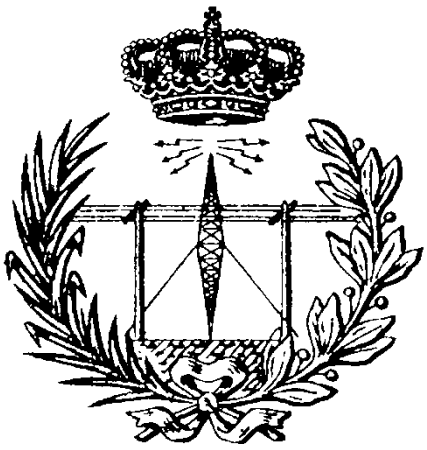
UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA



PROYECTO FIN DE CARRERA

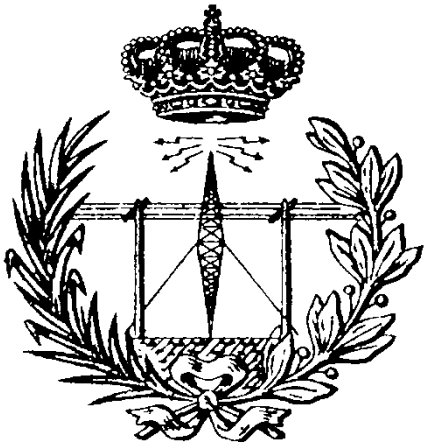
INTERFAZ PARA EL DESARROLLO DE APLICACIONES MÓVILES UTILIZANDO LA NUBE COMO BACKEND

Autor: Borja García González
Tutor: Dr. Luis Hernández Acosta
Fecha: Agosto 2018

[2]

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA



PROYECTO FIN DE CARRERA

INTERFAZ PARA EL DESARROLLO DE APLICACIONES MÓVILES UTILIZANDO LA NUBE COMO BACKEND

HOJA DE FIRMAS

Alumno:

Fdo.: Borja García González

Tutor:

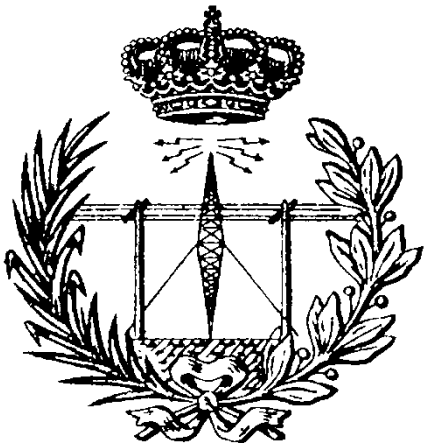
Fdo.: Dr. Luis Hernández Acosta

Fecha: Agosto 2018

[4]

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA



PROYECTO FIN DE CARRERA

INTERFAZ PARA EL DESARROLLO DE APLICACIONES MÓVILES UTILIZANDO LA NUBE COMO BACKEND

HOJA DE EVALUACIÓN

Calificación: _____

Presidente

Fdo.:

Vocal **Secretario/a**

Fdo.: Fdo.:

Fecha: Agosto 2018

[6]

Contenido

Tabla de ilustraciones

Memoria

[12]

Introducción

1.1 Introducción

Una buena *aplicación para dispositivos móviles* inteligentes (smartphones y tablets) *es clave* para una empresa. Puede ser el elemento central del negocio, como servicios de mensajería instantánea; complementar la actividad, como un servicio para pedir a domicilio de un restaurante; o simplemente servir como publicidad, anunciando ofertas, productos o servicios.

Sin embargo, *el desarrollo* de estas aplicaciones requiere de *muchas horas de trabajo* especializado, lo que se traduce en un *coste muy elevado*. Por ejemplo, el desarrollo una aplicación sencilla para Android que ofrezca publicidad *puede alcanzar fácilmente los 5.000€* [1], sin contar con el mantenimiento necesario durante la vida útil de la aplicación. Cualquier añadido incrementa rápidamente el precio, especialmente si se desea que funcione en *diferentes sistemas operativos*.

Una solución es *dirigir la aplicación a unos pocos sistemas operativos*: el *IDC* (*International Data Corporation*) estima que se venderán alrededor de 1.386 millones de smartphones con ***Android*** o ***iOS*** en 2015 (95.8% del mercado), cifra que superará los 2.000 millones (93.2%) en menos de 5 años [2]. Reduciendo el objetivo a sólo los dos sistemas operativos más populares no sólo se *reduce la cantidad de trabajo*, sino que es *más fácil encontrar desarrolladores*.

Otra opción es utilizar *nuevos entornos* que permiten pasar una aplicación web a móvil, como ***Ionic*** [3] o ***React*** [4], lo que permite a desarrolladores web trabajar con móvil. ***Ionic*** ejecuta las aplicaciones utilizando un pequeño navegador, por lo que las aplicaciones son sencillas de desarrollar a cambio de un coste en rendimiento. ***React***, por otro lado, es capaz de convertir la aplicación web en nativa, pero su curva de aprendizaje es elevada.

Aun así, es necesario un equipo de desarrollo, por lo que el *coste económico* sigue siendo elevado, el *tiempo hasta la publicación* puede hacer perder una ventaja competitiva y, a pesar de todo, existe la posibilidad de *no obtener el producto deseado*.

En respuesta a estos problemas, los desarrolladores buscan formas de agilizar y abaratar costes como las *plantillas o los módulos*. *Una plantilla es un esqueleto de aplicación* que se personaliza para cada cliente; mientras que *un módulo es una parte funcional*, como una vista o un componente complejo. A cambio de perder flexibilidad (modificarlas en exceso

haría que perdiese la ventaja de estar preparado) se *reduce el tiempo necesario* para el desarrollo y *el precio baja* hasta las pocas centenas de euros para aplicaciones poco exigentes. Sin embargo, el *factor humano* propicia que se cometan errores durante la adaptación, y se mantiene el *problema de entendimiento* entre desarrolladores y clientes.

El último paso dado en esta dirección es el de *automatizar la creación de aplicaciones* utilizando *módulos*. *Mediante una interfaz el cliente seleccionará los elementos que desea en la aplicación y el sistema se encargará de conectarlos*. Aunque quedan corregidos los problemas de tiempo de desarrollo, entendimiento y adaptación, el entorno debe ser lo suficientemente sencillo para ser usado por el cliente y lo suficientemente completo como para permitir tener aplicaciones útiles.

Un ejemplo que se aproxima a este último caso es el ***IUMATI Framework***, un sistema desarrollado por *Luis Hernández* y *José Quinteiro* que crea aplicaciones para *Android* e *iOS* usando módulos estructurados en listas y descritos en un *"lenguaje" propio*. Este lenguaje se pensó para ser *funcional*, por lo que tiene una estructura más *orientada al código* que al manejo humano. Por ello, en este proyecto *se ha creado una **interfaz web** que permite **diseñar** la aplicación al cliente y **convertirla** a la estructura del framework*, teniendo así un *sistema de creación de aplicaciones*.

1.2 Objetivos

El *objetivo principal* de este proyecto ha sido el de *crear una herramienta que permita que un usuario sin conocimientos de programación diseñe una aplicación para dispositivos basados en Android e iOS, a través del framework IUMATI*. A esta herramienta se puede acceder a través de un navegador web.

Además, para adaptarse a actualizaciones en el framework, *la herramienta debía ser capaz de aceptar modificaciones* por parte del administrador sin que éste tenga que cambiar su código.

Para ello se propuso cumplir con tres objetivos parciales:

1.2.1 Estudiar las opciones propuestas por el tutor

Dado que no es el único proyecto que pretende facilitar el uso del framework IUMATI, a la hora de ofertarse el proyecto se propusieron dos ideas: implementarlo con el software de ***Bonita BPM*** [5], utilizado para la gestión de documentos de una organización mediante formularios, o mediante un ***plugin para Wordpress*** [6], una plataforma de gestión de contenido.

Por ello deben ***estudiarse las dos opciones*** y ***escoger la que se considere más adecuada***.

1.2.2 Estudio de las herramientas

Una vez decidida la opción que se va a seguir, es necesario ***estudiar las herramientas*** que serán necesarias para llevar a cabo el proyecto. En el caso de *Bonita* será el entorno, y en el caso del *plugin de Wordpress* serán los lenguajes, librerías y entornos apropiados.

1.2.3 Diseñar la herramienta que permita cumplir los dos objetivos principales

Los objetivos principales son tener una *interfaz de usuario sencilla* y *permitir a un administrador modificar la herramienta sin cambiar el código*. Por tanto, se deberá diseñar una herramienta que cumpla con esta contradicción de sencillez y complejidad.

1.3 Peticionario

Actúa como peticionario de este proyecto la Escuela de Ingeniería de Telecomunicación y Electrónica (EITE), de la Universidad de las Palmas de Gran Canaria (ULPGC), siendo la realización de este Proyecto de Fin de Carrera requisito indispensable para la obtención del título de Ingeniero de Telecomunicación.

1.4 Estructura de la memoria

El presente documento está dividido en *cinco capítulos*, además del pliego de condiciones y anexos.

El primer capítulo, en la que se encuentra este apartado, es la ***introducción***. En él se ha intentado poner en conocimiento del lector las razones que han llevado a la propuesta y realización de este proyecto.

A continuación, se explicarán las ***tecnologías software*** que tienen relación con el proyecto, sea por similitud o por haber sido utilizadas para la realización. Se tratará de introducir lo necesario para poner en contexto, sin pretender ser una explicación detallada de ninguno.

Para empezar a concretar, se explicará el funcionamiento del framework IUMATI en ***Arquitectura del sistema***. Esto permitirá entender las decisiones tomadas en el diseño del plugin, que se introduce por primera vez al final de este capítulo.

El cuarto capítulo, ***Plugin IAC***, es la descripción completa del plugin, explicando el uso y funcionamiento de cada una de sus partes.

Por último, se exponen las ***conclusiones*** extraídas al finalizar el proyecto, basándose en el grado de cumplimiento de los objetivos y la experiencia en el desarrollo de la herramienta.

Tecnologías software

En este capítulo se introducen con cierto detalle las tecnologías y *herramientas relacionadas con este proyecto*. En primer lugar, se hablará de aquellas tecnologías que se usaron en el proyecto, aunque no son específicas del desarrollo de aplicaciones; pasando después a aquellas que sí son específicas del desarrollo, sean lenguajes o herramientas.

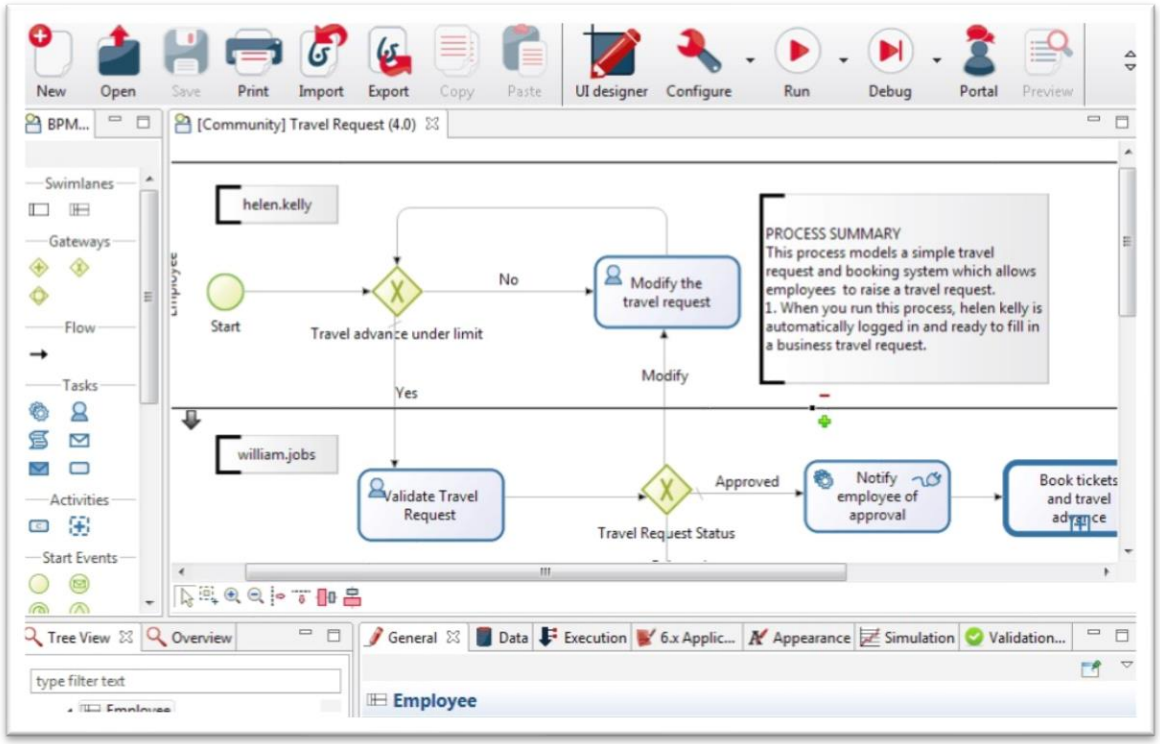
2.1 Elección del entorno

Como se introdujo en los objetivos, el primer paso en la realización de este proyecto era el *estudio y elección de las dos opciones* propuestas por el tutor. Dado que *esta elección condiciona todo lo demás* se considera oportuno explicar en primer lugar qué decisión se tomó y cuáles fueron las razones para ello.

La opción en *Wordpress* consistiría en integrar un código propio en forma de plugin, por lo que se tenía la certeza de que sería viable realizar el proyecto de esa manera. Por ello, se decidió empezar estudiando Bonita, buscando saber si era una mejor opción.

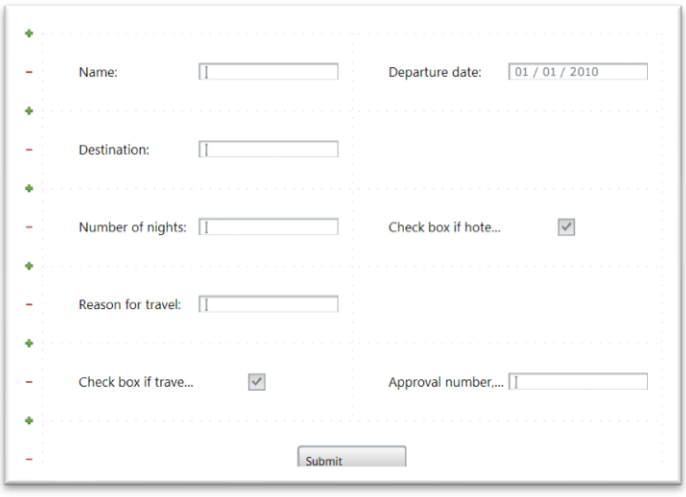
Además, al ser un entorno *utilizado por muchas organizaciones*, como CISCO, Orange, entidades bancarias, … Por tanto, independientemente de la elección, su estudio supondría de por sí una ventaja al poder utilizarse en un futuro laboral.

Bonita es un entorno para desarrollar flujos de trabajo. El diseñador puede crear el proceso mediante un diagrama donde cada elemento representa una acción o evento. Por ejemplo, en un *proceso para reservar viajes de negocios* un empleado accede al servicio. Se le pedirá que *rellene un formulario*, y si cumple con los requisitos establecidos *se enviará al supervisor*. Éste podrá solicitar que sea *modificado*, *rechazarlo* o *aprobarlo*, en cuyo caso se *notificará al empleado* y se iniciaría el *proceso de reserva*.

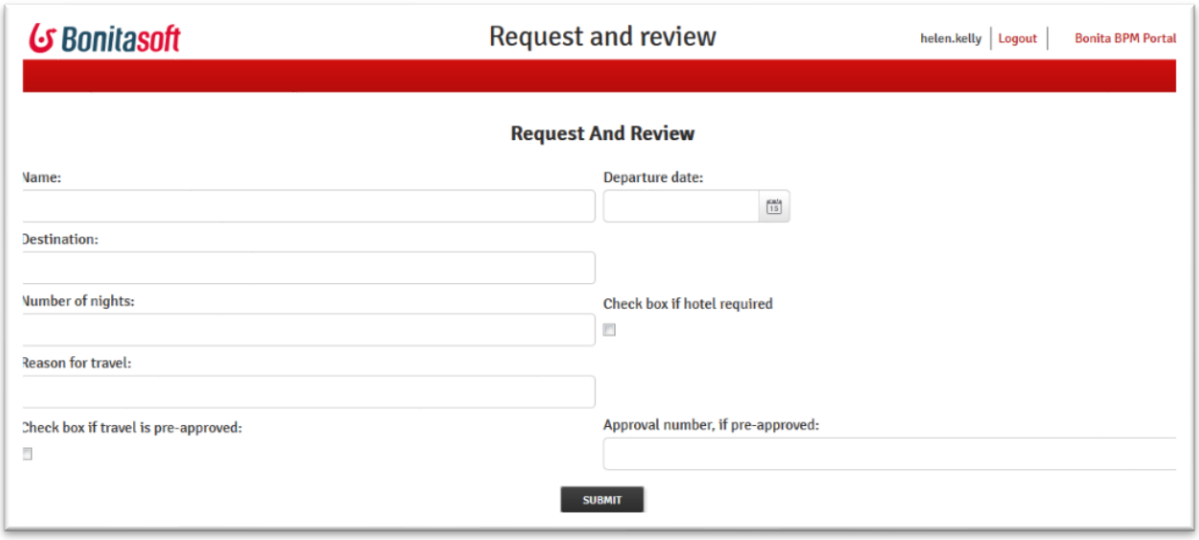


1. Bonita BPM: Diseño del flujo de procesos mediante diagrama de eventos

Este diseño se integra con otros componentes del entorno, como la gestión de la organización, que permite definir roles, grupos, puestos, y relaciones jerárquicas entre ellos.



2. Bonita BPM: Proceso de diseño de un formulario



3. Bonita BPM: vista de un formulario

El *estudio se realizó conociendo primero la herramienta* a través de los *tutoriales* y *documentación* que ofrece Bonita. La documentación y los ejemplos de uso de Bonita se centran en el diseño de herramientas para la gestión de procesos de una empresa. Al pretender dar un uso distinto, *la información disponible era de utilidad reducida.*

Por ello se recurrió después a realizar *pruebas conceptuales* de las distintas partes que podrían ser necesarias y buscar soluciones en foros de la comunidad. La conclusión de este estudio fue que, si bien ciertas partes podrían funcionar, otras limitaban en exceso las posibilidades y no encajaban como se deseaba.

Por ejemplo, la *gestión de usuarios está diseñada pensando en los usuarios de una organización*, no siendo tan accesible para que otros usuarios se registren y utilicen el producto. Era posible, pero artificioso.

Otro ejemplo sería la interfaz con el usuario, que estaría *obligado a introducir toda la información mediante formularios con campos simples*; y la única forma en la que el administrador pudiera añadir contenido sería mediante el propio entorno de Bonita, teniendo que aprender su uso.

Por lo tanto, *la opción escogida fue desarrollar un plugin de Wordpress*.

WordPress es un *sistema de gestión de contenido enfocado a la creación de cualquier tipo de página web*. Originalmente alcanzó una gran popularidad en la *creación de blogs*, para convertirse con el tiempo en una de las principales herramientas para la creación de páginas web comerciales. Una de las razones de su popularidad es permitir el *uso de complementos* (plugins) que añaden o modifican el funcionamiento de la página y temas.

El proyecto se implementó como uno de estos plugins, utilizando a Wordpress como un entorno en el que mostrar su contenido, y que le *facilita la gestión de usuarios y de base de datos*, liberándose pues de estos trabajos.

2.2 Tecnologías generales

Javascript

Javascript [7] (abreviado comúnmente JS) es un lenguaje de programación interpretado, que implementa el estándar *ECMAScript* (actualmente, versión 6). Se define como orientado a objetos, basado en prototipos, imperativo, débilmente tipado y dinámico.

Javascript se diseñó con una sintaxis similar a C, aunque adopta nombres y convenciones del lenguaje de programación Java. Sin embargo, Java y JavaScript tienen semánticas y propósitos diferentes.

Se utiliza principalmente en el lado del cliente, implementado como parte de un navegador web permitiendo mejoras en la interfaz de usuario y páginas web dinámicas, aunque existe una forma de JavaScript del lado del servidor. Su uso en aplicaciones externas a la web, por ejemplo, en documentos PDF o aplicaciones de escritorio (mayoritariamente widgets) es también significativo.

Tradicionalmente se venía utilizando en *páginas web HTML* para realizar operaciones y únicamente en el marco de la aplicación cliente, sin acceso a funciones del servidor. Se interpreta en el agente de usuario al mismo tiempo que las sentencias van descargándose junto con el código HTML.

Posteriormente se utilizó para enviar y recibir información del servidor con ayuda de otras tecnologías, como AJAX. Con la aparición de intérpretes fuera del navegador como NodeJS se puede utilizar Javascript para implementar servidores.

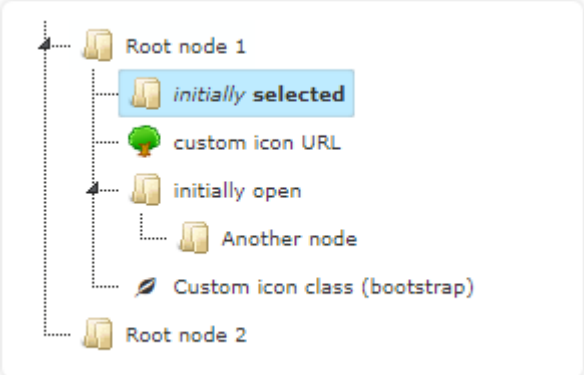
Aunque Javascript puede ser utilizado de forma nativa, existen potentes herramientas que facilitan enormemente el desarrollo. Por ejemplo, *jQuery* es una librería que provee de métodos para acceder al DOM y se ha convertido casi en obligatorio en todos los proyectos. También hay frameworks que van aún más allá, como *Angular* o *VueJS*, que componentes que se añaden a la página como etiquetas HTML, o los ya mencionados *ReactJS* e *Ionic*, que permiten utilizar Javascript para aplicaciones móviles.

JS es el lenguaje más utilizado en este proyecto, por ser el apropiado para una interfaz web dinámica, apoyándose en dos frameworks: ***AngularJS*** y ***jsTree***.

AngularJS [8] es un framework mantenido por Google para desarrollo de aplicaciones de una página (Single Page Application, SPA). No debe confundirse con el framework *Angular*, también mantenido por Google, pero es muy diferente a AngularJS.

Implementa un modelo vista-controlador (MVC) donde la vista es el contenido de la página (DOM) y el modelo es el código escrito para AngularJS. Al cargar el documento se escanea el código buscando elementos propios de AngularJS y se vinculan automáticamente. De esta forma, se puede, por ejemplo, mostrar el contenido de una variable JS en pantalla, o almacenar en una el valor de un campo de un formulario.

jsTree [9] es un framework que permite representar y modificar datos en estructura de árbol. Sus funciones más llamativas son las de incorporar un menú contextual, la posibilidad de añadir datos propios a cada nodo del árbol y la posibilidad de adaptarlo integrando código en métodos ejecutados ante eventos tales como crear un nodo, renombrarlo, moverlo, …



4. jsTree: Ejemplo de árbol

PHP

PHP [10], acrónimo recursivo en inglés de *PHP Hypertext Preprocessor* (*Procesador de Hipertexto PHP*), es un lenguaje de programación de propósito general de código del lado del servidor originalmente diseñado para el desarrollo web de contenido dinámico. Fue uno de los primeros lenguajes de programación del lado del servidor que se podían incorporar directamente en un documento HTML en lugar de llamar a un archivo externo que procese los datos. El código es interpretado por un servidor web con un módulo de procesador de PHP que genera el HTML resultante.

PHP ha evolucionado, por lo que ahora incluye también una interfaz de línea de comandos que puede ser usada en aplicaciones gráficas independientes. Puede ser usado en la mayoría de los servidores web al igual que en muchos sistemas operativos y plataformas sin ningún costo.

En este proyecto se ha utilizado para implementar la parte del servidor, dado que es el lenguaje en el que trabaja Wordpress. El uso dado es para la instalación del plugin y comunicación entre la base de datos y el editor.

2.3 Desarrollo de aplicaciones

En esta sección se introducen las principales opciones disponibles actualmente para desarrollar una aplicación móvil.

Desarrollo mediante programación

El desarrollo de aplicaciones móviles mediante programación requiere de una *gran cantidad de conocimientos en el lenguaje de programación* utilizado y dedicar *tiempo* y *esfuerzo* al *desarrollo* y *mantenimiento* de esta.

A cambio, permiten *personalizar al máximo el contenido* y pueden tener el *mejor rendimiento* si se desarrollan correctamente.

Android: Java / Kotlin

Las aplicaciones Android pueden desarrollarse en cualquier lenguaje capaz de compilar en *bytecode* (un lenguaje interpretado por la máquina virtual de Java, JVM). Entre todos, destacan *Java* y *Kotlin*.

Java [11] es uno de los lenguajes más utilizados en el mundo, y es el lenguaje oficial de *Android*. Sin embargo, no es un lenguaje moderno, y a pesar de haber recibido actualizaciones importantes recientemente pasará un tiempo hasta que Android pueda beneficiarse de ellas.

Kotlin [12] es un lenguaje más reciente, diseñado para tener una curva de aprendizaje más suave que Java y toma lo mejor de la programación procedural y de la funcional.

Cabe destacar que todos los lenguajes aptos para Android se compilan del mismo modo a bytecode, por lo que un mismo proyecto puede tener partes escritas en distintos lenguajes.

IOS: Objetive-C / Swift

En iOS sucede algo similar: se puede programar con *Objective-C* o con *Swift*. [13]

Objective-C es una extensión de C, por lo que es fácil aprenderlo si ya se conoce éste. Además, obliga al programador a ser específico con lo que quiere hacer, siendo menos propenso a errores y necesitando menos conocimiento del contexto.

Swift, por otro lado, delega mucha carga al compilador. Esto hace que el código sea más corto, legible y fácil de mantener, pero requiere conocer bien cómo lo ejecutará el compilador. Es más rápido por no depender de librerías de C y puede añadir librerías dinámicas. Se puede probar en playgrounds.

Aunque la intención es que todas las aplicaciones se desarrollen en Swift, Objective-C sigue siendo muy utilizado por mantenimiento de aplicaciones antiguas y por ser conocido por los desarrolladores.

Desarrollo mediante plantillas de código

Las plantillas de código ofrecen una estructura de aplicación ya creada. Algunas tienen la aplicación completa, y el usuario sólo tendría que cambiar la estética y contenido. Otros ofrecen una aplicación empezada, con secciones que pueden ser modificadas para conseguir los objetivos propuestos.

Aunque pueden reducir considerablemente el tiempo de desarrollo sigue siendo necesario conocer el lenguaje y entorno de desarrollo, es propenso a errores

(especialmente cuando no se conoce el proyecto) y tiene los mismos problemas de mantenimiento.

Un ejemplo de mercado de plantillas es *CodeCanyon* [14], en su sección Mobile.

Desarrollo mediante interfaz web

Estas opciones son las más parecidas al objetivo de este proyecto. El funcionamiento es parecido entre todas ellas: se escoge un tema visual y se añaden módulos que aportan alguna funcionalidad, por lo que se limitará a comentar algunos aspectos clave, ventajas e inconvenientes de cada una.

King of App [15]

- La estética se elige mediante temas visuales.
- 27 módulos: algunos muy configurables (como HTML), muchos específicos de redes sociales.
- Permite añadir servicios.
- Tiene opciones gratuitas, de pago único o con mensualidad.
- Permite descargar el código para modificarlo, desarrollando incluso nuevos temas, módulos o servicios.

Adiante Apps [16]

- Estética fija, permite cambiar fondos, color principal e iconos.
- 22 módulos: orientados a una función (eventos, galería de fotos, formulario, …).
- Dos planes de precios: Android (sólo 1 app para Android) o premium (1 app para Android e iOS), con pago mensual/anual.
- Tiene aplicación móvil para ver el estado de la aplicación mientras se desarrolla.

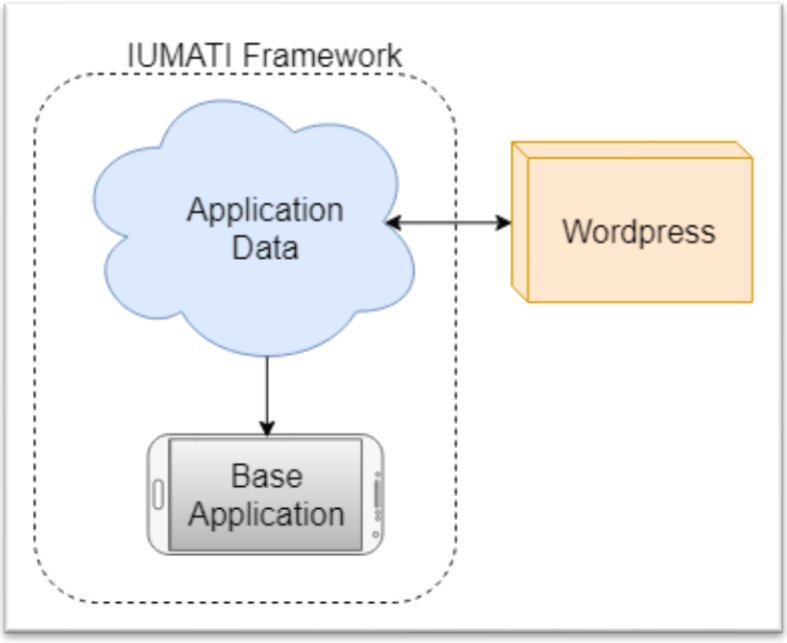
Aplicarium [17]

- Estética a elegir entre una paleta de colores.
- Módulos orientados a una función.
- Dispone de servicio de estadística de uso.
- Pago anual por aplicación (alto el primer pago, reducido a partir del segundo).

Arquitectura del sistema

Este proyecto es parte de un sistema que permitirá al usuario diseñar una aplicación en una página web y usarla en su móvil al instante. Dicha cadena consta de:

- *IUMATI Framework*: el *sistema existente* antes del proyecto.
    - *Aplicación base*: aplicación que el usuario instala en su dispositivo móvil para utilizar la aplicación diseñada. Se conecta a la base de datos donde está la aplicación y descarga la estructura.
    - *ParseDB*: el contenido de la aplicación está guardado en una base de datos no relacional ParseDB. Esta DB permite enviar eventos push (iniciados por el servidor) a la aplicación para notificar de actualizaciones.
- *Editor*: herramienta en Wordpress que aporta este proyecto, donde el usuario diseña su aplicación.
    - ***Editor IAC** (IUMATI App Creator)*: plugin que provee de una interfaz que facilita la creación y gestión de contenido para el framework.
    - *Base de datos*: la DB de Wordpress se utiliza para dar persistencia a las aplicaciones desarrolladas por el usuario.



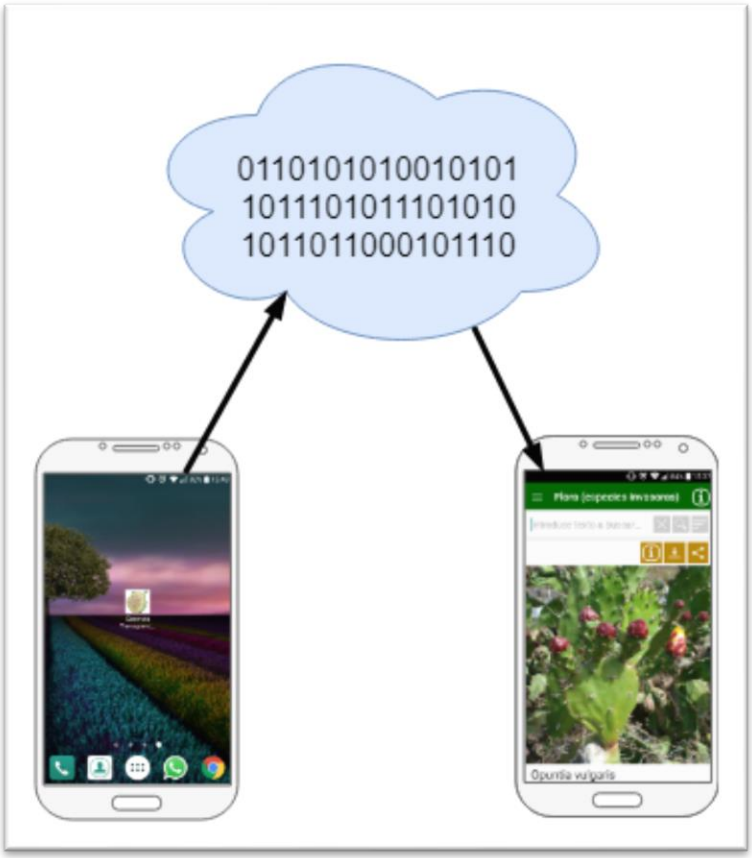
5. Arquitectura del sistema: framework IUMATI y plugin IAC

3.1 Arquitectura existente

3.1.1 Framework IUMATI

El framework IUMATI permite tener aplicaciones móviles actualizables sin necesidad de descargas. Consiste en una *aplicación móvil base* (actualmente para iOS y Android) que se sube a la tienda de aplicaciones correspondiente (Apple Store o Google Play) *configurada para acceder a una base de datos ParseDB* [8].

Cada vez que el usuario inicia la aplicación accede a la DB, descarga unas cadenas de caracteres que le indican cómo configurarse y muestran el contenido. Gracias a esto, el administrador puede editar los datos cuando quiera, viéndose reflejados los cambios la próxima vez que el usuario abra la aplicación, sin necesidad de descargar actualizaciones de la tienda.



6. Diagrama básico del funcionamiento del framework IUMATI

Un ejemplo real de aplicación creada con este framework es *Georruta Transgrancaria* [9], que puede encontrarse en la Play Store para dispositivos Android.

3.1.2 Estructura de una aplicación del framework IUMATI

Las aplicaciones basan su estructura en la de un plugin de Wordpress: *Spider Catalog Catalog* [19], otro plugin para Wordpress utilizado para organizar artículos en una tienda on-line.

Esta información consiste en dos listas de elementos: *categorías* y *productos*, donde las categorías son contenedores que dan forma y los productos el contenido que da valor a los contenedores.

ID	NAME	PARENT	IMAGE URL	DESCRIPTION	PARAMS	ORDERING	PUBLISHED
1	Category A	0			['type', 'color']	1	1
2	Category B	1			['type', 'subtype',...]	1	1
3	Category C	1			['type', 'subtype',...]	2	1

7. Ejemplo de categorías en DB

ID	NAME	CATEGORY ID	IMAGE URL	DESCRIPTION	PARAMS	ORDERING	PUBLISHED
10	Metadata A	[1]	http://...	Decription A	['leftMenu', '#3A3']	1	1
11	Metadata B	[2]	http://...	Decription B	['products', 'share']	1	1
12	Metadata C	[3]	http://...	Decription C	['products', 'share']	1	1
20	Product 1	[2,3]	http://...	A Product	['pdf', 'http://...']	2	1
21	Product 2	[3]	http://...	Another Prod...	['pdf', 'http://...']	3	1

8. Ejemplo de productos en DB

Categorías

- ID: Un identificador numérico único en la aplicación.
- La ID 1 se reserva para la categoría principal.
- Name: Nombre que se le da a la categoría. Puede utilizarse.
    - El nombre debe ser único entre categorías y productos con un mismo padre.
- Description: Descripción de la categoría. No se utiliza.
- Image_url: URL de la imagen asociada. No se utiliza.
- Parent: identificador (spider_id) de la categoría que la contiene.
- La ID 0 se reserva para la raíz. Sólo la usan unas pocas categorías.
- Param: una lista con los parámetros que deben definir los productos contenidos en la categoría.

- Ordering: orden en el que se mostrarán (el menor se muestra primero).
- Published: indica si la categoría es visible en la aplicación.

Productos

- Spider_id: identificador numérico, similar al de las categorías.
- Name: Nombre que se le da al producto.
    - El nombre Metadato está reservado.
- Description: Descripción del producto.
- Image_url: URL de la imagen asociada.
- Category_id: Lista de ID de las categorías a las que pertenece.
    - Un producto puede pertenecer a varias categorías.
- Param: valores de los parámetros definidos por la categoría.
- Ordering: orden en el que se mostrarán.
    - El producto Metadato tiene ordering 1.
- Published: indica si el producto es visible en la aplicación.

3.1.3 Controladores

Basándose en *categorías* y *productos*, el framework define unas unidades llamadas "***controladores***", que son interpretadas por la aplicación base para representar el contenido. La forma de indicar el tipo de controlador es mediante un parámetro presente en todos los controladores: *type*.

La mayoría de los controladores están formados por una *categoría* y un *producto asociado* (llamado *Metadato*), pero pueden ser un simple producto o estar formados por una compleja estructura de categorías y productos. Para explicarlo mejor se utilizarán algunos ejemplos.

Products

Un controlador del tipo *Products* muestra su contenido en forma de lista detallada. Puede utilizarse, por ejemplo, para ver la lista de plantas de una región o la carta de un restaurante.

Se forma con una categoría, que aporta el nombre, y un producto metadato, que aporta descripción, imagen y configuración adicional (como si muestra un cuadro de búsqueda o permite compartir elementos). El resto de productos contenidos se mostrarán en la lista.

PDF

El controlador del tipo *PDF* se utiliza para referenciar a un documento PDF. Su aspecto varía según su uso: si se añade al menú de la aplicación se abrirá el documento, pero si se utiliza como producto de una lista se ofrece la información y un enlace al documento.

La estructura también varía según el uso: si se añade a un menú se utiliza una categoría + metadato, pero basta con un producto cuando forma parte de otro controlador.

Existen otros similares con el mismo funcionamiento y estructura, como *URL* o *HTML*.

Gallery

El tipo *Gallery* muestra su contenido como una serie de imágenes, como las que se utilizan para poner enlaces a redes sociales.

Su estructura se compone también de una categoría y su metadato, pero en este caso puede contener controladores, como *Products, PDF,* …

Maps

Un controlador *Maps* muestra un mapa interactivo con puntos de ruta predefinidos. Este tipo es tiene una estructura muy compleja y su explicación detallada no aportaría nada al lector, pero se menciona como ejemplo de un controlador que requiere multitud de categorías y productos a distintos niveles para poder configurar todo lo necesario: imágenes del mapa, orografía, puntos de ruta, …

Filter

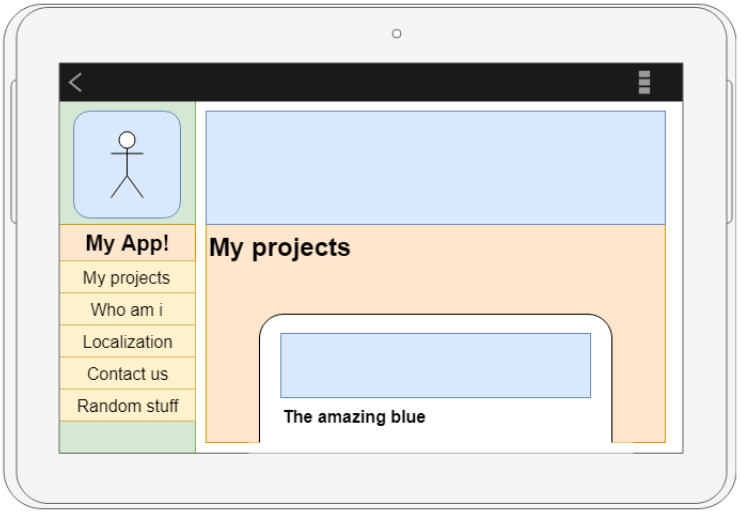
Los *Filter* son controladores especiales parecidos al *Products*, pero su finalidad es la de mostrar un subconjunto de productos. Por ejemplo, si se tiene una aplicación con una sección de flora se puede tener un *Filter* para “Árboles”, “Arbustos”, …

3.1.4 Arquitectura del plugin

Antes de hablar del plugin es conveniente advertir de un cambio necesario en la terminología. El framework IUMATI utiliza el término *controlador* para referirse a cada una de las opciones que se pueden utilizar para una pantalla de la aplicación. Por otro lado, el framework AngularJS también utiliza el término *controlador*, pero esta vez para referirse a la parte de código encargada de *controlar* la interacción entre vista y modelo.

Para evitar confusiones, en el desarrollo de este proyecto se decidió renombrar los controladores de IUMATI a ***tipos***, utilizando el término ***elemento*** cuando se quiere hacer referencia a una instancia de un tipo en la aplicación. Así, *Products* sería un *tipo*, y *Flora* sería un *elemento* del tipo *Products*.

Como se vio en la sección anterior, un tipo puede ser más o menos sencillo, pero en cualquier caso obligaría al usuario a conocer los detalles de implementación de cada uno de ellos. Por ello, se decidió realizar la interfaz con un aspecto similar al que podría tener una aplicación, con una vista maestro-detalle.

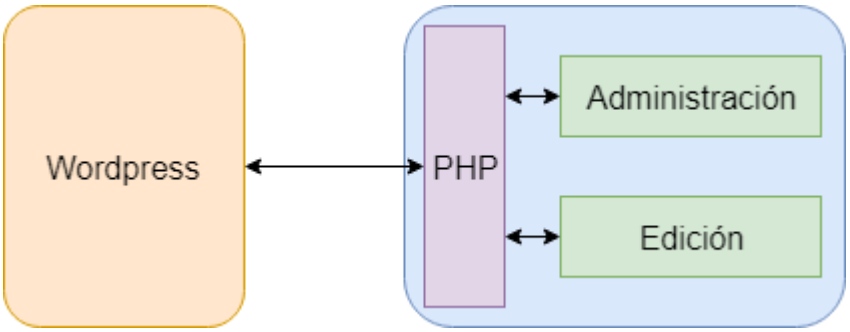


9. Idea de maquetación de la interfaz

Para que el usuario no tenga que conocer la estructura de los tipos se pensó en hacer su uso transparente al usuario, de forma que pueda añadir un *Products*, un *Gallery* o un *Maps* con sólo seleccionarlo en el menú.

Esto se consigue pidiendo al administrador que indique la estructura de cada tipo, que servirá al editor para crear y configurar adecuadamente las categorías y productos.

Por ello, el proyecto cuenta con dos partes principales: ***administración*** y ***edición***. Además, para la conexión con Wordpress es necesario crear una ***interfaz Wordpress-plugin*** en PHP que muestre la interfaz y gestione las conexiones con la DB.



10. Diagrama básico del plugin IAC

En el próximo capítulo se explicarán estas partes, comenzando con un ejemplo de uso donde fuera posible para familiarizar al lector con la herramienta, y finalizando con la explicación del funcionamiento.

Plugin IAC

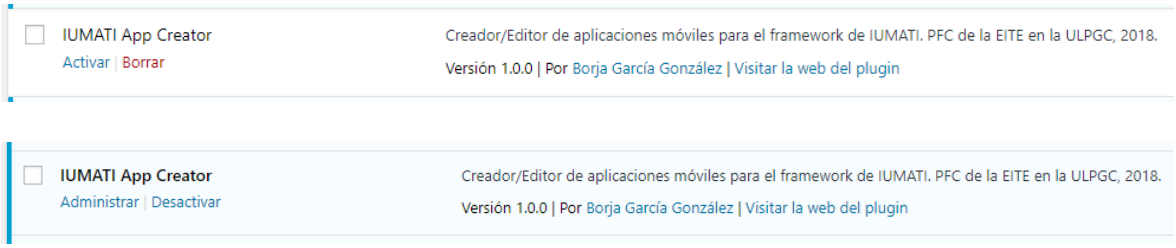
4.1 Sección de administración

Como ya se ha explicado, el plugin utiliza una estructura de *tipos* diseñada para que puedan ser creados y modificados sin necesidad de escribir código. Para facilitar la tarea se incorpora un menú de administrador dentro de los ajustes de Wordpress.

A continuación, se explicará el *uso del plugin por parte del administrador*, incluyendo la instalación, modificación y desinstalación. Hay que advertir que la explicación es para un plugin, y se parte de un servidor con Wordpress configurado.

4.1.1 Instalación / Desinstalación del plugin

Para utilizar el plugin el primer paso será *copiar el directorio del plugin* en {ruta-wordpress}/wp-content/plugins. Entonces aparecerá en la sección de plugins del menú de administración de Wordpress, pudiendo ser *activado pulsando el botón correspondiente*.



11. Plugin antes y después de activarlo en el menú de administrador

Al activar el plugin se ejecutará un código SQL que añadirá las tablas de aplicaciones (iac_*apps*) y de tipos (*iac_types*) a la base de datos, además de añadir algunos tipos predefinidos: *leftMenu*, *gallery*, *products*, *PDF* y *filtros*.

Para que el plugin sea accesible para el usuario debe escribirse un pequeño código en la página que el administrador elija. Se ha hecho así para que pueda elegir dónde añadirlo, incluyendo contenido antes o después si lo desea. Para ello se debe acceder a la sección "*Páginas*" del menú de administración de Wordpress, crear o editar una página e introducir el siguiente código:

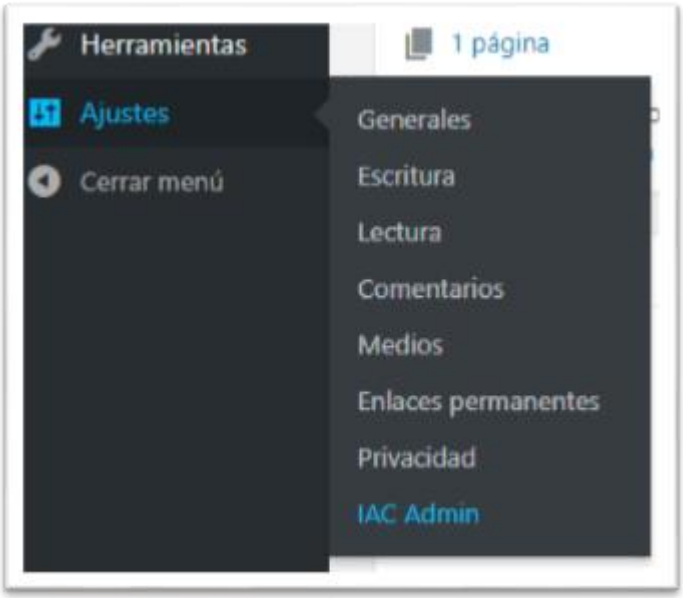
```
1.  <div ng-app='iac'>
2.      <div ng-view></div>
3.  </div>
```

No hay que olvidar que en caso de crear una página ésta debe añadirse a un menú para que sea accesible. Esto se puede hacer en la sección *Apariencias > Menús*.

Una vez activado podrá pulsarse el enlace "*Desactivar*" para que no se muestre el plugin. Se advierte que al desactivarse de borran las tablas creadas, perdiendo cualquier modificación que se hiciera a los tipos y cualquier aplicación creada. Además, la página creada seguirá existiendo, aunque no se mostrará nada.

4.1.2 Menú de administración

El plugin dispone de una *página de configuración* dentro del menú de administración de Wordpress. Para acceder se puede pulsar el enlace "*Administrar*" que aparece tras activar el plugin, o mediante la sección *Ajustes > IAC Admin*.



12. Ubicación del menú de administrador del plugin

El menú de administración del plugin permite *crear*, *modificar* y *eliminar tipos de controladores*. Para ello tiene un formulario en la parte superior, donde se introducirá la configuración del tipo, que se explicará más adelante; y una lista con los tipos existentes en la parte inferior.

Para crear un tipo bastante con introducir los datos y pulsar el botón de *Guardar*. El botón *Cancelar* vaciará el formulario.

Si se quiere editar un tipo existente se debe pulsar el botón correspondiente (*Editar*). Esto cambiará el título del formulario de "Crear tipo" a "Editar <nombre del tipo>" y rellenará el formulario con los datos del tipo, y al pulsar el botón de *Guardar* se modificará el

seleccionado. Pulsando el botón *Cancelar* se restablecerá el formulario, pasando de nuevo al modo “Crear tipo”.

Algunos tipos, como *leftMenu*, forman parte de la estructura de la aplicación, y la implementación del plugin depende de ésta. Por esta razón algunos tipos tienen restricciones en la edición o eliminación.

Configuración de los tipos

ID

La ID es una palabra (sólo letras, sin espacios) que identifica el tipo. Se corresponde con el type del tipo (controlador IUMATI). Por ejemplo: *leftMenu*, *products*, *gallery*.

Nombre

El nombre es el texto que se mostrará al usuario cuando quiera escoger el tipo a crear. No tiene por qué coincidir con el tipo, y sólo se utiliza para hacerlo más accesible. Por ejemplo, al tipo *products* se le ha puesto el nombre *Lista*.

Icono

El icono es la imagen que aparecerá junto al nombre del tipo (en el menú de creación) y junto al nombre del elemento (una vez creado), para indicar rápidamente al usuario de qué tipo se trata. Por defecto, los tipos preconfigurados tienen las iniciales en mayúscula sobre un fondo de color. Se indica mediante la URL a la imagen.

Por ejemplo, el tipo *products* tiene el icono 

, por la *L* de *L*ista, con la URL http://placehold.it/32/6e6/000?text=L.

Hijos permitidos

En las estructuras jerárquicas se llama *Padre* al elemento que contiene a otro, siendo éste el *hijo*. Por ello, en este campo se indica qué tipos pueden crearse dentro. Por ejemplo, la imagen corresponde al tipo *leftMenu*, que puede tener Lista (*products*) y Galería (*gallery*).

13. Campo de "Hijos permitidos" en el menú de administrador

El campo permite seleccionar varias opciones a la vez, utilizando las siguientes combinaciones:

Botón principal del ratón (BPR)	Selecciona sólo ese tipo.
Mantener y arrastrar BPR	Selecciona todos los tipos sobre los que se pase el cursor.
⇧SHIFT + BPR	Añade el tipo a los seleccionados.
CTRL | ⌘ + BPR	Selecciona todo entre el último seleccionado y el actual.

Productos permitidos

De forma análoga al campo anterior, en este caso se seleccionan los tipos que pueden crearse como productos. Los controles de selección son los mismos.

Número máximo de hijos

Permite limitar la cantidad de elementos hijo que puede tener un tipo. El valor por defecto es *-1*, que no establece limite. Cualquier valor superior limitará el total de hijos a esa cifra.

Params

Este campo se corresponde con la propiedad *params* de una categoría del tipo. Se introduce como una lista de palabras, siendo cada una de ellas un param.

La estructura a utilizar es la de *JSON* para listas: texto entre comillas dobles (“), separado por comas y con corchetes cuadrados ([ ]) encerrándolo. Por ejemplo:

```
1. ["type", "lightColor", "mediumColor", "strongColor", "fontType", "fontSize"]
```

ParamsConfig

Para cada uno de los *params* es necesario introducir cierta información de su comportamiento: ***default***, para el valor que tendrá, y ***heredable***, para indicar si este parámetro deben tenerlo sus hijos.

Nuevamente se introducen con la estructura JSON, pero esta vez para objetos: palabras entre comillas dobles, cada atributo se escribe como "clave": "valor", separando los atributos por comas, y todo encerrado en llaves curvas (*{ }*). Ejemplo:

```
1.  { "type": { "default": "leftMenu", "heredable": true }, "lightColor": { "default"
    : "#aaeeaa", "heredable": false }}
```

Plantilla para categoría

Es el contenido que verá el usuario al seleccionar un elemento de este tipo. Se trata de código HTML, pudiendo usar etiquetas *HTML*, *clases de Bootstrap* y *expresiones de AngularJS*.

Al utilizar *AngularJS* se puede acceder a las *variables del elemento* mediante la variable ***element***. Por ejemplo, puede acceder al nombre mediante `element.text`, a la descripción, imagen o params mediante las propiedades de `element.data: .description`, `.data.image_url` o `.data.params` respectivamente.

Además puede añadir variables en `element.data.variables`. Estas últimas están pensadas para permitir al usuario introducir información que se añadirá posteriormente a la estructura. Por ejemplo, *element.data.configuracion.usuario.pais*. Esta explicación se completará cuando se explique el campo ***Estructrura***.

Por otro lado, se han creado las etiquetas `<products-list>`, que muestra los productos como una lista de productos; y `<create-product>`, que añade un formulario para crear productos. De esta forma el administrador puede situarlos donde sea conveniente.

Por último, si la categoría necesita ser configurada se puede utilizar la etiqueta `<save-reset-edit>` que añade un pequeño panel para la edición. El formulario de edición debe ser creado por el administrador, y podrá usar el atributo `uib-collapse="!formData.config.editmode"` para mostrarlo y ocultarlo.

Ejemplo:

```
<h1>{{element.text}}</h1>
<save-reset-edit></save-reset-edit>
<div class="col-sm-12">
  Hoy hablaremos de
  <strong>
    {{element.data.variables.tema.titulo}}
  </strong> porque
  <strong>
    {{element.data.variables.tema.razon}}
  </strong>
  <div class="col-sm-12" uib-collapse="!formData.config.editMode">
    <div>Tema:
        <input ng-model='element.data.variables.tema.titulo'/>
    </div>
    <div>Razón:
        <input ng-model='element.data.variables.tema.razon'/>
    </div>

  </div>
</div>
<create-product></create-product>
<products-list></products-list>
```

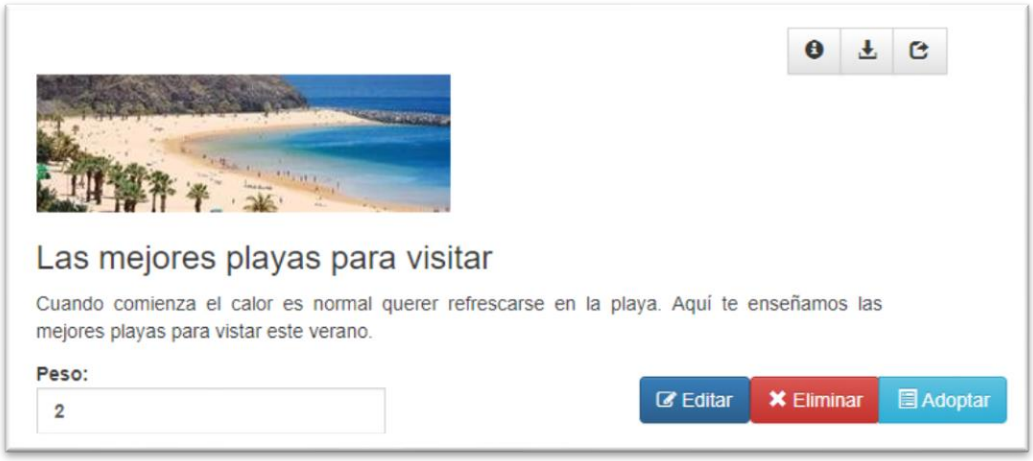
14. Ejemplo de plantilla de producto en el editor

Plantilla para productos

De forma análoga a la plantilla de categorías, el contenido de este campo será mostrado cuando el elemento sea creado como producto dentro de un elemento. En este caso, la variable que permite acceder a la información del producto es product:`.name`, `.image_url`, `.description` y `.params`.

También en este caso el formulario de edición será tarea del administrador, disponiendo de dos variables para ello:

- PC.editMode: equivalente a formData.config.editmode en la categoría, indica si se está editando.
- `PC.editProduct` como `product`: PC.editProduct.name,…



15. Ejemplo de producto en el editor

En el capítulo de Anexos se incluye el código que se usa para mostrar el tipo PDF, como en la imagen anterior.

Estructura

La estructura indicará al sistema la disposición de categorías y productos que debe tener el tipo para formar un tipo. Se compone de una lista de objetos JSON, siendo cada uno un elemento (categoría o producto) del tipo.

La mayoría de las propiedades de un elemento tienen una función intuitiva por estar directamente relacionados con una category o product del framework. Estas son: *ID*, *name*, *description*, *image_url*, *parent*, *params*, *ordering* y *published*. Además, hay dos propiedades nuevas:

- *Categ*ory: un boolean que indica si el elemento es categoría, siendo producto en su defecto.
- Struct: cuando se define una category, struct contiene los hijos de ésta.

Con esta configuración se puede configurar un tipo “estático”, es decir, que no contendrá más información que la suministrada por el administrador. Por ello se permite el uso de valores especiales que se sustituirán con datos introducidos por el usuario.

Estos valores van precedidos del prefijo "@@", que los distingue de un valor fijado por el administrador. Cada propiedad tiene unos valores especiales, por lo que se explicarán por separado:

- ***ID:***
  - @@id: asigna una nueva ID disponible.
  - @@hook: sólo para categorías. Asigna la ID del elemento en el editor, y fija la categoría como padre de los productos que haya podido definir el usuario y de los elementos que se añadiesen como hijo en el editor.
  - @@setRef@@nombre: Asigna una nueva ID y la guarda en un grupo con el nombre introducido. Se utiliza junto al valor @@getRef@@nombre para asociar varias categorías a un producto sin conocer de antemano qué IDs tendrán.
- ***Name***: @@text se sustituye por el texto del elemento en el editor.
- ***Description***: @@description se sustituye por la descripción del elemento en el editor.
- ***Image_url***: @@image_url se sustituye por la descripción del elemento en el editor.
- ***Parent:***
  - @@nodeParent: se sustituye por el parent del elemento en el editor.
  - @@structParent: se sustituye por el parent de la categoría o producto en la estructura.
  - @@getRef@@nombre: como se explicó anteriormente, asigna las IDs asociadas a ese nombre mediante @@setRef@@nombre.
- ***Params:***
  - @@categoryParams: se sustituye por la lista de params, como debe tener una category.
  - @@productParams: se sustituye por el valor de los params, con el formato adecuado.
  - @@customParams@@params: es equivalente a introducir los productParams de forma manual, pero admite valores especiales.
- ***Ordering***: @@ordering asigna un valor que mantiene el orden que se tenía en el editor.
- ***Published***: @@published introduce el valor del elemento en el editor. Al no utilizarse en el framework no se muestra en el menú, pero se ha añadido en previsión de un futuro.

Por último, se permite al administrador utilizar variables propias, tal y como se introdujo con las plantillas para categorías. Cualquier otra palabra que acompañe al prefijo se tomará

como variable, pudiendo además tener una estructura de objeto en JS. Siguiendo con el ejemplo de la plantilla, se utilizarían los valores especiales *@@tema.titulo* y *@@tema.razon*.

En el capítulo de Anexos se tiene un ejemplo de la estructura que tienen la mayoría de tipos, teniendo una categoría a la que añadir contenido (otros elementos y productos) y su metadato.

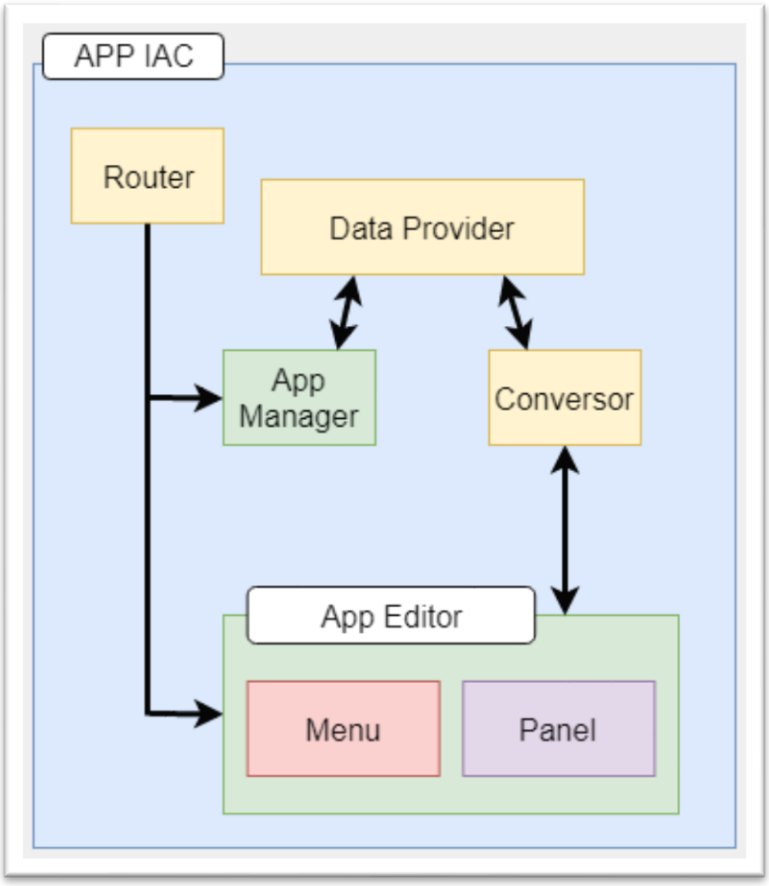
4.2 Sección de Usuario

La parte fundamental del proyecto es la interfaz de usuario para la creación y edición de aplicaciones. Ésta se implementa como una SPA (*Single Page Application*) con dos rutas: ***gestor de aplicaciones*** y ***editor de la aplicación***.

La aplicación de AngularJS consiste en 2 módulos principales (*gestor de aplicaciones* y *editor*) y otros *módulos* y *servicios* complementarios.

Si bien se explicará cada elemento por separado, se introducirá el sistema en conjunto para facilitar el entendimiento de cada parte:

1. El ***Router*** utiliza la ruta de la página para utilizar *AppManager* o *AppEditor*.
2. El servicio ***Data Provider*** realiza las peticiones AJAX a la *interfaz PHP* para obtener y entregar datos.
3. El servicio ***Conversor*** transforma la *estructura de datos* que se utiliza en *AppEditor* a la que utiliza el framework IUMATI.
4. ***AppManager*** y ***AppEditor*** son los módulos principales con los que trabajará el usuario.

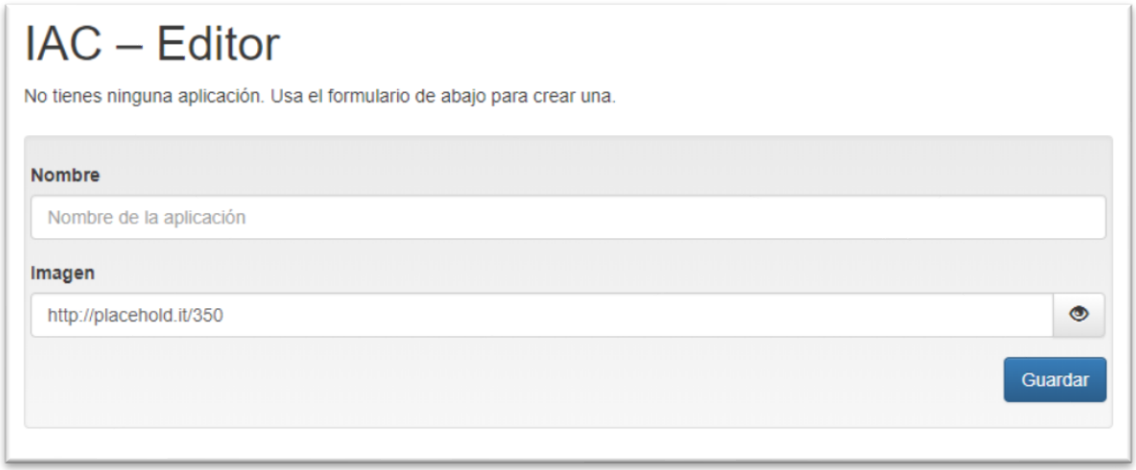


16. Estructura general del plugin

A continuación, se explicará el uso y funcionamiento de cada parte.

4.2.1 Gestor de aplicaciones

El gestor de aplicaciones permite que al usuario cree, elimine y modifique algunos aspectos de sus aplicaciones. La primera vez que acceda encontrará un formulario que le invita a crear su primera aplicación.



17. Editor: gestor de aplicaciones

Bastará con introducir el nombre de la aplicación y la ruta de la imagen principal. Pulsando el botón del ojo junto al campo de la imagen se podrá previsualizar.

Una vez creada, aparecerá una lista con la única aplicación, mostrando algunos datos, como la fecha de creación y de última modificación.

18. Editor: Ejemplo de aplicación creada

El enlace de Nombre lleva al editor, y los de Acción permiten editar el nombre y la imagen, o eliminar la aplicación.

Funcionamiento

El gestor de aplicaciones es un formulario *CRUD* (Create, Read, Update, Delete), que se limita a mostrar información sencilla y permitir modificarla mediante campos de texto simples, accediendo a la base de datos a través del *Data Provider*.

4.2.2 Editor de aplicaciones

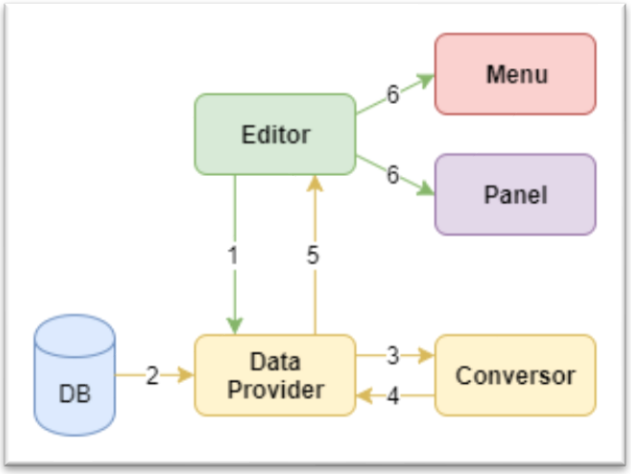
Basándose en la idea inicial de mostrar al usuario una vista similar a la que podría tener la aplicación en un dispositivo real, el editor cuenta con un menú a la izquierda para la gestión de elementos y un panel a la derecha donde ver y editar el contenido de cada uno de ellos. Además, tiene un enlace para volver al editor y un botón para guardar los cambios en la aplicación. Es importante señalar que el menú no se guardará si no se pulsa este botón, para permitir al usuario descartar los cambios.

19. Editor: vista de una aplicación creada

En esta imagen se ven los elementos por defecto (*LeftMenu* y una raíz para *Filtros*) en el menú y la plantilla para *leftMenu* definida por el administrador.

Funcionamiento

Al cargar el editor solicitará (1) los tipos de controladores y la aplicación mediante *Data Provider*. Éste los pedirá a la DB (2) y pasará la aplicación por el *Conversor* (3) antes de entregarla al editor (5). Si ha obtenido los datos de manera satisfactoria iniciará los módulos de *Menu* y *Panel*, suministrándoles los datos para que puedan configurarse (6).



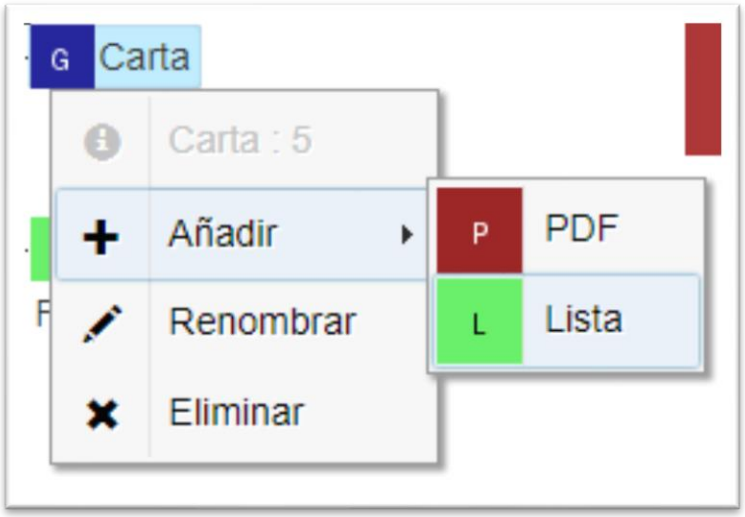
20. Proceso de carga del editor

4.2.2.1 Menú

El menú permite al usuario ver y gestionar fácilmente los elementos de su aplicación. Inicialmente contiene dos elementos:

- *LeftMenu*, configurada con el nombre de la aplicación.
- *Filtros*, que sirve de contenedor para los filtros.

Para añadir o modificar un elemento se dispone de un menú contextual al que se accede pulsando con el Botón Secundario del Ratón en un elemento.



21. Editor: menú contextual

Las opciones dependen del elemento seleccionado. Por ejemplo, el elemento *LeftMenu* no puede ser eliminado o renombrado (esto se realiza desde el gestor de aplicaciones), y sólo aparecerán en el sub-menú de Añadir los elementos indicados como hijos válidos.

También se pueden mover y copiar los elementos. Para ello basta con pulsar y mantener el Botón Principal del Ratón sobre un elemento y mover. Si la acción no fuese posible (el padre no permite ese hijo, hay otro elemento con el mismo nombre,…) el cursor mostrará una cruz roja , mientras que si es una acción permitida será una marca verde .

Si mientras se mueve se mantiene pulsado el botón Control / ⌘ la acción cambiará de Mover a Copiar.

Se pueden seleccionar varios elementos manteniendo pulsado CTRL antes de pulsar sobre cada uno de ellos, o ⇧Shift para seleccionar todos los intermedios. Hay que tener

en cuenta que en este caso todos los elementos deben poder realizar la acción o ninguno lo hará.

Funcionamiento

Cuando el *App Editor* llama al *Menu* con los tipos de controlador se configura *jsTree* con un árbol vacío, pero que contiene en su configuración los tipos, eventos y menú contextual. Posteriormente se llama al *Menu* con los datos de la aplicación, que se añaden al árbol. Como una aplicación recién creada está vacía, el *Menu* es el encargado de iniciar el árbol con los elementos básicos: *LeftMenu* y la raíz para filtros.

La estructura escogida para los elementos del *Menu* se basa en la que *jsTree* utiliza para sus nodos:

- ***ID***: cada elemento del árbol tiene una ID única. Aunque por defecto *jsTree* utiliza *IDs alfanuméricas*, es posible cambiársela en el momento de su creación y asignarle números.
  - Se puede asignar esta ID a la aplicación con *@@hook*.
- ***Text***: el nombre del elemento en el *Menu*.
  - Se puede asignar el texto a la aplicación con *@@text*.
- ***Type***: jsTree utiliza tipos en sus nodos, lo que permite personalizar su aspecto (como el icono que se muestra) o moderar su comportamiento (como el permitir que un tipo pueda contener a otro, cantidad de hijos, …)
  - Este valor se asigna automáticamente a la categoría con ID *@@hook*, pero también se puede asignar manualmente con *@@type*.

Todo lo demás debe añadirse a la propiedad *Data* de cada elemento, pues jsTree no permite más modificaciones en sus elementos.

Una ventaja de haber decidido trabajar con una estructura distinta a la del framework IUMATI es que los elementos no necesitan seguir la estructura categoría-producto, que complicaría el trabajo en una estructura de árbol. De este modo, un solo elemento contiene toda la información del tipo, incluyendo los productos que añada el usuario.

Esto se hace mediante la propiedad *Data* que jsTree pone en cada elemento. Ahí se guardarán los campos que faltaban (*params*, *description*, *imagel_url*) y algunos extras:

- ***Products***: una lista con todos los productos que el usuario añada a este elemento. Se explicarán con más detalle al hablar del Panel.
  - Se añaden automáticamente a la categoría con id *@@hook*.
- ***Variables***: como se vio al explicar la parte del administrador, éste puede utilizar variables para pedir al usuario datos necesarios para el tipo. Es por ello que se usaba `element.data.variables` para referirse a ellas.
  - Se añaden utilizando el valor especial @@ seguido de la ruta, a partir de variables.
- ***Instances***: una opción del framework es que una categoría puede pertenecer a varios padres. jsTree no permite esto, por lo que se utilizan las instancias: cuando se quiere tener un elemento en varios sitios se crea uno nuevo y se anotan sus IDs en esta lista. Cuando un elemento se modifica se realiza el mismo cambio en todas sus instancias, a excepción del borrado. Esto se realiza automáticamente cuando se copia un elemento.
  - Las instancias se transforman en el *Conversor* de forma automática.

Todo esto se genera de forma transparente al usuario, sea en la creación del elemento mediante el ***menú contextual*** o en las llamadas ante los ***eventos*** que se producen en respuesta a acciones del usuario.

El ***menú contextual*** se genera cada vez que se pulsa el Botón Secundario del Ratón  , y depende de sobre qué elemento se ha pulsado. Por ejemplo, la opción eliminar no aparece para LeftMenu, pues es un elemento necesario. La opción Añadir cuenta con un submenú, cuyas opciones dependen de los hijos válidos (valid children) que tenga el elemento.

Los ***eventos*** de jsTree pasan por dos fases:

- ***Comprobación de permisos***: antes de finalizar una acción jsTree comprueba si será posible terminarla en el estado actual. Las comprobaciones son:
  - *Mover/Copiar un elemento*: se permite si no hay otro elemento con el mismo nombre y el nuevo padre admite este tipo como hijo. Para mostrar  o  en el cursor, la comprobación se realiza con cada movimiento del ratón.
  - *Renombrar un elemento*: se permite si todas las instancias pueden renombrarse sin tener el mismo nombre que un elemento hermano.

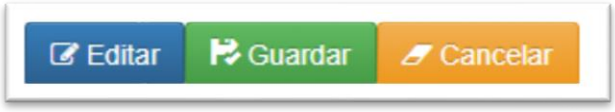
- *Crear o eliminar un elemento*: no es necesario comprobar si se puede realizar, pues sólo puede crearse o eliminarse mediante las herramientas diseñadas para ello, que aportan el control necesario.

- ***Evento de respuesta***: si la acción se pudo completar se lanza un evento, al que se asoció un método durante la configuración de jsTree. Los eventos a los que se reacciona son:
  - *Rename*: cuando se renombra un elemento se comprueba si tiene instancias, para renombrarlas también.
  - *Delete*: se elimina su ID del resto de instancias (si existen) y del category_id de cada producto que tuviera.
  - *Move*: mover un elemento sólo afecta a su parent, y esto lo gestiona jsTree. La única acción que se lleva a cabo es estética, y consiste en asegurarse de que el elemento padre está "abierto" (mostrando sus hijos).
  - *Copy*: jsTree se encarga de copiar el contenido, pero es necesario actualizar nuestra estructura añadiendo la nueva instancia a la lista de instances y su ID a la category_id de los productos.
  - *Changed*: este nombre puede llevar a error, pues sólo notifica cambios en la selección. Cuando se cambia el elemento seleccionado se llama a un método que compara el elemento antes y después y actualiza todos los elementos necesarios.

4.2.2.2 Panel

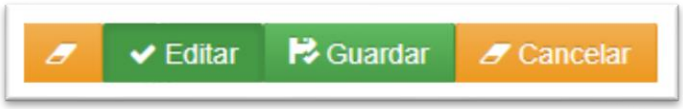
El panel es la otra parte del editor de aplicaciones que utilizará el usuario. En él se mostrará la plantilla diseñada por el administrador para el tipo del elemento seleccionado en el menú. Por ello, lo único que se puede explicar sobre su uso son los componentes que se han puesto a disposición del administrador.

El primero es el grupo de botones para gestionar el elemento. En su estado inicial tiene un botón para desplegar la vista de edición, uno para guardar el estado actual del elemento y otro que lo revierte al último estado guardado.



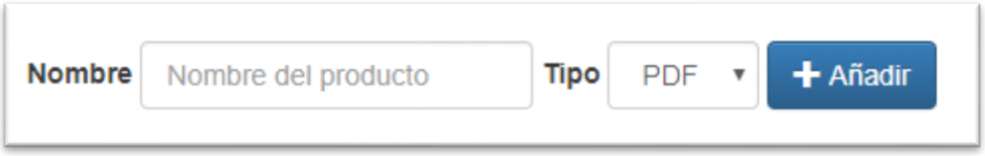
22. Editor: botones de gestión del elemento

Al pulsar el botón de Editar se desplegará la vista que el administrador haya diseñado para tal fin con el atributo `uib-collapse`, y cambiará el grupo de botones para añadir el de deshacer. Este botón, similar a cancelar, deshará todos los cambios en la edición del elemento. Por el contrario, volver a pulsar el botón de Edición guardará los cambios.



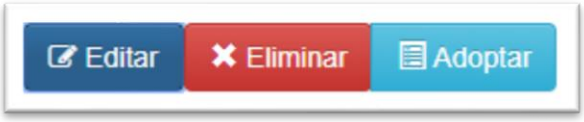
23. Editor: botones de gestión de aplicaciones, desplegado

Otro componente es el de crear producto, que consta de un campo para el nombre y un selector de tipo. En este selector sólo aparecerán los indicados como *valid products* del elemento seleccionado.



24. Editor: formulario de creación de productos

Por último, cada producto cuenta con un grupo de botones para gestionar su información, similar al del elemento.



25. Editor: botones de gestión del producto

El botón editar abrirá la vista de edición del producto, y añadirá un botón para deshacer los cambios. En caso de pulsar nuevamente Editar se guardarán los cambios.

26. Editor: botones de gestión del producto, desplegado

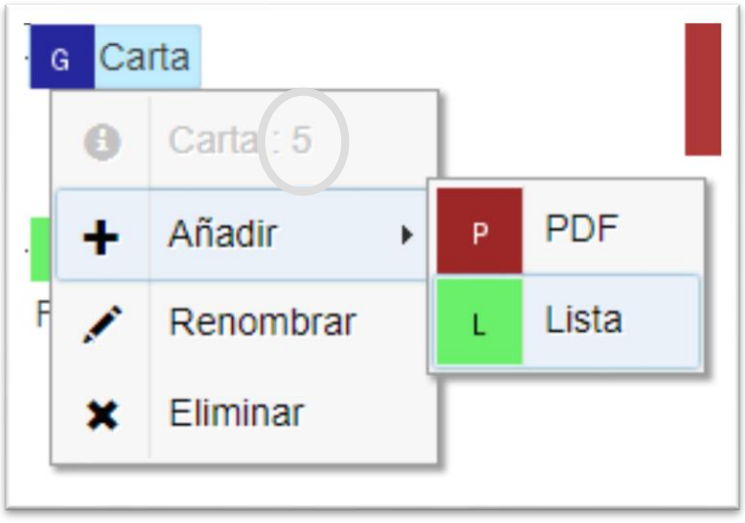
El botón de adoptar sirve para añadir el producto a otras categorías. Al desplegarse aparecerán sólo las categorías que cumplan los requisitos para tener este producto:

permiten el tipo del producto, no tienen ninguno con el mismo nombre y no han alcanzado el máximo de hijos.

27. Editor: selector de categorías padre de un producto

En este ejemplo se ve que se puede añadir a los filtros Para vegetarianos y Para celíacos, pues los dos admiten productos tipo PDF como el actual.

Los números que acompañan a los nombres son las IDs de los elementos, que puede ser más de una si existen varias instancias. Esto se indica para evitar confusiones en caso de tener dos categorías con el mismo nombre en distintas partes de la aplicación. Puede consultarse la ID en el menú contextual:



28. Editor: ubicación de la ID de un elemento

Hay que tener en cuenta que ninguno de estos cambios se guardará hasta que no se pulse el botón Guardar introducido al principio de esta sección o al cambiar de elemento en el menú.

Funcionamiento

El Panel se inicia junto con el resto del editor, cargando las plantillas de los tipos existentes para poder mostrarlas.

Cuando se selecciona un elemento en el *Menú*, éste lo indica al *Editor* y éste a su vez al *Panel*, que extrae su tipo y datos, carga la plantilla en la vista y la rellena con la información. Debido a que la plantilla puede contener etiquetas que deben ser procesadas por AngularJS no basta con añadir el código HTML, sino que debe compilarse.

A partir de este momento todos los cambios que se hagan en el *Panel* será sobre una copia del elemento, de forma que es posible revertir los cambios. La copia se pasará al *Menú* cuando seleccione un nuevo elemento o se pulse el botón de *Guardar*.

Los componentes para gestión de elemento y producto funcionan de una forma muy similar: al pulsar el botón de edición se muestra la parte de edición y se rellena con una copia de la información. Si no se hiciera esta copia, modificar el formulario se reflejaría inmediatamente en el elemento o producto, por el funcionamiento de AngularJS. Al aceptar los cambios se vuelcan sobre la versión del Panel.

Por último, la función de *Adoptar* necesita llamar al *Menú* a través del *Editor* para saber qué elementos pueden contener un producto.

4.2.3 Otros componentes

4.2.3.1 Data provider

Todos los datos pasan a través del servicio de *Data Provider*, sean de entrada o salida. De esta forma se centraliza la forma en la que se accede a ellos.

En el caso de este plugin, esto se realiza mediante *peticiones AJAX* a Wordpress, a los métodos escritos en PHP que se explicarán más adelante.

Como parte de este flujo de datos, se le añadió la responsabilidad de llamar al *Conversor*.

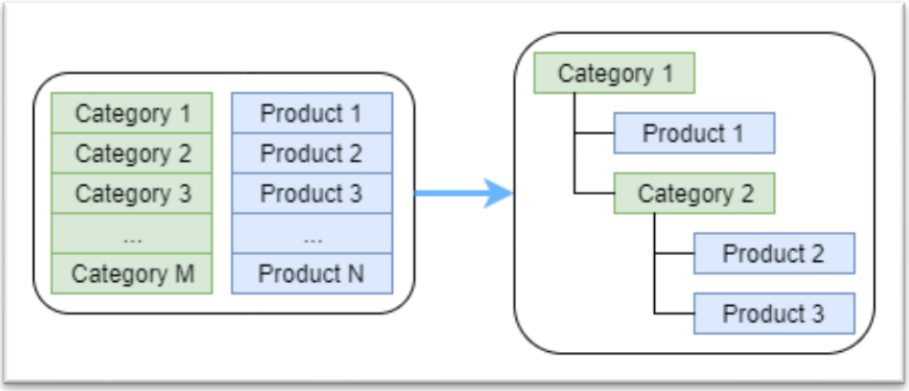
4.2.3.2 Conversor

El *Conversor* es un servicio que permite pasar entre las dos estructuras de datos que se han explicado: la del framework IUMATI y la del plugin IAC.

Conversión IUMATI → IAC

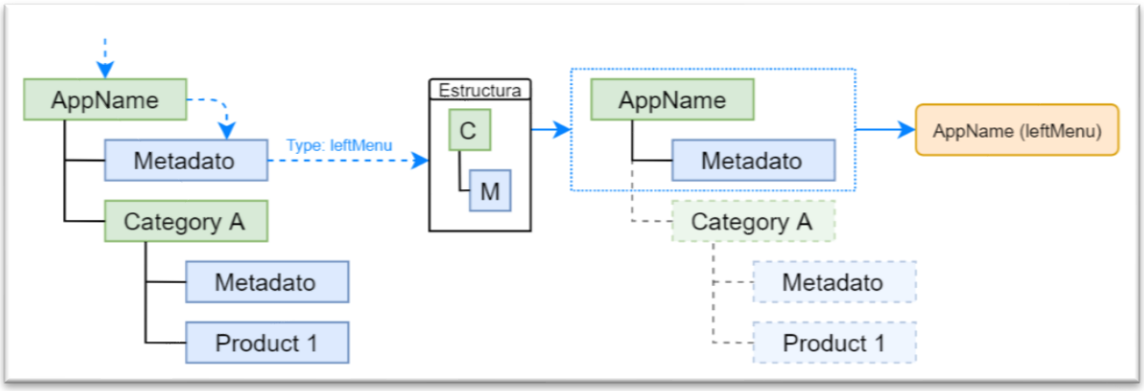
Se implementan cuatro etapas para pasar de dos listas de categorías y productos al árbol que gestiona el Menú:

- ***Convertir las listas en un árbol***: jsTree relaciona elementos mediante un campo *children*, mientras que IUMATI utiliza una referencia a los padres. El primer paso es recorrer cada categoría y producto, añadiéndolos a un árbol. Para ello se busca su campo de padres y se añade en éstos la categoría o producto como hijo.



29. Conversor: esquema de conversión lista-árbol

- ***Procesar el árbol por niveles***: se recorre el árbol recursivamente desde la raíz hasta las hojas, nivel por nivel. El primer paso es identificar el tipo de elemento al que pertenece la categoría, sirviéndose para ello del type de su metadato. Una vez conocida, se recorre la estructura indicada por el administrador, buscando qué categorías y productos coinciden.



30. Conversor: esquema del procesado por niveles

- ***Añadir productos del usuario***: al procesar buscando equivalencias en la estructura, los productos añadidos por el usuario no han sido procesados. Con la estructura terminada es fácil añadirlos a sus padres.
- ***Encontrar instancias***: el framework no indica internamente las instancias, por lo que la única forma es asumir que dos elementos idénticos (sólo se diferencian en la ID y el padre) son instancias. Cuando se encuentra un caso de instancia se van relacionando.

Conversión IAC → IUMATI

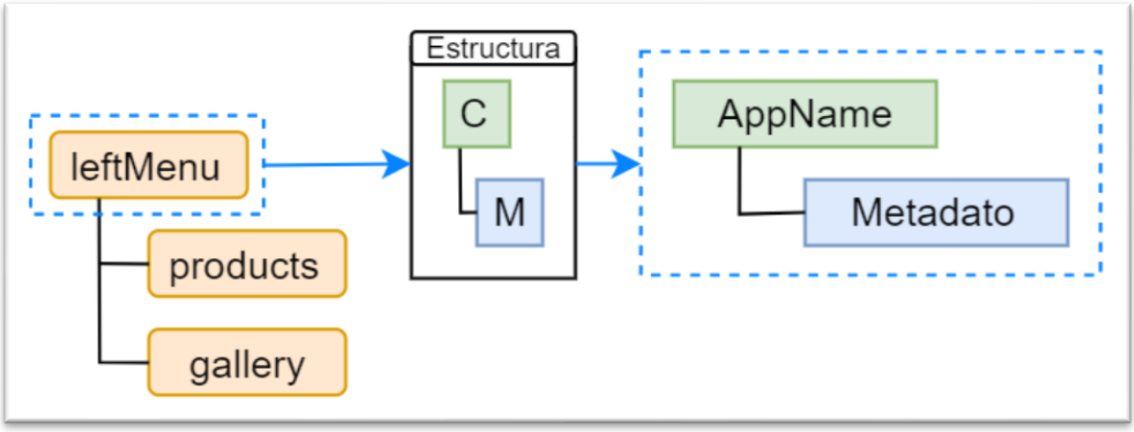
La conversión en el sentido contrario es algo más compleja debido a dos condiciones del framework:

1) Los parámetros de una categoría se heredan a sus descendientes.
2) Todos los padres de un producto deben tener los mismos parámetros.

Con la posibilidad de tener un gran cruce de productos, es necesario organizar las categorías en grupos, en función de los parámetros que debe implementar. Por ejemplo, un producto que pertenece a tres categorías (A, B, C), donde una de ellas (C) es, además, descendiente de otra (B). A y B compartirían sus parámetros, que se heredarían a C, y éste los compartiría con B, lo que afectaría al resto de descendientes.

Para poder cumplir con las dos condiciones, antes de la conversión se agrupan los elementos, de forma que los de un mismo grupo tienen los mismos parámetros.

Una vez relacionadas basta con recorrer los elementos y añadir la estructura correspondiente a su tipo y procesando sus parámetros.



31. Conversor: esquema de la conversión elemento - categorías y productos

4.3 PHP

Como ya se ha explicado, es necesario incluir código PHP para incorporar el plugin en Wordpress, tanto en el proceso de instalación/desinstalación como para atender las peticiones a la base de datos.

Para integrarlo en Wordpress se ha utilizado un *boilerplate*, una especie de plantilla de código. El boilerplate escogido es el de *WPPB* (WordPress Plugin Boilerplate), que se genera mediante un formulario en su página web y dispone de lo necesario para gestionar el plugin sin necesidad de conocer en profundidad el desarrollo en Wordpress.

Siguiendo la estructura de este boilerplate, el código añadido o modificado se reparte en tres directorios:

- *Includes*:
  - *class-iac.php* registra los métodos que deben ser llamados por Wordpress para que el plugin se integre.
  - *iac-activator.php* e *iac-deactivator.php* se encargan de activar y desactivar el plugin respectivamente. Las únicas operaciones necesarias son las de añadir y rellenar las tablas de datos y aplicaciones en la base de datos al activar, y las de eliminarlas al desactivar.
    - Create-db-tables.sql y add-element-types.sql: código SQL que se ejecuta al activar el plugin.
- *Admin*:
  - *Class-iac-admin.php*: importa el código JS y el css que se usará en la sección de administración, e implementa los métodos que conectan JS con la base de datos.
- *Public*
  - *class-iac-public.php*: importa el código JS y el css que se usará en la sección de usuario, e implementa los métodos que conectan JS con la base de datos.

La integración se realiza mediante hooks, una funcionalidad de Wordpress que permite registrar métodos propios que deben ser llamados en momentos específicos. Se pueden encontrar en includes/class-iac.php, en los métodos define_public_hooks() y define_admin_hooks(). Los hooks utilizados son:

- ***Admin_enqueue_scripts / wp_enqueue_scripts***:
  - Se ejecuta cuando Wordpress está añadiendo scripts a la página.
  - Es el momento apropiado para incorporar el código JS creado para este plugin y las hojas de estilo CSS.
- ***Admin_menu***:
  - Se ejecuta cuando Wordpress crea el menú de administrador.
  - Añade la página de administrador de IAC en el menú de ajustes.
- ***Plugin_action_links***:
  - Se ejecuta en el administrador de plugins.
  - Añade un enlace a la página de administrador de IAC bajo el plugin.
- ***Wp_ajax_[action]***:
  - Se ejecuta cuando se realiza una petición AJAX con el action indicado. Por ejemplo, *wp_ajax_iac_add_type* responderá cuando se realice una petición con *action = iac_add_type*.
  - La acción depende del método solicitado.

Las respuestas a estas peticiones AJAX son métodos que acceden a la base de datos. Por ello comprueban previamente que los valores recibidos son válidos. Por ejemplo, cuando un usuario pide una app comprueba que le pertenece. En caso de ocurrir un error devolverá una respuesta debidamente formateada, siendo de especial importancia el campo code, pues indicará al código JS qué error se ha producido.

Conclusiones

Para finalizar la memoria del proyecto se presentarán las conclusiones en base a los objetivos propuestos y en qué medida se han cumplido. Por último, se comentarán posibles líneas para mejoras del proyecto.

Objetivos

Estudio de las opciones

Inicialmente se pretendía estudiar Bonita para ver si era una opción viable y, de paso, tener conocimientos en una herramienta que potencialmente sería útil en un futuro laboral. Lamentablemente, siendo Bonita una plataforma tan grande se tuvo que centrar el aprendizaje en los elementos necesarios para decidir la viabilidad de la opción, por lo que, si bien se tienen conocimientos puntuales, no se ha conseguido aprender a utilizar la herramienta de forma útil.

Estudio de la opción escogida

La experiencia adquirida con Bonita sirvió para no dedicarle tanto tiempo al estudio de la plataforma de Wordpress, que únicamente se utilizaría como entorno al que incorporar un código prácticamente independiente. Aun así, se aprendió las bases del uso de plugins y de cómo éstos se integran en Wordpress.

Desarrollo del proyecto

Cuando se comenzó el proyecto se tenía una idea básica de lo que se tendría que hacer, basándose en las indicaciones y recomendaciones del tutor. Sin embargo, durante el desarrollo surgieron retos que no podían ser previstos con anterioridad, y que supusieron modificaciones en la idea.

La primera idea consistía en un menú en forma de árbol que jerarquizara las categorías y productos y permitiese editar sus propiedades en el panel. La documentación inicial de la que se disponía presentaba controladores (tipos) sencillos, por lo que la gestión era viable.

Cuando se conoció la complejidad de otros controladores, como el Maps, se entendió que sería muy complicado para un usuario crear la estructura necesaria. Por ello surgió la segunda idea: agrupar los controladores en elementos que hicieran transparente la estructura interna al usuario.

Esto sumó muchos retos, como la estructura interna de los tipos para que fueran definidos por el administrador, mostrados al usuario y convertidos a la estructura del framework; y la gestión de los propios elementos, que pasaban de ser categorías y productos con información propia a estructuras que contenían todos los datos.

De todo esto se extrae la conclusión de que el diseño de la estructura de datos es vital para un buen desarrollo del proyecto, y que debe estar abierta a modificaciones para poder amoldarse a los cambios que suelen surgir con el tiempo.

Líneas futuras

Los principales cambios en los que se podría trabajar para mejorar este proyecto son:

Mejorar la estética

Al empezar el proyecto se contaban con los conocimientos básicos de maquetación para realizar una interfaz sencilla, y se decidió priorizar la funcionalidad una vez alcanzado un mínimo de diseño.

Como resultado, la interfaz cumple con su cometido de facilitar la tarea de desarrollar aplicaciones para el framework, pero sería interesante mejorar la formación en diseño y maquetación y tener una interfaz que sea tanto útil como atractiva.

Menú de administrador

La primera aproximación a la creación de tipos por parte del administrador fue importarlos desde ficheros con el objeto JSON. Como no siempre será posible ni fácil añadir contenido al servidor, se decidió añadir el menú de administrador.

Siendo por tanto una función accesoria no se le pudo dedicar el tiempo necesario para brindar una interfaz más robusta. Por ejemplo, se podría facilitar el manejo de listas y objetos para los params y la estructura, o incorporar un editor visual para las plantillas.

Conexión con una base de datos Parse

Las categorías y productos de la aplicación deben estar en una base de datos Parse para ser utilizable en un dispositivo móvil. Lamentablemente, ParseDB dejó de dar servicio y ofreció en su lugar el código para montar un servidor propio.

Siendo el objetivo de este proyecto crear una interfaz para el framework se considera fuera del alcance el estudio de la infraestructura y software necesarios para disponer de un

servidor de ParseDB accesible desde Internet, que además alargaría el desarrollo del proyecto.

Bibliografía

[1] Yeeply, «Cálculo del coste de una aplicaicón móvil según el número de horas,» 16 Noviembre 2015. [En línea]. Available: https://www.cuantocuestamiapp.com/.

[2] Forbes, «Penetración de mercado de los Smartphone Android e iOS,» 16 11 2015. [En línea]. Available: https://www.forbes.com/sites/dougolenick/2015/05/27/apple-iosand-google-android-smartphone-market-share-flattening-idc/2/.

[3] Ionic, «Ionic Framework,» [En línea]. Available: https://ionicframework.com/. [Último acceso: 25 07 2018].

[4] Facebook, «React,» [En línea]. Available: https://reactjs.org/. [Último acceso: 25 07 2018].

[5] Bonitasoft, «Bonitasoft | Plataforma de desarrollo de applicaciones de código abierto | BPM,» [En línea]. Available: https://es.bonitasoft.com/. [Último acceso: 2015 11 15].

[6] Wordpress, «Wordpress,» [En línea]. Available: https://es.wordpress.org/. [Último acceso: 16 11 2015].

[7] Wikipedia, «Javascript - Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/JavaScript. [Último acceso: 25 02 2016].

[8] Google, «AngularJS,» [En línea]. Available: https://angularjs.org/. [Último acceso: 01 03 2016].

[9] I. Bozhanov, «jsTree,» [En línea]. Available: https://www.jstree.com/. [Último acceso: 01 03 2016].

[10] PHP Groupo, «PHP: Hypertext Preprocessor,» [En línea]. Available: http://php.net. [Último acceso: 12 03 2016].

[11] Wikipedia, «Java (lenguaje de programación) - Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n). [Último acceso: 20 06 2018].

[12] Wikipedia, «Kotlin (lenguaje de programación) - Wikipedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Kotlin_(lenguaje_de_programaci%C3%B3n). [Último acceso: 20 03 2016].

[13] campusMVP, «Swift vs Objective-C,» [En línea]. Available: https://www.campusmvp.es/recursos/post/Swift-vs-Objective-C-5-preguntas-indispensables-para-aprender-a-programar-para-iOS.aspx. [Último acceso: 20 07 2018].

[14] Envato, «Buy Plugins & Code from CodeCanyon,» [En línea]. Available: https://codecanyon.net/. [Último acceso: 18 07 2017].

[15] King of app, «King of App - Empieza a crear tu app para Android e iOS,» [En línea]. Available: https://kingofapp.es/. [Último acceso: 20 07 2018].

[16] Adiante ventures, «Crea tu app sin programar - adiante apps,» [En línea]. Available: https://www.adianteapps.com. [Último acceso: 20 07 2018].

[17] Iliberi, «aplicarium | Crea tu aplicación móvil,» [En línea]. Available: http://aplicarium.com/. [Último acceso: 20 07 2018].

[18] WebDorado, «Spider Catalog by WD,» [En línea]. Available: https://es.wordpress.org/plugins/catalog/. [Último acceso: 19 04 2018].

[19] Parse, «Parse + Open Source = ❤,» [En línea]. Available: https://parseplatform.org/. [Último acceso: 15 11 2015].

[20] L. H. Acosta, «Georruta Transgrancanaria,» [En línea]. Available: https://play.google.com/store/apps/details?id=es.ulpgc.iumati.android.georutatransgc&hl=es. [Último acceso: 15 11 2015].

Pliego de condiciones

Recursos necesarios para su utilización

Para el correcto uso de este proyecto es imprescindible el uso de un ordenador con pantalla, dispositivo señalizador (ratón o equivalente) y conexión a Internet. Se recomienda el uso de teclado para introducir texto.

Dado que el proyecto se entrega como plugin de Wordpress, el hardware y software necesarios serán aquellos que necesite Wordpress para funcionar. Queda a discreción del administrador la elección de las capacidades, pero debería componerse al menos de un equipo con un servidor web, PHP y SQL y una instalación Wordpress correctamente configurada. En este proyecto se ha utilizado el servidor *XAMPP v3.2.2*.

Para acceder al servicio es necesario un navegador que admita *HTML5*, *CSS3* y *JS2015* (que implementa ES6). No se garantiza la visualización con una resolución inferior a 370px, siendo óptima a partir de los 990px. Esto puede cambiar si en futuras versiones de Wordpress se modifica el espacio disponible para el contenido. En este proyecto se ha utilizado los navegadores *Google Chrome v67.0.3396.99 64bits* y *Mozilla Firefox v60.0.2 64bits*.

Concesión de licencia

Este Proyecto Fin de Carrera es propiedad de la Universidad de las Palmas de Gran Canaria y cualquier usuario debe estar de acuerdo en obligarse por los términos y condiciones establecidas en esta licencia del proyecto aceptando todas sus cláusulas. El uso de los distintos programas o ficheros, o de una copia en cualquier ordenador o computadora sólo podrá darse bajo la autorización expresa del autor, el tutor del proyecto y de la Escuela de Ingeniería de Telecomunicación y Electrónica de la Universidad de Las Palmas de Gran Canaria.

Derechos de autor

Este proyecto y su documentación asociada están protegidos por las leyes de propiedad intelectual que le sean aplicables, así como las disposiciones de los tratados internacionales. Por consiguiente, el usuario deberá utilizar el material relativo al proyecto como cualquier producto protegido por derechos de autor. Sin embargo, se permitirá que el usuario pueda realizar una copia de los códigos fuente de programación y de la documentación del proyecto siempre que exista una autorización previa del autor, el tutor

del proyecto y de la Escuela de Ingeniería de Telecomunicación y Electrónica perteneciente a la Universidad de Las Palmas de Gran Canaria.

Limitación de responsabilidad

En ningún caso serán el autor, el tutor o la Escuela de Ingeniería de Telecomunicación y Electrónica de la Universidad de Las Palmas de Gran Canaria responsables de los perjuicios directos, indirectos, incidentales o consiguientes, gastos, lucro cesante, pérdida de ahorros, interrupción de negocios, pérdida de información comercial o de negocio, o cualquier otra pérdida que resulte del uso o de la incapacidad de usar los ficheros o la documentación del proyecto. El usuario conoce y acepta que los derechos de licencia reflejan esta asignación de riesgo como el resto de cláusulas y restricciones. Asimismo, el autor y el tutor de este proyecto rechazan cualquier otra garantía que no haya sido indicada anteriormente.

Otros

En el supuesto de que cualquier disposición de esta licencia sea declarada total o parcialmente inválida, la cláusula afectada será modificada convenientemente de manera que sea ejecutable una vez modificada, plenamente eficaz, permaneciendo el resto de este contrato en vigencia y regido por las leyes de España.

Finalmente, el usuario acepta la jurisdicción exclusiva de los tribunales de este país en relación con cualquier disputa que pudiera derivarse de la presente licencia.

Presupuesto

Don Borja García González, autor del presente Proyecto Fin de Carrera,

DECLARA QUE:

El proyecto Fin de Carrera con título “Interfaz Web para el Desarrollo de Aplicaciones Móviles Utilizando la Nube como Back-end”, realizado a petición de la Escuela de Ingeniería de Telecomunicación y Electrónica de la Universidad de Las Palmas de Gran Canaria y en un periodo de 24 meses, tiene un coste total de ***CINCUENTA Y OCHO MIL QUINIENTOS NOVENTA Y DOS EUROS CON DIECISIETE CÉNTIMOS (58.592,17 €)*** correspondiente a la suma de las cantidades consignadas a los apartados considerados a continuación.

Firmando la presente para que así conste a los efectos oportunos.

Autor del Proyecto:

Borja García González

Las Palmas de Gran Canaria, a 03 de Agosto de 2018

Desglose del presupuesto

El presupuesto del proyecto realizado se ha generado según los precios de mercado actual, y de las indicaciones del COIT (Colegio Oficial de Ingenieros de Telecomunicación) y de la AEIT (Asociación Española de Ingenieros de Telecomunicación), a efectos de visado.

Los conceptos en los que se reparte el presupuesto son los siguientes:

1. Amortización del inmovilizado material.
   a) Amortización del material hardware empleado.
   b) Amortización del material software empleado.
2. Trabajo tarifado por tiempo empleado.
3. Redacción del proyecto.
4. Derechos de visado del COIT.
5. Gastos de tramitación y envío.
6. Aplicación de impuestos.

Amortización del inmovilizado material

Se trata de la corrección de valor por la depreciación del inmovilizado material del proyecto realizada de acuerdo con un plan sistemático definido con anterioridad.

Normas de valoración

El inmovilizado material se ha registrado a su coste de adquisición. No se han capitalizado ningún tipo de gastos financieros. No se han incurrido en gastos de reparación, conservación o mantenimiento.

A esta categoría pertenece la amortización del hardware y del software empleado en la realización del Proyecto Fin de Carrera. El sistema de amortización empleado ha seguido el método lineal, que distribuye el coste de los activos entre los años de vida útil de los mismos de forma constante. Asimismo, dichos recursos están ubicados en la categoría de "Útiles, herramientas y equipos para tratamiento de la información, sistemas y programas informáticos" de las tablas de amortización propias del I.R.P.F.

La amortización se practicará elemento por elemento, si bien cuando se trate de elementos patrimoniales integrados en el mismo grupo de la tabla de amortización, la amortización podrá practicarse sobre el conjunto de ellos, siempre que en todo momento pueda conocerse la parte de la amortización correspondiente a cada elemento patrimonial.

Asimismo, los elementos de inmovilizado material nuevos cuyo valor unitario no exceda de 601,01 euros, podrán amortizarse libremente, hasta el límite de 3.005,06 euros anuales.

Según la tabla "*Útiles, herramientas y equipos para tratamiento de la información, sistemas y programas informáticos*", el número de años mínimos y máximos de amortización en esta categoría es de entre 2.5 y 5 años, por lo que, para este proyecto, se ha estipulado en 3 años para cada uno de los elementos de tipo hardware empleados.

Teniendo en cuenta que la duración del proyecto ha sido aproximadamente de 24 meses, y sabiendo que el cálculo del coste de amortización se constituye en un periodo de 3 años, los costes de amortización de la mayoría de los recursos utilizados se calcularán para el primer y segundo año.

Finalmente, la cuota de amortización anual será calculada haciendo uso de la siguiente ecuación:

$$Cuota\ de\ amortización\ anual = \frac{Valor\ de\ adquisición - Valor\ residual}{Nº\ de\ años\ de\ vida\ útil}$$

Donde el valor residual es el valor de cada uno de los elementos en cuestión después de su vida útil, teniendo en cuenta los índices de depreciación actual.

Amortización del material hardware

Puesto que la elaboración del proyecto ha precisado de 24 meses de trabajo y el cálculo del coste de amortización se estipula en un periodo de tres años, los costes serán calculados como los derivados del tiempo de utilización que se ha requerido por cada uno de los elementos hardware.

En la siguiente tabla se muestra la relación del elemento hardware con su valor de adquisición, su valor residual y el coste de amortización finalmente obtenido.

Elementos Hardware	Coste	Valor residual (3 años)	Amortización (24 meses)
Ordenador de sobremesa: CPU AMD FX-8350, GTX960, 8 GB RAM, HDD 1TB, MOBO GA-970A DS3P, 1 Monitor LED	660€	200€	306€
Total	**660€**	**200€**	**306€**

Por tanto, el coste total del hardware empleado en el proyecto asciende a la cantidad de TRESCIENTOS SEIS EUROS.

Amortización del material software

De forma análoga, en este apartado se procederá a calcular la amortización del inmovilizado material de tipo software del proyecto.

Elementos Software	Coste	Valor residual (3 años)	Amortización (24 meses)
Microsoft Office 356	149€	0€	99,33€
Microsoft Windows 10 Pro, 64 bits	259€	0€	172,66€
Sublime Text	0€	0€	0€
XAMPP	0€	0€	0€
Wordpress	0€	0€	0€
Google Chrome	0€	0€	0€
Total	**408€**	**0€**	**272€**

Por tanto, el coste total del hardware empleado en el proyecto asciende a la cantidad de DOSCIENTOS SETENTA Y DOS EUROS.

Trabajo tarifado por tiempo empleado

Normas de valoración

La aproximación del importe de las horas empleadas en la realización del presente proyecto con respecto a los honorarios finales estimados está indicada en la fórmula de recomendación del COIT mostrada en la siguiente ecuación:

$$H = 74{,}88 * H_N * C_T + 96{,}72 * H_E * C_T \text{ euros}$$

En donde:

- H son los honorarios por tiempo.
- HN son las horas trabajadas dentro de la jornada laboral.
- HE son las horas especiales trabajadas.
- CT es un factor de corrección en función del número de horas trabajadas.

Asimismo, el valor del factor de correlación según el número de horas trabajadas vendrá dado por la siguiente tabla:

HORAS TRABAJADAS	**FACTOR DE CORRECCIÓN C_T**
Hasta 36 horas	1,00
De 36 a 72 horas	0,90
De 72 a 108 horas	0,80
De 108 a 144 horas	0,70
De 144 a 180 horas	0,65
De 180 a 360 horas	0,60
De 360 a 540 horas	0,55
De 540 a 720 horas	0,50
De 720 a 1080 horas	0,45
Más de 1080 horas	0,40

En el cálculo del tiempo dedicado a la realización del presente proyecto se debe considerar que la dedicación no ha sido completa. Además, se deben descontar los periodos festivos o donde se tuvo que paralizar el trabajo, con una duración total estimada de 6 meses.

Con esto en consideración, el total ha sido de 1800 horas (5 horas diarias * 5 días a la semana * 4 semanas al mes * 18 meses) normales y 0 especiales, al que se le aplica un factor de corrección de 0,40.

$$H = 74{,}88 * 1800 * 0{,}40 = 53.913{,}60 \text{ euros}$$

La tarifa final por tiempo de ejecución es de CINCUENTA Y TRES MIL NOVECIENTOS TRECE EUROS CON SESENTA CÉNTIMOS.

Derechos de visado

El COIT establece que para la redacción de proyectos y trabajos en general, los derechos de visado se calculan de acuerdo con la siguiente expresión:

$$V = 0{,}006 * P * C$$

Donde P representa el presupuesto total y C es el coeficiente reductor en función de dicho presupuesto.

El presupuesto total se obtiene de la suma de las secciones anteriores correspondientes al trabajo tarifado por tiempo empleado, la amortización del inmovilizado material y la redacción del proyecto, como se observa en la siguiente tabla:

Concepto	**Coste**
Amortización de recursos hardware	306€
Amortización de recursos software	272€
Trabajo tarificado por tiempo empleado	53.913,60€
Total	**54.491,60€**

En función del presupuesto obtenido, se extrae el valor del coeficiente reductor del presupuesto C, que, según el COIT, para presupuestos de más de 30.050€ y menos de 90.150€ viene definido con un valor de 0,8. Por tanto:

$$V = 0{,}006 * 54.491{,}60 * 0{,}8 = \ 261{,}56 \text{ euros}$$

Finalmente, los costes por derecho de visado del proyecto ascienden a DOSCIENTOS SESENTA Y UN EUROS CON CINCUENTA Y SEIS CÉNTIMOS.

Gastos de tramitación y envío

Los gastos de tramitación y envío son fijos y se estipulan por el COIT en 6,01€ por cada documento en un visado digital.

Por ello, los gastos de tramitación y envío ascienden a SEIS EUROS Y UN CÉNTIMO.

Presupuesto antes y después de impuestos

Sumando todos los conceptos calculados hasta el momento, se obtiene el total del presupuesto previo a la aplicación de impuestos, como se muestra a continuación en la siguiente tabla.

CONCEPTO	COSTE
Amortización de recursos hardware	306€
Amortización de recursos software	272€
Trabajo tarificado por tiempo empleado	53.913,60€
Derechos de visado del COIT	261,56€
Gastos de tramitación y envío	6,01€
Total sin impuestos	**54.759,17€**
Aplicación de impuestos (7% IGIC)	3.833,14€
Total con impuestos	**58.592,31€**

El presupuesto calculado antes de impuestos asciende a CINCUENTA Y CUATRO MIL SETECIENTOS CINCUENTA Y NUEVE EUROS CON DIECISIETE CÉNTIMOS, y el presupuesto después de impuestos es un total de CINCUENTA Y OCHO MIL QUINIENTOS NOVENTA Y DOS EUROS CON DIECISIETE CÉNTIMOS.

Las Palmas de Gran Canaria, a 03 de Agosto de 2018

Fdo: Borja García González

Anexo

Contenido del DVD

Junto con esta memoria se adjunta un DVD en el que se recopila el trabajo realizado a lo largo de este Proyecto Fin de Carrera. El contenido de este DVD es el siguiente:

- Memoria en formato PDF.
- Plugin realizado para Wordpress.
- Instalador de Wordpress y tema utilizado.

Estructura y descripción del contenido

Se ha organizado el contenido en los siguientes directorios: *Memoria*, *Plugin* y *Wordpress*.

- **Memoria**: contiene la versión digital del presente documento, incluyendo memoria, bibliografía, pliego de condiciones y anexos.
- **Plugin**: contiene el directorio *iac* que debe ser copiado en el directorio *wp-content/plugins* de Wordpress.
- **Wordpress**: contiene el instalador con la versión utilizada de Wordpress (4.2.9_es-ES) y el tema recomendado para la correcta visualización del plugin, que debe ser copiado en el directorio *wp-content/themes* de Wordpress.

Plantilla para Producto PDF

```
<div class="row row-centered row-m-tb">
  <!-- View area -->
  <div class="col-sm-12 col-centered">
    <div class="col-sm-12">
      <div class="input-group-btn text-right">
        <button class="btn btn-default" title="Información">
          <i class="glyphicon glyphicon-info-sign"></i>
        </button>
        <a class="btn btn-default"
        ng-href="{{product.params.resource}}" target="_blank"
        alt="{{product.params.action}}">
          <i class="glyphicon glyphicon-download-alt"></i>
        </a>
        <button class="btn btn-default"
        ng-show="product.params.subtype=\'shareable\'"
        title="Compartir">
          <i class="glyphicon glyphicon-share"></i>
        </button>
      </div>
    </div>
    <div class="row">
      <div class="col-sm-12">
        <img ng-src="{{product.image_url}}"
        alt="Imagen de {{product.name}}" align="middle"
        style="max-width: 300px">
      </div>
    </div>
    <div class="row">
      <div class="col-sm-12">
        <h3>{{product.name}}</h3>
        <div class="row">
          <div class="col-sm-12 text-justify">
            {{product.description}}
          </div>
        </div>
      </div>
    </div>
  </div>
  <!-- Edit area -->
  <div class="col-sm-12 col-centered">
    <div class="row" uib-collapse="!PC.editMode">
      <div class="col-sm-12">
        <form>
          <div class="form-group">
            <label>   Nombre:
              <input type="text" class="form-control"
              ng-model="PC.editProduct.name"
              placeholder="Nombre del producto" required />
            </label>
          </div>
          <div class="form-group">
            <label>   Imagen:
              <input type="text" class="form-control"
              ng-model="PC.editProduct.image_url" placeholder="http://..." />
            </label>
          </div>
          <div class="form-group">
            <label>   Contenido:
              <textarea class="form-control"
              ng-model="PC.editProduct.description" rows="3">
```

```
              </textarea>
            </label>
          </div>
          <div class="form-group">
            <label>   Dirección del PDF:
              <input type="text" class="form-control"
              ng-model="PC.editProduct.params.resource"
              placeholder="Sin http://" />
            </label>
          </div>
          <div class="form-group">
            <label>   Texto del botón de descarga:
              <input type="text" class="form-control"
              ng-model="PC.editProduct.params.action"
              placeholder="Texto del botón" />
            </label>
          </div>
        </form>
      </div>
    </div>
  </div>
</div>

```

Ejemplo de Struct

```
[
  {
    "category": true,
    "id": "@@hook",
    "name": "@@text",
    "description": "",
    "image_url": "******0",
    "parent": "@@nodeParent",
    "params": "@@categoryParams",
    "ordering": "@@ordering",
    "published": true,
    "struct": [
      {
        "category": false,
        "id": "@@id",
        "name": "metadato",
        "description": "@@description",
        "image_url": "@@image_url",
        "category_id": "@@structParent",
        "params": "@@productParams",
        "ordering": 1,
        "published": true
      }
    ]
  }
]
```