>
        <p>
            <label for="user_beta">Administrador:</label><br />
            <input type="text" name="user_beta" id="user_beta" value="<?php echo($gSync->UserBeta); ?>" maxlength="60" />
        </p>
        <p>
            <label for="passw_beta">Contraseña:</label><br />
            <input type="password" name="passw_beta" id="passw_beta" value="<?php echo($gSync->PasswBeta); ?>" maxlength="60
            " />
        </p>
    </fieldset>
    <div class="btnbox">
        <button id="savebtn" title="Guardar cambios">Guardar</button>
    </div>
</form>
```

En este caso, el interfaz de usuario se construye totalmente en el backend. El frontend es muy básico. Incluiremos el código javascript que se encargue del evento clic del botón “Guardar”, con una sencilla validación para garantizar que los campos (los valores en los controles input) no sean vacíos. Utilizaremos para ello la librería jQuery [8].

La implementación de dicho código podemos encontrarlo en el archivo “js/pb_config.js”.

A la derecha mostramos el fragmento del código fuente que implementa el botón guardar. Se observa que utilizamos Ajax para procesar la petición.

```
$("#savebtn").bind('click', function (e) {
    if (validarDataUser()) {
        $("#infoprocess").html('Procesando informaci&oacute;n, espere por favor ...');
        $.ajax({
            url: 'http://alandalus.today/sync/wp-admin/admin-ajax.php',
            dataType: 'json',
            type: 'POST',
            data: {
                url_main: $("#url_main").val(),
                url_beta: $("#url_beta").val(),
                user_main: $("#user_main").val(),
                user_beta: $("#user_beta").val(),
                passw_main: $("#passw_main").val(),
                passw_beta: $("#passw_beta").val()
                action: 'set_config_data'},
            success: function(resultado, aTextStatus, aJqXHR) {
                if (resultado.status == '0')
                  alert('La peticion no ha podido ser procesada!');
                else
                  alert('Los datos se han guardado correctamente.');
                $("#infoprocess").html('');
            },
            error: function (jqXHR, textStatus, errorThrown) {
                alert('Se ha producido un error: ' + errorThrown);
                $("#infoprocess").text('');
            }
        });
    }
    return false;
});
```

[8] jQuery es una librería JavasCript, rápida, pequeña y muy rica en funciones para la manipulación de los elementos del DOM. Podemos consultar su documentación en: https://api.jquery.com/

La librearía React y su uso con la API REST

El interfaz de usuario de la página de acceso a la monitorización de la sincronización de los posts, así como a los logs, ha sido creado utilizando React. A diferencia del caso anterior, aquí se utiliza la librería React para construir el propio interfaz de usuario.

React es una biblioteca javascript creada por el equipo de desarrollo Facebook para construir interfaces de usuario en entornos web.

React está basada en componentes encapsulados que manejan su propio estado. La lógica de los componentes está escrita en JavaScript, como hemos explicado, y no en plantillas, pudiendo de esta forma pasar datos a dichos componentes de forma relativamente sencilla, manteniendo el estado fuera del DOM.

Por lo tanto, en términos generales, para escribir interfaces de usuario React, escribimos componentes React que corresponden a elementos de la interfaz. Podemos escribir elementos como un botón o un campo input como un componente React, pudiendo dicho componente incluir además uno o más componentes en su salida. Luego organizamos estos componentes dentro de componentes de nivel superior, que definen la estructura de nuestra aplicación.

React utiliza JSX (aunque no es obligatorio). Se trata de una extensión de Javascript que ayuda a describir la interfaz de usuario, ya que se parece mucho a los elementos html que utilizamos para definir las plantillas web. El código JSX necesitará un proceso de compilado (transpilado) para ser convertido a Javascript al crear el componente.

En nuestro caso, como veremos, no hemos construido interfaces de usuario complejas, que requieran interacciones difíciles de implementar, así como cambios de estado en la propia interfaz. Tanto en la monitorización de la sincronización, como en los logs de eventos del backend, se trata de la creación de una tabla que muestre información proporcionada por el backend a través de la API REST de la propia instalación local de WordPress.

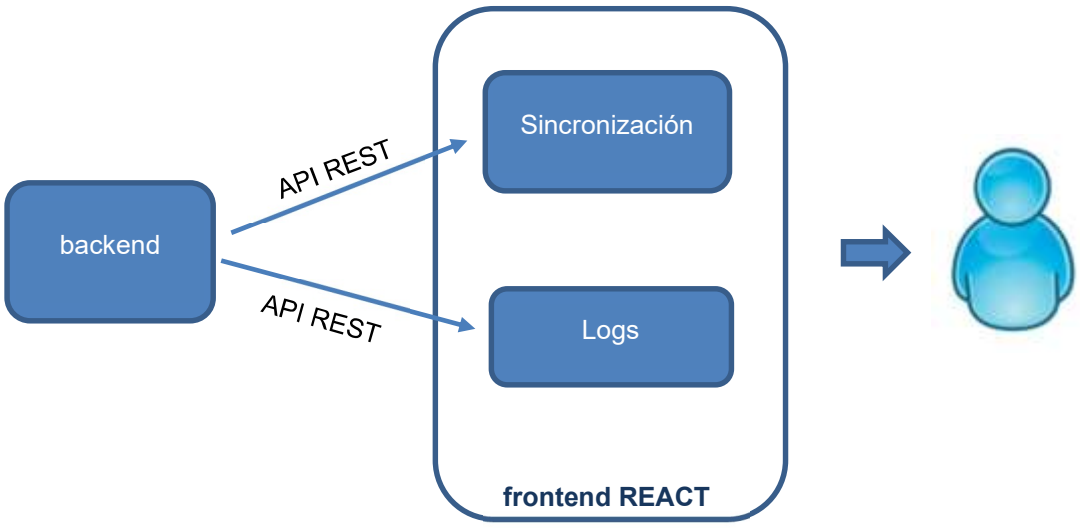


Figura 6: Esquema del Front-end utilizando REACT.

Cargar las librearías React en el front-end de WordPress

A continuación, mostramos el código fuente que carga las librerías React, cuando las páginas a las que se accede son la página de monitorización de la sincronización y la página de logs. Este código se encuentra en la función “pbLoadScripts” dentro del archivo “pbfn-general.php”.

```
// Cargo REACT y Babel
wp_register_script ('babel', 'https://cdnjs.cloudflare.com/ajax/libs/babel-standalone/6.26.0/
  babel.min.js', NULL, FALSE, TRUE);
wp_enqueue_script('babel');
wp_register_script ('pb_reactlib', 'https://unpkg.com/react@16/umd/react.development.js', array('babel
  '), FALSE, TRUE);
wp_enqueue_script('pb_reactlib');
wp_register_script ('pb_reactlib_dom', 'https://unpkg.com/react-dom@16/umd/react-dom.development.js',
  array('pb_reactlib'), FALSE, TRUE);
wp_enqueue_script('pb_reactlib_dom');
```

La función “wp_register_script” registra en el sistema una librería o un simple script, javascript en nuestro caso. Para ello le pasamos como parámetros, de izquierda a derecha: el nombre con el que deseamos registrar la librería, la url o path de acceso a la librería, un array con las librerías que deberían ser cargadas antes que ella (librerías de las que depende), la versión (si deseamos especificarla) y un campo booleano para indicar al sistema si queremos que la librería se cargue en el <head> del documento html, o al final, antes del cierre </body>.

Como se puede observar, registramos y cargamos las librerías, en este orden:

- **Babel**: Es la herramienta que se encarga del compilado del código JSX de los componentes React.
- **ReactJS**: Es la librería REACT propiamente dicha (react-development.js).
- **ReacDOM**: Es la librería que nos permite conectar React con el DOM.

Ejemplo de uso, con la monitorización de la sincronización:

Analizamos ahora el código fuente del archivo “**pb_sync.js**”, que es el que se encarga de construir el interfaz de usuario usando React.

La url de acceso a la página de monitorización de la sincronización es:

```
http://alandalus.today/sync/synchronization-post-state/
```

Esta página carga la plantilla definida por el archivo “**sync-post-state.php**”, existente en nuestro tema “**Sync Theme**”. Este archivo con tiene el siguiente fragmento de código html:

```
<div id="root"></div>
```

Nuestro script React va a cargar el interfaz de usuario en ese elemento “div” cuyo atributo “id” es “root”.

En el archivo “**pb_sync.js**” podemos encontrar la definición del componente React, definiendo una clase que hereda de React.Component:

```
class App extends React.Component {
```

Hemos denominado “App” a la clase que representa nuestro componente. Al final del archivo indicamos que dicho componente se debe renderizar en el div cuyo id es “root”, de la siguiente forma:

```
ReactDOM.render(
  <App />,
  document.getElementById('root')
);
```

Analicemos el constructor del componente:

```
constructor(props){
  super(props);

  this.state = {
    total: 0,
    isLoaded: false,
    posts: [],
    errorTxt: '',
    dataRoute: 'http://alandalus.today/sync/wp-json/sync/v1/sync-status/'
  }
}
```

En el constructor definimos el estado del componente. Este estado sólo podrá ser modificado posteriormente invocando al método *setState*. Entre otros campos, dentro del estado guardaremos la siguiente información importante para la carga de los datos de sincronización:

- posts: Lista con información de los post existentes en los sitios principal y beta, tras su comparación.
- dataRoute: El endopoint personalizado definido en la API REST, explicado en el punto 4.1, utilizado para obtener la información que nos permita monitorizar el estado de la sincronización.

El método “componentDidMount” es invocado justo después de que el componente haya sido montado en el DOM. Es el método perfecto para invocar lanzar la petición a la API REST que nos permita cargar la información sobre el estado de la sincronización:

```
componentDidMount(){
  fetch(this.state.dataRoute)
    .then(res => res.json())
    .then(
        (resultado) => {
          this.setState({
            isLoaded: true,
            total: resultado.total,
            posts: resultado.posts.map(this.mapPosts)
          });
        },
        (error) => {
          this.setState({
            isLoaded: true,
            errorTxt: error
          });
        }
      )
}
```

Como se observa, si la petición tuvo éxito, utilizando el método *setState* obtenemos los datos de los posts, así como otra información que luego utilizaremos en la renderización.

El método de renderización del componente es el siguiente:

```
render() {
  if (this.state.total == 0) {
    return <p>No hay datos que sincronizar</p>;
  }

  // Si hay datos que sincronizar, paso a mostrarlos
  return (
    <div className="App">
      <div className="headerApp">
        <div className="btnBlock">
          {this.printButton()}
        </div>
        <div className="leyendaBlock">
          <ul>
            <li>Leyenda:</li>
            <li className="mainColor">Sólo en el sitio principal.</li>
            <li className="bothColor">En el sitio principal y beta.</li>
            <li className="betaColor">Sólo en el sitio beta.</li>
          </ul>
        </div>
      </div>
      <div className="posts-container">
        <table className="table-posts">
          {this.printTableHeader()}
          {this.printTableRows()}
        </table>
      </div>
    </div>
  );
}
```

En este método se puede observar el código JSX, muy similar a los elementos de las plantillas html. Se observa como dentro puedo incluir sentencias de control condicionales, así como llamadas a funciones de la propia clase que realizan la renderización de elementos concretos.

Veamos, por ejemplo, el método “printButton”:

```
printButton() {
  return (
    <button
      id="syncbtn"
      className="button button-primary"
      onClick={(e) => this.handleBtn(e)}
    >Sincronizar</button>
  );
}
```

Se observa como en la renderización del botón puedo indicar, entre otras cosas, cual es la clase css asignada, el atributo id, o cual es el método que se va a encargar de manejar el evento click.

Veamos a continuación el método “printTableRows”:

```
printTableRows() {
  return (
    <tbody>
      {this.state.posts.map((post, index) =>
          <tr className={`post-tr-${post.where}`} key={ index }>
            <td className="td-chb td-border-right"><input type="checkbox" id={`chb-${post.index}`} mainid={ post
              .id } betaid={ post.idBeta } className="sync-chb" onChange={(e) => this.handleChangeCheckbox(index
              , e)} /></td>
            <td className="td-chb status td-border-right">{ post.estado }</td>
            <td className="td-slug td-border-right">{ post.slug }</td>
            <td className="td-id">{ post.id }</td>
            <td className="td-title">{ post.titulo }</td>
            <td className="td-fecha">{ post.creacion }</td>
            <td className="td-fecha td-border-right">{ post.modificacion }</td>
            <td className="td-id">{ post.idBeta }</td>
            <td className="td-title">{ post.tituloBeta }</td>
            <td className="td-fecha">{ post.creacionBeta }</td>
            <td className="td-fecha">{ post.modificacionBeta }</td>
          </tr>
      )}
    </tbody>
  );
}
```

Aquí vemos como utilizamos la lista *state.posts* y su método *map* para definir cada fila de la tabla, y en cada fila los datos de cada celda, correspondiente al post en curso.

4.3 Base de datos

Se han creado únicamente dos tablas en la base de datos, una para dar soporte a la configuración de los datos de sincronización, y otra para registrar los logs.

Descripción de las tablas:

- **/configsync**
  - **id**: identificador único del registro de configuración.
  - **user_main**: nombre de usuario administrador en el sitio principal.
  - **user_beta**: nombre del usuario administrador en el sitio beta.
  - **passw_main**: contraseña del usuario administrador en el sitio principal.
  - **passw_beta**: contraseña del usuario administrador en el sitio beta.
  - **url_main**: dominio del sition principal.
  - **url_beta**: dominios del sitio beta.

- **/logssync**
  - **id_log**: identificador único de log.
  - **id_user**: identificador único del usuario que está usando la herramienta durante el registro del log.
  - **fecha**: fecha en la que se ha registrado el log.
  - **hora**: hora en la que se ha registrado el log.
  - **procedencia**: ip del navegador del usuario que está usando la herramienta.

- **proceso**: acción que se estaba realizando en el momento de registrase el log (por ejemplo, solicitando datos remotos, creando un post, etc.).
- **mensaje**: texto descriptivo del log (por ejemplo, creación del post correcta, ha fallado la petición de datos remotos, etc.).

5. Plataforma de desarrollo

5.1 Software:

Los 3 portales desarrollados están creados utilizando como base WordPress.

WordPress es un sistema que permite crear gestores de contenidos para entornos Web, de código abierto, publicado bajo licencia GPL. Un gestor de contenidos es una herramienta que facilita la administración de los aspectos más importantes de un sitio web, como el contenido, su apariencia, la gestión de usuarios, etc., todo ello sin necesidad de tener conocimientos técnicos de programación.

Aunque originalmente WordPress se creó como un sistema de blogging, es decir, una herramienta para crear blogs, hoy en día permite el desarrollo de prácticamente cualquier tipo de portal Web.

A continuación, indico las tecnologías en las que está basado WordPress, detallando las que hemos utilizado específicamente:

- **PHP (back-end):**

  PHP es el lenguaje utilizado por WordPress en el lado del servidor. En nuestro caso, la versión usada es PHP 7.1.33.

- **REACT y jQuery (front-end):**

  React y jQuery son las librerías JavaScript usadas en el frontend. Uso la versión 16.12 de React. Respecto a jQuery uso la versión 1.12.4 que incorpora WordPress por defecto.

- **Sistema Gestor de Base de Datos.**

  Como SGBD podemos usar MySQL o MariaDB. El sistema y versión usado en nuestro caso es MySQL 5.6.

- **Servidor Web.**

  Como servidor web se recomienda Apache o Nginx, pero se puede usar cualquiera que soporte PHP y MySQL. El servidor web usado en nuestro caso es Apache 2.

El sistema operativo:

El sistema operativo del servidor es Debian GNU/Linux 8.11 (jessie). Podríamos haber utilizado otra de las distribuciones Linux de propósito general (SuSE, Red Hat, etc.), o alguna de las últimas versiones de Windows Server, sin prácticamente ninguna variación en el proyecto [9].

[9] Quizá alguna variación en la configuración o en las versiones del software utilizado (PHP, Apache o MySQL), para asegurar aspectos de compatibilidad, u optimización del funcionamiento, pero sin cambios en el código desarrollado.

5.2 Hardware:

La herramienta desarrollada no requiere grandes recursos hardware en el servidor, ya que:

- Será utilizada por una persona, o muy pocas personas simultáneamente.
- La instalación de WordPress en la que se implanta no utiliza un gran número de plugins, y los que usa no consumen muchos recursos.
- El theme creado es muy ligero en cuanto a consumo de recursos, y el código fuente ocupa muy poco espacio (poco más de 100 KB).
- Una instalación de WordPress básica (como la nuestra) ocupa aproximadamente 45 MB.

No obstante, indicamos los recursos hardware usados en el servidor, para cada hosting:

- Servidor Dell R230 (con procesador Intel Xeon E3).
- 4 GB de RAM DDR4.
- Espacio en disco: 4 GB.

6. Perfiles de usuario

WordPress incorpora en su núcleo un sistema de gestión de usuarios basados en **capacidades**. La idea es poder definir distintos **roles** de usuarios en función de las capacidades asignadas.

Por ejemplo, el usuario con el rol “Editor” tiene, entre otras capacidades, las de crear, modificar y eliminar contenidos, pero no tiene la capacidad de instalar, desinstalar, activar o desactivar plugins.

6.1 Perfil de usuario necesario para usar la aplicación:

Para poder usar la aplicación desarrollada se exige tener el rol de “Administrador”. Un usuario con dicho rol tiene todas las capacidades definidas por defecto en una instalación de WordPres básica.

Por lo tanto, sólo un usuario que haya iniciado sesión y tenga el rol “**Administrador**” podrá acceder a las funcionalidades desarrolladas: página de configuración, página con el log de eventos y página de monitorización y sincronización.

7. Usabilidad/UX

El proyecto tendrá un interfaz de usuario (front-end) para la configuración y, en su caso, la monitorización de la sincronización de los dos portales web, el portal principal y el portal beta.

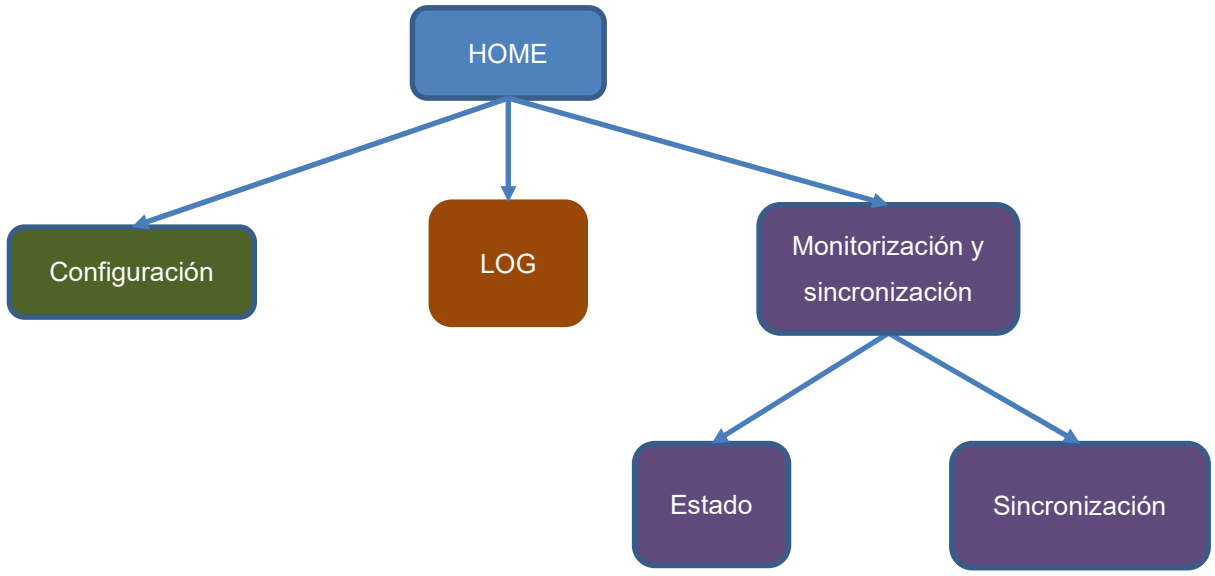
Por lo tanto, tendremos dos espacios diferenciados:

- Configuración de todos los aspectos que tengan que ver con los accesos de un portal a otro: identificación de los portales (dominios) y datos de autenticación de los usuarios administradores en los portales principal y beta.
- Activación de la sincronización (su puesta en funcionamiento), así como la monitorización del estado de dicha sincronización.

Además, el usuario tendrá acceso a un tercer apartado con la siguiente funcionalidad:

- Visualización de un log de los eventos producidos en el back-end durante los procesos de monitorización del estado de sincronización, así como de los propios procesos de sincronización.

En un acercamiento a la estructura del interfaz de usuario, desde el punto de vista de su navegación por él, podemos establecer el siguiente diagrama:



El interfaz de usuario para las funcionalidades de monitorización y sincronización, al menos en esta primera versión del proyecto, aunque en el diagrama se ha diferenciado, se encontrará en la mima pantalla (en la misma página dentro del portal web).

8. Instrucciones de instalación/implantación

Para instalar [10] la herramienta simplemente hay que copiar el theme "**Sync Theme**" en el directorio de themes de WordPress y activarlo.

La activación del theme dispara la action "**load-themes.php**". Hemos implementado el hook "**pb_init_theme**" para que se dispare durante dicha action. Este detalle se puede encontrar en el archivo *functions.php* del theme y la línea en la que defino el hook es:

```
add_action('load-themes.php',  'pb_init_theme');
```

La función "**pb_init_theme**" comprueba que existan las tablas de configuración de la herramienta y de registros de los logs. Si no existen se crean. Además, después de la creación de la tabla de configuración, se inserta un registro que es el que almacenará la información de configuración.

La implementación de esta función se encuentra dentro del theme, en el archivo *pbfn-general.php*.

[10] Este aspecto fue también detallado en el punto 3.5 de esta memoria.

9. Instrucciones de uso

El acceso a las distintas partes de la herramienta es el siguiente:

- Configuración de la herramienta:
  `http://alandalus.today/sync/config/`
- Registro de logs:
  `http://alandalus.today/sync/logs/`
- Sincronización:
  `http://alandalus.today/sync/synchronization-post-state/`

9.1 Configuración:

La página de configuración tiene como objetivo poder modificar la siguiente información, utilizada internamente en los procesos de monitorización y sincronización:

- Los dominios de los sitios web a sincronizar (principal y beta).
- Los datos que identifican a los usuarios (nombre y contraseña) que acceden a los sitios remotos que se pretenden sincronizar.

Ningún dato puede quedar en blanco.

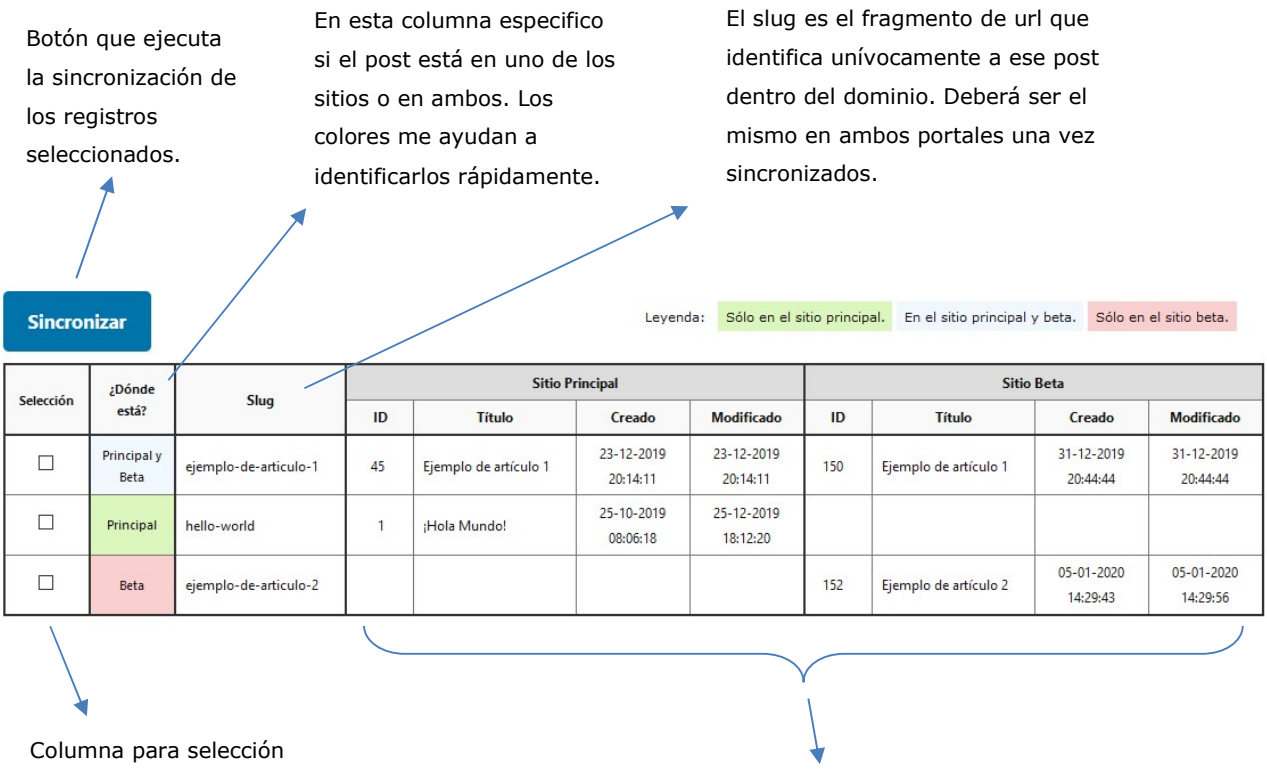
Respecto a los datos de identificación de los usuarios, es conveniente realizar la siguiente puntualización:

- Para acceder a los posts del sitio principal no es necesario, realmente, en esta primera versión del trabajo, autenticarse, ya que simplemente se accede a la lectura de los datos de los posts, y su acceso es público a través de la API REST.
- El usuario especificado para el acceso al sitio beta debería ser un usuario registrado en dicho portal con la capacidad de crear y eliminar contenidos (un administrador tiene dichas capacidades).

9.2 Sincronización:

En esta página se puede acceder a una tabla con el estado de la sincronización de los portales web. En dicha tabla podemos observar qué artículos están en un portal web y en el otro no, y qué artículos están en ambos portales. He incluido una leyenda para que sea más fácil identificarlos utilizando colores:

A continuación, paso a describir brevemente los elementos que aparecen en la interfaz de usuario.



Selección	¿Dónde está?	Slug	Sitio Principal				Sitio Beta			
			ID	Título	Creado	Modificado	ID	Título	Creado	Modificado
☐	Principal y Beta	ejemplo-de-articulo-1	45	Ejemplo de artículo 1	23-12-2019 20:14:11	23-12-2019 20:14:11	150	Ejemplo de artículo 1	31-12-2019 20:44:44	31-12-2019 20:44:44
☐	Principal	hello-world	1	¡Hola Mundo!	25-10-2019 08:06:18	25-12-2019 18:12:20				
☐	Beta	ejemplo-de-articulo-2					152	Ejemplo de artículo 2	05-01-2020 14:29:43	05-01-2020 14:29:56

¿Qué hace la sincronización?

Cuando seleccionamos uno o varios posts en la primera columna y hacemos clic en el botón “Sincronizar”, el proceso que se ejecuta es el siguiente:

- Si el post está únicamente en el portal principal, el proceso de sincronización realiza una copia de dicha publicación en el portal beta.
- Si el post está únicamente en el portal beta, el proceso de sincronización elimina el post en el portal beta.
- Si el post está en ambos portales (ya fue sincronizado), el proceso de sincronización elimina el post en el portal beta y a continuación vuelve a copiarlo.

10. Conclusiones

Las conclusiones que puedo extraer de este trabajo, en este momento, tienen dos vertientes;

Por un lado, me siento moderadamente satisfecho por los conocimientos adquiridos, teniendo en cuenta que existía la motivación de conocer y experimentar con una API REST, más concretamente con la de API REST de WordPress, a la hora de elegir la temática del trabajo.
En este sentido, puedo afirmar que las expectativas se han visto sobradamente cumplidas, ya que me llevo una idea bastante fiel del uso y posibilidades de esta API, ya no sólo por el trabajo desarrollado, sino también por la gran cantidad de artículos que he leído sobre el tema, y la gran cantidad de test y pruebas que he realizado.

Por otro lado, respecto al producto final creado, he de reconocer que se queda corto en funcionalidades respecto a las expectativas de las que partía, que eran las de la sincronización total de dos portales web hechos con WordPress. En este sentido, quedan cosas por hacer, como el tratamiento de las taxonomías o los elementos multimedia, tareas para las cuales sólo necesitaría más tiempo, ya no conocimientos.

El trabajo ha supuesto un reto personal, ya que partía de unos conocimientos muy básicos y sólo teóricos sobre la herramienta, y porque la documentación existente en WordPress relacionada con la API REST es vaga, puramente esquemática en algunos aspectos, y sólo llegas a comprender el funcionamiento después de realizar muchas pruebas.

Anexo 1. Entregables del proyecto

Se entrega un archivo comprimido en el que podremos encontrar los siguientes archivo y directorios:

- **Memoria:**
  PAC_FINAL_mem_BlancoBerguño_Pablo.pdf

- **Código de la aplicación:**
  PAC_FINAL_prj_BlancoBerguño_Pablo.zip

  Nota: Este archivo comprimido contiene 3 directorios:
  - Sitio_beta: Directorio con el código WordPress del portal web beta.
  - Sitio_desarrollado: Directorio con el código WordPress del portal web desarrollado. El código fuente de la aplicación desarrollada estará aquí, dentro del directorio wp-content/themes/sync-theme.
  - Sitio_principal: Directorio con el código WordPress del portal web principal.

- **Presentación escrita-visual:**
  PAC_FINAL_prs_BlancoBerguño_Pablo.pdf

- **Autoinforme de evaluación:**
  PAC_FINAL_informe_autoevaluacion_BlancoBerguño_Pablo.pdf

2. Código fuente (extractos)

Definición de endpoints en la API REST

Destacamos este fragmenteo, ya que, como hemos comentado, aunque no era estrictamente necesario, para experimentar con la creación de endpoints personalizados, hemos definido dos endpoints para el acceso desde el frontend creado con REACT a los datos que muestran el estado de la sincronización y de los logs.

A continuación, podemos observar el código php que define en WordPress dichos endpoints, dentro del archivo *includes/pbfn-general.php*:

```php
/**************** ENDPOINTS DE LA API REST *****************/

/**
 * Configuración de endpoints en la API REST de la aplicación desarrollada.
 */

function status_endpoint( $request_data ) {
    global $gSync;
    $mData = $gSync->getSyncStateData();
    if ($mData)
        return $mData;
    else
        return '';
}

function logs_endpoint( $request_data ) {
    return getLastLogs(100);
}

function pb_rest_api_local_init () {
    // Definición del endpoint para el acceso al estaod de la sincronización
    register_rest_route( 'sync/v1', '/sync-status/', array(
        'methods' => WP_REST_Server::READABLE,
        'callback' => 'status_endpoint'
    ));

    // Definición del endpoint para el acceso a los logs
    register_rest_route( 'sync/v1', '/sync-logs/', array(
        'methods' => WP_REST_Server::READABLE,
        'callback' => 'logs_endpoint'
    ));
}
```

Anexo 3. Bibliografía

WordPress

- Páginas principales del manual online y de la referencia WordPress para desarrolladores.
  https://codex.wordpress.org/
  https://developer.wordpress.org/reference/

- Referencia de la clase wpdb, para gestión de la base de datos.
  https://codex.wordpress.org/Function_Reference/wpdb_Class

- Referencia de las *actions* en WordPress:
  https://codex.wordpress.org/Plugin_API/Action_Reference

- Referencia de los *themes* en WordPress:
  https://developer.wordpress.org/themes/

API REST

- Referencia de la API REST en WordPress.
  https://developer.wordpress.org/rest-api/

- Schema de los post usando la API REST en WordPress.
  https://developer.wordpress.org/rest-api/reference/posts/

- Guía para principiantes de la API REST de WordPress.
  https://www.hostinger.es/tutoriales/guia-para-principiantes-api-rest-wordpress/

- Cuatro formas de empezar a usar la API REST de WordPress.
  https://torquemag.io/2017/05/ways-start-using-wp-rest-api

- WP REST API: Setting Up and Using OAuth 1.0a Authentication.
  https://code.tutsplus.com/tutorials/wp-rest-api-setting-up-and-using-oauth-10a-authentication--cms-24797

- How to Setup and Use WordPress REST API: Basic Authentication.
  https://www.cloudways.com/blog/setup-basic-authentication-in-wordpress-rest-api/

- List Posts From WordPress Using REST API With cURL In PHP.
  http://niralar.com/list-posts-from-wordpress-using-rest-api-with-curl-in-php/

- A simple post insert using WP REST API and PHP:
  https://gist.github.com/andrewahead4/489e6422feb5be901143

REACT

- Documentación oficial de React:
  https://es.reactjs.org/docs/getting-started.html

- Componentes y propiedades en React:
  https://es.reactjs.org/docs/components-and-props.html

- Construyendo la primera aplicación con React:
  https://medium.com/learning-new-stuff/building-your-first-react-js-app-d53b0c98dc

- Construir un plugin en WordPress con React:
  https://gopangolin.com/building-wordpress-plugin-with-react-part-1/

- Cómo hacer submit en forms y guardar datos usando React y Node:
  https://html5hive.org/how-to-submit-forms-and-save-data-with-react-js-and-node-js/

- Tying a React SPA to WordPress as a Backend:
  https://snipcart.com/blog/reactjs-wordpress-rest-api-example

- How to Enqueue React JSX File in WordPress:
  https://milandinic.com/2015/12/01/using-react-jsx-in-wordpress/

- How to create a modern web app using WordPress and React:
  https://www.freecodecamp.org/news/wordpress-react-how-to-create-a-modern-web-app-using-wordpress-ef6cc6be0cd0/