UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

PROYECTO FIN DE CARRERA

HERRAMIENTA PARA EL PROTOTIPADO DE APLICACIONES MÓVILES USANDO MOCKUPS

Autor: D. Mario Bañares Colastra
Tutores: Dr. D. José María Quinteiro González
Dr. D. Luis Hernández Acosta
Fecha: Noviembre 2016

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

PROYECTO FIN DE CARRERA

HERRAMIENTA PARA EL PROTOTIPADO DE APLICACIONES MOVILES USANDO MOCKUPS

HOJA DE FIRMAS

Alumno/a

Fdo.: Mario Bañares Colastra

Tutor/a

Fdo.: José María Quinteiro González

Tutor/a

Fdo.: Luis Hernández Acosta

Fecha: Noviembre 2016

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

PROYECTO FIN DE CARRERA

HERRAMIENTA PARA EL PROTOTYPADO DE APLICACIONES MOVILES USANDO MOCKUP

HOJA DE EVALUACIÓN

Calificación: _____

Presidente

Fdo.:

Vocal **Secretario/a**

Fdo.: Fdo.:

Fecha:

TABLA DE CONTENIDO

BLOQUE PRIMERO. MEMORIA

Agradecimientos

Quiero agradecer en primer lugar a mis padres, por su paciencia y dedicación durante todos estos años. A mi tía Marina y mi tío Rafael, por alojarme durante un año en su casa y tratarme como a un hijo más. A mi tia Peque, siempre esta cuando la necesito. A mi abuela, por su ayudita siempre que hizo falta.

A mis tutores, por toda la ayuda y dedicación del proyecto, que ha sido incluso más difícil realizándolo desde el extranjero. Por siempre liberar el calendario cada vez que visitaba la isla y sobre todo por responder a los mensajes instantáneos siempre, y solventando las dudas de las maneras más complicadas.

A mis amigos y allegados, a los que viven en Canarias, o en cualquier parte del mundo. A los que compartimos Helsinki. Gracias.

And special thanks for you. Thanks for being there always. Without you it would have been impossible. This is only thanks to you. Ačiū, ačiū labai.

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

BLOQUE PRIMERO

MEMORIA

1. Introducción

1.1. Introducción

Cada día en España, la utilización por parte de usuarios de los denominados móviles inteligentes, llamados comúnmente Smartphone en su habla inglés, es mayor. Tanto es así que actualmente 8 de cada 10 adultos españoles posee un Smartphone y 4 de cada 10 una Tablet o tableta en español [1]. Estos dispositivos, no solo facilitan el acceso a internet, sino que también son una herramienta muy potente para que las empresas lleguen a posibles consumidores. Es por esto que cada día aumenta el número de aplicaciones disponibles en las principales tiendas de descarga, como Google Play o App Store.

Sin lugar a dudas, esta explosión de las aplicaciones móviles ha conllevado que numerosas empresas de desarrollo vean como este nicho de mercado puede ser explotado, no solo por todos los desarrolladores que poseen una gran experiencia en la creación de aplicaciones, sino por todos los usuarios que poseen grandes ideas de aplicaciones móviles que pueden llevar a conseguir numerosas descargas.

Entre los actores que buscan o necesitan desarrollar aplicaciones móviles, podemos destacar algunos, los cuales son más próximos al objetivo de este proyecto.

En un lado se encuentran las empresas que quieren tener una mayor penetración en el mercado y buscan a los clientes / usuarios en el mundo de las aplicaciones móviles, o buscan expandir sus servicios y el valor añadido de sus productos mediante la salida a este mercado. Como demuestra el estudio realizado por Accenture sobre Inversión en Marketing y Publicidad Móvil [2], la tendencia de las empresas está en el desarrollo de aplicaciones móviles para aumentar su presencia y llegar a más usuarios. Para el correcto desarrollo de la aplicación móvil, estas empresas tienen dos vías principales para el desarrollo.

La primera es la de contratar desarrolladores e incorporarlos a la dinámica de la empresa, formando parte de ella. La segunda, contratar un tercero que pueda proveer los servicios de desarrollo y soporte de aplicaciones móviles.

Por otro lado, nos encontramos con los usuarios con un nivel bajo de conocimientos de lenguajes de programación, diseño de aplicaciones, etc. En general estos usuarios suelen tener ideas que podrían generar buenas aplicaciones móviles, pero se enfrentan al reto de aprender estos lenguajes o a contratar los servicios de empresas que, por lo general, tienen unos honorarios demasiado elevados para estos usuarios.

En estos dos casos, el tiempo que transcurre desde que la persona o empresa tiene la idea y decide dar el siguiente paso, ya sea contratar una empresa o desarrollarlos por ellos mismos, es elevado. Esto se hace incluso aún más difícil cuando no solo hace falta programar en un lenguaje de programación, dado que plataformas como iOS, Android o Windows Phone, usan diferentes lenguajes de programación.

La realización de este proyecto no pretende solucionar todos estos problemas, sino mitigarlos y tendrá como consecuencias la creación de una aplicación móvil que contará dos funciones principales:

- Ayudar a los usuarios de nivel bajo – medio a poder realizar aplicaciones móviles básicas, a través de herramientas de diseño gráficas y sin necesidad de conocer ningún lenguaje de programación, pudiendo obtener las aplicaciones móviles funcionales al menos en dos sistemas operativos, iOS y Android.
- Dar un nuevo soporte de análisis de datos y de compresión visual a los desarrolladores y analistas para facilitar y acortar los pasos iniciales y el *"time to market"*.

1.2. Objetivos

Los principales objetivos perseguidos y que han motivado la realización de este Proyecto Final de Carrera (PFC) han sido los siguientes:

- El aprendizaje de nuevos lenguajes de programación enfocados en el desarrollo de aplicaciones en entornos Apple, tanto aplicaciones de escritorio como aplicaciones móviles.
- El inicio en técnicas de desarrollo como el modelo – vista – controlador u otros patrones de diseño.

- El estudio de tendencias actuales empresariales en el desarrollo de aplicaciones y sus procesos internos.
- Introducirse en el desarrollo de aplicaciones utilizando Software Kit Developments (SDK) y Frameworks.

1.3. Estructura de la memoria

La memoria del PFC se encuentra divida en tres secciones principales: memoria, pliego de condiciones y presupuesto.

El primero de ellos, la memoria, se encuentra dividido en 11 capítulos.

El segundo bloque, el pliego de condiciones se divide en dos secciones:

- El pliego de condiciones donde se detallan los lenguajes de programación y los programas utilizados durante el desarrollo del PFC. Así como los marcos de desarrollo utilizados
- Una breve guía para los usuarios.

En un tercer bloque se encuentra el presupuesto, en el que se detallan las fases en las que se divide el proyecto, la duración de cada una de ellas y los costes necesarios para su implementación.

Por último se incluyen los anexos, en los que están todas las tablas de ilustraciones, tablas y código.

2. Estado del arte

En este apartado se realiza un breve repaso a todas las aplicaciones o herramientas existentes en el mercado para la realización de aplicaciones móviles sin conocimientos de programación o con un conocimiento bajo/medio. De la misma forma, se presentaran aquellas herramientas que son utilizadas en la elaboración de mock-up. De esta forma, se trata de cubrir las dos fuentes que se quieren combinar en el proyecto.

Por otro lado, se hará un pequeño recorrido por las plataformas más elegidas para el desarrollo de aplicaciones móviles y su alcance en el mercado.

2.1. Herramientas para el diseño de aplicaciones móviles online

Actualmente el mercado está lleno de aplicaciones y herramientas que permiten y prometen a los usuarios crear sus propias aplicaciones móviles sin necesidad de tener conocimientos de programación o de requerir un nivel medio de conocimientos. El mercado se divide principalmente en varios segmentos, entre los que destacan sobre todo las aplicaciones por suscripción, en la que los usuarios generan sus aplicaciones y pagan suscripciones mensuales o anuales para mantener su aplicación online. En este apartado se presentan algunas de ellas.

2.1.1. Adiante apps

Esta herramienta proporciona a los usuarios distintos niveles para el diseño de la aplicación basado en módulos. Estos módulos pueden ser de dos tipos, view controllers o motores. La principal diferencia es que en los de tipo view controllers es el usuario el que rellena la información del view controllers y los motores son los que unen diferentes view controllers y permiten realizar conexiones entre estos o enlaces externos [3]. En total cuentan con 22 módulos.

Ilustración 1 - Ilustración de Adiante Apps

Uno de los puntos más fuertes que tiene esta herramienta es que cuentan con aplicaciones plantilla para distintos sectores, inmobiliario, hostelería, etc. Por otro lado, cuentan en las principales tiendas de aplicaciones móviles con un visualizador de aplicaciones en la que los clientes pueden ver un borrador de su aplicación. Entre los aspectos negativos es su modularidad y diferencia de precios entre ellos. Del mismo modo, el pago de la aplicación es mediante pagos anuales y tiene coste de mantenimiento.

2.1.2. Aplicarium

Aplicarium trata también de usar la modularidad como herramienta para que los usuarios construyan su aplicación. Cuentan también con distintos módulos, pero en este caso todos ellos son de tipo view controller. Es decir, solo pueden rellenar la información de la aplicación, pero las conexiones son fijas. Esta herramienta también tiene una tarifación anual y se puede gestionar mediante el navegador web.

Entre los puntos en los que menos destaca, sería la menor cantidad de módulos que permite y las limitaciones a la hora de poder cambiar y personalizar una vez la aplicación ha sido publicada en la tienda.

2.1.3. App Press

Esta herramienta tiene un enfoque totalmente diferente a los anteriores. En este caso, el usuario posee una herramienta muy similar a Photoshop. El usuario no tiene módulos sino "bienes" que puede ir colocando en la interfaz gráfica y puede ir dando contenido a cada pantalla. En este caso la herramienta está más enfocada al diseño que a la

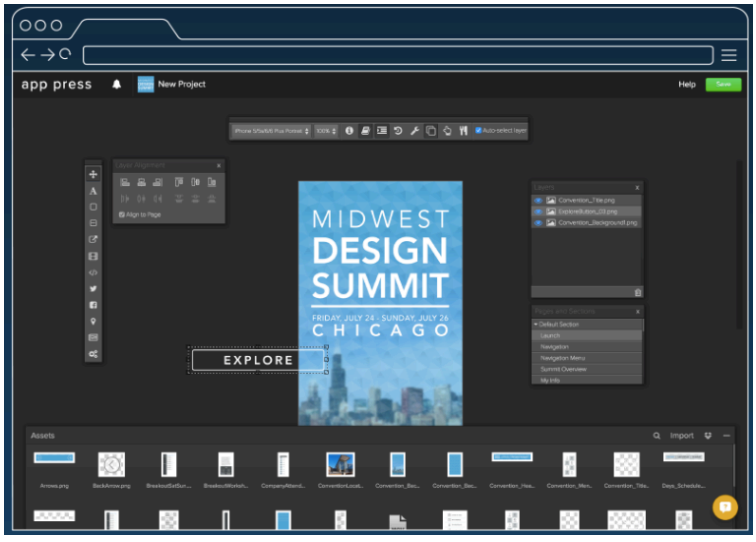


Ilustración 2 - Ilustración de App Press

funcionalidad. Cada aplicación será diferente de la otra debido a que cada diseñador puede optar por colocar los bienes en distintas posiciones.

El mayor problema que presenta es el tiempo que hay que dedicar para diseñar la aplicación, pero por otro lado, proporciona suficiente libertad para que cada cliente pueda diseñar la aplicación y no solo darle el contenido.

2.1.4. King of App

Esta herramienta es totalmente distinta. Los creadores se han basado en proporcionar al usuario el código de la aplicación base y es el usuario el que teniendo un conocimiento limitado proporciona el contenido o modifica el código.

La principal desventaja de usar esta herramienta es que el usuario ha de saber que está haciendo y puede eliminar código que resulte de vital importancia para la aplicación. Por otro lado, si se tienen unos mínimos de conocimientos puede ser una aplicación muy interesante, dado que los desarrolladores tendrían una plantilla de la que pueden empezar para programar sus aplicaciones, pero normalmente, todos los desarrolladores ya cuentan con sus plantillas, por lo que puede ser un enfoque equivocado.

Por ultimo cabe destacar, que esta empresa es beneficiaria de una subvención europea bajo el marco Horizon 2020. Para más información sobre esta subvención se puede consultar la siguiente página web:

https://ec.europa.eu/research/participants/portal/doc/call/h2020/smeinst-13-2016-2017/1710102-smeiphase2-beneficiaries-feb_2016_en.pdf

2.2. Herramientas para el diseño de mock-up de aplicaciones móviles

Por el otro lado, haremos un análisis de las aplicaciones disponibles para la realización de mock-up. Del análisis de estas aplicaciones, escogeremos la que mejor se adapte a las características del sistema que queremos desarrollar. En este apartado se representará una selección de las aplicaciones que se analizaron.

2.2.1. Balsamiq Mockups

Esta aplicación para desarrollar mock-up es la más popular, apareciendo en las búsquedas de los buscadores online en las primeras posiciones y siempre en múltiples ocasiones. La principal característica de esa aplicación es la sencillez de uso. Los elementos que se pueden usar parecen estar dibujados a mano y se puede usar la función de "drag and drop" para agarrar los elementos y arrastrarlos hasta la posición deseada. La biblioteca de elementos que posee es muy amplia, conteniendo elementos que pueden ser reutilizados tanto en aplicaciones móviles como en aplicaciones de escritorio o incluso páginas web.

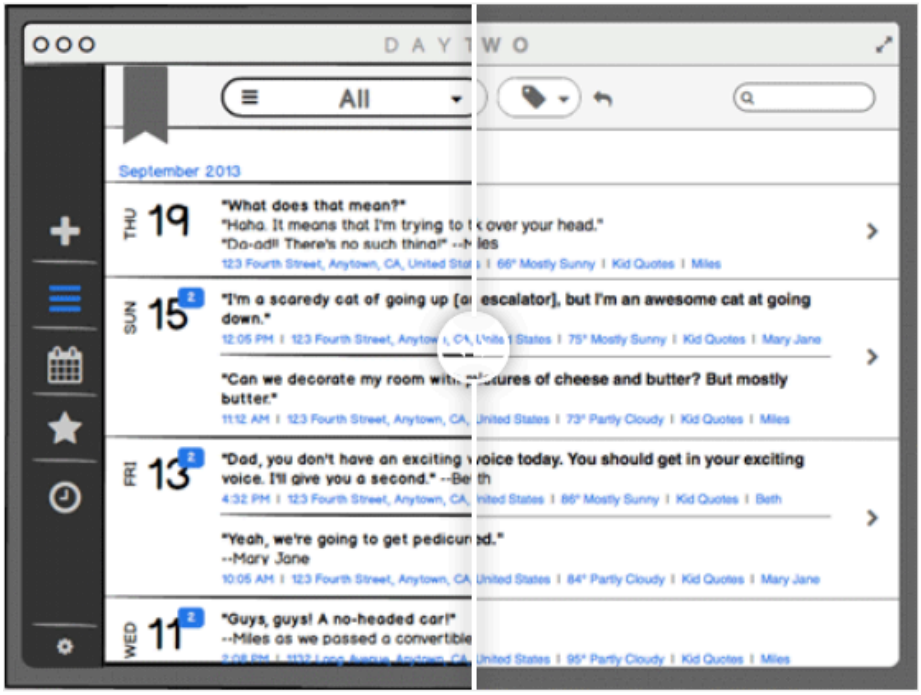


Ilustración 3 - Ilustración de Balsamiq Mockup

Quizás la funcionalidad más importante que posee Balsamiq es que puede exportar los diseños en formato .XML o .BMML. Esta funcionalidad está pensada para que los usuarios puedan compartir sus diseños o usarlos en otros ordenadores. Para el desarrollo del proyecto, esta funcionalidad puede ser crítica.

Además, Balsamiq Mockups está disponible no solamente en versión de escritorio sino también en versión online, por lo que se puede trabajar desde cualquier ordenador y salvar los diseños en la nube. La versión online es gratuita para universitarios y estudiantes,

mientras que para la versión multiplataforma de escritorio es necesario pagar una cuota mensual.

2.2.2. Mockingbird

Esta herramienta se usa para el prototipado de aplicaciones o páginas web y se usa a través de un navegador web, guardando todos los contenidos en la nube. La principal característica de la herramienta es el énfasis que se hace para enlazar y compartir los prototipos pudiendo controlar el flujo que los usuarios hacen y poder de esa manera probar la viabilidad de la aplicación y su uso. Del mismo modo, los posibles clientes pueden validar el trabajo realizado.

Otra característica que comparte con Balsamiq es que se pueden crear equipos de trabajo y trabajar de forma simultánea en los diseños, aunque su principal problema es la biblioteca de elementos con los que se puede trabajar, menor que en otros. Aunque sin lugar a dudas el principal problema es que se trata de una aplicación de pago, en el que el periodo de prueba dura solo 6 días.

2.2.3. Mockup Builder

Otro problema para el uso de esta aplicación es su suscripción de pago. Solo se puede desarrollar un proyecto con el plan de pago más pequeño. Aunque esta aplicación dispone de una de las bibliotecas más grandes, es un gran inconveniente que no dispone de una versión lite de uso gratuita. Además, el número de desarrolladores simultáneos y de personas con las que compartir también depende del plan de pago. La versión de prueba dura, como en Mockingbird, tan solo unos días.

2.2.4. Gliffy

Aunque Gliffy se centra en la creación universal de diagramas, también es usada por muchos usuarios para la creación de mock-up de aplicaciones o páginas web. Además parte de su uso es gratuito, por lo que lo convierte en una herramienta que puede ser usada por cualquier persona.

Gliffy se usa desde el navegador web y contiene una librería de elementos bastante mayor, pero dichos elementos son más generales y simples que los que poseen otras herramientas para el diseño de mock-ups, aunque permite la inserción de imágenes y diseños por parte del usuario, por lo que hace que sea una librería dinámica más adaptable al usuario que las demás plataformas.

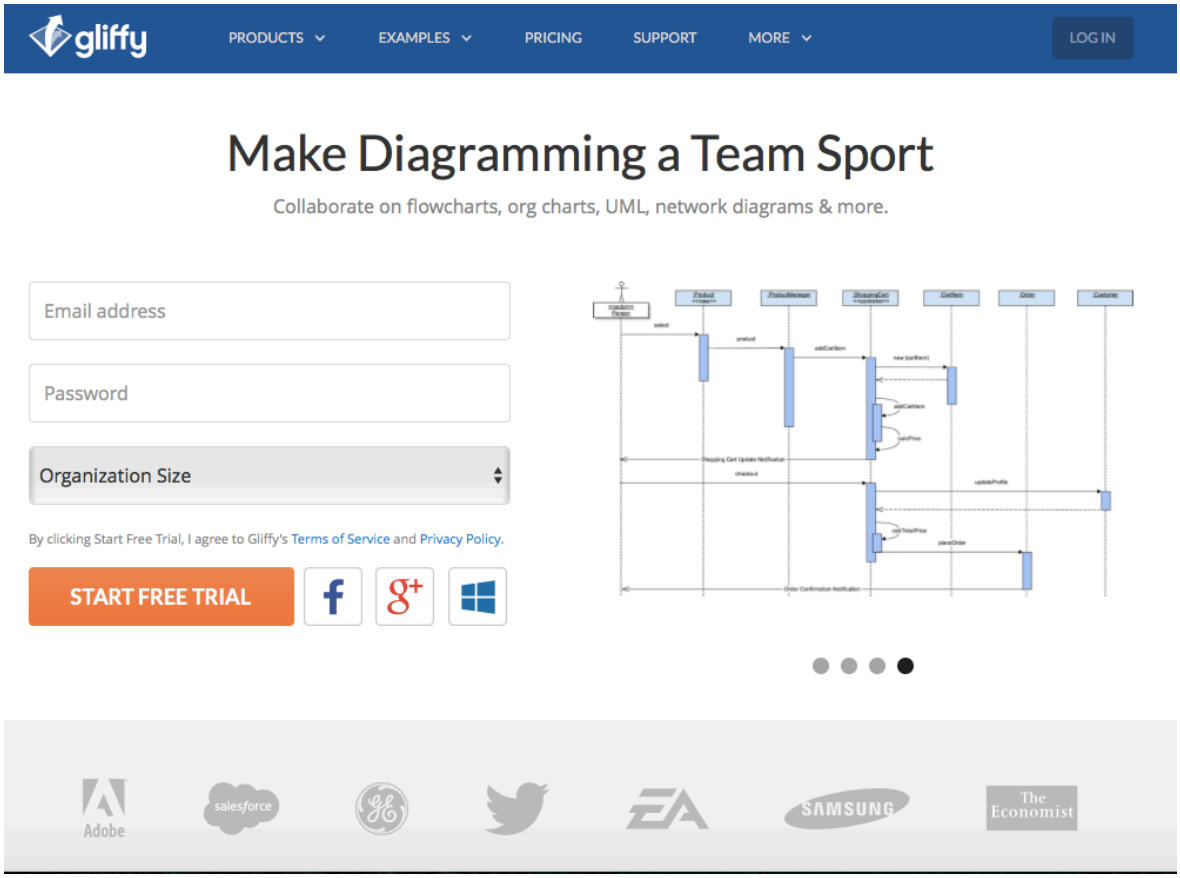


Ilustración 4 - Ilustración de Gliffy

Esta se trata, sin lugar a dudas, de la plataforma más universal, permitiendo hacer pequeños diseños arquitectónicos como UML, aunque la estructura de la interfaz de usuario se parece más a la de cualquier software de creación de diagramas. Además, Gliffy puede ser integrada con multitud de plataformas y está disponible para las herramientas de control de software y de Agile más comúnmente usadas como Jira, Confluence o GitHub.

El mayor problema que presenta Gliffy para los usuarios es su periodo de prueba y el no disponer de un plan gratuito. Gliffy pasa a ser de pago a los 30 días naturales. El principal problema de usar la versión gratuita de Gliffy es que todos los diseños que se realizan pasan

a una biblioteca pública, donde todo el mundo puede ver los diseños, por lo que para desarrolladores semi-profesionales no es la mejor opción.

2.3. Android, iOS y otros

Actualmente, el mercado de los Smartphone lo dominan, en términos de sistemas operativos, Android e iOS. Android es claramente el primer sistema operativo, dominando con poco más de un 82% el total de cuota de mercado, mientras que iOS tiene casi un 14%. El resto de los sistemas operativos abarcan el resto del mercado, 4% [4].

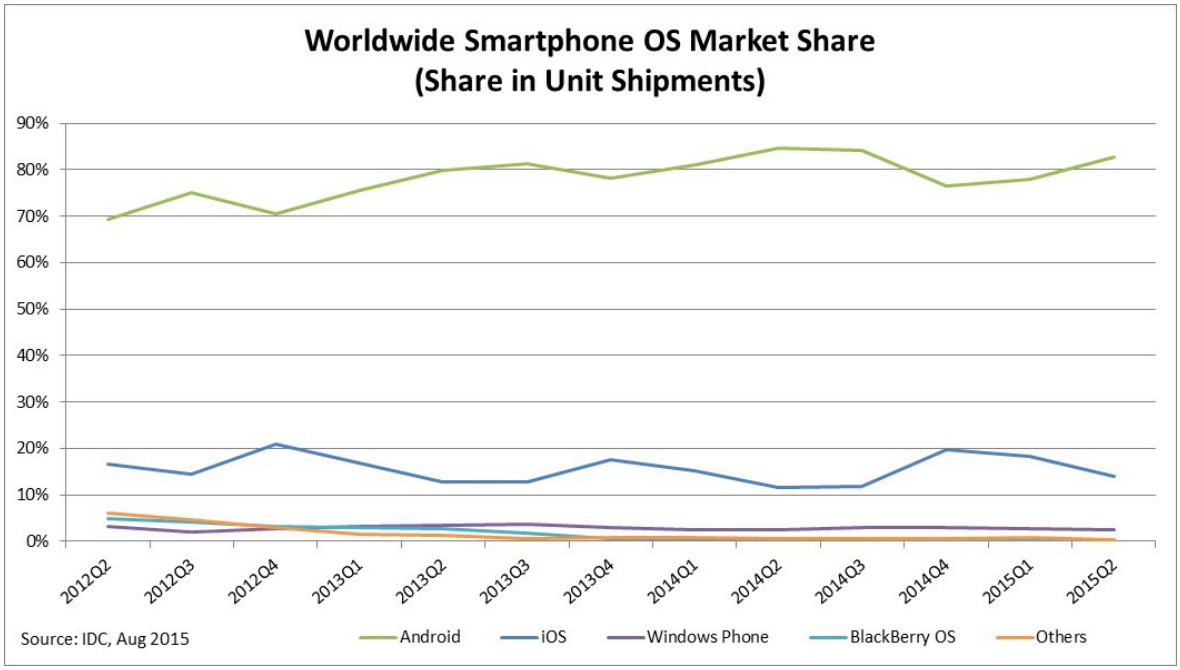


Ilustración 5 - Distribución del uso de Sistemas Operativos en Smartphones

El que Android posea una cuota de mercado tan importante es debido a que la mayoría de los fabricantes de hardware usan este sistema operativo. Según un informe publicado por Open Signal en 2014, casi 19.000 modelos diferentes de teléfonos inteligentes usan Android como sistema operativo. Esto se debe principalmente a que Samsung, es el fabricante del 43% del total de terminales que usan Android [5].

Estos datos demuestran que estos dos sistemas operativos son los que los usuarios van a demandar en sus aplicaciones móviles, siendo Android la clara primera elección. Por otro lado, los desarrolladores de aplicaciones móviles lo tienen más sencillo para programar en Android, tratándose de un sistema operativo más abierto que iOS.

3. Arquitectura del Sistema

3.1. Introducción

El proyecto final de carrera "*Herramienta para el prototipado de aplicaciones móviles usando mock-up's*" tiene como propósito facilitar a los usuarios sin nociones de programación, el diseño de su propia aplicación móvil a través de una interfaz gráfica de fácil uso. Una vez la aplicación está diseñada, estos usuarios pueden tener de forma instantánea su aplicación móvil para la plataforma que el usuario desee, tanto iOS como Android.

De la misma manera, las distintas herramientas que se han generado para conseguir este objetivo, pueden ser usadas de manera independiente, ampliando las funcionalidades y añadiendo un mayor valor añadido.

Este proceso para el usuario estará basado en varias fases:

1. Diseño gráfico de la aplicación móvil. El usuario deberá hacer uso de una interfaz de diseño de mock-up gratuita preseleccionada. El usuario podrá abrir las diferentes plantillas de view controllers que ya están diseñadas y personalizarlas como desee. En ese punto, el usuario deberá salvar y exportar todos los datos usando el manual de usuario y almacenarlas en la plataforma de almacenamiento de datos "Dropbox".
2. Análisis de datos. El usuario podrá ahora usar la aplicación móvil diseñada para seleccionar el directorio donde ha almacenado los datos procedentes de las plantillas y seleccionar a que destino final desea enviarlas. El usuario puede elegir entre Parse o Xmind. Si el usuario desea incluirlo en Parse, obtendrá en la aplicación que se le proporcionará, una aplicación totalmente funcional si el analizador de datos no ha detectado ningún fallo en los datos. Si por otra parte, el usuario ha decidido obtener los datos en Xmind, este obtendrá un archivo .mm que podrá importar en Xmind y obtener el árbol jerárquico de la aplicación.

Del mismo modo, los usuarios pueden revertir el proceso y realizar modificaciones tanto en Parse como en Xmind y pasar estos archivos por el proceso y obtener, tanto desde Parse

el Xmind o los mock-up de la aplicación o desde Xmind se puede obtener el Parse y el mock-up de la misma. Es por esto que el sistema es bidireccional.

En la siguiente figura, podemos ver una representación del sistema que se pretende diseñar.

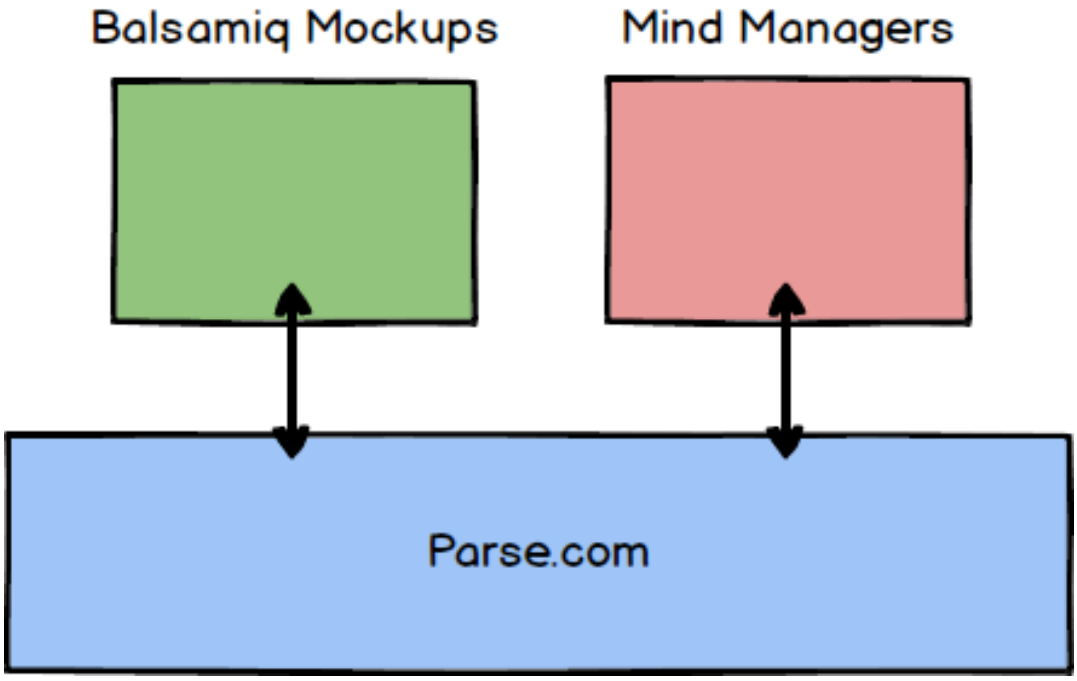


Figura 1 - Diagrama de Bloques del Sistema que se pretende diseñar

Con el diseño de este sistema se busca que sea bidireccional en todos sus ramales y que además las ramas sean independientes unas de otras, es decir que se pueda alcanzar cualquiera de los extremos de punto a punto.

3.2. Arquitectura del sistema existente

La arquitectura de este proyecto está basada en la existencia de un sistema diseñado por el IUMA, el cual contiene todas las herramientas necesarias para la creación de aplicaciones móviles. Actualmente, ese sistema tiene la posibilidad de que a partir de la plataforma online Parse puede, puede procesar los datos almacenados en forma de tabla y convertirlos en una aplicación móvil gracias al uso del framework del IUMA En la figura 2 se representa el diagrama de flujo actual del sistema.

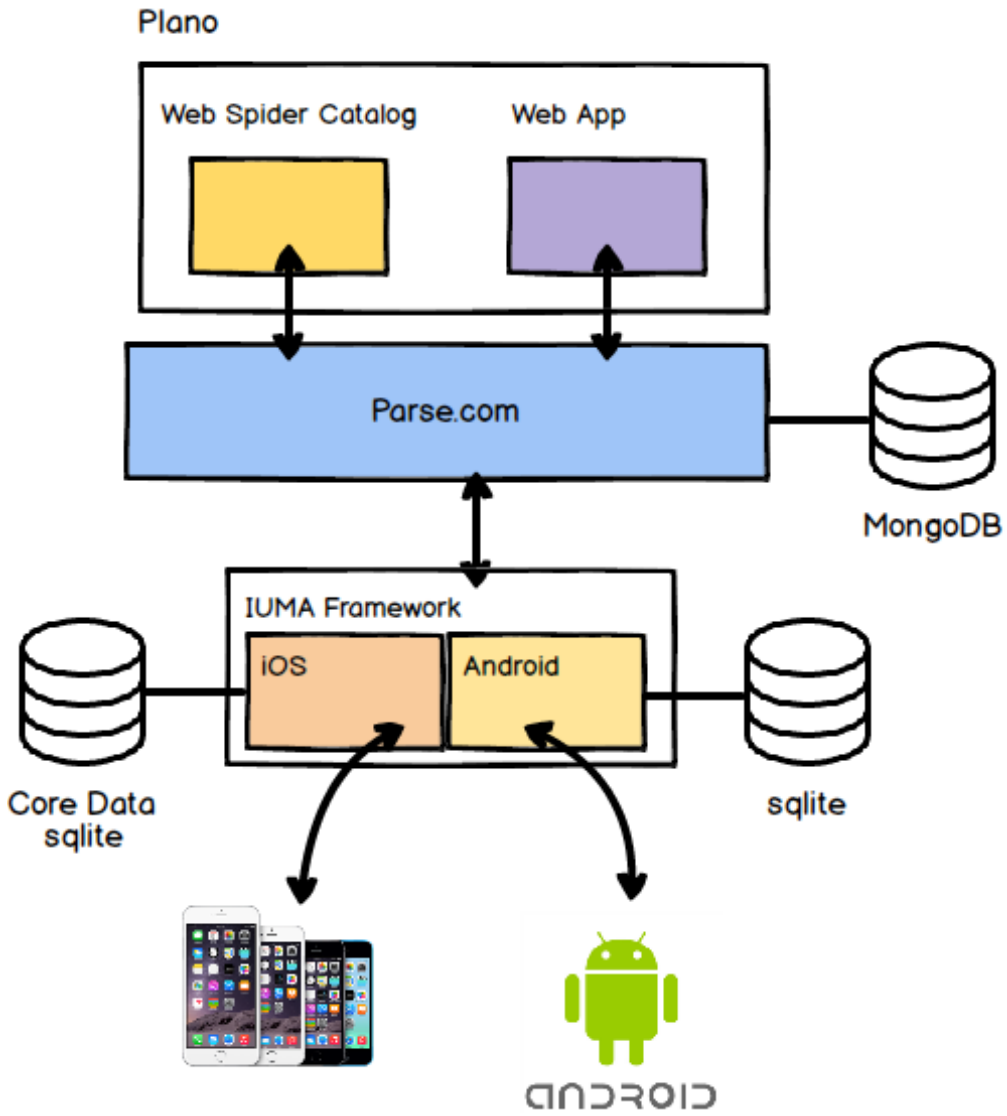


Figura 2 - Diagrama de Bloques del Sistema Existente

Como se observa, desde distintos tipos de web, uno desde Wordpress y otro desde una web app, se puede alimentar actualmente a Parse.com o en realidad a la base de datos MongoDB sobre la que está basado Parse.

La principal característica que posee el framework diseñado por el IUMA es que es capaz de interpretar los datos almacenados en Parse para dar contenido a diferentes view controllers previamente diseñados. De esta forma, ya solo con lanzar el framework e indicar las claves de acceso de Parse, los usuarios podrán obtener la aplicación móvil. El framework del IUMA ayuda que las aplicaciones estén disponibles tanto para iOS como para Android. La forma de interpretar los datos es la misma para ambos sistemas, tratándose solo de un canal que une los datos con la representación gráfica de los mismos. Es más, realmente no es solo un framework, sino que existen dos frameworks diseñados

por diferentes personas, que dan como resultado una misma aplicación, pero para diferentes sistemas operativos móviles. Por tanto, aunque la forma de tratar los datos es distinta, tanto al comienzo como al final del proceso los resultados son los mismos.

La forma y contenido que han de tener los datos en Parse, se detallará más adelante en el capítulo 5. Este punto es crucial, dado que la forma en la que se representan los datos en la aplicación depende de cómo los datos están almacenados en Parse y es por eso que los demás puntos del sistema han de estar coordinados.

Actualmente, Parse se puede alimentar usando Wordpress. Este sistema se optimizó para trabajar con un plugin llamado SpiderCatalog. Este plugin es el que da la forma a Parse. Parse se debe completar de la misma forma en la que SpiderCatalog está diseñado.

3.3. Componentes del sistema

El sistema inicial se plantea con la idea de conectar dos puntos: Mockup de las aplicaciones con la aplicación, es decir, Balsamiq con la aplicación móvil. Más tarde durante el desarrollo del proyecto, se plantea poder incorporar otros componentes que resultan interesantes para analizar la información y poder dotar al proyecto de un cuerpo más sólido. De esta forma pasaron a formar parte de componentes del sistema los gestores de mapas / árboles, más conocidos como mind managers.

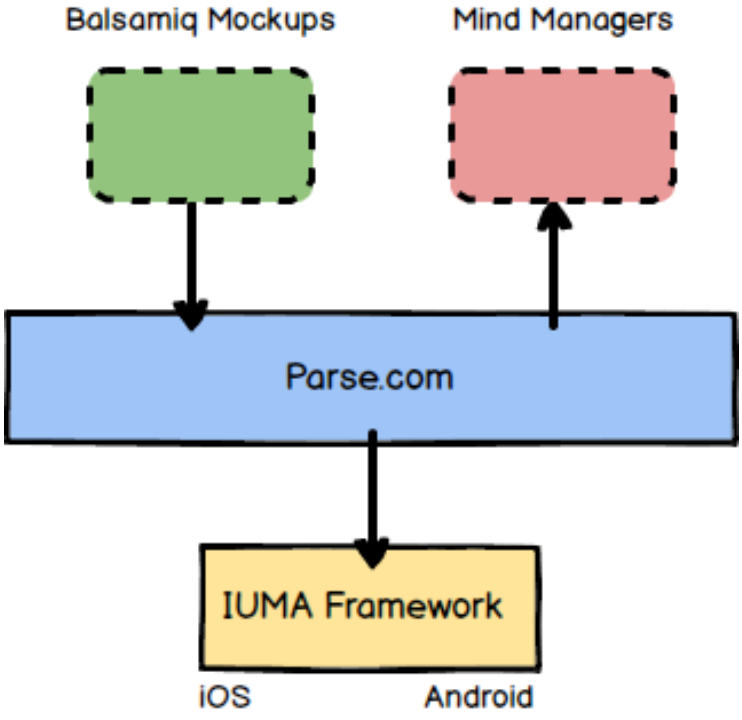


Figura 3 - Diagrama de Bloques del sistema en una dirección a diseñar

De esta forma, podemos hacer una lista de los diferentes componentes de los que se compone el conjunto del proyecto. El diagrama de bloques de la página anterior, representa el sistema desde Balsamiq hasta los distintos puntos.

Como se aprecia, Parse es la piedra angular desde la que se alimentan varios sistemas. Estos sistemas son representaciones distintas de la información. Los colores en el diagrama representan diferentes formas de representar la información. En Azul se encuentran los Mockup, en rojo violeta la información completa las diferentes pantallas de la aplicación, en verde la información está en forma de tabla y, por último, el rojo la información se representa mediante un árbol. Por tanto, de esta representación podemos obtener varios componentes:

- **Balsamiq to Parse**. Este componente extrae los datos de los diferentes Mockup generados con Balsamiq y completará la información en Parse según las necesidades del sistema.
- **Parse to Xmind – Freemind**. Aunque se puede dividir en dos, el sistema está diseñado para obtener en un formato libre que Freemind puede abrir. Por su parte, Xmind permite importar a su formato, cerrado, los archivos en formato Freemind. Esto hace que una vez obtenido el .mm (formato usado en Freemind), se puede obtener el .xmind que es el formato que entiende el programa Xmind.
- **Parse to iOS – Android**. Este componente no forma parte de este proyecto, dado que ya está diseñado por el IUMA y otros proyectos finales de carrera ya han probado su correcto funcionamiento para diferentes aplicaciones.

Por otro lado, como se menciona anteriormente, el propósito de este proyecto es el de poder tener un sistema bidireccional. Cabe destacar que debido a que el componente de Parse a iOS o Android no forma parte de los objetivos de este proyecto, no se implementará el camino inverso.

De la figura de la siguiente página podemos obtener los otros dos componentes que se integran en el sistema:

- **Parse to Balsamiq**. Este componente descarga la información que se ha almacenado en Parse y genera los diferentes Mockup en el formato adecuado. Una vez se han obtenido los ficheros estos se pueden usar con la aplicación de escritorio de Balsamiq o se pueden importar a la versión web del mismo programa.

- **Xmind – Freemind to Parse**. Este es simplemente el camino inverso. Aunque parece que es simplemente el inverso, la estrategia que se sigue es totalmente distinta, dado que la forma de obtener los datos cambia radicalmente.

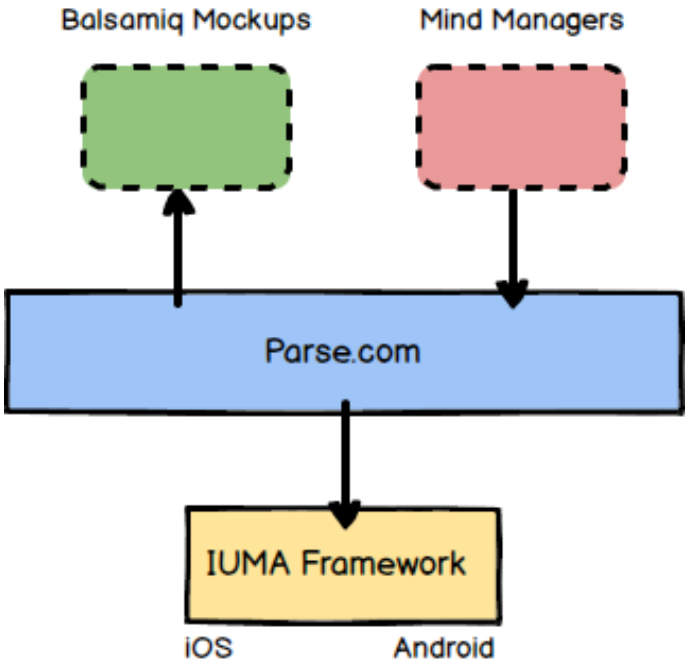


Figura 4 - Diagrama de Bloques del segundo sistema a diseñar

Estos serían los 5 principales componentes en los que se basa el sistema para obtener, a partir de los Mockup la aplicación móvil. Como se observa, el sistema tiene diferentes entradas y da como resultado diferentes salidas. Estas a su vez pueden formar parte nuevamente como entradas, dando como resultado un sistema totalmente recursivo.

3.4. Diseño del sistema completo

La siguiente figura, representa como queda el sistema completo, después de integrarlo con el resto.

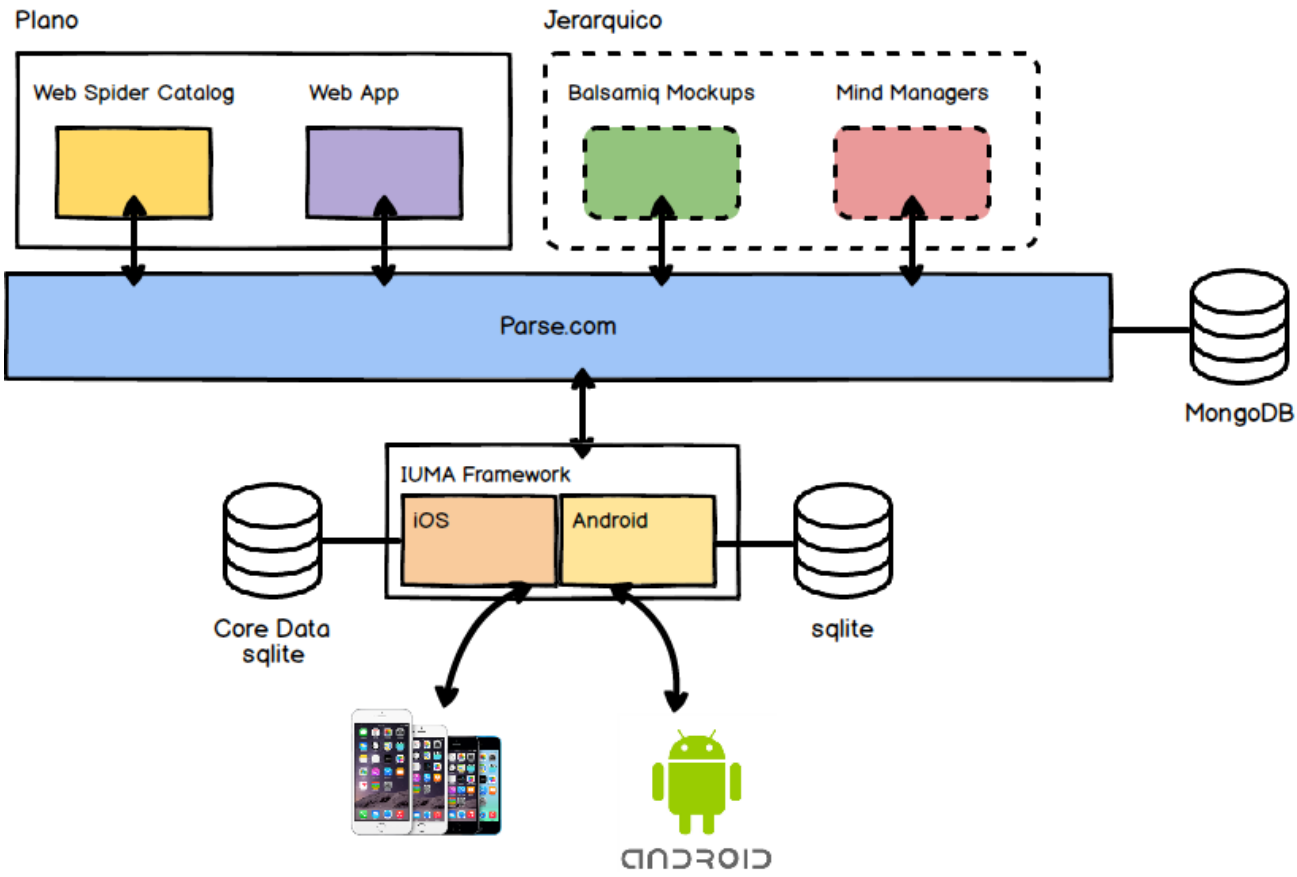


Figura 5 - Diagrama del sistema total integrado en el existente

3.5. Plataforma de soporte del sistema

Inicialmente, y como se indica en el anteproyecto de este PFC, se planteó realizar la herramienta como aplicación de escritorio. Esta idea fue rechazada durante los primeros meses de desarrollo, dado que uno de los principales escollos que tenía era el del framework del IUMA que se pretende usar. Este framework está diseñado para trabajar con iOS y no con OSX. Aunque en un principio se pensó que la compatibilidad podía ser solucionada, después da varios test y de intentar hacerlos compatibles, se decidió junto con los tutores del proyecto en realizar la herramienta como una aplicación móvil.

Aunque en un principio resultaba un poco extraño el hecho de que una aplicación móvil pudiera realizar este tipo de aplicaciones, pronto se encontraron ventajas al desarrollo en móvil y no en escritorio. Las más importantes son las siguientes:

- No hay necesidad de realizar ningún cambio en el framework del IUMA y en caso de que haya una actualización del mismo o una corrección de errores, no haría

falta volver a pasar por el proceso de compatibilidad y sería compatible desde el momento cero.

- Si se usan herramientas en la nube, el usuario puede en cualquier momento ejecutar la aplicación y obtener los cambios en la aplicación móvil que está parseando.
- Desde un principio se sabía que para que la aplicación funcionara correctamente, esta debía conectarse con los servidores de Parse. Para esto, se requiere que exista una conexión a internet que permita que la aplicación y los servidores de Parse se comuniquen entre ellos. Los ordenadores, ya sean portátiles o estacionarios requieren de tener una conexión vía Wi-Fi o vía cable (Ethernet) para poder tener conexión a internet, mientras que los teléfonos inteligentes, de por sí tienen una tarjeta SIM facilitada por el operador. Estas pueden tener contratadas un plan de datos que permite la conexión a internet. Por tanto las probabilidades de que se pueda trabajar y parsear las aplicaciones usando como motor un teléfono móvil son mayores.
- Como motivo personal, la experiencia anterior hace más fácil el diseño de la aplicación móvil que la aplicación de escritorio para OSX, donde la experiencia y conocimiento anterior era muy limitada.

Por tanto, la aplicación móvil que se ha programado para el proyecto tendrá diferentes funciones, entre las que destaca la de unión entre todos los componentes. Todos estos componentes mencionados en el punto anterior, están programados en la aplicación móvil y son usados por el usuario según sus necesidades.

La aplicación, a la que se le ha dado el nombre de Parapp Studio, está diseñada para funcionar con iOS. Se decidió así debido a la experiencia previa, y principalmente, a que los recursos materiales son los necesarios para programar para iOS y no para Android. Esto no quiere decir que las aplicaciones resultantes del uso de Parapp Studio sean exclusivas para iOS, dado que es el framework del IUMA el que se encarga de generar y rellenar las aplicaciones y no hay diferencia en Parse.com entre iOS o Android.

4. Diseño y estructura de la aplicación Parapp Studio

Para la creación de la aplicación móvil se va a usar la herramienta de desarrollo Xcode, así como para creación de los Mockup que se usarán de guía en la programación del storyboard se usará Balsamiq 3.0 versión de escritorio.

Inicialmente la aplicación se empezó a desarrollar para trabajar sobre iOS8 pero como posteriormente salió la versión de iOS9, este será el sistema operativo sobre el que se han realizado las pruebas. Por otro lado, se decidió no cambiar a Swift y continuar programando en Objective-C dado que era el lenguaje que estaba más extendido en el inicio de la programación de la aplicación.

En este capítulo se pretenden explicar varios aspectos, así como presentar diferentes metas que se resumen a continuación:

- Las estrategias usadas tanto en el diseño inicial como durante el transcurso del desarrollo.
- Presentar los diseños Mockup y explicar todos los cambios que se realizaron desde el Mockup a la aplicación final.
- Modularización de la aplicación para usar los diferentes componentes sin dependencias entre ellos.
- Uso de base de datos y principales preferencias que se han de configurar.
- Principales diseños usados para la interfaz de usuario, así como logotipo, imágenes empleadas, paleta de colores, etc.
- Mostrar el resultado final de la aplicación

4.1. Diseño inicial y Mockup de la aplicación

Como se comentó anteriormente, la aplicación va a contener todos los diferentes motores para alojar los diferentes componentes de los que se compone el PFC. El diseño de la aplicación tiene como objetico que el usuario, con tan solo unos pocos clics sea capaz de realizar la acción completa, ya sea para parsear un Mockup como para generarlo.

Debido a que se poseen varios motores, la estrategia para elegir el motor, es dar a elegir al usuario cual será el punto de partida. Una vez el usuario ha elegido el punto de partida los diferentes escenarios disminuyen, pero este tendrá que configurar, sino lo ha hecho antes, los parámetros necesarios para poder ejecutar el motor. Estos parámetros se explicarán más adelante en la sección dedicada a cada componente.

La idea de la aplicación es que el usuario pueda seguir el proceso de análisis, y pueda darse cuenta de los posibles errores que hay. Una vez que el usuario ha elegido el punto de partida y ha configurado de manera general los parámetros necesarios, este puede elegir lanzar el componente y este se ejecutará dando feedback al usuario, si el proceso se ha completado con éxito o no.

Hay que tener en cuenta que la aplicación en sí no se utiliza para representar información o para que el usuario obtenga los resultados. Los posibles resultados de ejecutar la aplicación pueden ser dos: datos enviados a Parse.com donde se almacenaran o generación de ficheros con diferentes extensiones que el usuario debe usar con otras aplicaciones, como pueden ser Balsamiq, Xmind, editores de texto.

Es por tanto que el diseño de la aplicación está pensada para que esta sea un canal de comunicación entre el usuario y el resultado que este espera. Todos los contenidos de la misma se han pensado para que sean lo más simples posibles. Para ello se inspiró en los diseños de aplicaciones que tienen simplemente lo que el usuario necesita, intentando reducir la información presente al mínimo posible, como puede ser la página inicial de Google o los diseños de la página de ajustes del Smartphone iPhone. Como se indica en el título de este proyecto, lo que se pretende es tener una herramienta para generar prototipos, pero esto no quiere decir que se pierda de vista el diseño y la apariencia. Se intenta conseguir un estilo minimalista para que el usuario no pierda el foco de atención en lo que necesita usar en la aplicación.

Teniendo en cuenta estas premisas, la pantalla principal debe tener el menor número de elementos posibles. Como se quiere que se elija desde un principio cual será el punto de partida, de entre los 3 posibles (Balsamiq, Xmind o Parse), se dará al usuario únicamente esas opciones.

Del mismo modo, se necesita que el usuario tenga un control de la interfaz en la que se encuentra, se pondrá en la parte superior un controlador para la navegación y que el usuario pueda navegar entre las distintas interfaces hacia delante y haca atrás. En la siguiente imagen se representa el Mockup de la pantalla de inicio que se debería encontrar el usuario.

Ilustración 6 - Mockup pantalla inicial de Parapp Studio y de la idea de cómo mostrar las opciones

Una vez que el usuario ha seleccionado la fuente de datos, pasará a otra pantalla donde podrá ejecutar dos acciones, una será la de configurar los parámetros o la de ejecutar el motor que llevará los datos a un destino final. Esto se puede ver en la imagen derecha anterior, en la que se muestra el Mockup diseñado para esta función.

Como se explicó en el capítulo 3, sección 3.3, la aplicación está compuesta por varios componentes, pero todos ellos tienen en común que en el camino tienen que pasar por Parse.com que se trata de un punto intermedio que hace de junta con todos los demás. Es por esto que aunque existan varias opciones y entre ellas este la de “Parsear a Parse” y la de “Parsear a Xmind”, en general, si se selecciona la segunda opción, la aplicación ha de

pasar por Parse primeramente, por lo que deberá estar correctamente configurado para funcionar en cualquier dirección.

El siguiente Mockup representa cuales pueden ser las opciones dependiendo de la opción seleccionada y cuál sería la siguiente interfaz.

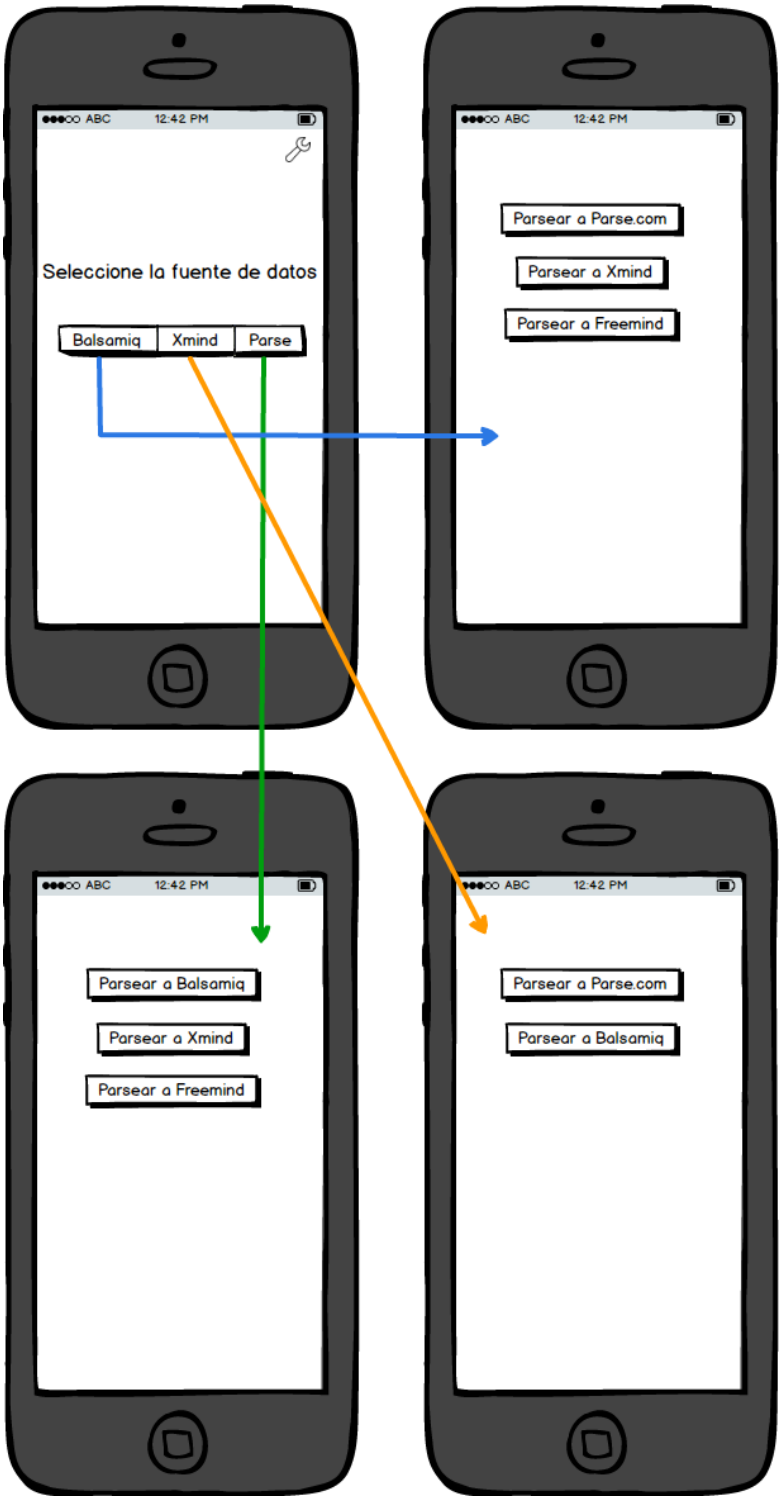


Ilustración 7 - Mockup con indicación de siguiente pantalla, dependiendo de la opción elegida

Como se observa en la figura, no todas las opciones están presentes en todas las interfaces, existiendo una, que tiene una opción menos. Esto se debe a que pasar desde Freemind a Xmind no tendría sentido dado que los propios programas permiten esta opción de forma nativa.

Por otro lado, en la pantalla inicial se ubica un botón para configurar la aplicación en la parte superior derecha. Cuando se pulsa este botón, la aplicación nos llevará a la pantalla de configuración, que tendrá un aspecto similar al siguiente.

Ilustración 8 - Mockup del menú de opciones

Estos son los parámetros de configuración iniciales que se pensaron útiles para la aplicación, aunque con el desarrollo de la aplicación, estos pueden cambiar. Se ha separado en la imagen el archivo en dos partes, dado que el menú no entrada en una sola pantalla.

Los parámetros de configuración se dividen en 4 secciones. La primera sección es la que se encarga de la base de datos, donde almacenaremos toda la información. Las tres siguientes secciones están dedicadas a las diferentes plataformas en las que podemos obtener los datos. La primera de ellas es Balsamiq, donde solo hace falta indicar dónde queremos obtener los resultados, como ocurre también con la última, xmind. Por último, tenemos los parámetros de configuración para Parse.com, que son un poco más

complicados, dado que estos son los que nos van a permitir usar la aplicación, dado que sin pasar por Parse.com la aplicación no tiene ninguna utilidad.

Uno de los parámetros de configuración de la parte de la base de datos, no es un parámetro, sino un botón que nos lleva a ver los archivos que se han sincronizado entre la base de datos y la aplicación. El aspecto de la aplicación será el siguiente.

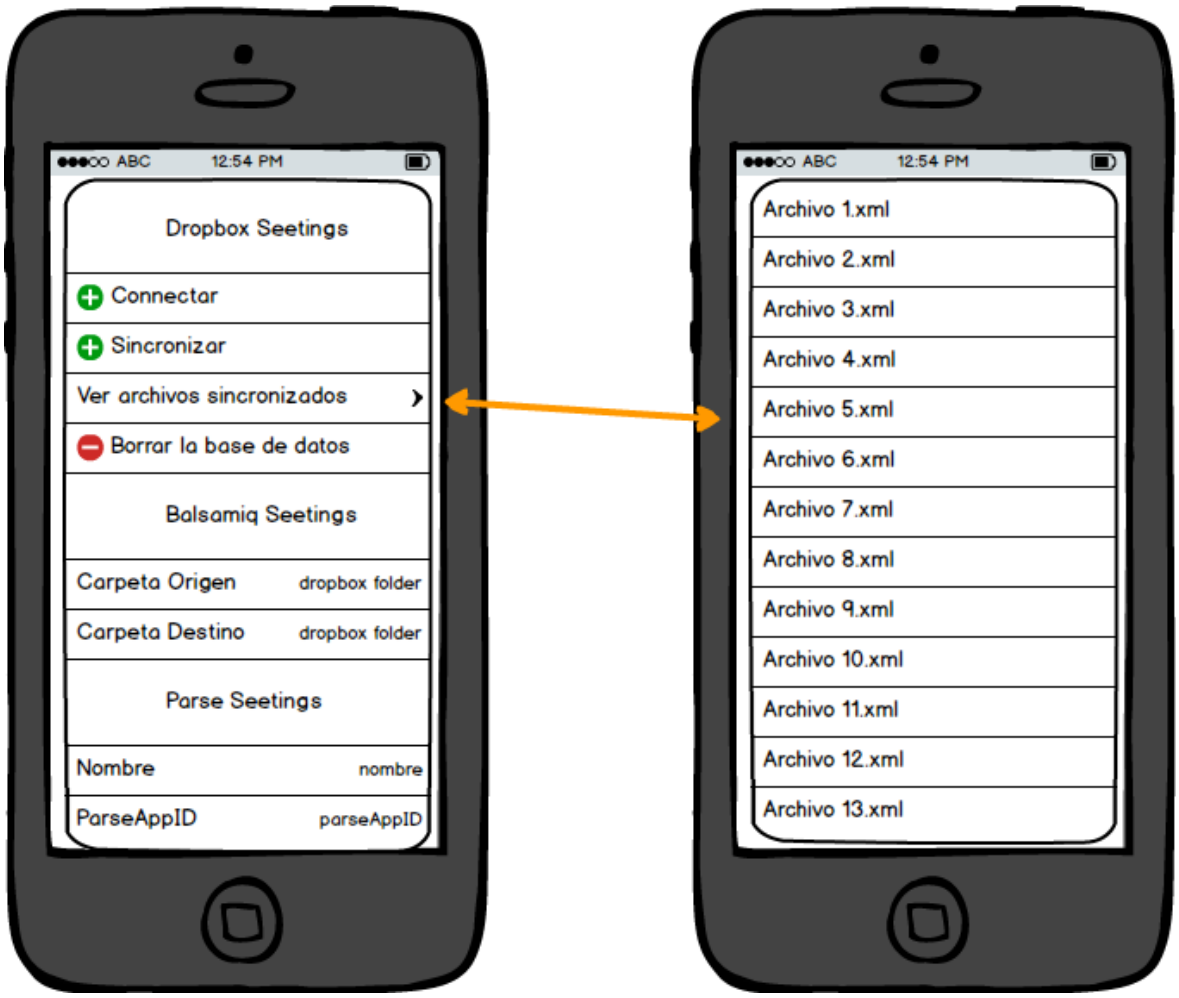


Ilustración 9 - Mockup de una de las opciones del menú de configuración

4.2. Arquitectura de la aplicación - MVC

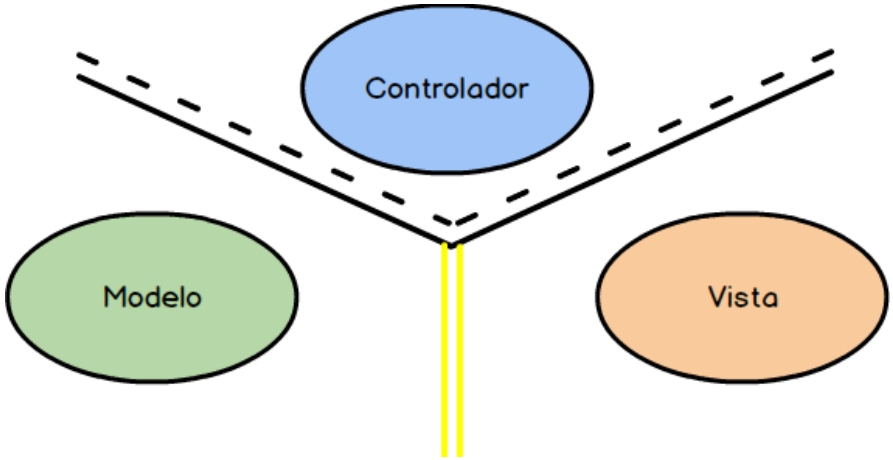
Como se va a trabajar usando el programa Xcode, y usando los frameworks que este ya posee, se ha decidido usar el patrón de diseño Modelo – Vista – Controlador (MVC) en su variación de Modelo – Vista – Presentador (MVP). Este patrón de diseño tiene una larga historia, siendo usado por primera vez en 1979. Se utiliza dado que permite separar y definir de forma independiente el Modulo (Objetos del negocio), la vista (UI, interfaz de

usuario) y el controlador, que guía la aplicación. Básicamente conseguimos separar la lógica de la aplicación de la vista de la aplicación [6] [7]. Este proporciona al desarrollador una gran flexibilidad.

Para entender este patrón hace falta entender cada capa [8]:

- La capa modelo. Contiene el núcleo de la funcionalidad. Es decir, implementa la lógica de la aplicación. Este obtiene o transforma los datos de forma que le da sentido. También se encarga de procesar, validar, asociar y en general, cualquier acción que requiera la manipulación de datos. Contiene toda la base de datos de la aplicación, así como todas las instrucciones de cómo manipular y adaptar los datos de esta para que sean entendibles en un ámbito concreto.
- La capa vista. Presenta el modelo. Como su nombre indica, contiene la parte del código que va a producir toda la visualización de una o varias interfaces graficas que componen la aplicación. Puede existir una capa vista o varias. Esto no quiere decir que la capa vista sea con la que se comunica el modelo. Esta capa presenta los datos del modelo, pero no los obtiene a través de él. De hecho, la vista es independiente de los datos y no sabe lo que está representando, solamente da forma a unos datos para el usuario.
- La capa controlador o presentador. Reacciona a las peticiones del usuario, ejecutando la acción adecuada. Podría ser llamado como el administrador de la aplicación. Esta siempre esperando a que el usuario genere una petición. Este valida que la petición se puede procesar y se la envía al modelo para que procese los datos que el usuario necesite. Una vez el modelo actualiza los datos, el controlador puede usarlos para actualizar la vista con los datos del modelo.

Ilustración 10 - MVC



Esta imagen representa claramente las distintas relaciones que puede haber entre el modelo, la vista y el controlador. Las líneas intermedias son representadas como las líneas de tráfico de las carreteras. Como se puede observar, entre el modelo y la vista la comunicación está prohibida. No puede haber comunicación entre ellos. Esto sucede porque este patrón de diseño pretende separar el modelo de interfaz gráfica, de forma que los datos sean independientes.

Por otro lado se observa que la comunicación siempre parte desde el controlador. Es por esto que también es conocido a veces como administrador. Es el que reacciona a todos los eventos de la aplicación y maneja al modelo. Por tanto la comunicación en este modelo parte siempre del controlador, que actualiza la vista con la información que ha obtenido del modelo a petición del controlador.

En la siguiente imagen, las flechas rojas indican que comunicación está prohibida y las flechas verdes indican que comunicación está permitida.

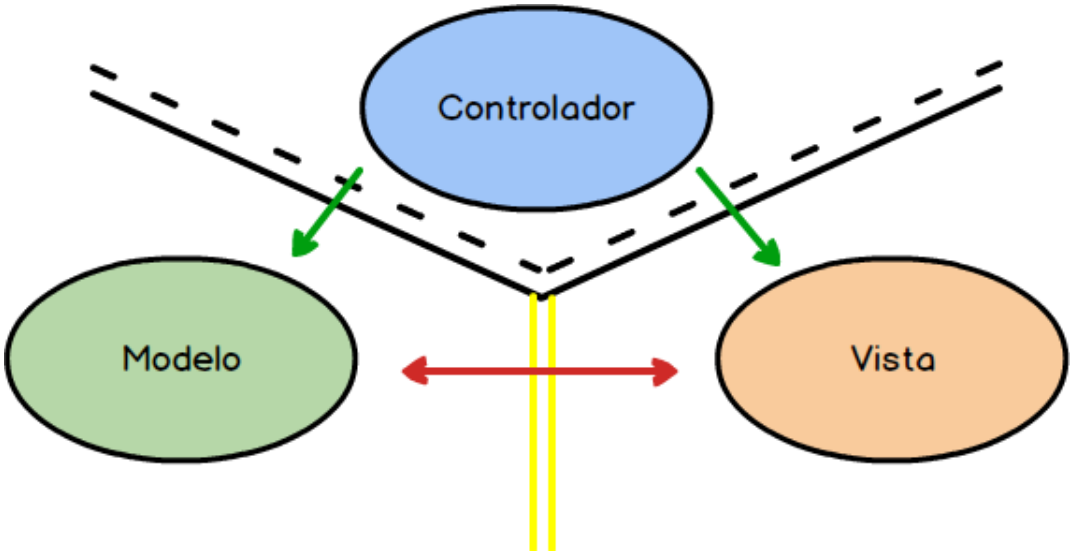


Ilustración 11 - Relaciones entre los componentes del MVC

Del mismo modo, la pregunta que nos viene a la mente, es como el modelo o la vista indican al controlador que, por parte de la vista, el usuario ha hecho una acción o va a realizar alguna, así como comprobar las reglas y limites que tiene el usuario y por el otro lado, cuando el modelo se ha actualizado y el controlador lo debe saber.

Por un lado, imaginemos que el controlador crea una diana para una acción concreta. En ese caso, cuando la vista tenga esa acción, la lanzara hacia la diana del controlador. Este recibe dicha acción y tomará las medidas que sean necesarias.

Pero esta no es la única forma de comunicarse por parte de la vista, también puede usar lo que se denomina delegate. Básicamente, este delegate lo que hace es sincronizar la vista con el controlador, con lo que el usuario debe hacer o no debe hacer, o hará. En este caso, este delegate se encuentra en el controlado y se asegura la sincronización entre ambos. Este delegate se realiza mediante un protocolo.

El ultimo posible caso de comunicación entre la vista y el controlador es con lo que se denomina la fuente de datos. Como ya explicamos antes, la vista no es dueña de ningún dato, ni tampoco sabe qué tipo de datos está mostrando al usuario. El mejor ejemplo es el de la música que se tiene en el reproductor del móvil. La vista, representa todas las canciones que tenemos disponibles y podemos seleccionar una u otra dependiendo de la que queremos escuchar en ese momento. Pero sería poco inteligente, que la vista supiera de todas y cada una de las canciones que tiene que mostrar. Lo que la vista necesita saber únicamente es cuantas y que valores ha de mostrar. Las canciones son parte del modelo, pero el controlador puede saber cuántas y puede estar atento, por qué se puede comunicar con el modelo, a posibles incrementos o descenso del número de canciones. Por tanto, la fuente de datos, es decir, todos los valores necesarios para representar la información en la vista, también se encuentran en el controlador, y la vista simplemente los usa para generar la vista en cada momento.

La siguiente imagen representa las tres formas de comunicación que hay entre la vista y el controlador.

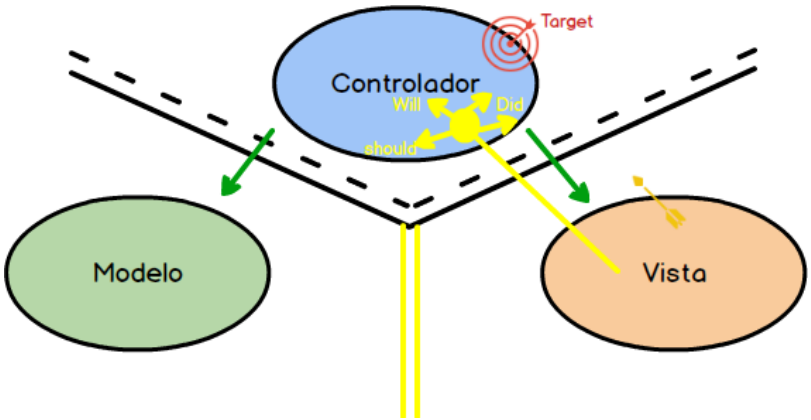


Ilustración 12 - Relación entre la Vista y Controlador

La comunicación entre el modelo y el controlador no está permitida en el sentido modelo-controlador, solo como explicamos antes en el sentido, controlador modelo. Pero algunas veces no es el controlador el que necesita recoger información, sino es el modelo, que puede estar sincronizado con otras fuentes, el que le diga al controlador, tengo nueva información que necesitas. Esto se complica, si tenemos en cuenta que el modelo no sabe de la existencia del controlador, por lo que necesita de otros métodos para comunicarse o para informar de que tiene nueva información o que la base de datos se ha actualizado.

Para llevar a cabo este sentido podemos asemejar el sistema de comunicación a un sistema de radio, donde el modelo tiene la estación base y el controlador un receptor sintonizado a la misma frecuencia. Cuando el modelo tiene nueva información, hace un broadcast, para que todos los que están interesados en la información que tiene, queden informados y de esta forma todos estén sincronizados. Por su parte el controlador se subscribe a ese broadcast de los modelos que le interesan. La siguiente figura representa el sistema de comunicación entre modelo y controlador.

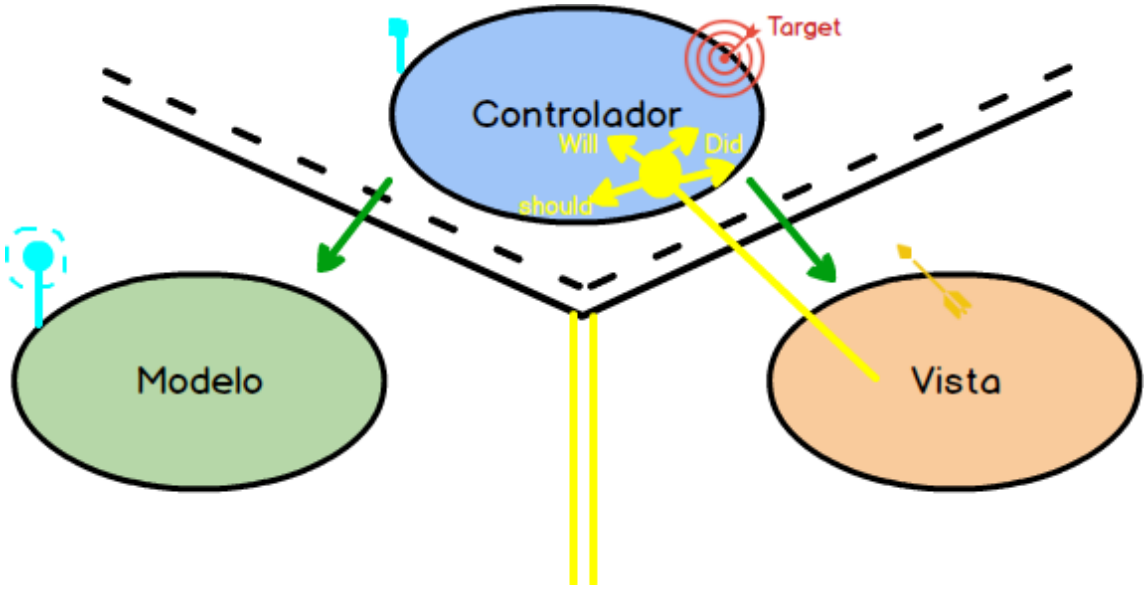


Ilustración 13 - Relación entre el modelo y el controlador

Aunque es posible que la vista incluya también este sistema para quedar informado de las actualizaciones del modelo, esta implementación no está permitida en esta arquitectura, quedando claramente identificado, que toda el intercambio de información ha de pasar por el controlador.

Por qué se elige este patrón de diseño y no otros tiene una fácil explicación. Este patrón es el más usado en el diseño de aplicaciones móviles y además permite la fácil modulación de los diferentes sistemas, permitiendo del mismo modo un sencillo mantenimiento y en caso de ser necesario incluir mejoras o aumentar el alcance del mismo.

Por último, y dependiendo de la aplicación móvil que se desea desarrollar es probable que no haya solo un modelo o solo un controlador o solo una vista. Pueden existir varios de cada uno de ellos y estar relacionados. Normalmente hay varios controladores y varias vistas y un solo modelo. Lo único que hay que tener claro en este caso, es que las relaciones y las comunicaciones han de seguir todas las reglas, no pudiendo incumplirlas.

Además de las explicadas anteriormente, cuando hay varios controladores, varias vistas y varios modelos, estos pueden hablar entre sí, es decir la comunicación entre modelos está permitida, así como comunicación entre controladores y entre vistas. Aunque es posible este tipo de comunicaciones, estás deben ser controladas, para que el sistema no se convierta en un sistema con un costo de mantenimiento alto. La siguiente imagen representa un sistema con múltiples modelos, controladores y vistas y como son las comunicaciones de unos con otros.

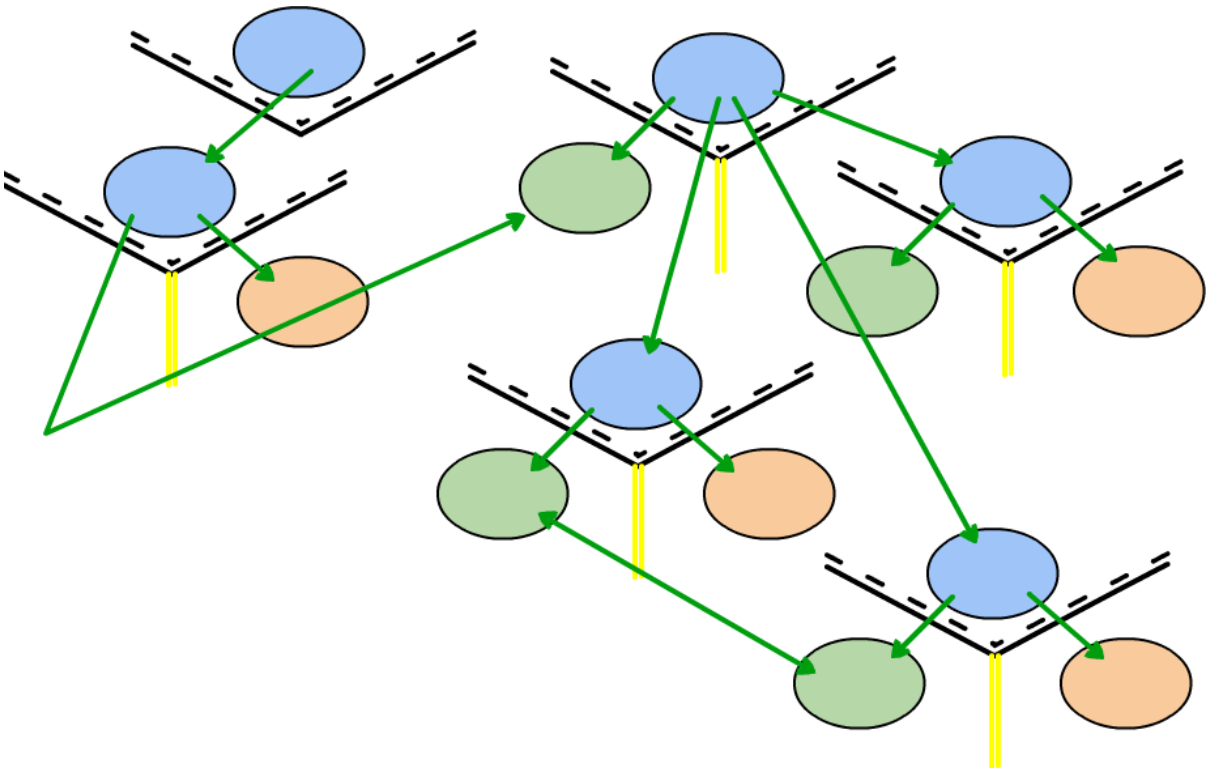


Ilustración 14 - Relación entre diferentes MVC

4.2.1. Modelo

En Parapp Studio, la capa modelo de la aplicación contiene todos los componentes de los que hemos estado hablando anteriormente, es decir, contiene todos los analizadores de la información que convertirán la información desde un modo de representación a otro. Normalmente, la capa modelo también implementa la relación con la base de datos. En el caso de esta aplicación y como no formaba parte de los objetivos del proyecto inicialmente, se planteó la idea de no usar una base de datos como tal, sino un servicio de alojamiento de archivos en la nube que tuviera disponible un API para su implementación y que además fuera popular en el mercado. Haciendo una pequeña investigación en el mercado, se puede encontrar que existen dos principales proveedores de servicios: Dropbox y Drive, aunque también hay que tener en cuenta algunos menos populares pero que pueden aportar alguna funcionalidad que sea de gran valor añadido para la aplicación.

Todos los servicios que la capa modelo incluye serán presentados en los siguientes capítulos excepto los servicios de almacenamiento que se presenta un análisis de los mismos en el capítulo 4.4. Los diferentes modelos que se incluyen en el modelo son los siguientes:

- Parse to Balsamiq
- Balsamiq to Parse
- Parse to Xmind/Freemind
- Xmind/Freemind to Parse

Aunque los servicios de almacenamiento se presentan en este capítulo, será responsabilidad de cada modelo el actualizar y obtener la información de la base de datos. Las claves y direcciones para acceder a la base de datos serán proporcionadas por el usuario, por lo que será responsabilidad del controlador de informar al modelo de donde quiere recibir los resultados y de donde tiene que recoger la información. El controlador, será único, y controlará todos los modelos que se han incluido. Por tanto, es de destacar que no hay un modelo que se comunique con la base de datos o que sea responsable de la comunicación.

Esto se ha diseñado de esta manera, dado que se pretende que todos los modelos interactúen con la base de datos de acuerdo a las necesidades de estos y no que sea un único modelo el que defina como tienen que ser las comunicaciones.

A priori, esta decisión parece incoherente con las directrices generales de programación, que recomiendan tener un protocolo único de comunicación y de que dos modelos no accedan simultáneamente a la base de datos, dando la responsabilidad de acceso a un único modelo. Pero en nuestro caso, dado que todos los modelos corresponden a un mismo controlador y a que el controlador no ejecutará la siguiente orden proveniente del usuario hasta que el modelo anterior no haya acabado todas las operaciones, nos aseguramos de que los modelos están relacionados entre sí, de forma de que no intentaran acceder a la base de datos (nube) hasta que el anterior no haya acabado todas las operaciones.

4.2.2. Vista

La capa vista es la que, como se explicó anteriormente, se encarga de representar mostrar la información. Es decir, es nuestro storyboard. Esta aplicación tiene muy pocas interfaces de usuario diferentes. Como se mosto en el punto anterior con los Mockup, solo tendremos 4 pantallas.

La siguiente imagen representa como queda representado en la aplicación la pantalla inicial.

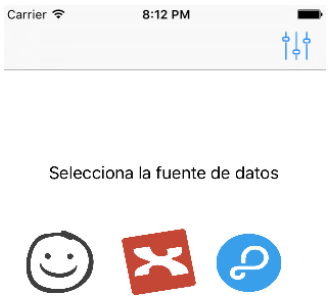


Ilustración 15 - Vista de la pantalla inicial

Como se explicó con los Mockup, la idea es tener una aplicación lo más sencilla posible, por lo que los botones de texto se han sustituido con los logotipos de los distintos posibles puntos de partida.

Ilustración 16 - Significado de los botones

De este modo, se muestra que la aplicación tiene una gran cantidad de espacio blanco, intentando asemejar el diseño a las principales pautas de diseño de aplicaciones de nivel bajo, donde cuanto menor número de botones y estos sean sencillos, más fácil será para el usuario usar esta aplicación.

Si se selecciona Balsamiq, la pantalla que resultará será la que se muestra a continuación.

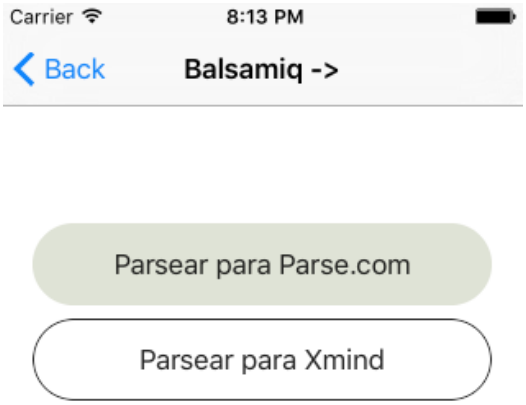


Ilustración 17 - Pantalla parseo desde Balsamiq

Como se puede ver en la imagen, y repitiendo la misma estrategia que en el caso de la pantalla principal, el número de botones es el menor posible, intentando satisfacer todas las posibles necesidades.

Una vez que se pulsa uno de los dos botones, para indicar el destino que se desea conseguir, el controlador indicará al modelo que acciones debe tomar y cuáles son los pasos a seguir. Una vez que el modelo ha acabado, este avisa al controlador que a su vez indica a la vista que debe desplegar un mensaje indicando al usuario que se ha acabado el analizado de los datos y que puede obtener los archivos en el directorio indicado.

Volviendo a la primera pantalla, si se pulsa tanto el logotipo de Xmind como el de Parse, se obtienen pantallas similares que se muestran a continuación.

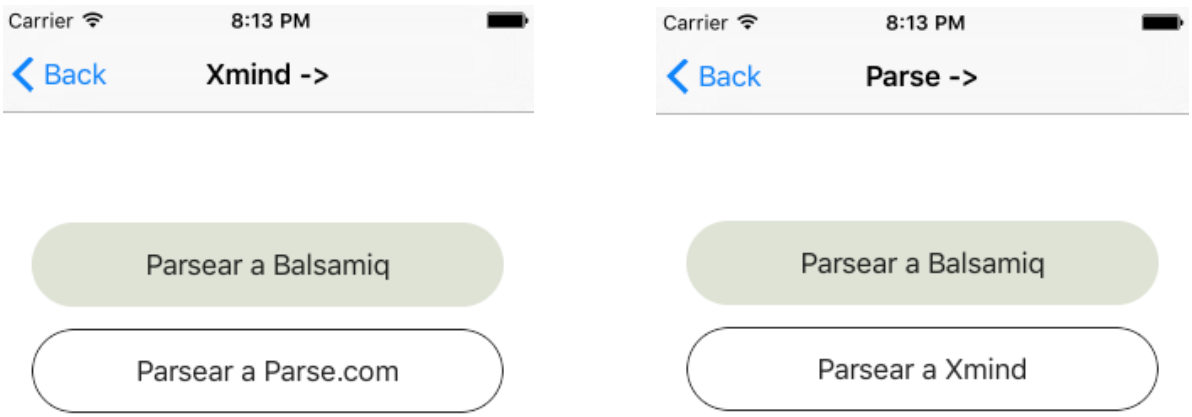
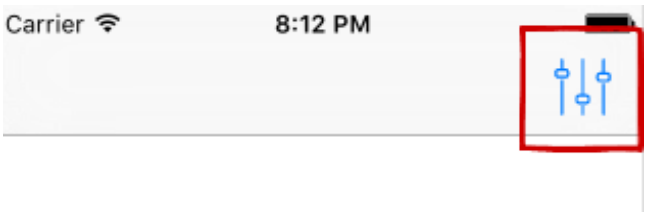


Ilustración 18 - Pantalla de parseo desde Xmind y desde Parse

Como se puede observar en las capturas de pantalla de la aplicación, en el título de cada pantalla se incluye una palabra con una flecha indicando cual es el inicio.

Aunque el usuario indique el inicio y a donde quiere parsear, la aplicación, internamente siempre pasará a Parse.com y luego pasará al destino seleccionado. En ningún caso puede funcionar si el usuario no ha configurado correctamente las claves para acceder a Parse. Para llevar a cabo la configuración, el usuario debe pulsar el icono de configuración que se encuentra en la pantalla inicial.



Cuando el usuario pulsa el icono, este le lleva a poder ajustar la configuración de la aplicación. En caso de que el usuario no configure correctamente la aplicación, esta puede tener un funcionamiento erróneo o que no se corresponda con lo que el usuario pretenda. Las dos imágenes que se muestran a continuación son las pantallas de configuración.

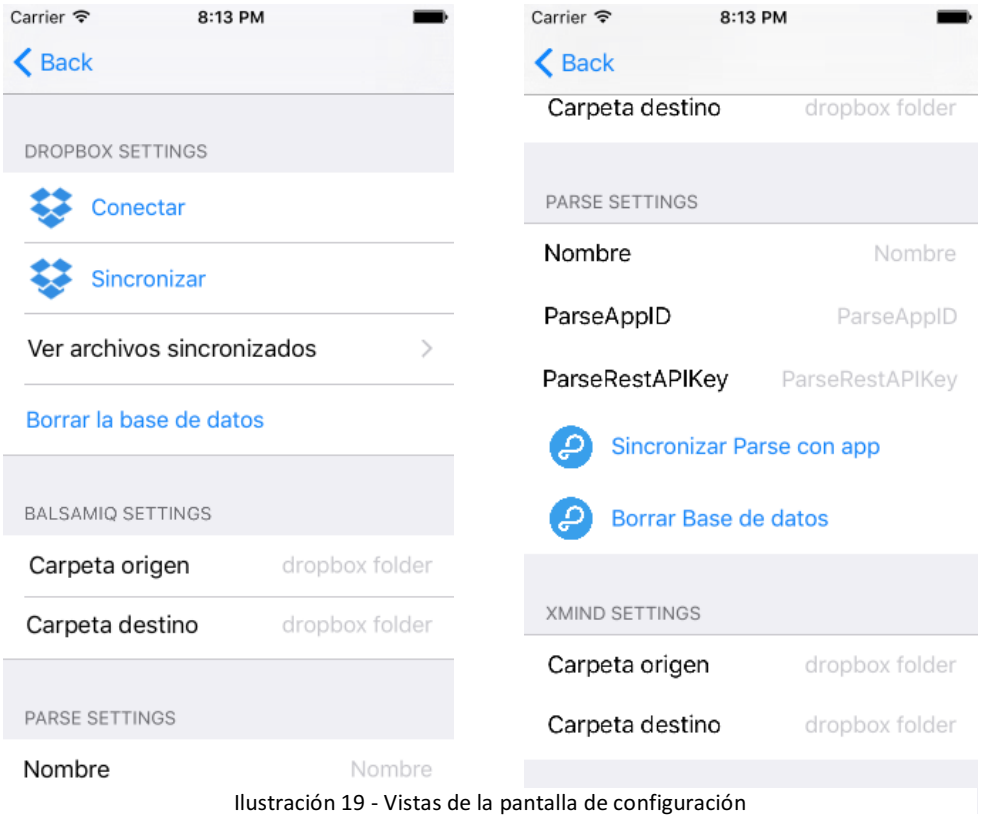


Ilustración 19 - Vistas de la pantalla de configuración

De igual manera, en los parámetros de configuración se pueden realizar las tareas de sincronización con la base de datos, así como la obligatoria tarea de conectar la cuenta de usuario de Dropbox con la aplicación y permitir que la aplicación escriba datos y lea de la misma.

Todo este proceso se lleva a cabo a través de la API de Dropbox, por lo que no se incumple con ninguna normativa de protección de datos, dado que la aplicación no guarda ningún dato de carácter personal. Por último, la vista incluye la posibilidad de que el usuario tenga acceso a comprobar que los datos sincronizados entre Dropbox y la aplicación sean los correctos. Cuando el usuario pulsa sobre el botón de "Ver archivos sincronizados", este le lleva a visionar en una tabla el nombre de todos los archivos que el usuario ha sincronizado, aunque no puede acceder a ellos para ver el interior de los mismos, dado que no tendría ningún uso.

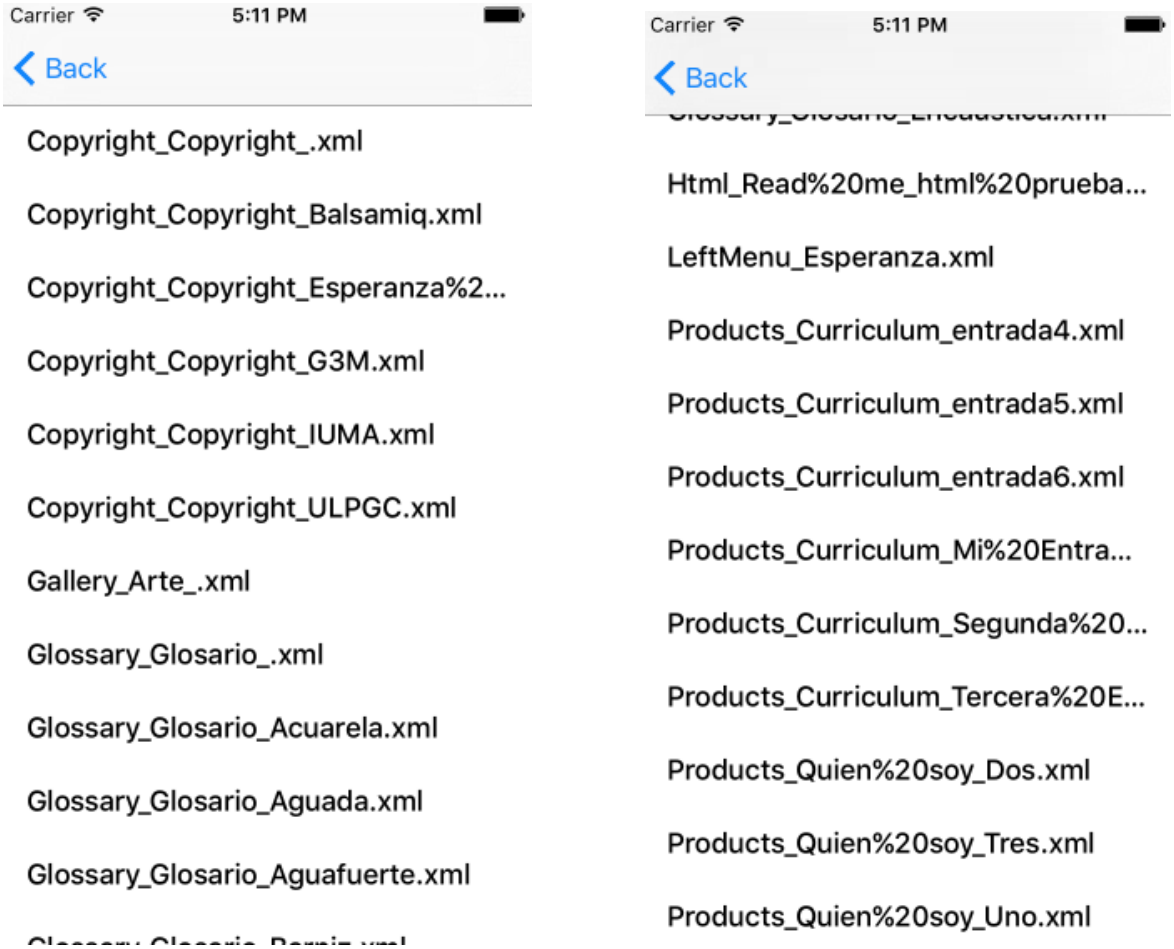


Ilustración 20 - Vista pantalla de configuración. Opción 1

4.2.3. Controlador / Presentador

Por ultimo tenemos el controlador o presentador. Este es el encargado de llevar a cabo todas las acciones y desde el parten todas las comunicaciones. En el caso de esta aplicación, tendremos tres controladores, que se encargaran de diferentes vistas.

View Controller

Este es el controlador que viene por defecto en el entorno de desarrollo de Xcode. Se ha reutilizado como controlador principal y es el que va a realizar las llamadas a los distintos modelos. Por tanto es lógico, que este controlador tenga varias propiedades muy importantes, como son las de los datos que se han parseado. Es decir, que este tendrá varias propiedades que almacenaran los resultados.

Aunque anteriormente se explicó que los resultados de salida de cada una de las llamadas al modelo serían a través del servicio de almacenamiento y en diferentes ficheros, debido a la complejidad de algunas operaciones, si el usuario ha obtenido previamente unos datos que pueden ser reutilizados en otro proceso, se almacenan para poder ser reutilizados y tener un acceso más rápido a ellos.

Estas propiedades son las siguientes:

```
@interface ViewController () <DBRestClientDelegate>

@property (nonatomic, strong) NSData *data;
@property (nonatomic, strong) NSData *dataParsed;
@property (nonatomic, strong) NSString *dataParsedXML;

@end
```

Código 1 - Propiedades del controlador principal

Como este controlador es el que más relación tiene con los modelos y con los diferentes servicios, este se explicará en conjunto con estos en los siguientes capítulos.

Settings View Controller

Este controlador está asociado solamente a una vista, que es la de la configuración. Este controlador es de tipo Tabla y contiene todos los outlets que son necesarios para guardar toda la información que el usuario ha introducido en la pantalla y que este no tenga que

introducir otra vez, cuando inicie la sesión por segunda vez. Esto se realiza sincronizando estos datos con el modelo.

Además de tener todos estos datos, incluye varias funcionalidades más, como la de sincronización de los servicios de almacenamiento y la de sincronizar con Parse.com todos los datos.

Por este motivo, estas son las propiedades que debe incluir este controlador:

```
@interface MBSettingsViewController() <DBRestClientDelegate>

@property (nonatomic, strong) DBRestClient *restClient;
@property (nonatomic, strong) Product_categories *category;

@property (weak, nonatomic) IBOutlet UITextField *balsamiqOriginFolder;
@property (weak, nonatomic) IBOutlet UITextField *balsamiqSendFolder;
@property (weak, nonatomic) IBOutlet UITextField *parseName;
@property (weak, nonatomic) IBOutlet UITextField *parseAppID;
@property (weak, nonatomic) IBOutlet UITextField *parseRestApiKey;
@property (weak, nonatomic) IBOutlet UITextField *xmindOriginFolder;
@property (weak, nonatomic) IBOutlet UITextField *xmindSendFolder;

@end
```

Código 2 - Propiedades del Controlador de Settings

Todo el código que incluye este controlador se encuentra disponible en los anexos. Aunque cabe destacar que las principales acciones que lleva a cabo este controlador son las de sincronizar con Dropbox y borrar la base de datos interna de los archivos sincronizados, conectar con Dropbox y por ultimo realizar las mimas acciones pero con Parse.com, además de cómo se mencionó anteriormente incorporar las distintas propiedades necesarias para el correcto funcionamiento de la aplicación.

Files Table View Controller

Este controlador está asociado solamente a una vista, que es la de la ver los archivos que tenemos sincronizados. Este controlador es de tipo Tabla y contiene todos los métodos necesarios para pedir al modelo que información ha de mostrar y mostrarla.

Como explicamos anteriormente, la forma de comunicarse entre el controlador y el modelo puede ser de diversas maneras y una de estas es cuando la vista delega en el controlador el control de saber qué información desplegar y como configurarse para poder mostrar la información de la manera más correcta posible. De esta manera, este

controlador pide al modelo que le informe de cuál será la información, cuantas filas debe generar, cuantas secciones, etc.

Este es un controlador básico para mostrar informaciones de una tabla y su código es muy sencillo y corto.

```
@interface MBFilesTableViewController()
@property (strong, nonatomic) NSMutableArray *nameOfFiles;
@end

@implementation MBFilesTableViewController

- (NSMutableArray *) nameOfFiles{

    if (!_nameOfFiles){
        _nameOfFiles = [[NSMutableArray alloc]init];
        MBParseador *parser = [[MBParseador alloc]init];
        _nameOfFiles = parser.loadFiles;
    }

    return _nameOfFiles;
}

- (NSInteger) numberOfSectionsInTableView:(UITableView *)tableView{
    return 1;
}

- (NSInteger) tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section{
    return self.nameOfFiles.count;
}

- (UITableViewCell *) tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    UITableViewCell *cell = [tableView
dequeueReusableCellWithIdentifier:@"Cell" forIndexPath:indexPath];

    cell.textLabel.text = [self.nameOfFiles objectAtIndex:indexPath.row];

    return cell;
}

@end
```

Código 3 - Controlador de archivos sincronizados

4.3. Diseños de la aplicación

En el siguiente apartado se detallan los distintos diseños que se han llevado a cabo para la aplicación. Estos son:

- Logotipo de la aplicación.
- Pantalla de inicio (Launch Screen)
- Botones usados

4.3.1. Logotipo de la Aplicación

El logotipo de la aplicación se ha diseñado con la intención de darle una imagen corporativa a la aplicación. Del mismo modo se ha intentado seguir la misma paleta de colores que usa Apple en sus iconos, intentando de esta forma integrar el logo entre los distintos logotipos de los teléfonos móviles.

También y rompiendo con los logotipos tradicionales, se ha incluido el nombre de la aplicación en el logotipo para hacerlo más sencillo de encontrar. El logotipo se ha diseñado con la ayuda de Photoshop. El resultado inicial es el siguiente:

Ilustración 21 - Logo de Parapp Studio

Uno de los principales problemas que se encontró durante la inclusión del logotipo en el proyecto, fue que no se conseguía adecuar el icono a los tamaños requeridos por Xcode. Para solventar este problema, se recurrió al uso de la aplicación online “makeappicon” [9].

Esta aplicación, usando como base cualquier icono, genera los archivos necesarios, con las calidades necesarias que Xcode requiere, ya sea para el desarrollo de aplicaciones en iPhone, iPad o iWatch.

El resultado de la inclusión de las imágenes, es que cuando se genera la aplicación en distintos teléfonos, estos tienen el icono en la calidad requerida.

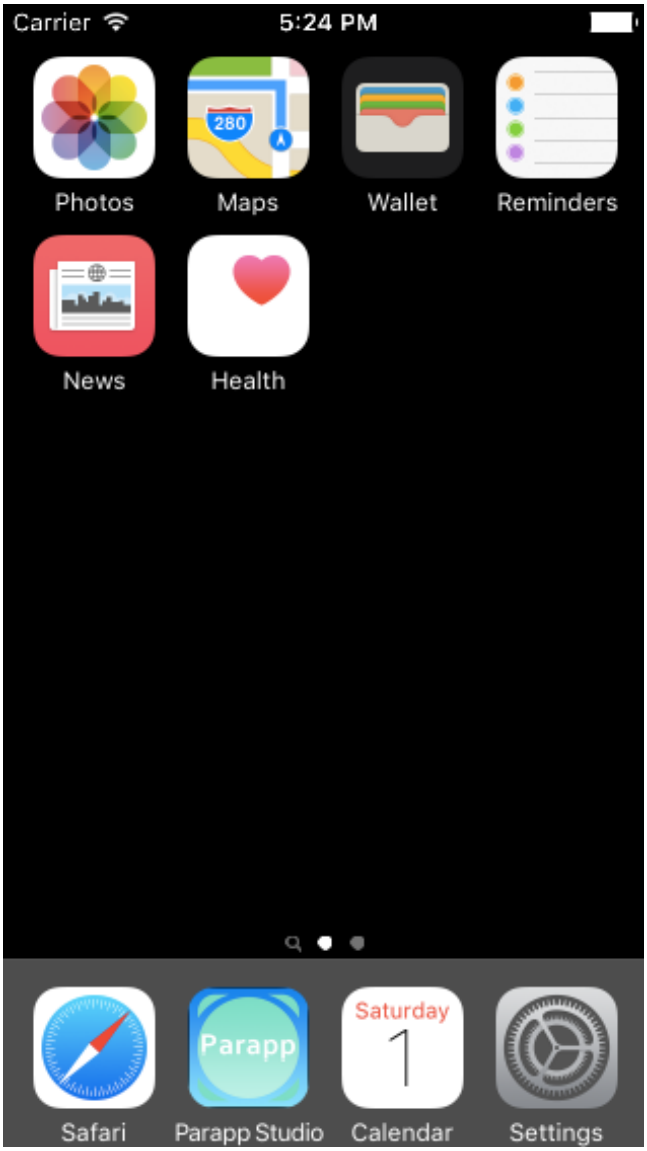


Ilustración 22 - Apariencia de la aplicación junto con las otras aplicaciones

Este es un ejemplo de la aplicación, ejecutada en un iPhone 5s. Una vez solucionado este problema, se añadieron las imágenes requeridas en la carpeta images.xcassets.

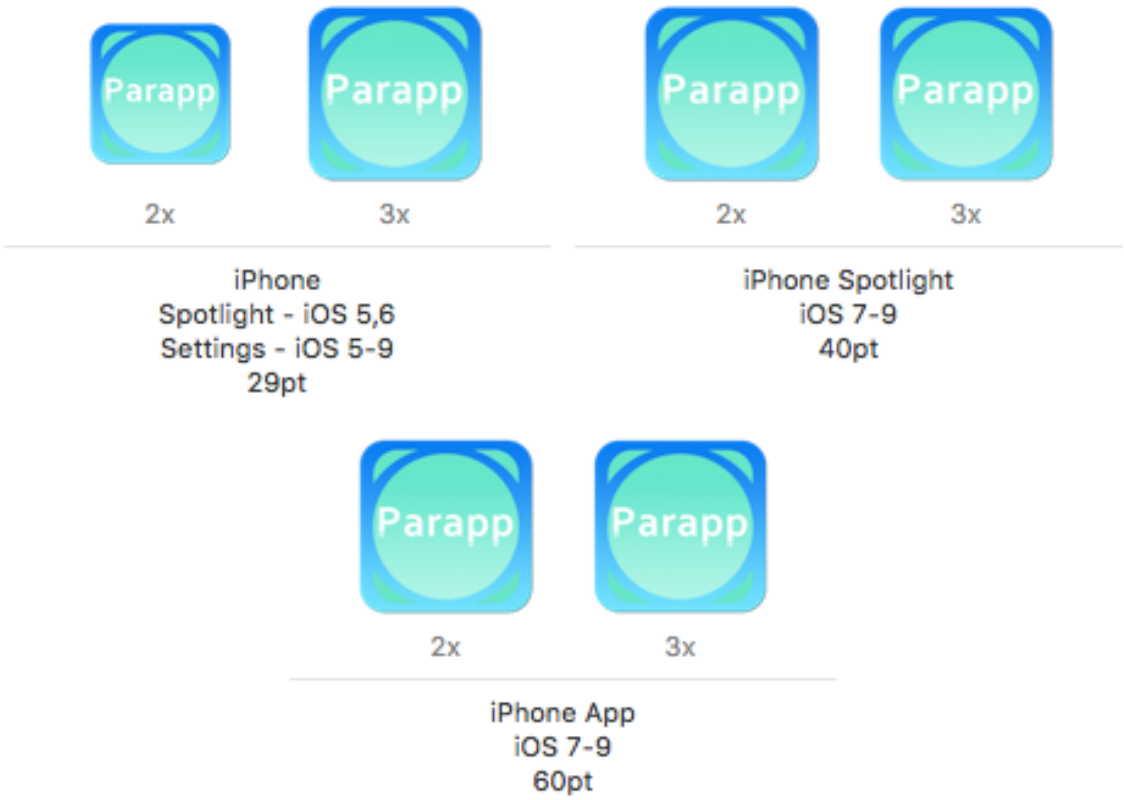


Ilustración 23 - Distintos tamaño de logos para las distintas versiones de iPhone e iOS

4.3.2. Pantalla de inicio (Launch Screen)

La pantalla de inicio de la aplicación o launch screen se diseñó utilizando como base los mismos colores que el icono de la aplicación, incluyendo el mismo en la parte superior.

Por definición, la pantalla de inicio es la que muestra el teléfono móvil mientras está realizando los procesos internos. Una vez que termina, la aplicación se dirige a la pantalla de inicio que se ha seleccionado en el storyboard. Esta pantalla es de transición y se mostrará en mayor o menor medida dependiendo de lo que el teléfono requiera hasta que termine todas las operaciones. Normalmente, este proceso no tarda más de unos cuantos segundos, por lo que la imagen no debería contener ningún tipo de información relevante para el uso de la aplicación, aunque si puede incluir información importante.

Aunque, no se incluye dentro de los objetivos de la aplicación, la pantalla de inicio puede ser también estática, y puede dar al usuario unas nociones básicas de cómo manejarse con la aplicación para que este sepa los pasos que hay que dar para su uso. Esto puede ser

interpretado como un manual de usuario para este. Lo importante en ese caso es que el usuario pueda elegir seguir viendo la pantalla de inicio o no.

La imagen de la derecha es el la imagen diseñada para ser usada en el launch screen, mientras que la imagen de la izquierda es el resultado de la pantalla de inicio en el simulador de Xcode.

Ilustración 24 - Pantalla de inicio diseñada y en el simulador

4.3.3. Diseño de Botones customizados

Para el diseño de los botones se decidió no usar los botones que vienen por defecto en Xcode, y con la ayuda de diferentes videos y libros para el diseño de botones en Swift se diseñó el controlador para los diferentes botones.

Aunque no es problema de técnico, sino más bien de estética, siempre es interesante desvincularse de los típicos botones que se usan y tratar de usar algún botón un poco más diferente. La única diferencia que se ha añadido a estos botones es la posibilidad de seleccionar un radio a las equinas del botón, de forma que con esto el botón tenga forma

redondeada. Para hacer esto se creó un nuevo controlador para los botones de esta clase. Este controlador hereda de la clase UIButton, por lo que solo los objetos de esta clase pueden usar este controlador.

```
import UIKit

@IBDesignable
class CustomButton: UIButton {
    @IBInspectable var borderColor: UIColor? =
UIColor.clearColor() {
        didSet {
            layer.borderColor = self.borderColor?.CGColor
        }
    }
    @IBInspectable var borderWidth: CGFloat = 0 {
        didSet {
            layer.borderWidth = self.borderWidth
        }
    }
    @IBInspectable var cornerRadius: CGFloat = 0 {
        didSet {
            layer.cornerRadius = self.cornerRadius
            layer.masksToBounds = self.cornerRadius > 0
        }
    }

    required init?(coder aDecoder: NSCoder) {
        super.init(coder: aDecoder)
    }

    override init(frame: CGRect) {
        super.init(frame: frame)
    }

    override func drawRect(rect: CGRect) {
        self.layer.cornerRadius = self.cornerRadius
        self.layer.borderWidth = self.borderWidth
        self.layer.borderColor = self.borderColor?.CGColor
    }
}
```

Código 4 - Clase UIButton 'CustomButton

Con esto se consiguen los botones como los mostrados en el apartado 4.2.2.

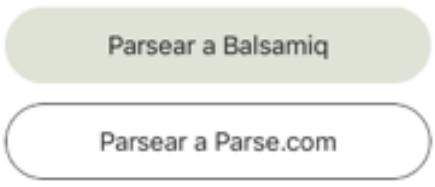


Ilustración 25 - Botones de la clase CustomButton

Una vez el en el storyboard, se selecciona que el botón es de la clase “CustomButton”, un nuevo campo para seleccionar el radio de las esquinas aparece en el inspector del botón.

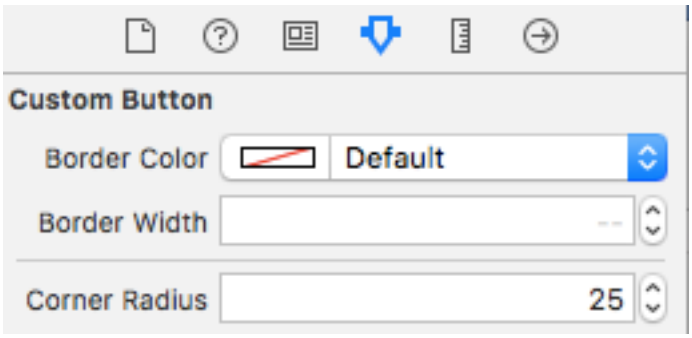


Ilustración 26 - Nuevo Campo para seleccionar el radio de las esquinas

Todos los botones de la aplicación comparten las mismas características y tiene un radio de 25.

4.4. Servicios integrados en la aplicación

En esta sección se explicarán y detallaran los diferentes servicios integrados en la aplicación.

4.4.1. Servicios de almacenamiento

Los servicios de almacenamiento en la nube se han convertido en aplicaciones muy populares, no solo para compartir archivos con otros usuarios sino también para almacenar copias de seguridad o para usar intercambiar archivos entre los dispositivos que el usuario tiene. Esto se debe a las diferentes API que los proveedores de servicios han lanzado en conjunto con las aplicaciones y que permite la proliferación de aplicaciones que integran su uso y hacen que la experiencia del usuario sea mucho más amigable.

Esto es lo que se pretende conseguir con la integración de un servicio de almacenamiento en la nube. Principalmente, el usuario tiene que usar diferentes programas gratuitos y guardar los archivos, con sus correspondientes extensiones en su ordenador para después procesarlos usando la herramienta diseñada en este PFC. Como la popularidad y la facilidad en el uso y registro de este tipo de aplicaciones, así como la gratuidad de almacenamiento básico, han ido creciendo, se pensó que usar como

herramienta de almacenamiento de los archivos generados por el usuario este tipo de servicio [10].

A continuación se presentan las principales ventajas y desventajas de cada uno de estos proveedores de servicios de almacenamiento en la nube [11] [12].

	Dropbox	**Google Drive**	**One Drive**
Compatibilidad	Android, BlackBerry, iOS, Linux, OS X, Windows NT, Windows Phone, Windows 10	Windows, OS X, Chrome OS, Android and iOS	Windows, OS X, Chrome OS, Android, iOS and BlackBerry
Almacenamiento Encriptado	Yes	Yes	No
API (iOS)	Yes	Yes	Yes
Sincronización de Archivos	Yes	Yes	Yes
Capacidad de almacenamiento máximo	132 GB	1 TB	207 GB
Capacidad de almacenamiento gratuito	18 GB	15 GB	25 GB

Tabla 1 - Comparación de servicios de almacenamiento

Aunque la facilidad de cara al usuario y sus principales características son importantes, a la hora de desarrollar y de integrar estos servicios en la aplicación móvil, lo que más importa es la API. No importa que las funcionalidades sean geniales si la API es una pesadilla de integrar.

Lo primero que hay que pensar cuando se habla de API es en la facilidad para hacer la autenticación. A este nivel, tanto Dropbox como Drive piden los mismos datos, así que no hay ninguna diferencia entre ellas. Las principales diferencias se encuentra en que Drive reduce la complejidad a la hora de realizar funciones. Con la misma función se pueden

realizar muchas más operaciones que con Dropbox, siempre y cuando se conozca el ID del archivo que se desea manejar. Todos los archivos y carpetas tienen un ID propio, así que si se desea usar una función en varios archivos o en varias carpetas, solo hace falta añadir todos los ID y la función actuará sobre todos esos ID. Esto reduce drásticamente el número de operaciones y de iteraciones en la aplicación.

Sin embargo, crear nuevas carpetas o elegir nuevas localizaciones se puede convertir en un problema grande en Google Drive. La utilización de ID y no de direcciones de carpetas hace que si, por ejemplo, se quiere añadir un archivo a una carpeta que está en un 5º o 6º nivel, se necesita el ID de cada una de las carpetas de todos los niveles.

Por el contrario, usando Dropbox, se tiene una dirección (path), que permite que la navegación sea mucho más sencilla. Además en la última versión de la API de Dropbox se puede elegir entre el uso de ID o de directorios, por lo que mejora la experiencia de programación. Es más, cada método puede ser usado con cualquiera de las dos, por lo que en el mismo código puede integrar los dos, lo que hace totalmente adaptable a lo que se pretende conseguir con la llamada a esa función.

No se pensó en el uso de One Drive, dado que no es un servicio tan popular como Dropbox y Google drive, y porque sobre todo no se tenía experiencia previa con su API. Aunque la aplicación va a usar este servicio como medio de almacenamiento y no se pretende procesar los datos ni mover los archivos de una carpeta a otra, se ha decidido usar Dropbox como servicio, dado que se tiene un poco de experiencia con la API y ha sido la recomendada por uno de los tutores del proyecto.

La API de Dropbox que se ha usado ha sido la Dropbox CORE API v1. Esta API integra todos los servicios que son necesarios para la aplicación. Esto representa un problema a largo plazo dado que a mediados de 2017 esta API será depreciada por Dropbox y se usará solo la API v2. Como el proyecto se ha realizado con anterioridad, no entra dentro de los planes la integración con la API v2, aunque el proceso de integración no supone grandes cambios.

Para empezar a usar la API hay que instalar el iOS SDK. Una vez instalado y seleccionado la aplicación que se desea usar y de sincronizar la aplicación con Dropbox llega el momento de usar los métodos presentados en la API para la autentificación. Para ello hay que

proporcionar las dos claves que permiten el uso de la API: “app_key” y “app_secret”. Estos dos datos se tienen que añadir en el App Delegate de la aplicación. Una vez proporcionadas, se puede iniciar la sesión en la vista.

Además de esto, se tiene que crear un esquema de URL. El SDK de Dropbox se coordina perfectamente con Dropbox App, que se instaló previamente, pero para poder usar un sistema mucho más sencillo de usar y de navegar, se necesita crear un esquema URL único en la aplicación. Este proceso llevo un tiempo bastante largo dado que en un principio la guía de uso y de instalación de Dropbox estaba explicada para iOS8, pero cuando se decidió usar este servicio se estaba usando un simulador de iOS9.

Aunque en un principio parece que no debería ser un problema importante, se convirtió en un gran problema que se tardó bastante en solucionar y solo se pudo solucionar con la ayuda de otros desarrolladores y el uso de la plataforma “StackOverFlow”. Para crear el esquema para iOS8 o anterior hay que seguir los siguientes pasos:

- Añadir un nuevo tipo de URL en el proyecto.
- En el esquema de URL añadir uno nuevo único usando el “app_key”.

Hasta aquí todo funciona con iOS8, pero cuando se pasa a iOS9 hay que añadir los permisos a la lista de esquemas autorizados en la aplicación. Este fue el principal paso que complico el uso de esta API. Para añadir esta instancia a la lista de esquemas autorizados solo hay que dar dos pasos:

- Abrir la lista de info.plist y añadir una nueva llave “key”: LSApplicationQueriesSchemes
- Añadir sobre esta nueva llave un nuevo elemento con valor “dbapi-2”

Al final, el info.plist tiene que tener un aspecto como el siguiente:

```
<key>LSApplicationQueriesSchemes</key>
    <array>
        <string>dbapi-2</string>
        <string>dbapi-8-emm</string>
    </array>
```

Código 5 - Nuevo esquema del archivo info.plist

Pasado el tiempo, Dropbox incluyo en su documentación de inicio los pasos necesarios para sincronizar igualmente con iOS9 [13].

En general, el info.plist tiene que incluir todas las claves que se muestran en la siguiente imagen. Todas estas claves son necesarias para el correcto funcionamiento de la aplicación.

Key	Type	Value
▼ Information Property List	Dictionary	(16 items)
▼ LSApplicationQueriesSchemes	Array	(2 items)
Item 0	String	dbapi-2
Item 1	String	dbapi-8-emm
Localization native development re...	String	en
Executable file	String	$(EXECUTABLE_NAME)
Bundle identifier	String	$(PRODUCT_BUNDLE_IDENTIFIER)
InfoDictionary version	String	6.0
Bundle name	String	$(PRODUCT_NAME)
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.0
Bundle creator OS Type code	String	????
▼ URL types	Array	(1 item)
▼ Item 0 (Editor)	Dictionary	(3 items)
Document Role	String	Editor
URL identifier	String	dropbox
▶ URL Schemes	Array	(1 item)
Bundle version	String	1
Application requires iPhone enviro...	Boolean	YES
Launch screen interface file base...	String	LaunchScreen
Main storyboard file base name	String	Main
▶ Required device capabilities	Array	(1 item)
▶ Supported interface orientations	Array	(3 items)

Ilustración 27 - Vista completa del archivo info.plist

Por último, para que la aplicación este totalmente sincronizada y que se pueda trabajar con esta base de datos, solo hace falta incluir un poco más de código para completar el link entre la aplicación y Dropbox. Todo este código se incluye en el delegate de la aplicación, para asegurar de que antes de que se sincronice la aplicación no haya ningún tipo de error, y en caso de que lo haya se guarde el error que se ha producido. Todas estas comprobaciones son básicas y provienen de las propias recomendaciones de Dropbox para asegurar un correcto inicio de sesión.

```
NSString* appKey = @"wklae188huon03e";
NSString* appSecret = @"4dxvkocwt45msom";
NSString *root = kDBRootDropbox;

NSString* errorMsg = nil;
```

```
if ([appKey rangeOfCharacterFromSet:[[NSCharacterSet alphanumericCharacterSet]
invertedSet]].location != NSNotFound) {
   errorMsg = @"Asegúrese de que la app_key es correcta en AppDelegate.m";
} else if ([appSecret rangeOfCharacterFromSet:[[NSCharacterSet
alphanumericCharacterSet] invertedSet]].location != NSNotFound) {
   errorMsg = @"Asegúrese de que la app_secret es correcta en AppDelegate.m ";
} else if ([root length] == 0) {
   errorMsg = @"Incluya el root para usar App Folder of full Dropbox";
} else {
    NSString *plistPath = [[NSBundle mainBundle] pathForResource:@"Info"
ofType:@"plist"];
    NSData *plistData = [NSData dataWithContentsOfFile:plistPath];
    NSDictionary *loadedPlist = [NSPropertyListSerialization
propertyListWithData:plistData options:0 format:NULL error:NULL];
    NSString *scheme = [[[[loadedPlist    objectForKey:@"CFBundleURLTypes"]
objectAtIndex:0] objectForKey:@"CFBundleURLSchemes"] objectAtIndex:0];
    if ([scheme isEqual:@"db-APP_KEY"]) {
            errorMsg = @"Configure el esquema de URL correctamente en
DBRoulette-Info.plist";
    }
}

DBSession *dbSession = [[DBSession alloc]
                        initWithAppKey:appKey
                        appSecret:appSecret
                        root:root];

[DBSession setSharedSession:dbSession];
```

Código 6 - Inicialización de Dropbox

Con esto conseguimos que se inicie la sesión, pero no que el usuario se identifique e indique en que carpeta o que archivos son los que desea incluir en el análisis posterior. Esto se analizará más adelante, una vez explicado la vista y el controlador.

5. Arquitectura del módulo Balsamiq a Parse.com

Como se explico en apartados anteriores, el sistema se basa en varios módulos que se encargan de analizar la información y convertirla en el resultado esperado por el usuario. En este apartado se pretende analizar el modulo diseñado para convertir la información extraída de Balsamiq y convertirla en el formato que Parse.com puede entender.

Este se trata del módulo inicial con el que se planteó el proyecto. Los principales objetivos que se pretendían conseguir con este módulo son los siguientes:

- El usuario diseña la aplicación móvil usando los Mockup y una vez se han diseñado todos, se puede obtener la aplicación móvil gracias a otras herramientas diseñadas por terceros.
- El usuario no necesita tener conocimientos de programación para diseñar su aplicación.
- Creación de plantillas de aplicaciones para diferentes mercados, por lo que puede ser comercializada como SaaS[1].

Para llevar la creación correcta del módulo, hay que conocer de dónde viene el sistema que hace que de Parse.com podamos dar con aplicaciones móviles, de forma que le demos a Parse.com la información tal cuál la necesita.

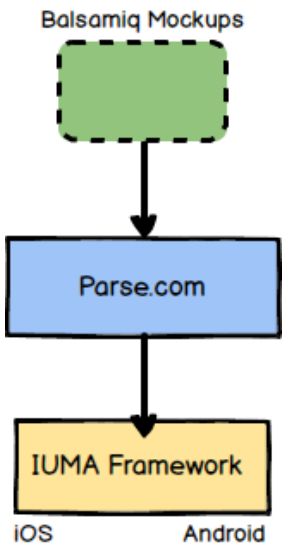


Ilustración 28 - Diagrama de Bloques del módulo Balsamiq

Como bien explicamos en el apartado 3, el módulo de Balsamiq se va a conectar a Parse.com, y de ahí se conecta el IUMA Framework. Los frameworks son software que se diseñan con la única función de ayudar en la creación de otros frameworks. En el caso que nos ocupa ayuda a crear aplicaciones móviles. Este sistema se creó usando como guía un plugin para Wordpress llamado SpiderCatalog. Este plugin trabaja sobre una base de

[1] Software as a Service (SaaS). Se trata de un servicio que tiene como modelo vender licencias de uso basadas en suscripciones que los usuarios pueden obtener de diferentes maneras. La principal característica es que el servicio está centralizado y si el usuario no paga las cuotas establecidas, el proveedor de servicios corta el acceso, pero no elimina los datos, pudiendo el usuario reestablecer los pagos y obtener los servicios nuevamente. [14] [15]

datos en MySQL que modifica de forma que pueda trabajar sus propios sistemas de archivos. Aunque inicialmente se usó este plugin, posteriormente se han usado otro tipo de módulos para alimentar de datos Parse.com de una forma lineal.

En este capítulo se presentarán los siguientes puntos.

- Presentación del plugin SpiderCatalog y su arquitectura. En este apartado se explicará el formato que han de tener los datos en Parse.com.
- Presentación de Balsamiq Mockups y de las principales herramientas que se van a utilizar.
- Descripción de cada una de las vistas de la aplicación que se pretenden parsear.
- Arquitectura del módulo y descripción de las principales funciones. Descripción de los patrones de diseños implementados. De desarrolla el método de conexión con Parse.com así como las alternativas en caso de que la conexión no estuviera disponible.
- Desarrollo de un ejemplo para mostrar el funcionamiento individual del módulo.

5.1. SpiderCatalog.

SpiderCatalog es un plugin de Wordpress que permite y facilita al usuario la organización jerarquizada de productos por catálogos. Para realizar esta función, todos sus datos se dividen en dos elementos principales, que son las categorías y los productos. Las categorías representan conjuntos de productos y estos a su vez pueden pertenecer a una o más categorías de las que se compone el total del catálogo.

Las categorías pueden estar jerarquizadas, por lo que podemos tener una mejor organización del catálogo, de esta manera una categoría puede pertenecer a otra, que a su vez pertenece a otra. Cada categoría tiene una serie de parámetros, que permiten ver toda la información relevante de la misma. La siguiente tabla muestra todos los parámetros de los que se componen las categorías así como una breve explicación de cada uno de ellos.

PARÁMETROS DE LAS CATEGORÍAS	
Nombre	**Descripción**
Name	Nombre de la categoría. No ha de ser único.
Description	Descripción de la categoría.
Image_url	Dirección URL de la imagen asociada a la categoría.
Parent	Categoría padre a la que pertenece. Se trata de un carácter numérico y se coge el ID de dicha categoría.
Param	Parámetros por los que se define la categoría. Estos serán explicados más adelante.
Spider_id	Identificador de la categoría. Ha de ser único en todo el catálogo.
Ordering	Orden de la categoría. Normalmente utilizado para indicar como han de ser representadas las categorías.
Published	Indica si la categoría debe ser visible o no. En la mayoría de los casos, el framework del IUMA ignora este parámetro.

Tabla 2 - Parámetros de las categorías

Por otro lado, los productos, son todos los elementos del catálogo que se clasifican en función de la categoría o categorías a las que pertenecen. No hay límite de categorías a las que puede pertenecer un mismo producto. De igual manera que las categorías tienen una serie de parámetros, los productos también tienen parámetros por los que proporcionan toda la información necesarias sobre ellos mismos. La siguiente tabla representa los parámetros disponibles en las categorías.

PARÁMETROS DE LOS PRODUCTOS	
Nombre	**Descripción**
Name	Nombre del producto. No ha de ser único.
Description	Descripción del producto.

Category_id	Identificador de la categoría a la que pertenece. Pueden ser una combinación de ellos si depende de varios y han de estar separados por comas. El identificador se toma el spider_id de la categoría a la que pertenece
Image_url	Dirección URL de la imagen asociada al producto.
Spider_id	Identificador del producto. Ha de ser único.
Cost	No se utiliza para este proyecto.
Param	Parámetros por los que se define la categoría. Estos serán explicados más adelante.
Market_cost	No se utiliza para este proyecto
Ordering	Orden del producto. Normalmente utilizado para indicar como han de ser representados los productos.
Published	Indica si el producto debe ser visible o no. En la mayoría de los casos, el framework del IUMA ignora este parámetro.

Tabla 3 - Parámetros de los productos

Cabe destacar que los param de las categorías declaran cuales han de ser los param de los productos. Por esto, los principales param son los que se representan en la siguiente tabla.

PARAMETROS PARA EL PARAMETRO "PARAM"	
Nombre	**Descripción**
Type	Tipo de vista que se va a usar
Resource	Recurso, normalmente si contiene un link a un externo
Subtype	Se ignora este parámetro
Action	Indica el texto de algún botón en alguna pantalla. Se explica más adelante

Tabla 4 - Parámetros del parámetro parama

Aunque estos son los principales que tienen todas las categorías, hay un tipo de categoría que contiene algunos más, que normalmente es el que posee el spider_id igual a cero. Estos parámetros solo pueden estar presentes en una categoría y representan los valores más importantes del catálogo, colores, tipo de fuente, etc. Estos tipos de parámetros se añadieron posteriormente, cuando se adaptó el SpiderCatalog a la producción de aplicaciones móviles.

PARAMETROS PARA EL PARAMETRO "PARAM" ESPECIALES	
Nombre	**Descripción**
strongColor	Color usado para configurar la aplicación
mediumColor	Color usado para configurar la aplicación
lightColor	Color usado para configurar la aplicación
fontType	Tipo de fuente para configurar la aplicación
fontSize	Tamaño de fuente para configurar la aplicación

Tabla 5 - Parámetro del parámetro "param" especiales

Entre todos estos param, hay dos que son especiales y que son necesarios para todas las categorías y productos. Estos son los param "type" y "resource". Estos parámetros indican que tipo de categoría y que tipo de producto es el que tenemos. Dependiendo de esto, luego la vista que se representará en la aplicación será una u otra. Los principales tipos que existen son los siguientes.

Tipos de vistas disponibles	
Tipo (type)	**Descripción**
Mail	Pantalla para enviar un email, por defecto del teléfono
Phone	Tipo que permite conectar para realizar una llamada.
HTML	Tipo básico de texto HTML enriquecido
PDF	Lector de PDF
URL	Tipo que abre una dirección URL dada

Products	Vista que permite visualizar la información de productos
Gallery	Vista que permite visualizar en una lista los productos
LeftMenu	Menú de la aplicación, situado a la izquierda
Glossary	Glosario de palabras que dirigen a sus productos
Copyright	Menú para definir los copyright que pueda tener la app.

Tabla 6 - Tipos de vistas disponibles

Actualmente se están desarrollando más tipos, pero no entran dentro de los objetivos de este proyecto.

De esta forma, se ha dejado claro los principales conceptos sobre los que se basa SpiderCatalog. Como se ve, este ha sido adaptado para utilizar de una forma más eficaz su forma de catálogo. Uno de los principales inconvenientes es que una vez que los datos están introducidos en Parse.com estos adquieren una forma lineal o de tabla. Esto es un inconveniente, dado que la estructura con la que se trabaja es jerárquica, es decir en forma de árbol, como se representa en la siguiente imagen.

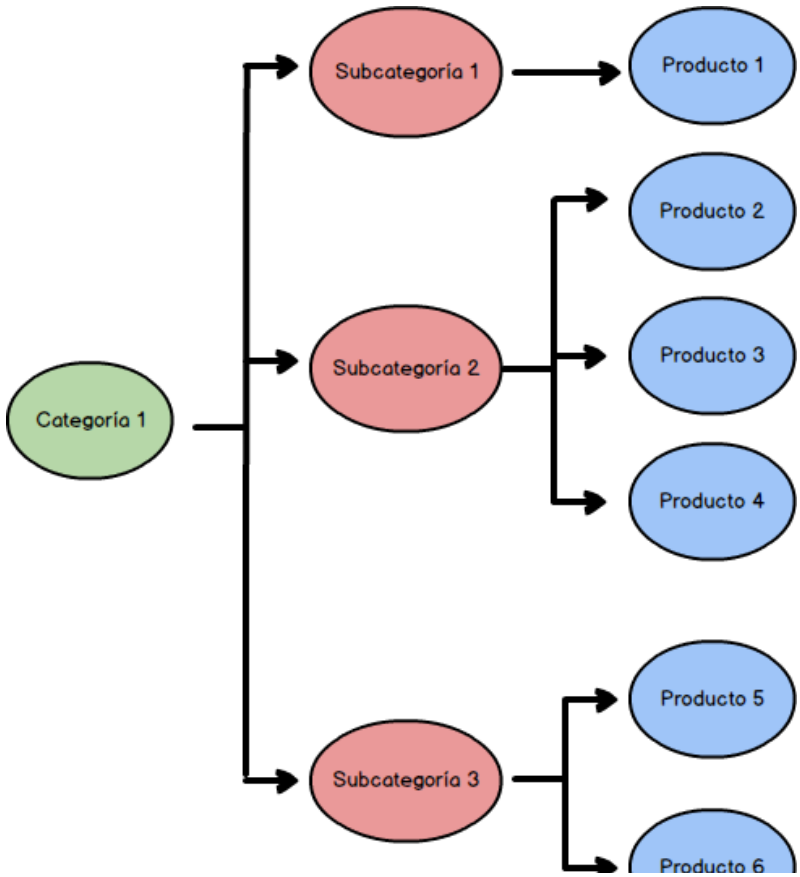


Ilustración 29 - Distribución jerárquica de SpiderCatalog

El modelo que se utiliza es el de categoría/parámetro y producto/valor. De esta forma, dentro de la estructura jerarquizada que estamos utilizando obtenemos distintas categorías y subcategorías que descienden de otras categorías. Estas categorías las vamos a utilizar para declarar parámetros, que serán usados en los productos. Estos productos, rellenaran con valores estos parámetros, dando como resultado la aplicación. En el caso de las subcategorías, estas heredarán los parámetros de la categoría de la que son hijos, por lo que es un factor a tener muy en cuenta a la hora de anidar categorías.

Cada categoría ha de tener al menos un producto. Este producto declara los principales valores de la categoría, y normalmente lo llamamos metadata. Estos metadata suelen compartir los mismos parámetros con muchas categorías, por lo que normalmente heredan de varias categorías. Si una categoría no tiene este producto, está estará incompleta.

En resumen, los dos elementos principales cumplen las siguientes funciones:

- **Categorías**. Son los elementos principales del catálogo. Podemos darles un nombre y orden en el que deben ser mostrados. Estas pueden estar anidadas en otras categorías, que denominamos subcategorías, por lo que es fácil ordenar y clasificar la importancia de las mimas, así como establecer niveles entre unas y otras. También podemos declarar los tipos de parámetros que queremos que sus hijos hereden, y establecer un vínculo jerárquico con ellos. Todas las categorías han de tener al menos un producto.
- **Productos**. Son los elementos que proporcionan el valor al catálogo. El nombre y la descripción suele ser la parte más importante de los mismos, así como los parámetros. Siempre existe un parámetro denominado metadata, que proporciona los parámetros de la categoría a la que pertenece. Al igual que las categorías, puede otorgárseles un orden de representación. Algunos parámetros no han sido integrados con el framework del IUMA, y aunque deben aparecer para respetar la estructura del catálogo, no tienen ninguna utilidad.

5.1.1. Relación con Parse.com

Parse es la plataforma que se ha decidido usar en el sistema como Backend para el almacenar la información de las aplicaciones en forma de tablas.

Esta plataforma ofrece un potente servicio de gestión de base de datos usando la nube que normalmente es usado para el desarrollo de aplicaciones móviles o de entornos web. Especialmente destaca por ser uno de los servicios más fáciles de uso y más rápidos como backend. La documentación que proporciona está bastante completa y ofrece un servicio de foros para la colaboración mutua.

Los principales servicios que ofrece son los siguientes:

- Creación de tablas no-SQL en la nube. Estas tablas pueden ser consultadas y modificadas, tanto por inserción o eliminación mediante el uso de la API que proporciona.
- Notificaciones push. Esta función no será usada en el proyecto.
- Potente cloud code, permitiendo la ejecución de código en la nube. Es muy útil para aumentar la seguridad de las aplicaciones, así como la posibilidad de actualizar los datos rápidamente. Esta función tampoco será utilizada en este proyecto.

El siguiente dibujo representa como funciona teoréticamente Parse.com.

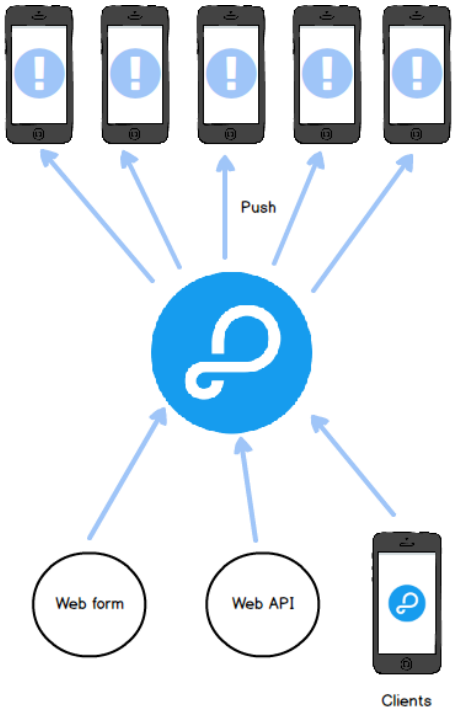


Ilustración 30 - Funcionamiento teórico de Parse.com

Para que Parse.com funcione correctamente, ha de ser configurado en la misma forma que requiere el framework del IUMA. Como Parse.com funciona con tablas, hay que crear dos tablas, una para los productos y otra para las categorías. Además los nombres de cada una de las tablas han de ser concretos. Para las categorías se usa: wp_spidercatalog_product_categories, mientras que para los productos se usa: wp_spidercatalog_products. Estas tablas incluirán toda la información que hace falta.

Como sabemos, las tablas están compuestas de columnas y filas. Las filas serán las que definan la información, mientras que las columnas indicaran que información define esa celda. Por ello hay que crear las columnas que necesitamos. Todas las columnas de Parse.com que usamos han de ser tipo String. Parse permite que se creen distintos tipos de columnas que tendrán formatos diferentes, como tipo date para expresar que es una fecha, o tipo int para expresar que es un entero. En nuestro caso, el framework del IUMA requiere que todos los campos sean de tipo String. La siguientes imágenes muestran cómo se representa la información en Parse.com para los productos y las categorías.

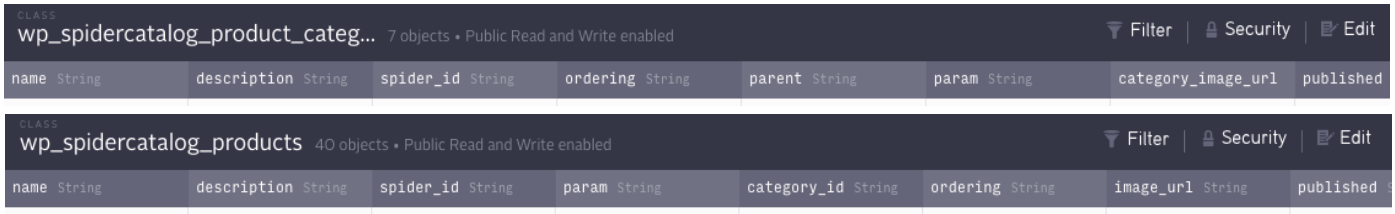


Ilustración 31 - Ejemplo de las clases que necesita Parse.com

Una vez que las tablas han sido generadas, Parse.com está listo para recibir la información y desde ahí distribuirla a las instalaciones en los teléfonos móviles. Cabe destacar que Parse.com ha anunciado que dejará de ofrecer el servicio de BaaS[2] a finales de enero de 2017. Aunque esto puede suponer un gran problema, Parse.com ha preparado una muy buena documentación de cómo generar tu propia base de datos y exportarlos para lanzarlo en la producción de todas las aplicaciones [14].

Por último, de Parse.com necesitaremos las claves que son necesarias para la conexión de la API con el servidor. Estas claves se pueden encontrar en la sección derecha de la página de la aplicación en la web, concretamente en la sección de settings. Dentro de settings, tenemos que ir a Security & Keys. Ahí tenemos todas las claves que podemos

[2] Backend as a service (BaaS). Es un servicio que ofrece a los desarrolladores web y de aplicaciones móviles de forma que estos pueden conectar sus aplicaciones con servicios de backend.

obtener para diferentes usos. Las claves que son necesarias para el uso de Parse.com y la aplicación que desarrollamos son las de Application ID o también conocida como Parse ID y la de REST API key. La siguiente imagen muestra cómo podemos encontrar las claves.

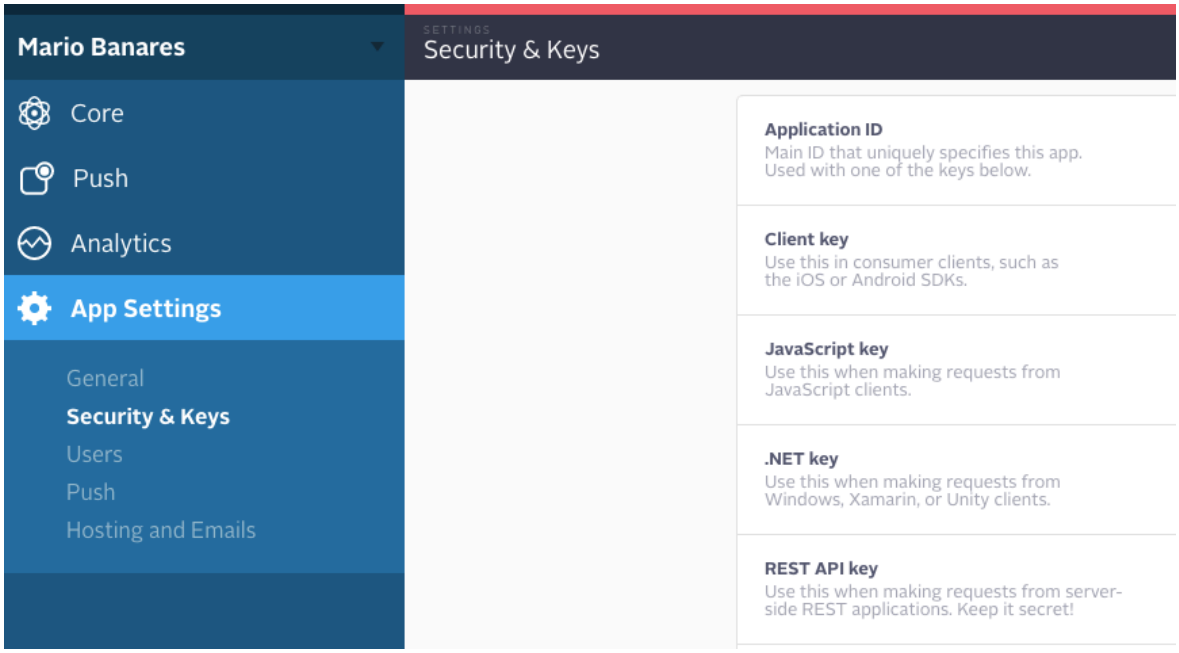


Ilustración 32 - Claves de Parse.com

5.1.2. Reglas de diseño de SpiderCatalog y Parse.com

Para poder usar correctamente SpiderCatalog y Parse.com hemos de seguir una serie de reglas en el diseño de las aplicaciones.

- Las categorías declararan los parámetros que han de contener los productos.
- Los productos han de contener todos los parámetros declarados por la categoría de la que heredan.
- La categoría principal lleva siempre el id 1. Este id dará unos privilegios a esta categoría. Las demás llevaran numeración consecutiva. Dos categorías no pueden tener el mismo id.
- La categoría principal de la aplicación ha de tener parent 0. Es el inicio del árbol. El resto de categorías han de tener un padre. El número del padre será tomado del id.
- Las subcategorías heredaran los parámetros de su categoría padre.

- Todas las categorías han de tener al menos un producto. Este producto puede ser llamado con el mismo nombre que su categoría o con el nombre de metadata. Esto indica que este producto declara los valores de los parámetros de la categoría de la que hereda.
- Puede existir categorías, subcategorías y productos con el mismo nombre, pero no dentro del mismo nivel y heredando del mismo padre.
- Un mismo producto puede pertenecer a una o varias categorías (como por ejemplo los metadatos). Puede ocurrir que estén incluso en diferentes niveles jerárquicos. En este caso, las categorías de ambos deben tener declarados los mismos parámetros.

5.2. Balsamiq Mockup.

Como se explicó en el capítulo 2 de esta memoria, Balsamiq Mockup es una herramienta que permite diseñar Mockup de aplicaciones, tanto web como móviles. Los Mockup son usados por los desarrolladores para tener un primer borrador de cómo desean que sean sus aplicaciones. Una vez que tienen diseñado el borrador pueden mostrarlo a las partes interesadas y obtener la confirmación de que lo que han diseñado es lo que están esperando.

Balsamiq tiene dos versiones, una versión web y una versión de escritorio. La versión de escritorio está basada en un sistema de suscripción, aunque también puede ser comprada. Para este proyecto nos centraremos en la versión web, que no es gratuita, pero de la que se consiguió tener una cuenta del IUMA para poder realizar el proyecto.

Una de las mejores características que tiene la versión web es la posibilidad de crear diferentes proyectos, de forma que se pueden agrupar los diferentes Mockup en diferentes proyectos, lo que para nosotros serán diferentes aplicaciones móviles. Otra buena característica es que en la vista por proyectos se puede ver cuantos Mockups incluye cada proyecto, por lo que cada Mockup es una vista. También se puede cambiar el color del contenedor del proyecto, por lo que se puede hacer otro tipo de división por colores para identificar los diferentes tipos de proyectos que se están manejando.

Por último, permite tanto crear un proyecto nuevo, como importar un proyecto. Esta última será usada a menudo durante el proyecto, y en caso de futuros usuarios también, dado que permite exportar e importar todo una colección de Mockups. Esta característica podría ser usada como importador de plantillas. La siguiente imagen muestra una captura de pantalla de la pantalla principal, remarcando los principales puntos comentados.

Ilustración 33 - Pantalla de proyectos de Mockup

Balsamiq contiene una amplia gama de objetos que pueden ser añadidos a los proyectos en su editor web. En el editor de escritorio, aparte de tener los mismos objetos que en la web, permite que el usuario añada todos los objetos extra que desee, y se quedan guardados en la memoria del programa para futuros proyectos.

El editor web es bastante completo y bastante sencillo de usar. Tiene tres secciones diferenciadas.

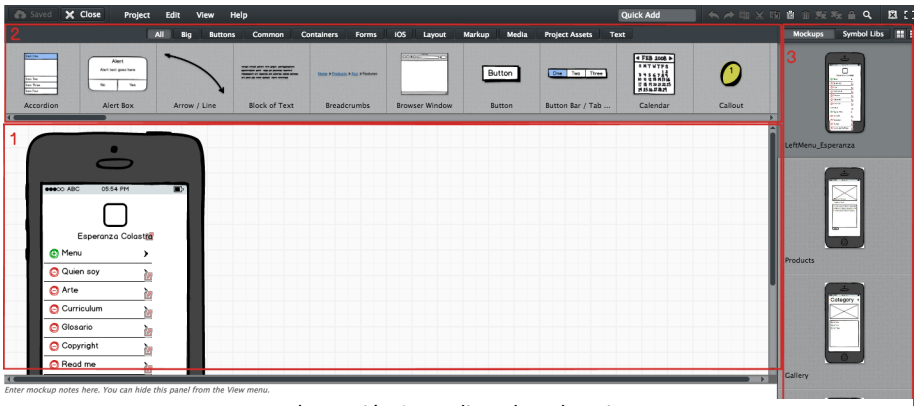


Ilustración 34 - Editor de Balsamiq

- La primera sección es la de edición del Mockup. En esta pantalla se puede dibujar cada una de las vistas que nuestro diseño va a tener. En el proyecto se usa una edición por cada pantalla que la aplicación tiene.
- La segunda es donde están todos los objetos que se pueden añadir. La siguiente imagen muestra más en detalle estos objetos.

Ilustración 35 - Objetos del editor

- La tercera es la sección para navegar por los diferentes Mockups que incluye el proyecto. En nuestro caso, cada Mockup es una vista diferente de la aplicación. No hay límite de unidades que se pueden tener por proyecto.

En el caso de este PFC, todos los Mockups están predefinidos y el usuario tendrá que importar las plantillas de las diferentes vistas, de las que podrá cambiar el texto e imágenes. En general, el usuario rellena de información las vistas que desea usar.

Pero sin lugar a dudas, la funcionalidad más importante que tenemos disponible en la versión web es la de exportar a XML los Mockup. Este fue el factor fundamental para seleccionar este programa y no otros. Los XML de cada uno de los Mockup comparten una misma arquitectura y todos son generados de la misma manera, por lo que el único problema era el de analizar el primer Mockup, y después todos los demás serían más sencillos de integrar. El siguiente código, muestra el ejemplo de uno de los Mockup que se presentarán más adelante, y como es su XML.

```
<mockup version="1.0" skin="sketch" fontFace="Balsamiq Sans" measuredW="342" measuredH="593" mockupW="282" mockupH="573">
 <controls>
  <control controlID="0" controlTypeID="__group__" x="60" y="20" w="282" h="573" measuredW="282" measuredH="573" zOrder="0" locked="false" isInGroup="-1">
   <controlProperties>
    <controlName>Menu%20Izquierda</controlName>
    <customData>App%2520Name</customData>
    <customID>MenuIzquierda</customID>
   </controlProperties>
   <groupChildrenDescriptors>
```

```
    <control controlID="0" controlTypeID="com.balsamiq.mockups::iPhone" x="0" y="0" w="-1" h="-1" measuredW="282" measuredH="573" zOrder="0" locked="false" isInGroup="0">
      <controlProperties>
       <model>IPhone5</model>
      </controlProperties>
     </control>
     <control controlID="3" controlTypeID="com.balsamiq.mockups::Image" x="52" y="128" w="178" h="107" measuredW="77" measuredH="79" zOrder="1" locked="false" isInGroup="0">
      <controlProperties>
       <text/>
      </controlProperties>
     </control>
     <control controlID="5" controlTypeID="com.balsamiq.mockups::Label" x="63" y="214" w="156" h="-1" measuredW="64" measuredH="21" zOrder="2" locked="false" isInGroup="0">
      <controlProperties>
       <text>Segunda Entrada</text>
      </controlProperties>
     </control>
     <control controlID="6" controlTypeID="com.balsamiq.mockups::Label" x="64" y="237" w="-1" h="-1" measuredW="50" measuredH="18" zOrder="3" locked="false" isInGroup="0">
      <controlProperties>
       <size>10</size>
       <text>Curriculum</text>
      </controlProperties>
     </control>
     <control controlID="7" controlTypeID="com.balsamiq.mockups::TextArea" x="52" y="262" w="178" h="166" measuredW="200" measuredH="140" zOrder="4" locked="false" isInGroup="0">
      <controlProperties>
       <text>Esto es una prueba de la segunda entrada, para probar como funciona exportar XML.</text>
      </controlProperties>
     </control>
     <control controlID="8" controlTypeID="com.balsamiq.mockups::Button" x="53" y="428" w="46" h="15" measuredW="42" measuredH="27" zOrder="5" locked="false" isInGroup="0">
      <controlProperties>
       <href>http%3A//esperanza@esperanzacolastra.com</href>

       <map>%3Carea%20shape%3D%22rect%22%20coords%3D%2253%2C428%2C99%2C443%22%20href%3D%22http%3A//esperanza@esperanzacolastra%22%20alt%3D%22http%3A//esperanza@esperanzacolastra%22%20id%3D%228%22/%3E</map>
```

```
        <size>11</size>
        <text>Mail</text>
       </controlProperties>
      </control>
     </groupChildrenDescriptors>
    </control>
  </controls>
</mockup>
```

Código 7 - Código HTML ejemplo de un Mockup

5.3. Mockup de las vistas integradas en IUMA framework

Las aplicaciones diseñadas debajo del paraguas del framework del IUMA se basan en varias vistas. Aunque actualmente, este se compone de más, el objetivo de este proyecto se centra en las vistas más usadas y más críticas, que en total son 6. Estas son las de los siguientes tipos:

Tipos de vistas disponibles
HTML
Products
Gallery
LeftMenu
Glossary
Copyright

Tabla 7 - Tipos de vistas disponibles

Además de esas vistas, existen vistas nativas de los Smartphone que no son necesarias de dibujar, como son las siguientes:

Tipos de vistas disponibles
Mail
PDF
URL

Tabla 8 - Tipos de vistas disponibles sin necesidad de usar una vista

Por tanto, si tenemos en cuenta todas las vistas, el alcance de los Mockup que se podrán analizar usando este motor es de 9. Todos y cada uno de los diferentes tipos tiene una o varias vistas, es decir, el tipo Glossary, consta de varias vistas, aunque para el usuario sea una única vista.

Para poder empezar a analizar los Mockup, lo primero que tenemos que hacer es diseñar todas las posibles vistas primeramente en Balsamiq. La siguientes imágenes representan a la izquierda la vista original, de la que se hizo una captura de pantalla en un iPhone 6 y a la derecha el resultado que se alcanzó usando Balsamiq Mockup.

- **Menú Izquierda**. También se conoce en el sistema como “leftMenu”.

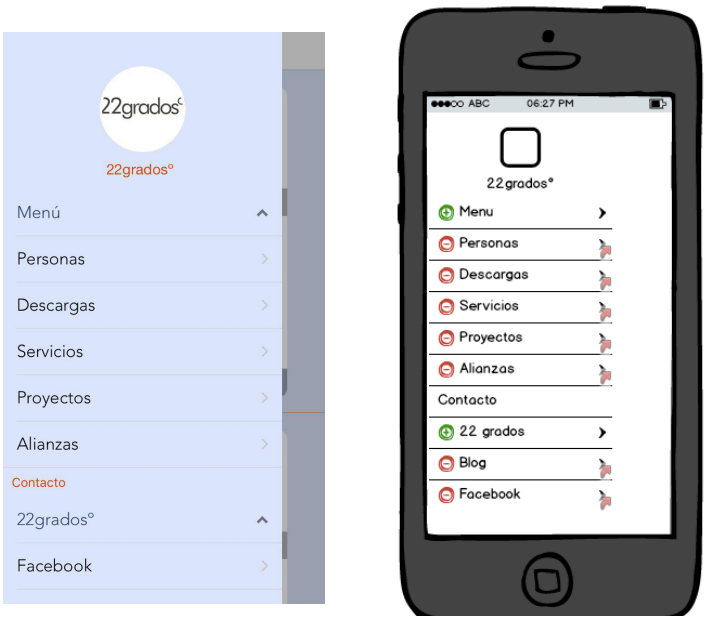


Ilustración 36 - Comparativa de leftMenu real con Mockup

- **HTML**. También se usa como complemento al tipo Glossary y al tipo Copyright.

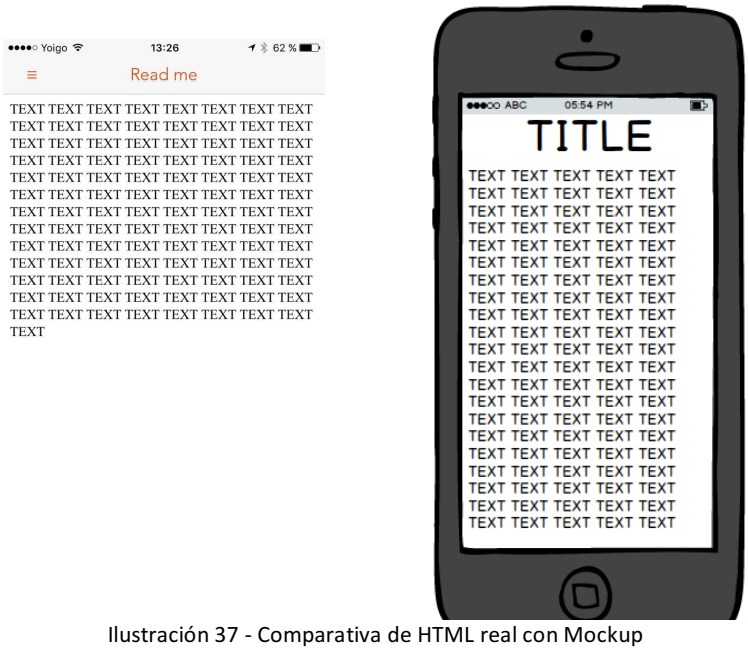


Ilustración 37 - Comparativa de HTML real con Mockup

- **Copyright**. Necesita complementarse con los Mockup de HTML, dado que cuando se selecciona una de las opciones que se incluyen en el Mockup, aunque no hará falta tener link internos en el Mockup.

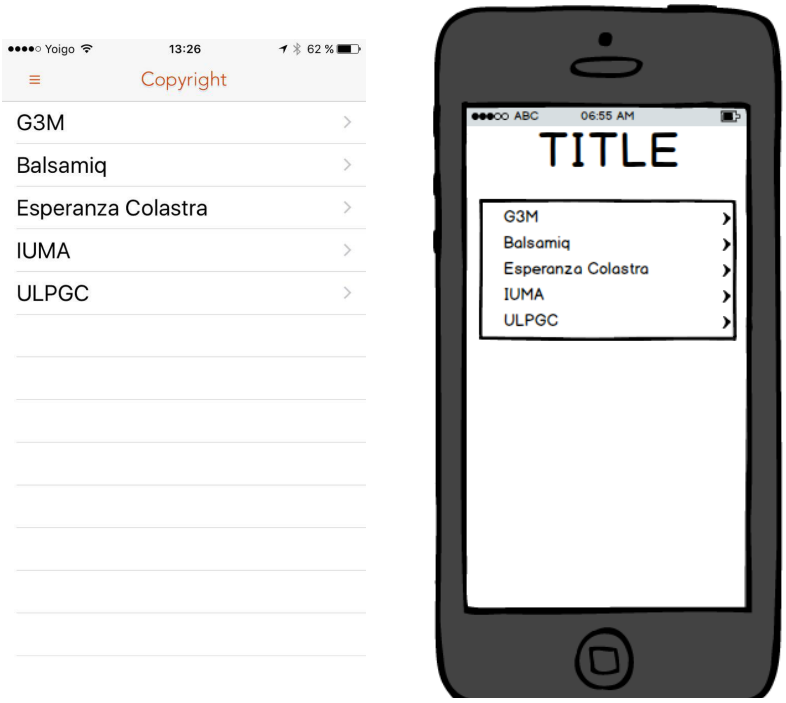


Ilustración 38 - Comparativa de Copyright real con Mockup

- **Products**. Se trata de uno de los Mockup más importante y más ampliamente utilizado en todas las aplicaciones.

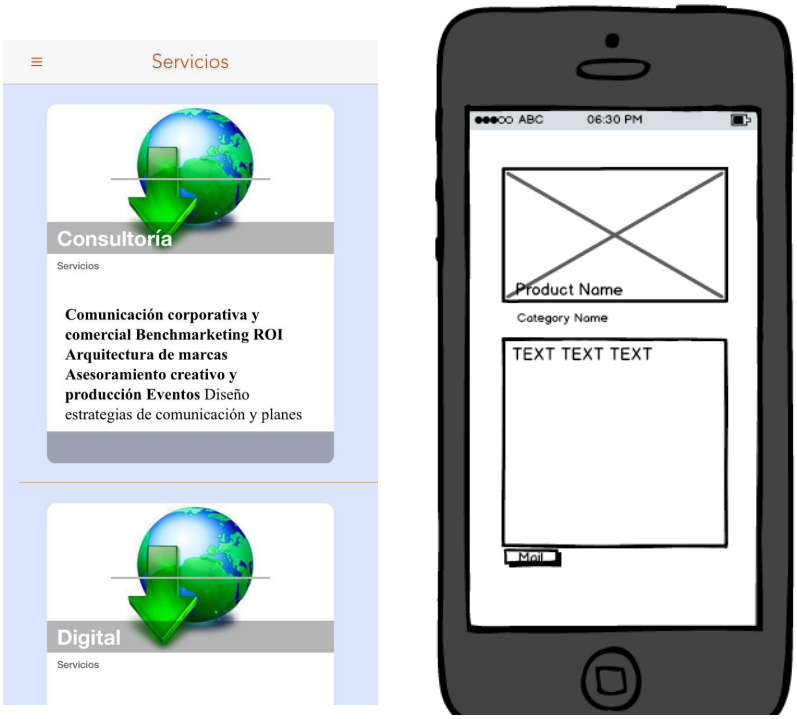


Ilustración 39 - Comparativa de Products real con Mockup

- **Gallery**. Junto con el tipo productos es el más importante de las vistas disponibles. Los diferentes ítems pueden conducir únicamente a diferentes URL.

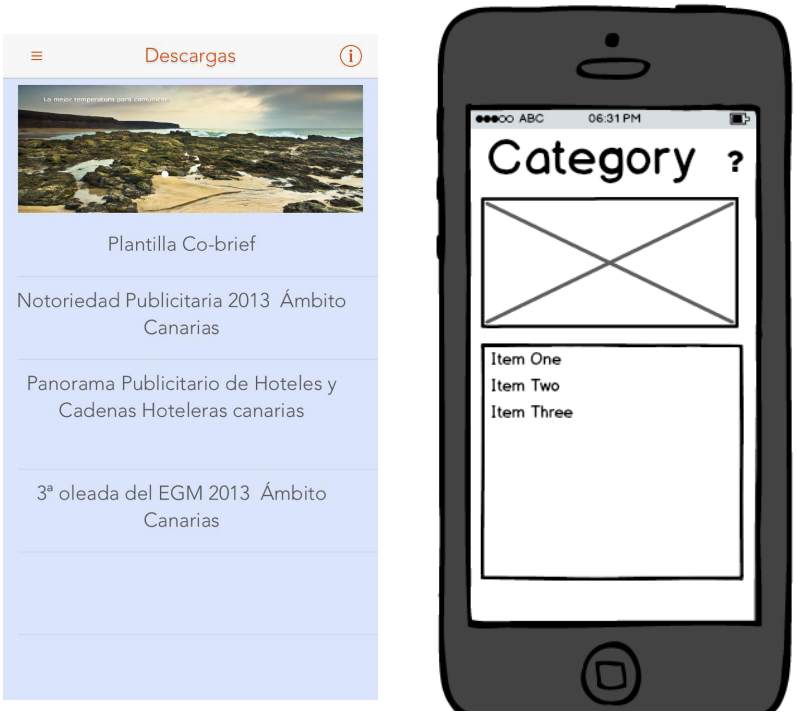


Ilustración 40 - Comparativa de Gallery real con Mockup

- **Glossary**. Los diferentes botones llevarán a otros Mockup de tipo HTML, pero no hará falta usar ningún tipo de link, dado que se usan los nombres.

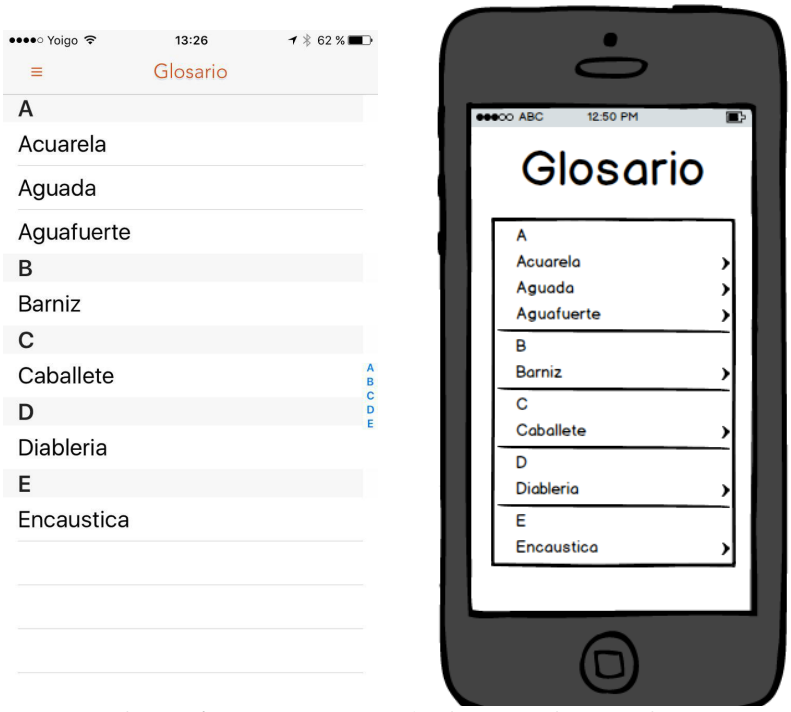


Ilustración 41 - Comparativa de Glossary real con Mockup

5.4. Análisis de los Mockup.

En la siguiente sección se procede a explicar cada uno de los diferentes Mockup que se diseñaron partiendo de las vistas de aplicaciones de ejemplo. El objetivo de esta sección es explicar cómo se relaciona la base de datos de Parse.com con cada uno de los diferentes Mockup.

El primer paso es entender que cada Mockup puede generar una o varias categorías o productos, pudiendo combinar varias categorías y varios productos. Aunque se pueda pasar de una a varias categorías, puede darse el caso de que con un Mockup no sea suficiente para incluir toda la información necesaria en la base datos y que se necesiten varios Mockup en el proceso. Lo que si hay tener claro es que todos los Mockup generan al menos un producto que denominamos meta data. En caso de que el Mockup no genere un meta dato significa que está generando otros productos y que no es necesario el metadato. Normalmente el metadato se necesita cuando el Mockup se basa en categorías.

Como vimos en la sección 5.1, los Mockup se relacionan con el formato que tiene un plugin usado en Wordpress, pero que se ha cogido la misma estructura para desarrollar la base de datos, por lo tanto hay que desarrollar los Mockup desde las vistas de la aplicación y una vez se tiene el Mockup se puede analizar para saber de dónde hay que extraer la información para añadirla en la base de datos de forma correcta.

Debido a esto, y a que cada Mockup de cada vista es diferente, no se usará un mismo analizador para cada una de ellas, por lo que hace falta analizar cada vista por separado.

5.4.1. Left Menu (Menú Izquierda)

Aunque normalmente en el catálogo se conoce esta vista con el nombre de leftMenu, en esta sección por simplicidad y entendimiento, llamaremos a esta vista Menú Izquierda. Esta vista proporciona un menú a la izquierda de la pantalla del Smartphone por el que se puede navegar y seleccionar a que otra vista se desea ir. Esta pantalla es accesible desde cualquier otra pantalla de la aplicación, por lo que podemos concluir que se trata de la vista más importante de la aplicación.

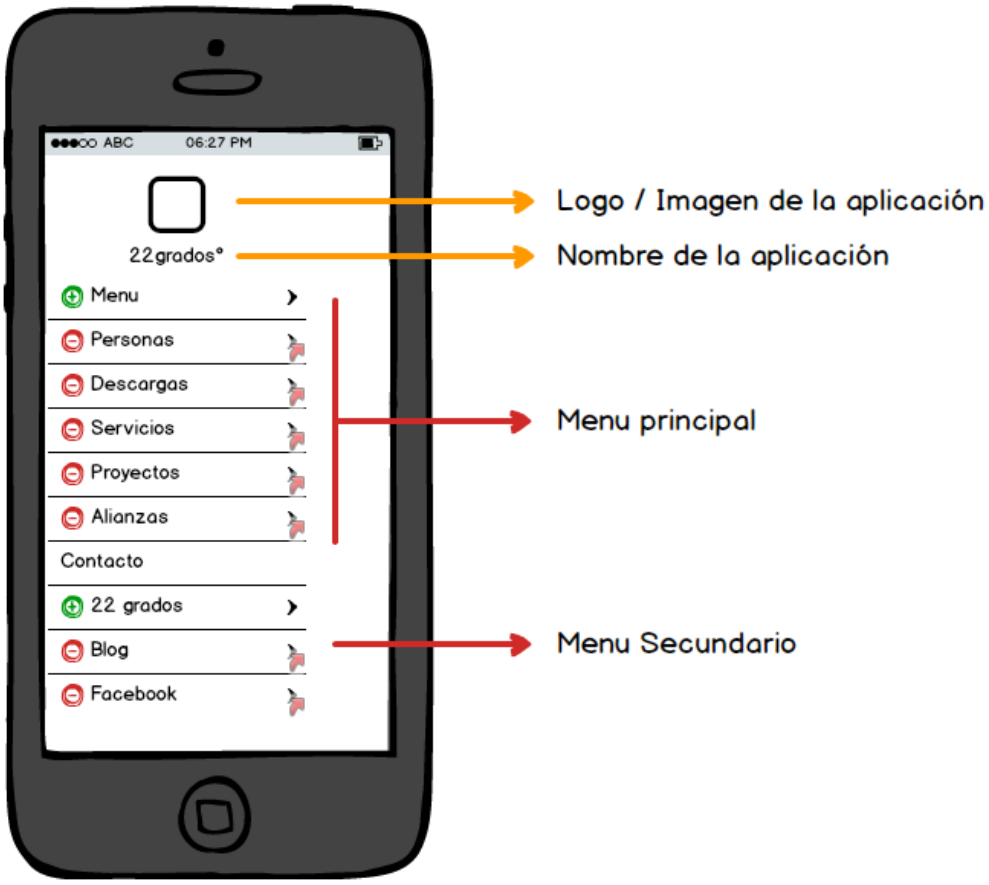


Ilustración 42 - Mockup de vista tipo leftMenu

En la imagen anterior, hemos dividido el Mockup en cuatro secciones.

- **Logotipo o Imagen principal**. Aquí el usuario pondría un link a la imagen que quiere usar como imagen de la aplicación o como logo. Esta imagen puede ser de cualquier tipo.
- **Nombre de la aplicación**. El nombre puede contener cualquier tipo de símbolo incluido en tabla ASCII. Este dará como resultado la categoría principal de la aplicación. De esta categoría saldrán todas las demás categorías y subcategorías, asi como los productos.
- **Menú principal**. Este menú es el que conducirá al usuario a diferentes vistas que habrá en la aplicación. Con esta descripción podemos entender que esto serán categorías de nuestro catálogo. Existe un límite máximo de 7 elementos que pueden ser introducidos debajo de “Menú”. La línea de “Menú”, representa un botón que expande o comprime todo el menú principal. Esta línea siempre tiene que estar presente y se usa delimitador de dónde empieza el menú. Para saber dónde acaba, hay que encontrar la línea de “Contacto”, que delimita donde

comienza el menú secundario. Las categorías de esta sección, serán realmente subcategorías de la categoría principal de la aplicación.

- **Menú Secundario**. Este segundo menú se usa para incorporar productos, normalmente de contacto. Se trata de un menú que solo puede mostrar los productos que se muestran aquí. Como explicamos anteriormente, estos productos deben tener una categoría de la que son hijos, por lo que en esta sección se tienen que incluir todos los productos que tienen como padre la categoría principal.

Si tomamos como ejemplo este Mockup de la imagen anterior, obtenemos que finalmente tendremos como resultado el siguiente árbol.

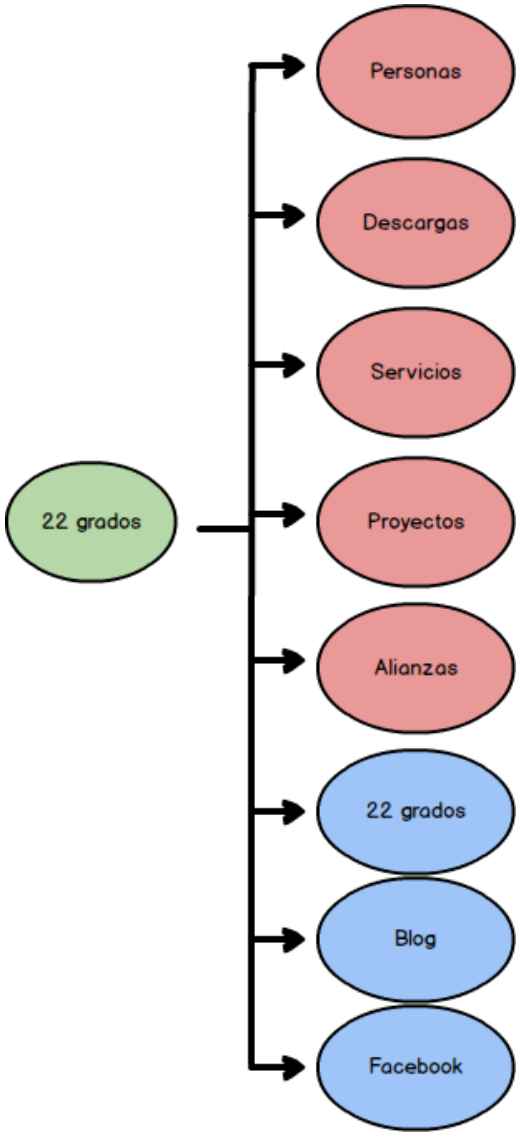


Ilustración 43 - Árbol ejemplo de un parseo de leftMenu

En color verde representamos la categoría principal, mientras que en color rojo las subcategorías de la aplicación y en color azulo los productos. Como explicamos anteriormente, todas las categorías tienen al menos un producto como hijos que llamamos metadato. Esto se aplica a todas las categorías, tanto a la principal como en las subcategorías. Debido a esto, el análisis de este Mockup se divide en tres partes:

- Categoría principal y su metadato.
- Subcategorías y sus metadatos.
- Productos

Categoría principal

Los parámetros que tenemos que obtener de la categoría principal son casi todos conocidos previamente, dado que han de responder a la configuración para la categoría inicial. Como hemos visto, podemos obtener de este Mockup, el nombre de la aplicación y la imagen que representa a la aplicación. Estos dos datos se toman del icono del Mockup, que al ser exportado a XML se obtiene el siguiente código.

```
<control controlID="2" controlTypeID="com.balsamiq.mockups::IconLabel" x="82" y="109" w="-1" h="-1" measuredW="118" measuredH="63" zOrder="2" locked="false" isInGroup="3">
  <controlProperties>
      <href>http%3A//logodemiapp.com/milogo.png</href>
      <map>%3Carea%20shape%3D%22rect%22%20coords%3D%2282%2C109%2C200%2C172%22%20href%3D%22http%3A//logodemiapp.com/milogo.png%22%20id%3D%222%22%20target%3D%22_blank%22/%3E</map>
       <text>22%20grados</text>
  </controlProperties>
 </control>
```

Código 8 - XML del icono de leftMenu

Como vemos del código XML, en el control tipo “IconLabel” podemos obtener de las etiquetas ´href´ y de ´text´ los dos valores que nos hacen falta para completar la categoría de esta y su producto. Para que se pueda obtener la etiqueta href, en cualquier tipo de objeto de Balsamiq, hará falta indicar el link al que se quiere unir ese objeto. Para ello tenemos que hacer uso de la herramienta inspectora de atributos de Balsamiq y en la sección de links, añadir la dirección URL de la imagen que queremos usar. La siguiente imagen muestra un ejemplo de cómo se hizo para esta imagen.

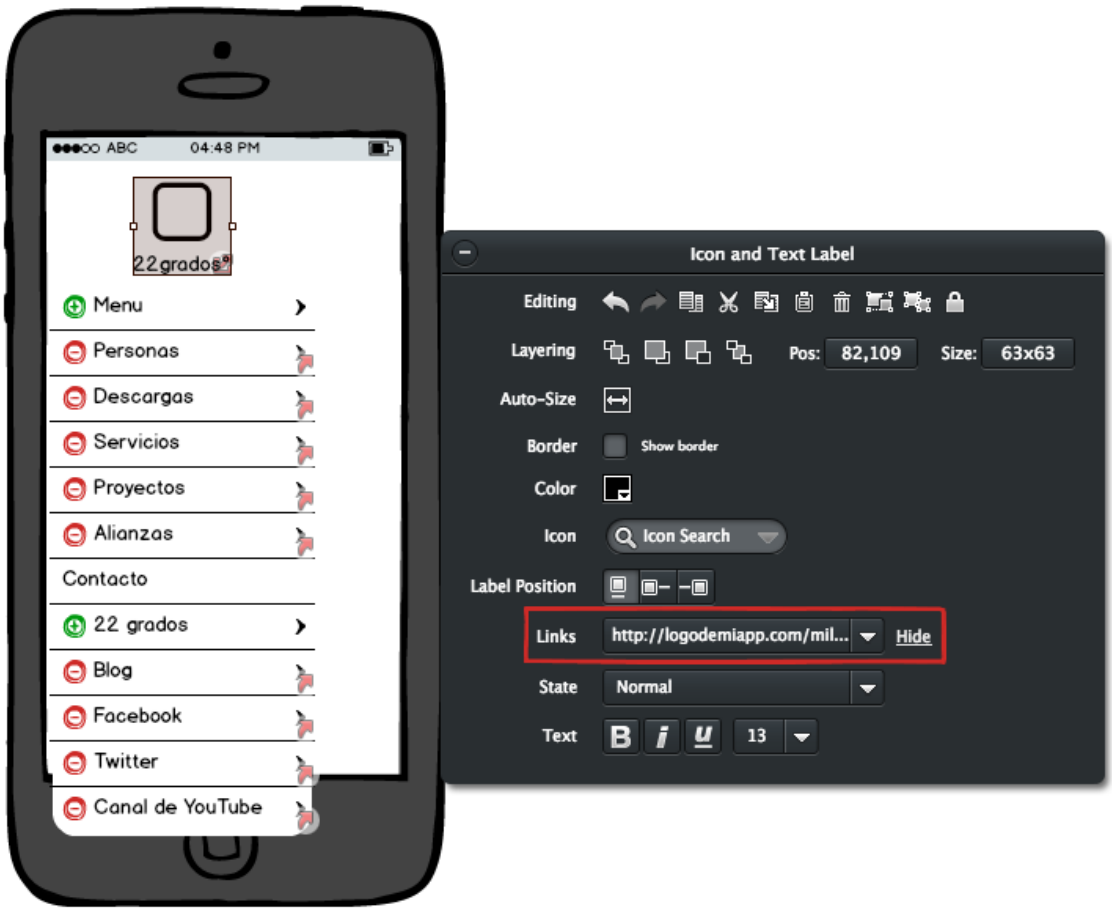


Ilustración 44 - Ejemplo de cómo añadir un link a un Mockup

Se decidió usar la etiqueta de href y no la de map, dado que era mucho más sencilla de usar y se obtenía el resultado de forma más directa.

La categoría queda definida como se muestra en la siguiente tabla.

Columna en Parse.com	Valor que obtiene del Mockup.
Name	Tomado de la etiqueta text <text>
Description	Este valor queda vacío
Image_url	Tomado de la etiqueta <href>
Parent	El parent de esta categoría no es nadie, por lo que ponemos ´0´
Param	Al tratarse de la vista de Menú Izquierda, los parámetros que esta define son los siguientes: "*type resource subtype strongColor mediumColor lightColor fontType action*"
Spider_id	Al ser la primera categoría, le damos el primer valor: ´1´
Ordering	Al ser la primera categoría, le damos el primer valor: ´1´

Tabla 9 - Tabla de parámetros de categorías tipo leftMenu

El producto metadato, quedaría definido como:

Columna en Parse.com	Valor que obtiene del Mockup
Name	Tomado de la etiqueta text <text>, o podría tener el valor por defecto de metadato.
Description	Este valor queda vacío
Category_id	Como se trata del producto de la categoría principal, tomará el mismo valor que el spider_id de la categoría, en este caso: ´1´.
Image_url	Toma el valor por defecto: ******0
Spider_id	Toma el valor siguiente al último producto o categoría añadido.
Param	Se explican en una tabla aparte.
Ordering	Como se trata del primer producto, toma el primer valor: ´1´.

Tabla 10- Tabla de parámetros de productos tipo leftMenu

Cabe destacar que hay más columnas en Parse.com, pero estas no se usan en este proyecto y se le dan a todas el mismo valor vacío. Por otra parte, los param del producto están definidos en la categoría y estos toman los siguientes valores:

PARAMETROS PARA EL PARAMETRO “PARAM”	
Parámetro	**Valor que obtiene del Mockup**
Type	Obtiene el valor del tipo de vista: leftMenu
Resource	Este valor queda vacío
Subtype	Este valor queda vacío
Action	Este valor queda vacío
strongColor	Se le da el valor por defecto: D44A0E
mediumColor	Se le da el valor por defecto: F66607
lightColor	Se le da el valor por defecto: DAE5FC
fontType	Se le da el valor por defecto: Avenir-Light
fontSize	Se le da el valor por defecto: 20

Tabla 11 - Tabla de parámetros del parámetro param del producto tipo leftMenu

El valor de param ha de ser formateado de forma especial. Para todos los productos, se tiene que dar el mismo formato. Esto se debe a como el framework del IUMA espera obtener los datos. Para ello, tomamos todos los datos y los escribimos en el formato deseado.

```
par_type@@:@@leftMenu par_resource@@:@@ par_subtype@@:@@
par_strongColor@@:@@D44A0E par_mediumColor@@:@@F66607
par_lightColor@@:@@DAE5FC par_fontType@@:@@Avenir-Light
par_fontSize@@:@@20 par_action@@:@@
```

Código 9 - Ejemplo de formato para los parámetros tipo param

Con todos estos datos obtenidos del Mockup, hemos completado la primera categoría y el primer producto.

Subcategorías y sus metadatos

En este caso, las subcategorías que obtenemos son parte del menú, y como se explicó en páginas anteriores, va desde la palabra ´Menú´ hasta la palabra ´Contacto´. El código XML que obtenemos tras la exportación es el siguiente:

```
<control controlID="1" controlTypeID="com.balsamiq.mockups::iPhoneMenu" x="29"
y="172" w="169" h="-1" measuredW="215" measuredH="358" zOrder="1"
locked="false" isInGroup="3">
   <controlProperties>
        <borderStyle>none</borderStyle>
        <hrefs>%2Chttp%3A//type.products%2Chttp%3A//type.gallery%2Chttp%3A//ty
        pe.products%2Chttp%3A//type.glossary%2Chttp%3A//type.copyright%2Chttp%
        3A//type.html%2Chttp%3A//type.products%2C%2Chttp%3A//type.url%2Chttp%
        3A//type.url%2Chttp%3A//type.url%2Chttp%3A//type.url</hrefs>
        <map>…</map>
        <text>+%20Menu%2C%20%3E%0A-%20Personas%2C%20%3E%0A-
        %20Descargas%2C%20%3E%0A-%20Servicios%2C%20%3E%0A-
        %20Proyectos%2C%20%3E%0A-
        %20Alianzas%2C%20%3E%0AContacto%0A+%2022%20grados%2C%20%3E%0A-
        %20Blog%2C%20%3E%0A-%20Facebook%2C%20%3E%0A-
        %20Twitter%2C%20%3E%0A-%20Canal%20de%20YouTube%2C%20%3E</text>
    </controlProperties>
</control>
```

Código 10 - XML del controlador Menu de leftMenu

Por razones de espacio, el texto de la etiqueta <map> se ha eliminado dado que no se utiliza en el análisis de este Mockup.

El principal problema que tiene el análisis de esta vista, es que el menú está en una sola pieza, por lo que hará falta dividirla en dos partes. Para ello usaremos el texto de inicio y final que es fijo en todas las vistas de este tipo. Por ello de la etiqueta <text> nos quedamos con la primera parte del texto hasta donde se indica contacto, que usaremos en la siguiente sección.

Como se ve en el texto, este está formateado con caracteres ASCII. Lo primero que debemos hacer es cambiar este formato, usando un parseador de ASCII. Una vez que tenemos el texto, dividimos cada línea, para ello nos ayudaremos del símbolo ´-´ que es insertado por Balsamiq. Cuando tenemos divido el menú y podemos saber cuántas categorías vamos a introducir nuevas, podemos coger de la etiqueta <href> los tipos de categorías que van a ser creadas, dado que de esta etiqueta obtendremos de que tipo es esta categoría.

Para ello, se aplica una regla de diseño preestablecida para poder usar esta herramienta. Cuando se añade una categoría en esta vista, en el inspector en la sección de link se introducirá una dirección web predefinida, dependiendo de qué tipo de categoría queremos que sea. Esta URL se compone de dos partes, una primera común para todas las URL y que servirá como indicador de que se trata de un link a otra categoría o producto y una parte que variará dependiendo del tipo de categoría queremos que sea. La parte común será la palabra “type”, seguido de un punto ´.´ y la parte variable será el nombre del tipo que queremos, ya sea products, gallery, copyright, etc. Un ejemplo sería type.products

Como ocurrió en la categoría principal, cada ítem dará como resultado una categoría y un producto metadato y además la mayoría de los parámetros son conocidos previamente y solo hay que dar valor a algunos de ellos. Las siguientes tablas, representan los valores que obtendrán las categorías, los productos y los parámetros de los productos.

Categorías	
Columna en Parse.com	**Valor que obtiene del Mockup.**
Name	Tomado de la etiqueta text <text>, después de que se haya dividido el menú en los diferentes ítems, cada uno de ellos dará un valor a cada categoría.
Description	Este valor queda vacío

Image_url	Toma el valor por defecto: ******0
Parent	Esta categoría toma el valor del parent de la categoría principal, que siempre será: ´1´
Param	Todas las categorías que no son del tipo Menú Izquierda, tienen los mismos parámetros, independientemente del tipo de categoría que sean: "*type resource subtype action*"
Spider_id	Toma el valor siguiente al último producto o categoría añadido.
Ordering	Toda el valor siguiente a la último producto añadido en esta categoría.
Productos	
Columna en Parse.com	**Valor que obtiene del Mockup**
Name	Tomado de su categoría, o podría tener el valor por defecto de metadato.
Description	Este valor queda vacío
Category_id	Se toma de la categoría a la que va a pertenecer.
Image_url	Toma el valor por defecto: ******0
Spider_id	Toma el valor siguiente al último producto o categoría añadido.
Param	Se explican en una tabla aparte.
Ordering	Toda el valor siguiente a la última categoría añadida
PARAMETROS PARA EL PARAMETRO "PARAM"	
Parámetro	**Valor que obtiene del Mockup**
Type	Obtiene el valor de la etiqueta <hrefs>, siendo el que corresponda con el nombre de la categoría.
Resource	Este valor queda vacío
Subtype	Este valor queda vacío
Action	Este valor queda vacío

Tabla 12 - Resultado de los valores de productos y subcategorías del tipo leftMenu

Como sucediera anteriormente, los param se tienen que formatear, quedando como ejemplo el siguiente formato:

par_type@@:@@tipo_categoría par_resource@@:@@ par_subtype@@:@@ par_action@@:@@

Tabla 13 - Formato de los param de leftMenu de los productos y subcategorías

Productos de la categoría principal

Por último quedaría los productos de la categoría principal que componen el menú secundario. Los valores de estos productos se obtienen también de la división del menú anterior y nos ayudamos igualmente del símbolo ´-´ para separarlos en unidades individuales. El tipo de productos normalmente es URL, pero también puede aceptar otros tipos como el tipo PDF o mail. Por ello es necesario también introducir los links de la misma forma que se hizo anteriormente en las subcategorías. Una vez que hemos dividido los links, podemos introducir uno a uno estos productos. La siguiente tabla representa como han de ser cogido estos datos.

Productos	
Columna en Parse.com	**Valor que obtiene del Mockup**
Name	Tomado de la etiqueta text <text>, después de que se haya dividido el menú en los diferentes ítems.
Description	Este valor queda vacío
Category_id	Se toma de la categoría a la que va a pertenecer, que este caso es la principal, por lo que será ´1´.
Image_url	Toma el valor por defecto: ******0
Spider_id	Toma el valor siguiente al último producto o categoría añadido.
Param	Se explican en una tabla aparte.
Ordering	Toda el valor siguiente a la último producto añadido en esta categoría.
PARAMETROS PARA EL PARAMETRO “PARAM”	
Parámetro	**Valor que obtiene del Mockup**
Type	Obtiene el valor de la etiqueta <hrefs>, siendo el que corresponda con el nombre de la categoría.
Resource	Este valor queda vacío

Subtype	Este valor queda vacío
Action	Este valor queda vacío

Tabla 14 - Productos de la categoría principal de leftMenu

5.4.2. Products.

Esta vista es una de las más utilizadas por los usuarios, dado que es la manera más sencilla de mostrar la información. Proporciona a los usuarios tres secciones bien divididas y de la que pueden extraer diferente información de cada una de ellas. La primera, la parte superior, donde se incorpora una imagen que representa este producto en concreto. Hemos de recordar, que este catálogo está pensado en forma de productos y categorías, por lo que cada vista, representa un producto.

La segunda parte, situado en el medio, representa el texto principal del producto, que básicamente será la descripción del mismo. Esta parte admite código HTML, por lo que se puede formatear usando un editor HTML. Esto también permite tener links, por lo que al pulsar en ellos se abriría la aplicación de navegación del Smartphone.

Por último, la parte inferior, donde se incorpora un botón que puede conducir a escribir un correo electrónico o a un archivo PDF. También puede llevar a una URL. En caso de que no conduzca a ningún lado, no aparecerá nada. Dependiendo el tipo de link al que conduzca, aparecerá un texto u otro.

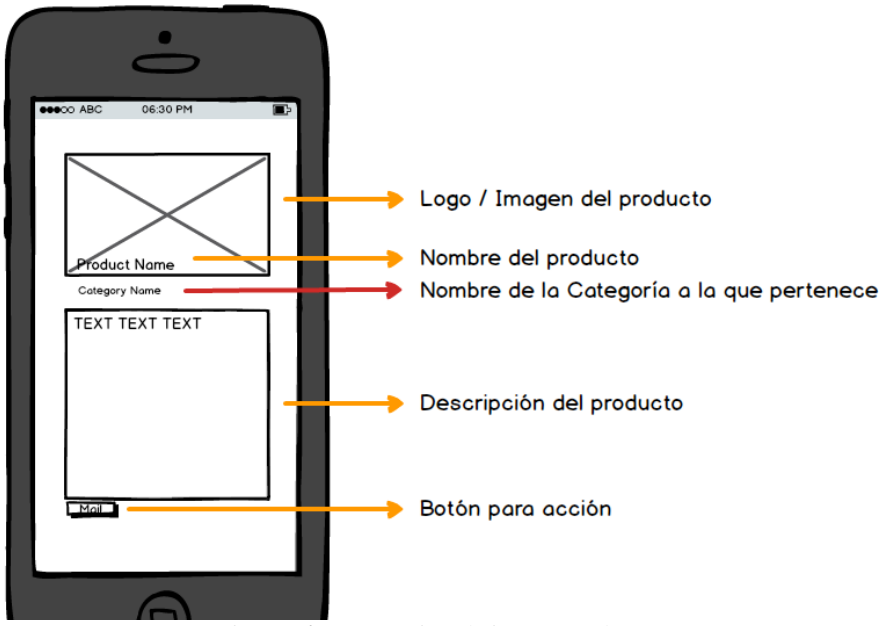


Ilustración 45 - Mockup de la vista Products

En la imagen anterior, se muestra el Mockup de esta vista con la descripción de cada una de las secciones en las que se divide. Los colores de las flechas indican si este objeto se va a usar durante el analizado del Mockup o no. Por tanto las flechas naranjas indican los objetos que se van a usar y las fechas rojas los parámetros que no se van a usar. Si dividimos el Mockup en las tres secciones anteriores:

- **Logo y Nombre**. En la parte superior encontraremos el logo del producto y el nombre que ha de tener. En el caso del nombre y puesto que este puede tener espacios u otros símbolos, debemos convertir el texto con el parseador a ASCII. Este se explica más adelante.
- **Descripción**. Se trata de un cuadro de texto situado en la parte central del Mockup. Este contiene toda la descripción del producto que vamos a introducir. También ha de pasar por el analizador de texto.
- **Recurso**. El botón situado en la parte inferior izquierda de la pantalla nos proporcionará el recurso, así como el tipo de producto del que estamos tratando.

Hemos de diferenciar entre una categoría de tipo producto y los productos. En una categoría de tipo producto, podemos encontrar numerosos productos, pero estos productos pueden ser de diversos tipos, como URL, PDF, mail, etc. Es por esto que en el caso de los productos que se representan dentro de la vista producto, el tipo del producto se coge de la sección del recurso.

Para analizar esta vista, usaremos las divisiones ficticias que hemos creado y posteriormente mostraremos como almacenar los datos en una tabla con el formato de SpiderCatalog y que entiende el framework del IUMA.

```
<control controlID="3" controlTypeID="com.balsamiq.mockups::Image" x="52" y="128" w="178" h="107" measuredW="77" measuredH="79" zOrder="1" locked="false" isInGroup="0">
  <controlProperties>
      <href>http%3A//22gradosmedia.com/wp-content/uploads/2013/09/Welovroi_400x400.png</href>
      <map>%3Carea…</map>
      <text/>
  </controlProperties>
</control>
```

```
<control controlID="5" controlTypeID="com.balsamiq.mockups::Label" x="63" y="214" w="156" h="-1" measuredW="64" measuredH="21" zOrder="2" locked="false" isInGroup="0">
  <controlProperties>
      <text>Mi%20Entrada%20con%20imagen</text>
  </controlProperties>
</control>
```

Código 11 - XML del objeto image del mockup tipo products

Como vemos, el código XML continua siendo bastante similar y podemos ver como del objeto Image podemos obtener una dirección URL a un archivo .PNG que es precisamente el tipo de archivos que estaríamos buscando para añadir como imágenes. En este caso, los productos aceptan tanto imágenes .PNG como .JPEG. En este caso, la dirección url al archivo se encuentra dentro de la etiqueta <href>

Por otro lado, el nombre del producto lo podemos obtener de la etiqueta <text> del objeto "label". Aunque si analizamos atentamente el Mockup, vemos que existen dos objetos denominados "label". Para diferenciar uno del otro, sabemos que el objeto label del nombre del producto, se encuentra siempre con un nivel menos que el objeto label del nombre de la categoría a la que pertenece.

Del área central de la vista obtenemos la descripción. El código XML que obtenemos es el siguiente.

```
<control controlID="7" controlTypeID="com.balsamiq.mockups::TextArea" x="52" y="262" w="178" h="166" measuredW="200" measuredH="140" zOrder="4" locked="false" isInGroup="0">
  <controlProperties>
      <text>Esto%20se%20trata%20de%20un%20texto%20de%20prueba%20para%20la%20memoria%20del%20proyecto%20y%20para%20demostrar%20donde%20se%20encuentra%20la%20descripcion%20del%20producto%20dentro%20de%20la%20vista%20de%20los%20productos.</text>
  </controlProperties>
</control>
```

Código 12 - XML del objeto TextArea del Mockup tipo product

Al igual que antes, pero esta vez del objeto TextArea podemos extraer de la etiqueta <text> el texto que describe al producto. Como podemos ver, también hay que usar el parseador a ASCII, dado que el exportador de Balsamiq a XML sustituye los espacios y en general todos los símbolos por su valor hexadecimal de la tabla ASCII.

Por ultimo tenemos el recurso que sacamos del botón situado en la parte inferior izquierda de la vista.

```
<control controlID="8" controlTypeID="com.balsamiq.mockups::Button" x="53" y="428" w="46" h="15" measuredW="42" measuredH="27" zOrder="5" locked="false" isInGroup="0">
  <controlProperties>
      <href>http%3A//esperanza@esperanzacolastra.com</href>
      <map>%3Carea%20shape%3D%22rect%22%20coords%3D%2253%2C428%2C99%2C443%22%20href%3D%22http%3A//esperanza@esperanzacolastra%22%20alt%3D%22http%3A//esperanza@esperanzacolastra%22%20id%3D%228%22/%3E</map>
      <size>11</size>
      <text>Mail</text>
  </controlProperties>
</control>
```

Código 13 - XML del objeto Button del mockup tipo product

De este objeto de tipo Button, obtendremos de la etiqueta <href> la dirección del recurso que debemos usar, y de la etiqueta <text> podemos obtener el tipo. En caso de que la etiqueta text no contenga explícitamente el tipo, pueden darse dos escenarios.

- El tipo se tiene que obtener analizando el recurso. Para ello miraremos si tiene una "@" o si acaba en ".com", ".pdf", ".jpeg" u otra extensión que pueda dar lugar a un tipo conocido.
- En caso de que analizando el recurso no podemos obtener el tipo o que no existiera recurso, se pondría un tipo por defecto, URL.

Una vez explicado el Mockup queda por saber cómo se sabe la categoría a la que pertenece este Mockup. Para eso, y aunque del Mockup se puede obtener, usaremos una de las reglas de diseño que se han diseñado para esta herramienta y se trata de la composición de los nombres de los archivos.

El nombre de los archivos se compone de tres partes, cada una de ellas separadas por el símbolo "_". La primera indica el tipo de vista que es, por ejemplo Products, o LeftMenu. Con esta información podemos saber y elegir el analizador correcto. La segunda indica la categoría a la que pertenece, por ejemplo si su categoría padre es "prueba", el nombre tendrá que ser "prueba". Por último, se encontrará el nombre que le queramos dar, que puede coincidir o no con el nombre del producto que estamos trabajando. Esta última parte

no se usa en ningún momento del proceso de analizado. Un ejemplo de nombre de archivo sería: Products_Prueba_Miprueba.XML

Una vez que tenemos la categoría, hemos de buscar si esta se encuentra dentro de nuestro sistema. Para eso recorremos nuestro árbol y si encontramos la categoría sabemos que podemos seguir analizando esta vista, sino, tendremos que parar y esperar hasta analizar las otras vistas. En caso de que una vez analizadas todas las vistas no tengamos esta categoría, avisaríamos al usuario a través de la vista de la herramienta de que se ha producido un error.

Queda por tanto claro, que cada vista analizada de este tipo dará como resultado un solo producto que colgará de su padre, como se muestra en la siguiente imagen.

Ilustración 46 - Árbol del tipo Producto

De toda la información que se ha explicado, hay algunos parámetros que serán siempre conocidos. Así quedaría la tabla genérica del analizado de una vista tipo “products”.

Producto	
Columna en Parse.com	**Valor que obtiene del Mockup**
Name	Tomado de la etiqueta <text> del objeto Label
Description	Tomado de la etiqueta <text> del objeto TextArea
Category_id	Se busca dentro de todos las categorías ya parseados anteriormente si existe una categoría que tenga el mismo nombre que el nombre que se obtiene de la segunda parte del nombre del archivo.
Image_url	Tomado de la etiqueta <href> del objeto Label. En caso de que no exista dicha etiqueta, se le otorgar el valor por defecto: ******0
Spider_id	Toma el valor siguiente al último producto o categoría añadido.
Param	Se explican en una tabla aparte.

Ordering	Toda el valor siguiente a la último producto añadido en esta categoría.
PARAMETROS PARA EL PARAMETRO "PARAM"	
Parámetro	**Valor que obtiene del Mockup**
Type	Obtiene el valor como se explicó anteriormente dependiendo de cuál sea la situación.
Resource	Se toma el valor de la etiqueta <href> del objeto "Button" en caso de que exista. Sino, se deja vacío.
Subtype	Este valor queda vacío
Action	Se le otorga el valor de la etiqueta <text> del objeto "Button"

Tabla 15 - Valores en Parse.com del tipo Producto

5.4.3. Gallery.

Gallery se trata de una vista que también es bastante utilizada por los usuarios, aunque en este caso en menor medida que la anterior. Esta vista es un poco más sencilla que la anterior, pero incorpora algunas diferencias. La primera de ellas es que admite en sus imágenes un vector de direcciones de diferentes imágenes, permitiendo moverse entre ellas.

Por otro lado, esta vista también incorpora una lista de los productos de los que se compone. Es decir, esta vista no genera un solo producto, sino que de la categoría de la que es hijo saldrán varios productos. Esta es una gran diferencia con respecto a la vista de tipo productos, dado que esta categoría se "cerrará" una vez analizada esta vista, porque sería incoherente que hubiera otra vista que tenga como padre la misma categoría.

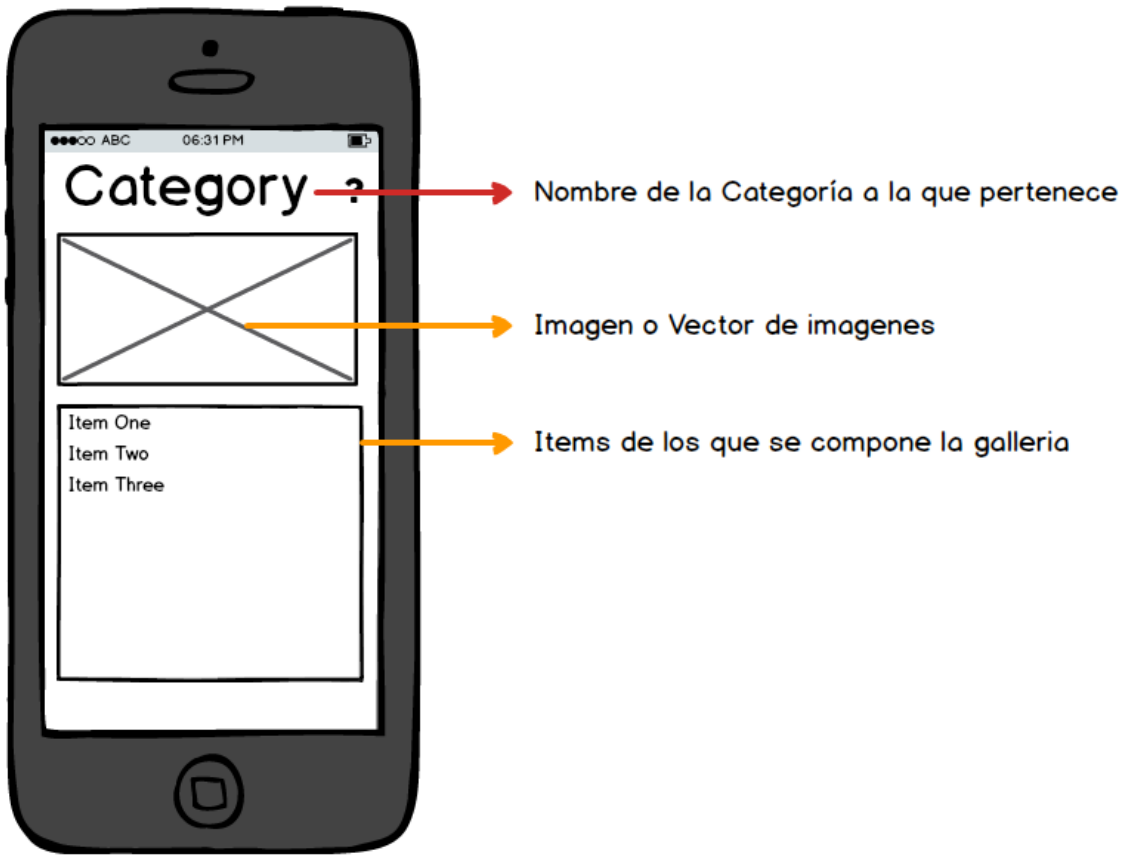


Ilustración 47 - Mockup tipo Gallery

Como podemos ver en el Mockup, solo hay dos secciones de las que estamos interesados. La primera de ellas es la imagen o imágenes que vamos a obtener del link que tenga la imagen, de forma que podemos tener una o varias imágenes que añadir. Dependerá principalmente del uso que se le quiera dar a esta aplicación.

Por otro lado, tenemos una lista de ítems de los que se compone la vista de tipo galería. Esta lista de productos puede ser tan larga como el usuario quiera y la única restricción es que todos los ítems han de tener un link a otra página, es decir que pueden tener un link de tipo URL, PDF o email. Cuando hablamos de URL, hablamos también que pueden ser links a imágenes que tengan una dirección URL.

Por tanto, esta vista se basa principalmente en los links que tiene cada ítem y el usuario tendrá que interactuar más con el inspector de los elementos que con el propio Mockup. También podemos observar que el nombre de la categoría no se utiliza, se muestra en rojo. Esto se debe a que usaremos en todos los Mockup el mismo sistema para identificar las categorías a las que pertenece.

Teoréticamente, el usuario podría introducir en estos ítems, links a otros productos, por lo que estos dejarían de ser productos y pasarían a ser subcategorías de la categoría principal. En el caso de esta aplicación, esta característica se dejó fuera del alcance del proyecto debido a que el principal objetivo es abarcar más vistas y tener unos requerimientos mínimos para poder validar la idea, que desarrollar todo el producto sin validar la idea.

El siguiente árbol muestra cuál sería el resultado obtenido después de analizar este Mockup.

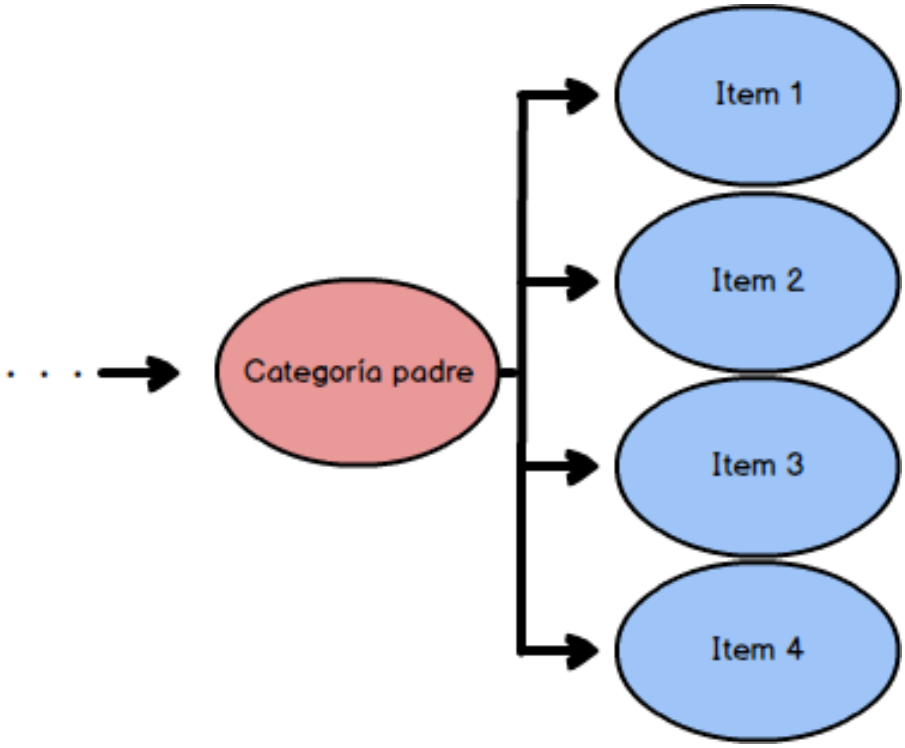


Ilustración 48 - Árbol del tipo Gallery

Del código XML, podremos extraer las diferentes partes que nos interesen. De la imagen:

```
<control controlID="1" controlTypeID="com.balsamiq.mockups::Image" x="35" y="153" w="204" h="104" measuredW="77" measuredH="79" zOrder="1" locked="false" isInGroup="0">
    <controlProperties>
        <href>http%3A//linkdeprueba.com/prueba.jpg%3Bhttp%3A//linkdeprueba.com
        /prueba2.jpg%3Bhttp%3A//linkdeprueba.com/prueba3.jpg</href>
        <map>%3Carea…</map>
        <text/>
   </controlProperties>
</control>
```

Código 14 - XML del objeto Image de la vista tipo Gallery

De la etiqueta <href> podremos extraer los link de todas las imágenes. Para separarlos, usaremos el símbolo ";", que en el XML es "%3B". Estos, han de ser añadidos en el producto

del metadato de la categoría y no en los productos, por lo que es necesario buscar en nuestra estructura el metadato de la categoría para actualizarlo.

Del mismo modo ocurre con cada uno de los productos, estos se crearan a partir de los ítems, como explicamos antes. Este sería la parte del código XML del que estamos interesados.

```
<control controlID="2" controlTypeID="com.balsamiq.mockups::List" x="35" y="269"
w="208" h="188" measuredW="100" measuredH="126" zOrder="2" locked="false"
isInGroup="0">
   <controlProperties>
      <hrefs>http%3A//linkdeprueba.com/prueba%2Chttp%3A//linkdeprueba.com/pr
      ueba.pdf%2Chttp%3A//linkdeprueba.com/pruebe.jpg</hrefs>
      <map>%3Carea…</map>
      <text>Item%20One%0AItem%20Two%0AItem%20Three</text>
   </controlProperties>
</control>
```

Código 15 - XML del objeto List de la vista tipo Gallery

Cada Item de la etiqueta <text> se separará en diferentes ítems, usando como referencia el "%0A". Una vez separado, se separaran los links usando la etiqueta <href> y como separador el "%2C". Una vez separados, se tendrá que analizar la URL para definir si se trata de un PDF o de una dirección URL. Dependiendo cuál sea, el tipo de este producto, será PDF o URL.

De toda la información que se ha explicado, hay algunos parámetros que serán siempre conocidos. Así quedaría la tabla genérica del analizado de una vista tipo "Gallery".

Categoría padre de los productos	
Columna en Parse.com	**Valor que obtiene del Mockup.**
Name	No hay modificaciones
Description	No hay modificaciones
Image_url	Toma los valores recogidos tras haber separado todos los links de la etiqueta <href>
Parent	No hay modificaciones
Param	No hay modificaciones
Spider_id	No hay modificaciones
Ordering	No hay modificaciones

Producto (tantos como diferentes ítems que contiene el objeto List)	
Columna en Parse.com	**Valor que obtiene del Mockup**
Name	Tomado de la etiqueta <text> del objeto List
Description	No aplica a esta vista
Category_id	Se busca dentro de todos las categorías ya parseados anteriormente si existe una categoría que tenga el mismo nombre que el nombre que se obtiene de la segunda parte del nombre del archivo.
Image_url	No aplica a esta vista
Spider_id	Toma el valor siguiente al último producto o categoría añadido.
Param	Se explican en una tabla aparte.
Ordering	Toda el valor siguiente a la último producto añadido en esta categoría.
PARAMETROS PARA EL PARAMETRO "PARAM"	
Parámetro	**Valor que obtiene del Mockup**
Type	Obtiene el valor como se explicó anteriormente dependiendo de cuál sea la situación. Podrá obtener los valores de URL o PDF
Resource	Se toma el valor de la etiqueta <href> del objeto "List" del ítem que corresponda. Sino, se deja vacío.
Subtype	No aplica a esta vista
Action	No aplica a esta vista

Tabla 16 - Categorías y productos de la vista tipo Gallery

5.4.4. Copyright.

Copyright se trata de una vista con un uso muy definido. Como su nombre indica, está pensada para ser usada como lista de copyright de los que la aplicación es responsable. Aunque este es el uso principal de esta vista, otros usuarios pueden encontrarle algún otro

tipo de uso. Esta vista solo puede aportar parte de la información, mientras que el resto de la información ha de ser proporcionada por la vista HTML.

Esta vista tiene una estructura muy similar a la de la galería. Esta vista genera una lista de productos que son asociados a la categoría padre, que se obtiene a través del nombre del archivo, por lo que el árbol que genera es idéntico al de Gallery.

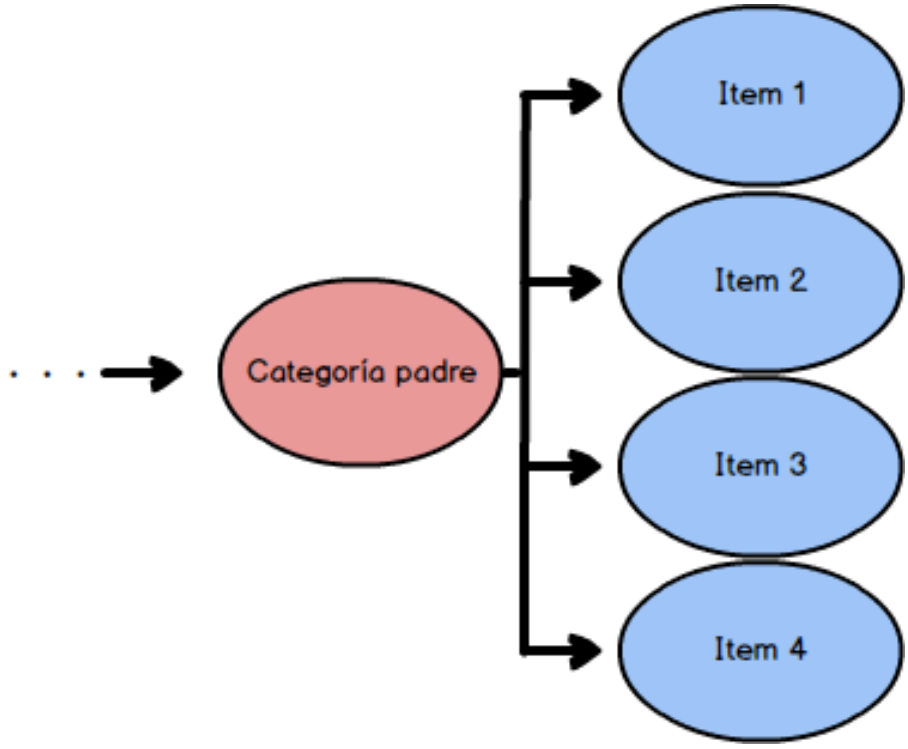


Ilustración 49 - Árbol de la vista tipo Copyright

Estos nuevos productos, quedarán a medio construir, y es responsabilidad del usuario el proporcionar el resto de la información necesaria para poder finalizar estos productos. En caso de que el usuario no proporcione estos ítems completos, la aplicación no fallará, y mostrará una información vacía, dado que no es necesario tener definido el producto para que la vista a la que conduce se pueda generar, con la información que obtenemos en esta vista es suficiente. Es más, la estrategia que se sigue en esta vista con respecto al objeto utilizado es un poco diferente con respecto al tipo gallery, usando el objeto Menu y no el objeto List. El Mockup es el siguiente:

Ilustración 50 - Mockup de la vista Copyright

Como vemos en el Mockup, solo tiene un objeto para ser analizado, por lo que este es más sencillo de analizar. Este objeto es de tipo Menú y permite al usuario introducir diferentes líneas para que actúen como diferentes ítems.

```
<control controlID="12" controlTypeID="com.balsamiq.mockups::Menu" x="37" y="158" w="208" h="-1" measuredW="150" measuredH="115" zOrder="2" locked="false" isInGroup="0">
   <controlProperties>
      <text>G3M%20%3E%0ABalsamiq%20%3E%0AEsperanza%20Colastra%20%3E%0AIUMA%20%3E%0AULPGC%20%3E</text>
   </controlProperties>
</control>
```

Código 16 - XML del objeto Menu de la vista Copyright

Para dividir todos los objetos usaremos el carácter %3E, que indica el carácter “>”. Una vez tenemos divido el menú en los diferentes objetos, podemos añadirlos a nuestro sistema, teniendo estos objetos la siguiente representación.

Producto (tantos como diferentes ítems que contiene el objeto Menu)	
Columna en Parse.com	**Valor que obtiene del Mockup**
Name	Tomado de la etiqueta <text> del objeto Menu
Description	Habrá que esperar al Mockup que describe este producto para poder actualizar la descripción. Para esta vista no aplica, pero este producto no está 100% acabado hasta que no ha sido descrito.
Category_id	Se busca dentro de todos las categorías ya parseados anteriormente si existe una categoría que tenga el mismo nombre que el nombre que se obtiene de la segunda parte del nombre del archivo.
Image_url	No aplica a esta vista
Spider_id	Toma el valor siguiente al último producto o categoría añadido.
Param	Se explican en una tabla aparte.
Ordering	Toda el valor siguiente a la último producto añadido en esta categoría.
PARAMETROS PARA EL PARAMETRO "PARAM"	
Parámetro	**Valor que obtiene del Mockup**
Type	Se usa el tipo "Copyright"
Resource	No aplica a esta vista
Subtype	No aplica a esta vista
Action	No aplica a esta vista

Tabla 17 - Producto de la vista tipo Copyright

Este se trata de uno de los Mockup más sencillos de analizar.

5.4.5. Glossary.

Glossary, al igual que copyright son Mockup muy específicos que tienen un uso limitado. Aunque en el caso de glossary, su análisis es un poco más complicado que el anterior, pero no tan complicado como los primeros. Primero, el usuario ha de configurar correctamente este Mockup, añadiendo siempre la primera letra de cada palabra como referencia y luego añadiendo como separador una línea. Eso es de vital importancia para que el analizador funcione correctamente.

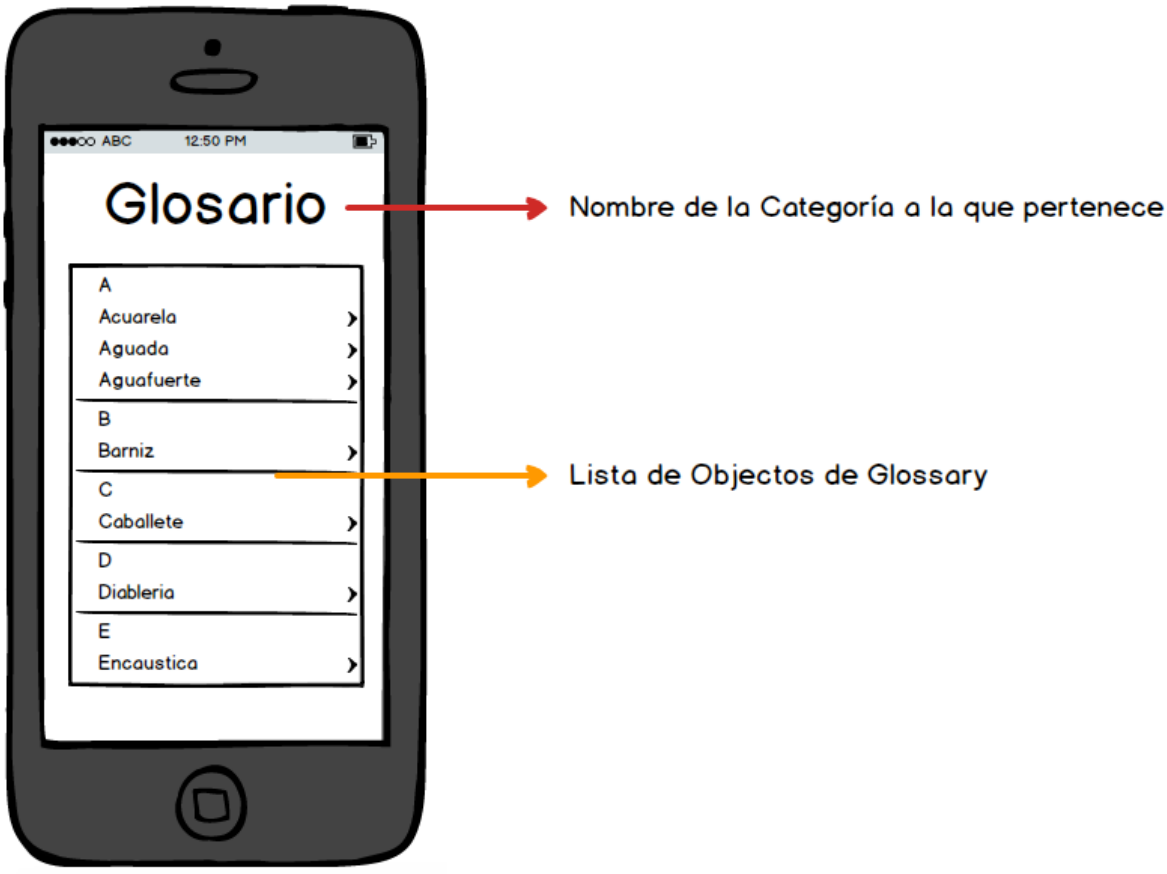


Ilustración 51 - Mockup de la vista tipo Glossary

Como antes, el nombre de la categoría a la que pertenece se toma del nombre y no del título del Mockup. Por otro lado, la lista de objetos se toma como anteriormente del objeto "Menu", dando como resultado múltiples productos. El código XML que hay que analizar de todo el Mockup es el siguiente:

```
<control controlID="2" controlTypeID="com.balsamiq.mockups::Menu" x="158" y="205" w="197" h="-1" measuredW="150" measuredH="282" zOrder="1" locked="false" isInGroup="-1">
   <controlProperties>
      <text>A%0AAcuarela%20%3E%0AAguada%20%3E%0AAguafuerte%20%3E%0A%3D%0AB%0ABarniz%20%3E%0A%3D%0AC%0ACaballete%20%3E%0A%3D%0AD%0ADiableria%20%3E%0A%3D%0AE%0AEncaustica%20%3E</text>
   </controlProperties>
</control>
```

Código 17 - XML del objeto Menu de la vista Glossary

Para dividir el Menu se usará la misma estrategia que anteriormente, pero como en este caso la primera letra, no hace falta analizarla, es decir, que la letra que indica que sección es, no hace falta introducirla en nuestro sistema, tenemos que asegurarnos que esta no se tiene en cuenta. Una vez separados todos los productos, el resultado final quedaría como el siguiente árbol.

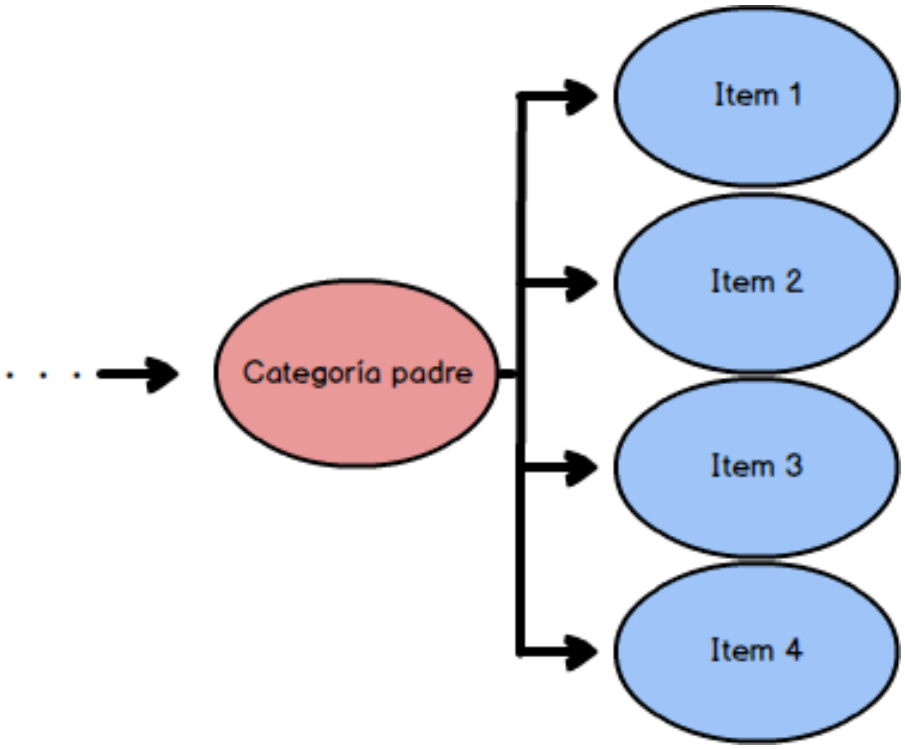


Ilustración 52 - Árbol de categorías y productos de la vista tipo Glossary

Como sucede con los anteriores Mockup, estos productos no quedarán completos hasta que el otro Mockup con toda la información sea analizado. Una vez analizado esta vista, todos los productos que se introducen en el sistema tendrán la siguiente estructura:

Producto (tantos como diferentes ítems que contiene el objeto Menú)	
Columna en Parse.com	**Valor que obtiene del Mockup**
Name	Tomado de la etiqueta <text> del objeto Menú
Description	Habrá que esperar al Mockup que describe este producto para poder actualizar la descripción. Para esta vista no aplica, pero este producto no está 100% acabado hasta que no ha sido descrito.
Category_id	Se busca dentro de todos las categorías ya analizadas anteriormente si existe una categoría que tenga el mismo nombre que el nombre que se obtiene de la segunda parte del nombre del archivo.
Image_url	No aplica a esta vista
Spider_id	Toma el valor siguiente al último producto o categoría añadido.
Param	Se explican en una tabla aparte.
Ordering	Toda el valor siguiente a la último producto añadido en esta categoría.
PARAMETROS PARA EL PARAMETRO "PARAM"	
Parámetro	**Valor que obtiene del Mockup**
Type	Se usa el tipo "Glossary"
Resource	No aplica a esta vista
Subtype	No aplica a esta vista
Action	No aplica a esta vista

Tabla 18 - Producto resultado de la vista tipo Glossary

5.4.6. HTML.

Por último, la última vista que hace falta analizar es la de tipo HTML. Esta vista puede tener tres objetivos distintos:

- Ser una vista HTML independiente.
- Ser una vista que depende de la vista Glossary.

- Ser una vista que depende de la vista Copyright.

Como vimos antes, la vistas Glossary y Copyright necesitan de otras vistas para ser completada. Para ello, se usa la vista HTML. Para saber qué tipo de de vista HTML es, tenemos que mirar el nombre del archivo. Si el segundo atributo del nombre del archivo es "HTML" se trata de una vista independiente, si es "Glossary", esta vista actualiza un tipo producto de tipo "Glossary" y si es "Copyright", actualiza un tipo producto de tipo "Copyright". El producto que actualiza es el tercer atributo del nombre del archivo.

Ilustración 53 - Mockup de la vista tipo HTML

En cualquiera de los tres escenarios, el código XML es el mismo para todos, por lo que único que hace falta es analizar una parte del mismo y añadirla al producto que se crea.

```
<control controlID="3" controlTypeID="com.balsamiq.mockups::Paragraph" x="153" y="205" w="206" h="244" measuredW="275" measuredH="80" zOrder="2" locked="false" isInGroup="-1">
   <controlProperties>
      <text>Denominaci%F3n%20com%FAn%20de%20los%20medios%20pict%F3ricos%20acuosos%2C%20en%20particular%2C%20se%20refiere%20a%20la%20pintu
```

```
        ra%20al%20agua%20compuesta%20por%20pigmentos%20org%E1nicos%20o%20inorg%E1nicos.%20Es%20una%20t%E9cnica%20de%20pintura%20sobre%20papel%20o%20cart%F3n%2C%20con%20pigmento*%20diluido%20en%20agua%20con%20una%20cantidad%20de%20un%20adhesivo%20suave%20e%20incoloro%2C%20normalmente%20goma%20ar%E1biga*%2C%20goma%20tragacanto*%20o%20cola%20de%20pescado*.%20Los%20colores%20utilizados%20son%20transparentes%2C%20lo%20que%20proporciona%20al%20procedimiento%20una%20luminosidad%20solo%20aventajada%20por%20las%20vidrieras*.%20Se%20emplea%20como%20blanco%20el%20del%20papel%20y%20no%20utiliza%20el%20blanco%20como%20pigmento.%20El%20papel%20debe%20lavarse%20y%20plancharse%2C%20para%20que%20no%20se%20arrugue%20demasiado%20durante%20la%20ejecuci%F3n%2C%20y%20no%20suelen%20ser%20de%20gran%20tama%F1o.%20Para%20extender%20el%20color%20se%20utiliza%20exclusivamente%20agua%2C%20por%20lo%20que%20seca%20r%E1pidamente%3B%20ello%20obliga%20a%20una%20t%E9cnica%20r%E1pida%2C%20suelta%2C%20que%20le%20da%20un%20aspecto%20caracter%EDstico%20y%20personal.%20Usada%20desde%20el%20Renacimiento%20para%20colorear%20trabajos%20de%20l%EDnea%2C%20a%20partir%20del%20XVIII%20se%20independiz%F3%20como%20t%E9cnica%20aut%F3noma</text>
    </controlProperties>
</control>
```

Código 18 - XML del objeto Paragraph de la vista HTML

Como vemos, la etiqueta <text> contiene un texto que será la descripción del producto y que contiene símbolos en HEX que habrá que analizar y convertir a ASCII con el analizador que se diseñó. Si esta vista en concreto actualiza una vista concreta anterior de copyright o glossary, el esquema del producto quedará así.

Producto	
Columna en Parse.com	**Valor que obtiene del Mockup**
Name	No se modifica con esta vista
Description	Valor tomado de la etiqueta <text>
Category_id	No se modifica con esta vista
Image_url	No se modifica con esta vista
Spider_id	No se modifica con esta vista
Param	No se modifica con esta vista
Ordering	No se modifica con esta vista
PARAMETROS PARA EL PARAMETRO "PARAM"	
Parámetro	**Valor que obtiene del Mockup**

Type	No se modifica con esta vista
Resource	No se modifica con esta vista
Subtype	No se modifica con esta vista
Action	No se modifica con esta vista

Tabla 19 - Producto de la vista HTML con las vistas glossary y copyright

Si por el contrario, esta vista genera un producto nuevo, el esquema quedará de la siguiente forma:

Producto	
Columna en Parse.com	**Valor que obtiene del Mockup**
Name	Se toma del nombre del archivo, segundo atributo
Description	Valor tomado de la etiqueta <text>
Category_id	Se busca dentro de todos las categorías ya analizadas anteriormente si existe una categoría que tenga el mismo nombre que el nombre que se obtiene de la segunda parte del nombre del archivo.
Image_url	No aplica a esta vista
Spider_id	Toma el valor siguiente al último producto o categoría añadido.
Param	Se explican en una tabla aparte.
Ordering	Toda el valor siguiente a la último producto añadido en esta categoría.
PARAMETROS PARA EL PARAMETRO "PARAM"	
Parámetro	**Valor que obtiene del Mockup**
Type	Se usa el tipo "HTML"
Resource	No aplica a esta vista
Subtype	No aplica a esta vista
Action	No aplica a esta vista

Tabla 20 - Producto de la vista HTML individual

5.5. Arquitectura del modelo.

La arquitectura de este modelo está basado en que a partir de una colección de archivos dados, somos capaces de analizarlos en el orden correcto y solo cuando son necesarios y obtener como resultado los mismos datos en un formato distinto, entendible por otra aplicación. Por ello, podemos dividir la arquitectura en varias secciones:

- Obtención de los archivos y ordenación de los mismos. Manejo de archivos no requeridos.
- Analizado de los archivos. Patrones de diseño utilizados para la implementación.
- Herramientas diseñadas para proporcionar escalabilidad al modelo.

Cuando el modelo obtiene la orden de analizar los archivos, este pasará todos y cada uno de los archivos por el proceso de ordenación y posteriormente por el proceso de parseado.

Antes de obtener los archivos, hemos de tener claro cómo vamos a guardar los datos. Como esta arquitectura se basa en el catálogo SpiderCatalog, en unión con Parse.com, usaremos una estructura similar que podamos posteriormente usar sin procesado para pasar los datos a Parse.com. De esta forma, tendremos dos vectores de objetos que serán diccionarios que contendrán toda la información ordenada por claves. Se necesitan dos vectores, dado que usaremos uno para las categorías y otro para los productos.

Ordenación de los archivos y clasificado

Una vez que los archivos han sido sincronizados desde la base de datos, estos se encuentran en la memoria interna del dispositivo. Como podemos acceder a esta en cualquier momento, solo vamos a acceder a ellos cuando los necesitemos, creando una copia de cada uno de los archivos, asegurándonos de esta manera que siempre que queramos podemos obtener los archivos nuevamente.

Si nos fijamos en la estructura que hemos presentado de cada una de las vistas, vemos que necesitamos formar un árbol, partiendo siempre de una categoría ´0´que podemos obtener únicamente de una de las vistas. Es por tanto claro, que necesitamos encontrar esta vista la primera y ponerla la primera en la cola para ser analizada.

Para el resto de vistas, a priori no nos importa el orden que han de tener. Pero si nos fijamos en como formamos el árbol, todas las demás vistas van a depender de los hijos que cuelgan de la categoría principal. En el caso de esta aplicación, no puede darse el caso de que una vista no tenga definido un padre que esté en la categoría principal. Además, dado como hemos planteado la estructura de los nombres de la aplicación, podemos rápidamente analizar todos los nombres y ordenarlos de forma que con una sola pasada por toda la lista, todos las vistas sean analizadas y no tengan que ser pospuestas por que no se encuentre un sitio donde enlazarlas.

Por lo tanto, es clave que la primera vista sea analizada y añadida al sistema de vectores, y una vez tengamos esa información podamos analizar el resto de vistas de una forma ordenada y sistemática.

Cabe recordar cómo se define el nombre de los archivos. Estos se componen de tres partes, separadas por un guion bajo, y en el caso de que incluya un espacio, se usara el código hexadecimal del espacio (20), antecedido del símbolo de porcentaje (%) para indicar que se trata de un numero hexadecimal y no parte del nombre.

Ilustración 54 - Nombre de los archivos XML

En la imagen anterior, podemos ver que la parte subrayada con naranja sería el tipo de vista que define este código, la segunda parte, el padre del que procede y por último el nombre del producto. En el caso de que la vista, no cree ningún producto, sino un vector de ellos, el nombre se deja vacío.

Por tanto, si tenemos en el nombre de los archivos, el padre del que proviene, podemos fácilmente ordenarlos y que a la hora de analizarlos todos estén en orden. En el caso de que una vez analizado la primera vista, que será la de Menú Izquierda y ordenados todos los archivos en el mismo orden que están en la vista, haya algún archivo que no tenga ninguna unión, estos archivos se marcaran como no válidos y no se analizarán.

El criterio para ordenar los archivos será ordenarlos en el mismo orden que están en la vista del menú. En el caso del tipo glossary y del tipo copyright, se buscará siempre poner primero la vista principal y después todos los HTML que definen los productos interiores de dichas vistas.

Analizado de los archivos.

El primer problema que nos encontramos a la hora de querer analizar los archivos es que cada una de las vistas es diferente de la anterior, por lo que no podemos hacer un único analizador. Tampoco podemos generar una estructura que no sea escalable y que requiera mucho mantenimiento posteriormente.

Para solucionar este problema se recurre al patrón de diseño Abstract Factory. Este patrón, permite generar diferentes objetos, todos los cuales proceden de una única familia. En el caso de nuestra aplicación, el usuario requiere analizar una vista y la factoría le proporcionará un parseador que entienda esa vista.

Con el nombre del archivo podemos saber qué tipo de vista es. Cuando obtenemos ese nombre, podemos pedirle a la factoría que nos cree una instancia a un parseador que entienda como parsear la vista a la que vamos a enfrentarnos. Por tanto, todos los parseadores han de ser objetos de la factoría y tener todos el mismo método para analizar la vista.

Una vez que tenemos la instancia al método que puede analizar la vista concreta, dependerá de cada uno de los diferentes parseadores, el analizar la información y proveer de uno o varios nuevos objetos al vector de categorías o al vector de productos de la forma que se explicó en secciones anteriores.

Para ello, a cada parseador hemos de pasarle cinco datos:

- El archivo que queremos que analice.
- El vector de categorías que tenemos formado.
- El vector de productos que tenemos formado.
- El ultimo valor de spiderID.
- Una variable de descarte.

La variable de descarte la usaremos para saber si, aun habiendo ordenado los archivos, hemos cometido algún error y esta vista no se puede conectar al árbol de ninguna manera.

Una vez que todos los archivos han sido analizados, y tenemos los dos vectores creados, podemos afirmar que hemos acabado el proceso y que es el momento de sincronizar nuestra base de datos con Parse.com. Para ello usamos la API que proporciona Parse.com

y para cada uno de los diferentes objetos del diccionario, hacemos un PUSH a la nube, donde quedarán guardados estos archivos. El método PUSH es muy sencillo y se muestra en el siguiente código.

```
- (void) updateProducts:(NSMutableArray *) products
andCategorys: (NSMutableArray *) categories{
    for (NSDictionary *product in products){
        [self updateProduct:product];
    }
    for (NSDictionary *category in categories){
        [self updateCategory:category];
    }

}

- (void) updateCategory: (NSDictionary *) dictionaryParse {
    PFObject *prueba = [PFObject
objectWithClassName:@"wp_spidercatalog_product_categories"];
    NSArray * allKeys = [dictionaryParse allKeys];
    for (NSString * key in allKeys) {
        [prueba setObject:[dictionaryParse objectForKey:key]
forKey:key];
    }
    [prueba saveInBackground];
}

- (void) updateProduct: (NSDictionary *) dictionaryParse {
    PFObject *prueba = [PFObject
objectWithClassName:@"wp_spidercatalog_products"];
    NSArray * allKeys = [dictionaryParse allKeys];
    for (NSString * key in allKeys) {
        [prueba setObject:[dictionaryParse objectForKey:key]
forKey:key];
    }
    [prueba saveInBackground];
}
```

Código 19 - Sincronización con Parse.com

Herramientas diseñadas.

Aunque la forma de analizar las vistas sean distintas, todas ellas están analizando archivos XML. Debido a esto, y para usar siempre las herramientas lo más universales posibles, hemos diseñado un objeto que contiene todos los métodos necesarios para parsear un archivo XML de Balsamiq.

Todas las funciones son básicas y pueden ser usadas por cualquier parseador. Aquí se presentan todos y se explica su función:

- **readFile**. Esta función obtiene el nombre de un fichero y retorna el contenido del mismo. Se usa para leer cualquier clase de fichero con cualquier extensión.
- **findCategory**. Aunque el nombre no es muy explícito, este método encuentra un objeto determinado en el XML y retorna la información alojada entre las etiquetas proporcionadas. Es uno de los métodos más usados, dado que siempre que se quiere obtener la información contenida en alguna etiqueta concreta se usa este método.
- **convertString**. Convierte una string en un NSNumber. Muy usado para poder manejar la base de datos correctamente, dado que los datos se almacenan en string, pero para trabajar con ellos necesitamos números.
- **generarCSV**. Este analizador permite generar archivos CSV si se quiere y no generar un update a la base de datos. Esta función se diseñó al principio, pero cuando se completó la sincronización con la base de datos no se usó más.
- **addNewCategory**. Existen dos funciones con el mismo nombre, una se usa para añadir una categoría y otra para manejar y añadir un producto y una categoría a los vectores. Otra función que se usa por todos los analizadores.
- **addNewProduct**. Cumple la misma función que la anterior pero con los productos.
- **setOrdering**. Busca cuál es el último ordering y retorna el siguiente valor para los productos.
- **setOrderingCategories**. Misma función que la anterior pero para las categorías.
- **setParamCategories**. Como los parámetros han de formarse de una forma particular, esta función toma todos los valores que han de tener los param y les da formato.
- **setParamProducts**. Misma función que la anterior pero usada para los productos.

Por otro lado, se diseñó también un objeto que convierte todos los símbolos que tiene el XML en hexadecimal a su valor en ASCII. Esta función es vital para poder convertir el código correctamente.

6. Arquitectura del módulo Parse.com a Balsamiq

El módulo de Parse.com a Balsamiq se basa en que el sistema debe ser full dúplex y que el usuario una vez tenga los datos en Parse.com, ya sea con el uso de esta herramienta o con las anteriores o incluso con futuras, pueda obtener los mockup.

Este módulo se ha pensado para que si esta herramienta se quiere incluir, por ejemplo en una herramienta web, cuando el usuario haga modificaciones en Parse.com, este pueda obtener una pre visualización de lo que va a aparecer en la aplicación de manera instantánea.

Hay que tener en cuenta, que en este módulo todo ocurre al contrario que en el módulo Balsamiq a Parse.com. Si en Balsamiq necesitamos un mockup para generar varios productos o categorías, ahora usaremos estas categorías todas juntas para generar un solo mockup. Esto hace que la estrategia sea un poco distinta. En el caso de esta aplicación, usaremos el IUMA framework realmente para descargar toda la información, dado que la estructura que nos proporciona y las funciones que nos permite usar, nos ayudarán a poder analizar de una manera más sencilla toda la información y extraer solo la información que necesitamos para los mockup.

Anteriormente, sabiendo el tipo de mockup que era, sabíamos qué tipo de información tenemos que poner en la categoría. Ahora ocurre al contrario, analizando la información que tenemos, sabremos qué tipo de mockup tendremos que generar.

Con toda esta información tenemos claro que realmente, la información que tenemos disponible en parse.com no es realmente la información que necesitamos para generar los mockup y que primeramente hay que analizar esta información para generar la información que realmente necesitamos. Otro dato importante es que aunque vamos a usar el IUMA framework, la información está en un formato que no es realmente fácil de navegar. Como ya tenemos experiencia con un formato, usado anteriormente en el módulo Balsamiq a Parse.com, trataremos primeramente de convertir la información de IUAM framework en un formato ya conocido de dos vectores, uno de categorías y otro de productos.

6.1. Diagrama de la arquitectura del módulo.

Si ponemos todo lo anterior descrito en un diagrama, obtenemos una visualización de cómo hemos montado la arquitectura de este módulo.

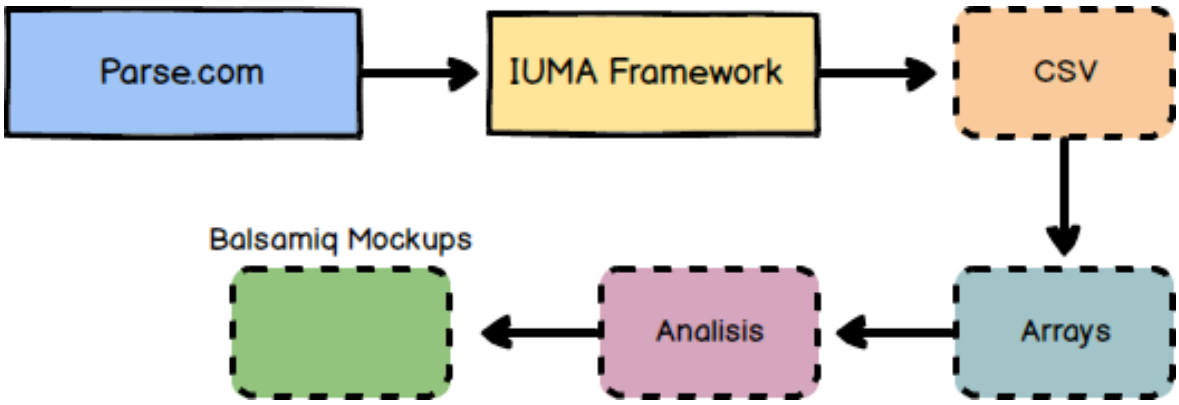


Ilustración 55 - Diagrama de funcionamiento del módulo Parse.com a Balsamiq

Vemos que este módulo tiene varias fases, y que para realmente obtener la información hemos de modificarla varias veces hasta poder obtener el resultado que queremos. Para eso, usamos el IUMA framework para descargar toda la información desde Parse.com y mediante las mismas herramientas que usamos antes y que usaremos también en los siguientes módulos, generamos dos archivos .CSV, uno con las categorías y otro con los productos.

Una vez que tenemos los dos archivos, los separamos en diferentes vectores para tener un diccionario de cada producto o categoría con la información que contiene el CSV. Ya aquí, tenemos casi la información que necesitamos y solo haría falta analizar para después generar todos los archivos .xml que necesitamos.

6.2. Análisis.

El proceso de análisis es justamente el contrario al proceso de análisis que vimos en el capítulo anterior. Para poder llevar a cabo este proceso, hemos de tener los datos con la información adecuada. Es por ello, que una vez hemos construido los dos vectores principales, cambiamos la información que tiene, por una información que nos proporciona todos los datos para construir nuestro mockup.

Realmente las categorías ya tienen toda la información de por sí, y solo hay que completar con los metadatos, pero los productos son un poco distintos, dado que en el mockup la información que se representa depende también de su categoría padre. El siguiente código, representa como serán los objetos que introduciremos en los nuevos vectores con la información que queremos.

```
NSDictionary *product=@{@"name":name,
                        @"type":type,
                        @"image_url":imageURL,
                        @"category_id":catID,
                        @"description":description,
                        @"action":parAction,
                        @"resource":parResource,
                        @"parent_name":parentName};
```

Código 20 - Descripcion de información necesaria de los productos

Como podemos ver, este producto es distinto a la información que almacena realmente en parse.com. La mayoría de la información, está en el mismo producto, pero por ejemplo el tipo que tiene que ser en el mockup o el nombre del padre es información que se saca de la categoría a la que pertenece o del metadato del mismo. De la misma forma, las categorías tendrán el siguiente formato:

```
NSDictionary *category = @{@"name":name,
                           @"type":parType,
                           @"image_url":imgURL,
                           @"spider_id":spiderID};
```

Código 21 - Descripción de información necesarias de las categorías

Una vez que tenemos todos los productos y todas las categorías con la información que queremos tener, podemos ir formando los diferentes mockup. Para empezar, hemos de montarlos de la misma forma en que la que montamos Parse.com, es decir, analizando primeramente el mockup principal, leftMenu.

6.2.1. Menú Izquierda (leftMenu).

Para analizar correctamente este mockup, tenemos que encontrar primeramente la categoría con spiderID igual a 1, dado que sabemos que es siempre la categoría de la que se parte. Una vez encontrado, buscamos todas las categorías que dependen de la misma, que en el caso de las capacidades de nuestra aplicación son todas. Del mismo modo, buscamos todos los productos que tienen como categoryID también el 1. Con esta

información, podemos ya generar el código XML que permite generar la categoría principal, dado que tenemos gracias a como hemos guardado la información, la imagen y el texto, así como los posibles links que pueden tener estos.

6.2.2. Product y HTML.

El tipo product es el más sencillo de analizar, dado que un producto se relaciona directamente con su mockup. Como vemos, tenemos toda la información en el diccionario para poder genera el mockup del que hemos partido, por lo que solo hace falta colocar la información en el sitio correcto del mockup y este estará finalizado.

Del mismo modo ocurre con el tipo HTML, que solo requiere de la información almacenada en el diccionario para poder formar el mockup. Es por esto que se eligió procesar la información primeramente, para poder ahórranos procesado posteriormente.

6.2.3. Gallery, Copyright y Glossary.

Estos tres tipos de mockup tienen en común que necesitan de varios productos para poder formar todo el conjunto. Se basan en dos conjuntos, un primer conjunto que digamos es el principal, de la que parten el resto de los productos que formarían el segundo conjunto. En el caso de gallery, está compuesto solo por el primero.

Para analizarlo correctamente, buscamos entre todos los productos los que deben ir en este conjunto y una vez que encontramos uno, formamos el segundo conjunto y añadimos la información necesaria de este conjunto al principal. Una vez que tenemos todos los productos del primer conjunto podemos decir que tenemos todo el tipo analizado.

6.3. Conclusiones.

Podemos concluir que la forma de analizar los datos para poder obtener los mockup ha sido mucho más sencilla después de haber obtenido la información del módulo Balsamiq a Mockup, dado que los datos que se procesan son los mismos, y esto solo supone el proceso inverso al primero.

Con esto, también aprendemos que para los siguientes módulos, será mejor acabar uno completamente, dado que cuando se obtiene un camino completo, revertirlo resulta un proceso mucho más sencillo.

Por último, cabe destacar, que una vez acabado el proyecto, se decidió modificar este módulo. Se observó que se eliminaban muchas de barreras y se otorgaba más universalidad si se usaba no el IUMA framework en si para generar el archivo CSV, sino los módulos que se explican en el capítulo 7. Con estos módulos también se obtienen archivos CSV y además, al sincronizar los dos módulos para trabajar con ellos, todas las mejoras que se incorporen en uno serán transferidas inmediatamente a este.

7. Arquitectura del módulo Parse.com a Mind map y viceversa

En capítulos anteriores hemos estado trabajando con archivos que hemos denominados jerárquicos, dado que fácilmente podemos ver la jerarquía que tienen unos con otros, pero realmente no son jerárquicos. Durante las primeras fases del desarrollo de la aplicación, era muy difícil poder entender cuál eran las relaciones que tenían unas vistas con otras y de donde salía cierta información y donde tenía que ser almacenada.

También se hacía difícil poder entender si la aplicación estaba bien diseñada. Además, cuando se acababa, es difícil modificar algo, dado que hay que abrir el Mockup y la información de un producto puede estar en varios Mockups, o nos hace falta cambiar varias vistas, por lo que no es muy sencillo de usar.

Lo que hacía falta era una herramienta con la que pudiéramos ver la información de la aplicación en general y de una sola vez de forma totalmente jerárquica y por otro lado que en esa vista, se pudiera modificar la información de manera sencilla. Para solucionar este problema, se recurrió a los programas que los tutores del proyecto usaban para pensar las aplicaciones y como debían ser, los mind maps. Estos mapas o árboles, permitían ver de forma sencilla toda la aplicación, también añadir, eliminar o modificar datos de forma muy sencilla. Además la herramienta usada para visionarlo, permitía importar y exportar estos mapas a un único fichero.

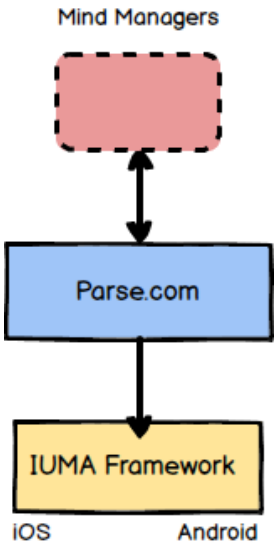


Ilustración 56 - Diagrama de flujo del módulo Xmind - Parse

Debido a esto, se decidió añadir a la herramienta la posibilidad de usar esta aplicación como parte del procesado de datos que permitía crear la aplicación, dotando al usuario de una forma fácil y sencilla de visionar los datos y de modificarlos. Por tanto, el usuario puede:

- Generar a partir de Parse.com un Mind map que puede abrir con las aplicaciones de Freemind o Xmind.

- Usar esta información para asegurarse de que la aplicación está correctamente construida y que todas las secciones tienen la información.
- Modificar la información del árbol, añadiendo, eliminando o modificando el árbol usando las herramientas que proporcionan las aplicaciones de Freemind y Xmind.
- Guardar y exportar la información al formato adecuado que la herramienta de analizado entienda.
- Actualizar Parse.com o generar los archivos que Parse.com pueda entender.

Por tanto, aunque dividimos este módulo en dos secciones, una que permite ir desde Parse.com al mind map y otra que va desde el mind map a Parse.com, se trata de un solo módulo, que no se entiende la utilidad de uno sin el otro.

En este capítulo se presentarán los siguientes puntos.

- Presentación de los Mind Managers que se usan en este PFC.
- Arquitectura y módulos utilizados para analizar Parse.com y parsear los datos al mind manager.
- Arquitectura y módulos utilizados para analizar el mind manager y parsear los datos a Parse.com.
- Desarrollo de un ejemplo para mostrar el funcionamiento individual del módulo.

7.1. Mind Managers.

Los mind maps son diagramas que se usan principalmente para organizar de forma visual la información, normalmente en forma de árbol. Un mapa de este tipo es jerárquico y muestra las relaciones entre todas sus partes. Siempre parte de un punto inicial y se desarrolla de forma que todos los conceptos siempre tienden a uno solo.

Para la ejecución de este proyecto se seleccionó el mind manager que los tutores estaban ya utilizando, dado que de esta forma era más sencilla la integración del mismo en la aplicación. El principal mind map es Xmind. Este programa permite la creación de forma intuitiva de árboles y además puede dotar a los árboles de diferentes iconos para hacer más visual los árboles.

Un ejemplo de árbol creado con Xmind es el siguiente.

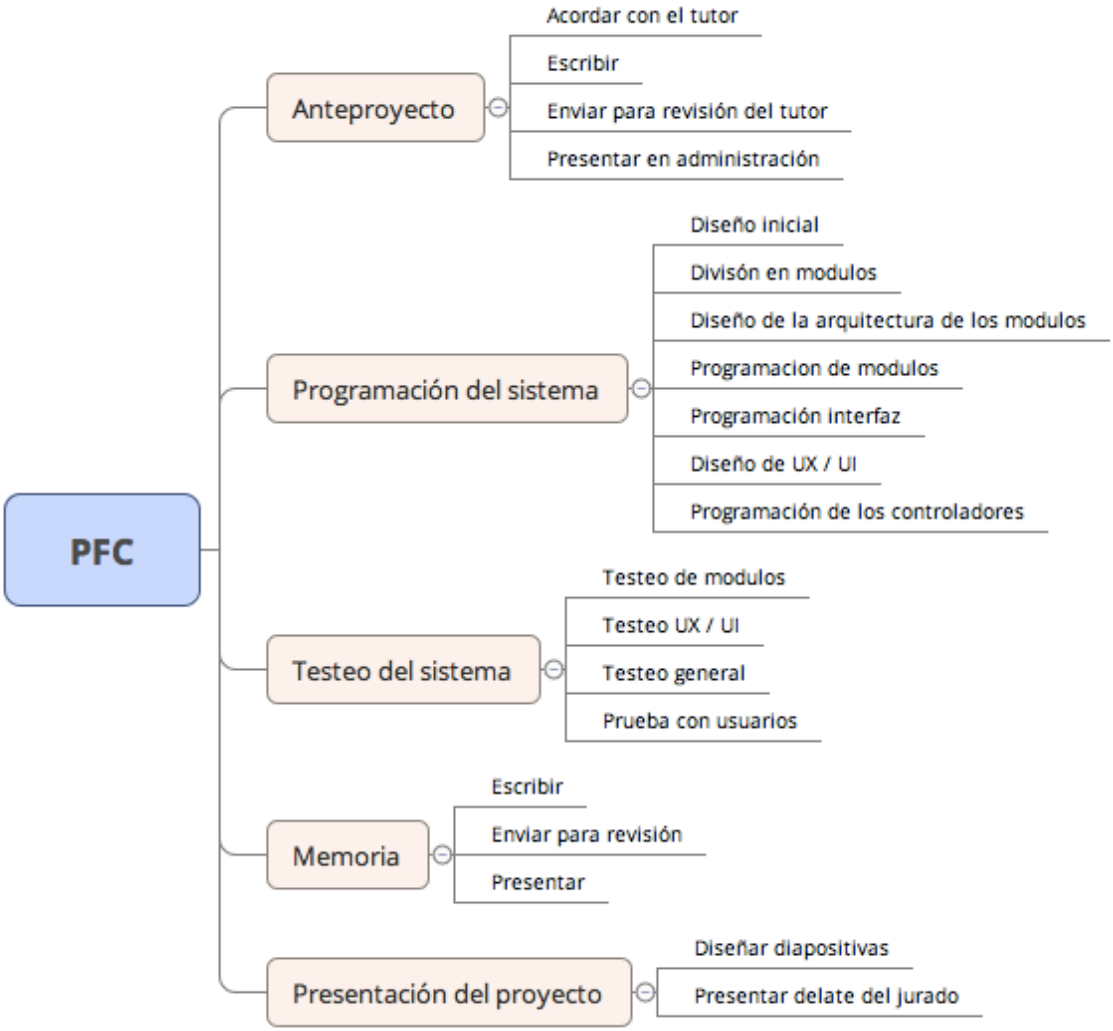


Ilustración 57 - Árbol ejemplo en Xmind

Este árbol muestra cómo sería la realización general de un PFC. Como vemos todos los subtemas refieren a un mismo tema, y todos las categorías de los subtemas, hacen posible el tema principal.

Una de las características que tiene xmind que resulta muy interesante para el proyecto es que se pueden comprimir y expandir cada hijo o padre, de forma que resulte más sencillo ver cómo está todo el árbol y cuáles son las principales características de los mismos. Esto resulta muy interesante, sobre todo para el futuro, cuando hagamos los árboles de aplicaciones que tengan muchas categorías y productos, donde todos ellos tendrán parámetros y será más difícil de ver si está todo expandido. De la misma forma, podemos comprimir una parte y expandir otra para centrarnos en una parte concreta del árbol.

Pero el problema principal que tiene este programa es que la forma en la que guarda los archivos es muy compleja y llevaría mucho tiempo diseñar una herramienta capaz de generar estos archivos. Para resolver este problema, usaremos un programa puente que es muy sencillo, dado que es de software libre, como es Freemind. Esta aplicación usa simple código XML jerárquico de representación similar a un JSON.

Este XML es muy sencillo y se parece mucho a lo que podemos asociar en programación a un diccionario, pero en estilo XML. Pero luego vendría el problema de cómo usar Xmind, que en términos generales es una herramienta mucho más potente que Freemind. Pero al ser más potente, la aplicación trae un importador – exportador a diferentes formatos y uno de ellos es Freemind.

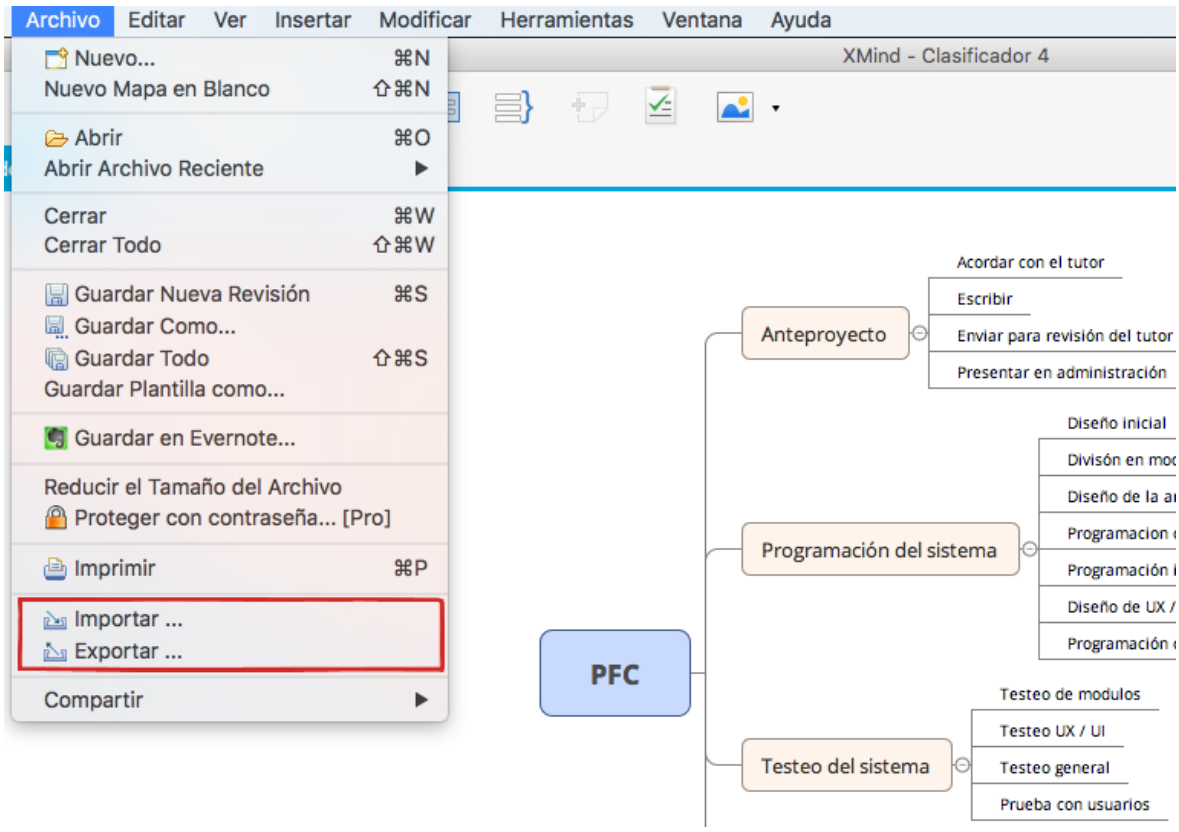


Ilustración 58 - Localización de las opciones Exportar e Importar

De esta forma podemos usar Xmind, habiendo generado primero archivos Freemind. Para el usuario no importa, no tiene que usar el programa de Freemind para nada, y de la misma forma, si prefiere puede usar también Freemind, dado que los archivos que generará nuestro modelo podrán ser usados en ambos programas.

En general, usaremos xmind durante el desarrollo de este PFC, sobre todo porque su interfaz gráfica es mucho más agradable y sencilla de usar.

Lo que se busca con este árbol es hacer exactamente los mismos árboles que anteriormente, pero de forma automática, y dejar de observar la información en forma de tablas, como se puede ver en Parse.com y ver la información en forma de árbol. Si tomamos como ejemplo una aplicación cualquiera que se ha usado durante el desarrollo del PFC, esta sería la forma:

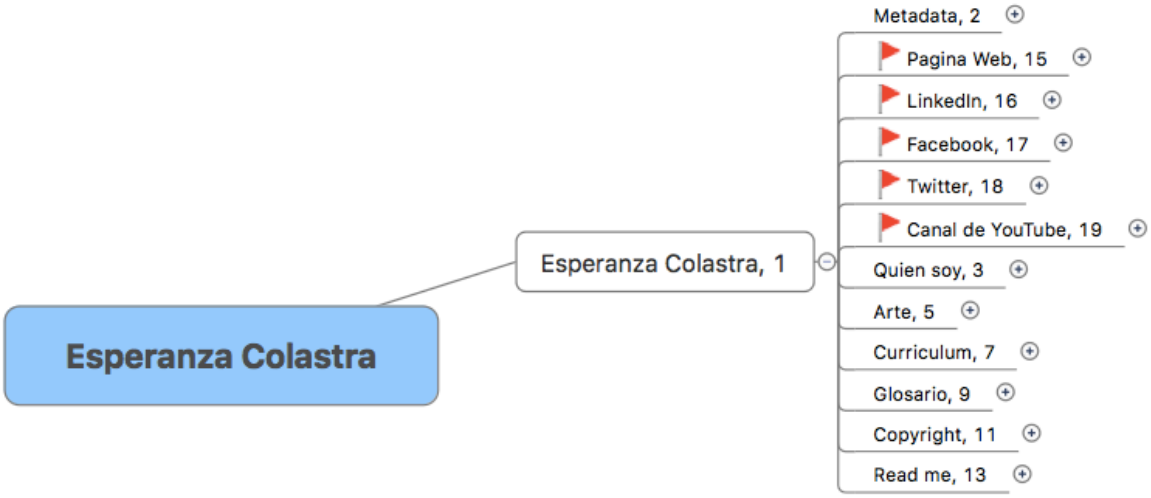


Ilustración 59 - Árbol ejemplo de una aplicación

Este árbol representa la aplicación creada durante este proyecto. Como se ve, este árbol tiene todo lo necesario, tanto productos como categorías. Para diferenciarlos y que sea fácilmente diferenciables, los productos llevan delante el símbolo de la bandera roja, mientras que las categorías no llevan nada.

De igual manera, queremos que la información sea claramente diferenciable. Para conseguirlo, hemos de tener la información más importante siempre lo más visible posible. Después de diferenciar entre productos y categoría, hay dos parámetros en nuestro sistema muy importantes, como son el nombre de la categoría o producto y su ID. Estos ID son los que usaremos para decir quién es su padre y quien es hijo de quien. También, para no confundir, el metadato de cada categoría, no tendrá el nombre igual que su categoría, sino que se le denominará “metadata”.

El árbol por tanto nos da una visión general de cómo está distribuida la aplicación y si ahora quisiéramos cambiar algo en varias categorías o productos, solo tendríamos que modificar los parámetros y volver para atrás en el proceso. Los parámetros se encuentran siempre dentro de los categorías, y hemos de profundizar hasta el final para encontrar estos. La siguiente figura representa los metadatos de cada uno de ellos.

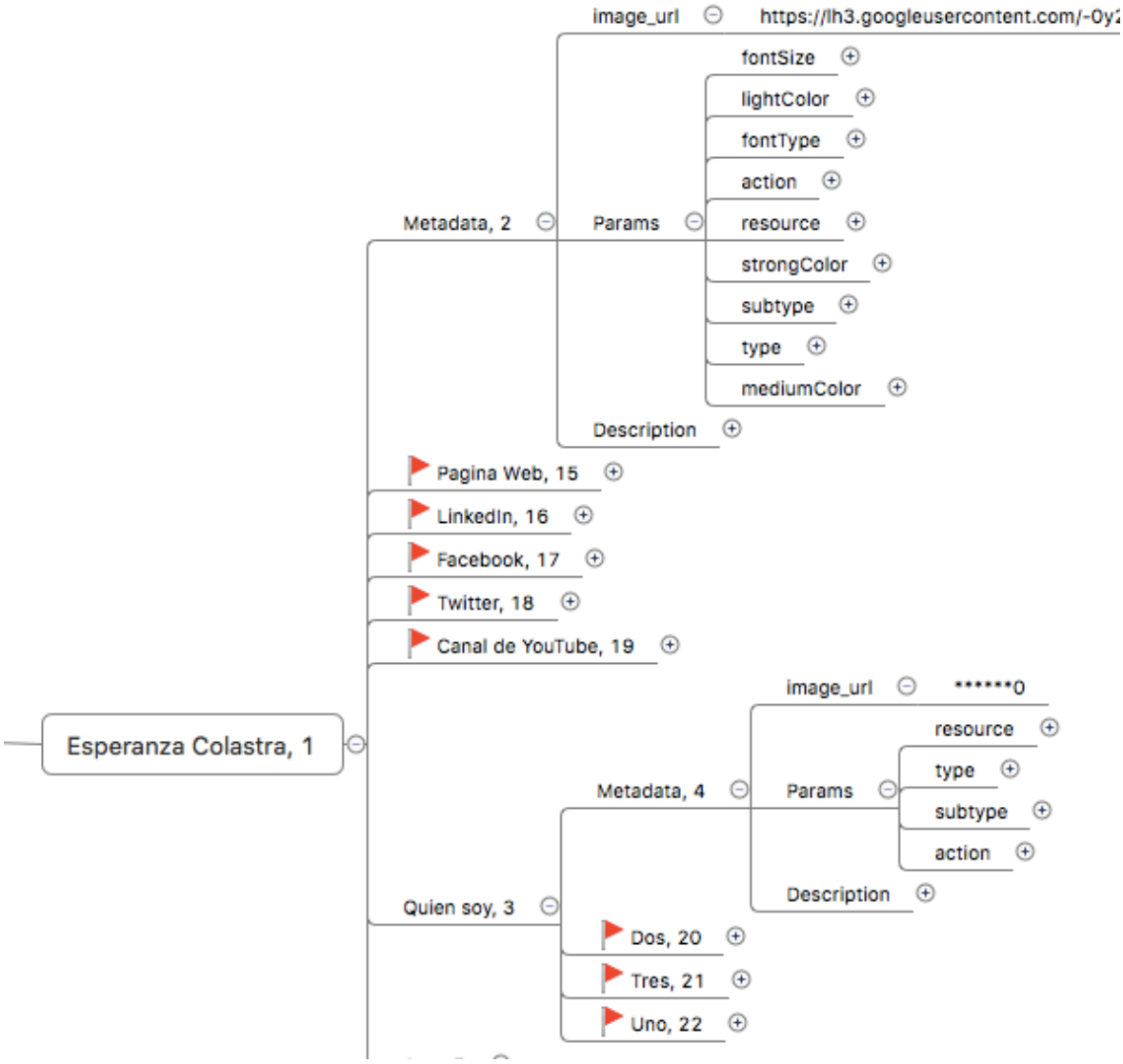


Ilustración 60 - Árbol ejemplo con parámetros

Como vemos, podemos ver todos los parámetros de una categoría o de un producto de una forma muy sencilla. Igualmente los parámetros internos también se agrupan y son más sencillos de reconocer en este tipo de árbol que en las tablas de Parse.com.

En general, conseguimos que se entienda mucho mejor e incluso si es necesario presentar cuál es la estructura de la aplicación un mapa jerárquico siempre es más sencillo

de entender que una simple tabla o la aplicación misma, dado que en la aplicación se puede tardar más tiempo en encontrar lo que queremos modificar o ver.

Una vez entendemos como queremos obtener el árbol, necesitamos programar el módulo que nos proporcionará esta información.

7.2. Módulo Parse.com a Xmind.

Para usar poder usar este módulo, usaremos el IUMA framework, dado que este framework trae consigo algunas funciones ya diseñadas, sobre todo para la extracción de los datos desde Parse que nos sirve para no tener que rediseñar todas las funciones.

Si pensamos que la información en Parse.com está en forma de árbol, entendemos que para poder analizar esta información de forma correcta, debemos primero entender como analizar árboles. Para recorrer árboles, hay que usar siempre recursividad, dado que es la única forma de recorrerlos de forma efectiva. Además, esta forma de recorrer el árbol nos va a servir también para ir generando nuestro XML, que como recordamos en Freemind usa una estructura en forma de árbol.

Si analizamos un XML genérico, generado por Freemind, vemos que su estructura se divide en tres partes generales, una primera parte donde se incluye la información general de qué tipo de XML es, una segunda parte que incluye todo el árbol y una tercera parte que cierra el árbol. Además, da la casualidad, que la primera parte y la tercera son siempre las mismas. Por esto, nos tenemos que centrar solo en analizar la segunda parte.

Esta parte es más difícil y no es siempre la misma, pero para analizar información, necesitamos encontrar patrones que sean siempre iguales. En el caso de nuestros árboles tenemos varios patrones:

- **Categorías**. La forma en la que se desarrollan las categorías son siempre las mismas, por lo que podemos siempre que nos encontremos una categoría, añadirla de la misma forma, con el mismo formato.
- **Productos**. Del mismo modo que las categorías, los productos que hay dentro de una categoría son siempre igual.

- **Parámetros**. Tanto los parámetros de las categorías como la de los productos tienen la misma estructura, pero con diferentes valores. Aun así, todos son comunes, por lo que siempre que tengamos parámetros, usaremos la misma estrategia.

Teniendo en cuenta que solo tenemos tres tipos de desarrollos dentro del XML, se hace más sencillo entender cómo podemos analizar Parse.com y convertirlo en xmind.

Recursividad de categorías.

Una vez que recorriendo el árbol, nos encontramos con una categoría, lo primero que tenemos que hacer es obtener su nombre. Con el nombre, podemos obtener el producto metadato que esta categoría tiene, usando una función del framework del IUMA, que se denomina "getMainProductFromCategory".

Una vez que tenemos el producto principal, podemos obtener sus parámetros. Esta sección se explicará con el analizado de los parámetros por separado.

Si vemos como está configurada una categoría, esta tiene una serie de parámetros que hay que escribir en forma de XML. Para ello, podemos usar siempre el mismo formato y cambiar simplemente la información que contiene. Al fin y al cabo, lo único que tenemos que generar es un string que contenga toda la información de la categoría, como la siguiente:

```
NSString *paramsCategoryXML = [NSString
stringWithFormat:@"\t\t\t\t<node FOLDER=\"true\" TEXT=\"%@\"
CREATED=\"1429767744093\" ID=\"%@\"
MODIFIED=\"1429767744093\">\r\t\t\t\t\t<node TEXT=\"image_url\"
CREATED=\"1429767744093\" ID=\"%@\"
MODIFIED=\"1429767744093\">\r\t\t\t\t\t\t<node TEXT=\"%@\"
CREATED=\"1429767744093\" ID=\"%@\" MODIFIED=\"1429767744093\"
/>\r\t\t\t\t\t</node>\r\t\t\t\t\t<node TEXT=\"Params\"
CREATED=\"1429767744093\" ID=\"%@\"
MODIFIED=\"1429767744093\">\r%@\t\t\t\t\t</node>\r\t\t\t\t\t<nod
e TEXT=\"Description\" CREATED=\"1429767744094\" ID=\"%@\"
MODIFIED=\"1429767744094\">\r\t\t\t\t\t\t<node TEXT=\"%@\"
CREATED=\"1429767744093\" ID=\"%@\" MODIFIED=\"1429767744093\"
/>\r\t\t\t\t\t</node>\r\t\t\t\t</node>\r\t",[NSString
stringWithFormat:@"Metadata, %@",mainProduct.sc_id],[[NSUUID
UUID] UUIDString],[[NSUUID UUID]
UUIDString],category.sc_category_image_url,[[NSUUID UUID]
UUIDString],[[NSUUID UUID] UUIDString],paramsXML,[[NSUUID UUID]
UUIDString],category.sc_description,[[NSUUID UUID] UUIDString]];
```

Código 22 - Ejemplo de cómo convertir un nodo en código XML

Este código, nos permite agrupar todos los parámetros que tiene una categoría y agruparlos en el formato que tiene que tener para poder pasar a Freemind. La única parte que no está incluida aquí es una variable llamada paramsXML que es la que explicaremos más adelante. Una vez que tenemos esta información podemos añadirla al final de nuestro XML, a continuación añadiríamos los productos que tiene (que se explican más adelante).

Pero una categoría puede tener subcategorías, por lo que una vez finalizados con los productos, tendremos que ver si hay subcategorías dentro de esta categoría y repetir el proceso con cada una de ellas y ponerlas cada vez que se acaba una al final del archivo, siempre siguiendo para cada subcategoría el mismo formato.

Una vez analizado un XML para una categoría, el resultado sería similar al siguiente:

```
<node TEXT="Esperanza Colastra" ID="1" CREATED="1429767744092"
MODIFIED="1429767744092">
        <node TEXT="Esperanza Colastra, 1" ID="1" CREATED="1429767744093"
        MODIFIED="1429767744093" POSITION="right">
                <node FOLDER="true" TEXT="Metadata, 2"
                CREATED="1429767744093" ID="CEE41C73-B36C-428A-8E93-
                7A768522A31D" MODIFIED="1429767744093">
                        <node TEXT="image_url" CREATED="1429767744093"
                        ID="690457AC-F99E-49C2-959E-4F37AE8AF49C"
                        MODIFIED="1429767744093">
                                <node TEXT="https://lh3.googleusercontent.com/-
                                0y22_K6ljRQ/AAAAAAAAAAI/AAAAAAAAAAA/8GY3
                                bJxrch0/photo.jpg" CREATED="1429767744093"
                                ID="8F934D50-CDD3-4948-B79A-B499FA37AAE5"
                                MODIFIED="1429767744093" />
                        </node>
                        <node TEXT="Params" CREATED="1429767744093"
                        ID="40D724F6-A96D-429F-A9BC-C5CC85E1A4CE"
                        MODIFIED="1429767744093">
                        ...
                        </node>
                        <node TEXT="Description" CREATED="1429767744094"
                        ID="BFA4611B-ED7C-4D17-A297-3F1FC585307B"
                        MODIFIED="1429767744094">
                        <node TEXT=" " CREATED="1429767744093"
                        ID="1002D51C-A3E3-4BE2-AD9E-2CB6CB0BC754"
                        MODIFIED="1429767744093" />
                </node>
        </node>
```

Código 23 - Código XML resultado

Por tanto, cada categoría finaliza cuando acaban sus subcategorías y cuando no quedan subcategorías cuando acaban sus productos.

Recursividad de productos.

Los productos siempre son parte de una categoría, por lo que para obtenerlos usaremos la función "getProductsFromCategoryWithoutRoot". Esta función nos proporciona un vector de productos sin incluir el producto metadato. Una vez que tenemos este, para cada uno de los productos que tendremos, podemos generar el código XML correspondiente:

```
NSString *nodeXML = [NSString stringWithFormat:@"\t\t\t<node
TEXT=\"%@\" CREATED=\"1429767744093\" ID=\"%@\"
MODIFIED=\"1429767744093\">\r\t\t\t\t\t<icon BUILTIN=\"flag\"
/>\r\t\t\t\t\t<node TEXT=\"Metadata\" CREATED=\"1429767744093\"
ID=\"%@\" MODIFIED=\"1429767744093\">\r\t\t\t\t\t\t<node
TEXT=\"image_url\" CREATED=\"1429767744093\" ID=\"%@\"
MODIFIED=\"1429767744093\">\r\t\t\t\t\t\t\t<node TEXT=\"%@\"
CREATED=\"1429767744093\" ID=\"%@\" MODIFIED=\"1429767744093\"
/>\r\t\t\t\t\t\t</node>\r\t\t\t\t\t\t<node TEXT=\"Params\"
CREATED=\"1429767744093\" ID=\"%@\"
MODIFIED=\"1429767744093\">\r\t%@\t\t\t\t\t\t</node>\r\t\t\t\t\t
\t<node TEXT=\"Description\" CREATED=\"1429767744094\" ID=\"%@\"
MODIFIED=\"1429767744094\">\r\t\t\t\t\t\t\t<node TEXT=\"%@\"
CREATED=\"1429767744093\" ID=\"%@\" MODIFIED=\"1429767744093\"
/>\r\t\t\t\t\t\t</node>\r\t\t\t\t\t</node>\r\t\t\t\t</node>\r",p
roductName,product.sc_id,[[NSUUID UUID] UUIDString],[[NSUUID
UUID] UUIDString],product.sc_image_url,[[NSUUID UUID]
UUIDString],[[NSUUID UUID] UUIDString],paramsXML,[[NSUUID UUID]
UUIDString],product.sc_description,[[NSUUID UUID] UUIDString]];
```

Código 24 - Ejemplo de conversión nodo producto metadato

Igualmente, aquí tenemos los parámetros que se explicarán más adelante. En el caso de los productos, estos no tienen subproductos debajo de ellos, por lo que cuando se añade cada producto al final del conjunto ya analizado, se va al siguiente producto y cuando se acaban todos los productos, retornamos un paso atrás y pasamos a la siguiente subcategoría. En el caso de que no haya más subcategorías, damos otro paso atrás y vamos a la siguiente subcategoría de esa categoría y así hasta que no queden más subcategorías y habremos acabado con todos los productos y las categorías.

Al final, el resultado es parecido al obtenido en la página anterior, pero con la información de un producto o productos y no de una categoría. Es más, la información que se añade es más o menos la misma, sin tener en cuenta que aquí se tiene un metadato y no un conjunto de productos.

Recursividad de parámetros.

Por último tendríamos los parámetros de cada producto. Estos parámetros, son los param de cada producto o categoría. El proceso para añadirlos es muy similar al proceso anterior, donde en este caso se tiene que hacer el mismo proceso para añadir los parámetros tanto de las categorías como de los productos.

Como los parámetros los tenemos dentro de un diccionario, para cada uno de los parámetros tendremos que hacer la misma operación. Primero convertir el parámetro en el formato de texto que queremos y después añadirlo al final del documento, para justo después coger el siguiente parámetro. Esto se haría de la siguiente manera:

```
for (NSString *key in [param allKeys]) {

        NSString *paramsXML = [NSString
stringWithFormat:@"\t\t\t\t\t\t<node TEXT=\"%@\"
CREATED=\"1429767744093\" ID=\"%@\"
MODIFIED=\"1429767744093\">\r\t\t\t\t\t\t\t<node TEXT=\"%@\"
CREATED=\"1429767744093\" ID=\"%@\" MODIFIED=\"1429767744093\"
/>\r\t\t\t\t\t\t</node>\r",key,[[NSUUID UUID]
UUIDString],(NSString *)param[key],[[NSUUID UUID] UUIDString]];
        [paramString appendString:paramsXML];
    }
```

Código 25 - Ejemplo de conversión de parámetros de un producto

De esta forma, aplicando esta simple lógica, obtendríamos como ejemplo un código como el siguiente:

```
<node TEXT="Params" CREATED="1429767744093" ID="FD22467F-9A5C-47CE-8A66-
2BCE58B4C877" MODIFIED="1429767744093">
        <node TEXT="resource" CREATED="1429767744093" ID="EE2F8EBA-4593-48CC-
        901E-6FABB258FCF5" MODIFIED="1429767744093">
                <node TEXT="" CREATED="1429767744093" ID="1EE41F91-4AE3-45D5-
                A633-681D678DF115" MODIFIED="1429767744093" />
        </node>
        <node TEXT="type" CREATED="1429767744093" ID="019D39C7-3D97-4696-
        9844-C4F9A8B51879" MODIFIED="1429767744093">
                <node TEXT="html" CREATED="1429767744093" ID="3947E7BE-C734-
                4A16-BD1D-9B208202EF2A" MODIFIED="1429767744093" />
        </node>
        <node TEXT="subtype" CREATED="1429767744093" ID="5A25BC92-511B-410A-
        9704-2CB2AF56BD94" MODIFIED="1429767744093">
                <node TEXT="" CREATED="1429767744093" ID="A4FD9D8C-2888-4EC3-
                ACED-C4A30B756477" MODIFIED="1429767744093" />
        </node>
```

```
<node TEXT="action" CREATED="1429767744093" ID="9BC01E1B-4840-4AE4-A771-B3E8B90780E4" MODIFIED="1429767744093">
    <node TEXT="" CREATED="1429767744093" ID="08AC5C5E-7191-4F6F-97C6-980AA31FA767" MODIFIED="1429767744093" />
</node>
```

Código 26 - Resultado de conversión de parámetros en XML

Una vez que se ha acabado con todos los parámetros, categorías y productos, añadimos el final que es común a todos los Freemind y la conversión de Parse.com a Xmind habría acabado.

Para acabar, solo habría que generar con el texto que tenemos un archivo y almacenar dicho archivo en la URL proporcionada por el usuario para guardar el archivo. El nombre de dicho archivo será el nombre de la categoría principal, añadiendo .mm como extensión para indicar de esta manera que es un archivo de tipo Freemind.

Para visionar el archivo, tenemos que abrirlo con la herramienta que queramos. Si elegimos xmind, tenemos que importar el archivo, seleccionar que queremos importar desde Freemind y darle a continuar. Xmind analizará el archivo y generará el árbol, como el que vimos anteriormente.

7.3. Módulo Xmind a Parse.com.

Este módulo se integró en la aplicación para poder, una vez modificados los datos en la el gestor de mapas, integrar nuevamente los datos en Parse.com. Hay dos decisiones importantes que se toman para integrar este módulo.

- Los datos no han de ser integrados en Parse.com inmediatamente, pero se generaran los archivos necesarios para que Parse los entienda. Esta decisión se toma, dado que como Parse.com anuncia que va a dejar de prestar servicios a principios de 2017, resulta más eficiente crear las mismas tablas en archivos CSV y cuando están listos los servidores del IUMA para esta aplicación, subirlos y actualizar la aplicación. Aunque podría haberse integrado, se prefirió dejar esta para el futuro desarrollo de la aplicación.
- Estos función se puede realizar por partes. El IUMA tiene diseñada una aplicación que convierte un JSON en los datos de Parse.com. Como esta función ya está diseñada, podemos usarla para tener la mitad del camino diseñado. Es

más, sabemos que en los últimos años los archivos XML no están siendo usados y se usan los archivos JSON, por lo que sería lógico trabajar con este tipo de archivos.

Basándonos en estas dos premisas diseñamos la arquitectura de este módulo. Buscando información en diversas web de desarrollo, ejemplos de conversores de XML a JSON, encontramos un programa diseñado por Troy Brand en 2010 que convierte un XML en un diccionario (código incluido en las librerías del proyecto y tomado bajo licencia de libre uso). Una vez examinado el código, se decide que el código funcione como un conversor general y totalmente autónomo, por lo que se decide usarlo para convertir todos los XML en diccionarios, dado que este código usa funciones generales incluidas en foundation.

Por último queda unir las dos partes, la que convierte de XML a diccionario y la que convierte de JSON a Parse.com diseñada por el IUMA. Para unir estas dos partes, se usa una función incluida también en foundation:

```
self.dataParsed = [NSJSONSerialization
dataWithJSONObject:xmlDictionary options:
NSJSONWritingPrettyPrinted error:&error];
```

Código 27 - Función conversión en JSON

Usando esta simple función, tenemos desde un diccionario convertido desde XML un JSON que pueda ser leído fácilmente, y posteriormente usar el conversor diseñado por el IUMA y proporcionado por los tutores para acabar de convertir los archivos. Únicamente haría falta obtener los archivos CSV de las categorías y de los productos y subirlos a la plataforma Parse.com

Con esta parte se acabaría la función completa de convertir primeramente desde Parse.com a xmind para examinar, analizar, modificar, añadir o eliminar información y posteriormente convertirla en información entendible por Parse.com.

Cabe destacar que para convertir de Xmind a Balsamiq o de Balsamiq a Xmind, hay que pasar forzosamente por Parse.com, por lo que estas funciones se reutilizaran en el controlador para obtener el resultado correcto, independientemente de cuál sea el camino que se desee tomar.

7.4. Ejemplo de aplicación

El siguiente árbol representa la aplicación que se obtiene el Parse.com, y que posteriormente se usará para generar los archivos CSV que se pueden encontrar adjuntos con la memoria de este proyecto.

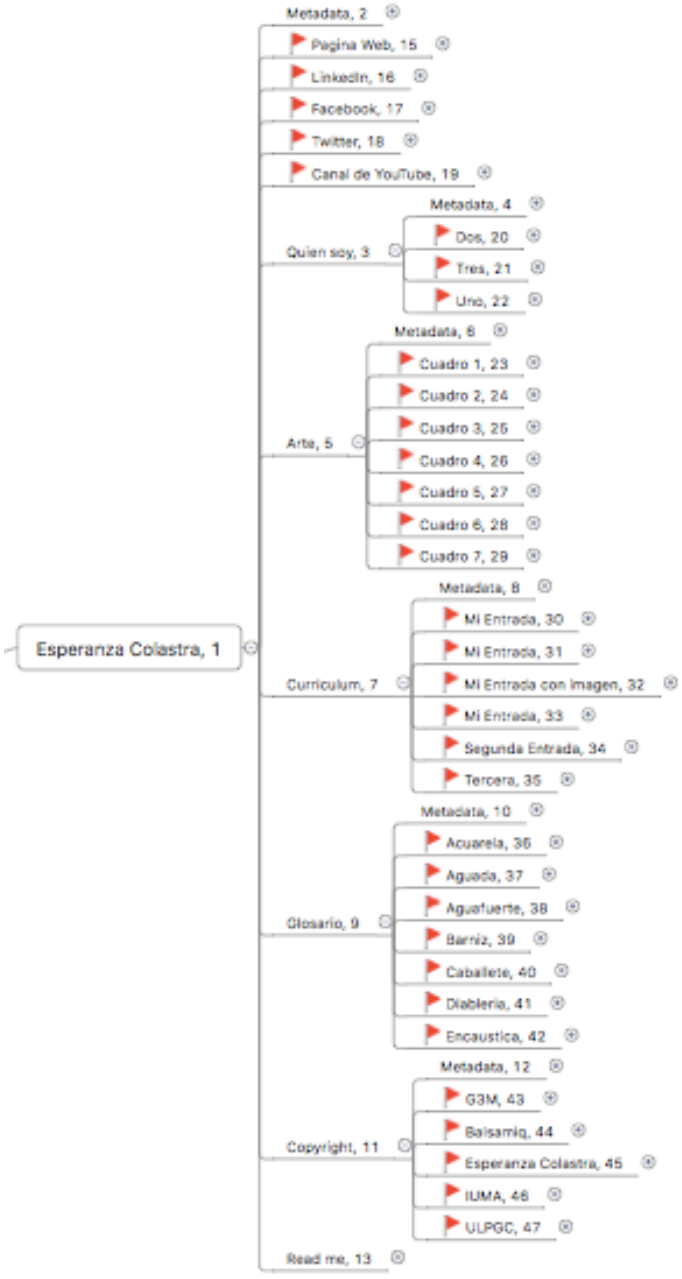


Ilustración 61 - Árbol resultado de aplicación ejemplo

Como los archivos, son demasiado grandes, se decide incluirlos únicamente en el disco del proyecto y no como parte de la memoria para no incluir más páginas que no sean entendibles

8. Estrategia de Negocio

Como hemos visto, la funcionalidad diseñada en esta aplicación es viable y se ha demostrado que se puede llevar a cabo, tanto la integración en los servicios ya existentes, como probablemente en nuevos servicios.

Si nos fijamos en las tendencias actuales del mercado, hemos visto como en el estado de arte, algunas empresas están fijando sus esfuerzos en poder realizar aplicaciones móviles basados en diferentes módulos, pero no con vistas. Otras proveen de herramientas a los desarrolladores de nivel intermedio para que pueden desarrollar las aplicaciones a un menor esfuerzo. En general, todas ellas intentan reducir el tiempo de lanzamiento al mercado de las aplicaciones, así como el tiempo de desarrollo, dado que el tiempo de lanzamiento se puede reducir también con frameworks de desarrollo de software como puede ser Scrum.

Según estudios de mercado, como el llevado a cabo por la firma Gartner [15], la demanda para el diseño y desarrollo de aplicaciones móviles crecerá al menos cinco veces más rápido que la capacidad que tienen las empresas que hay en el mercado para cubrir estas necesidades. Esto presenta un espacio en el mercado donde este tipo de herramientas pueden situarse.

Pero, actualmente hay otras empresas tratando de cubrir ese hueco, y llevar a cabo la misma estrategia y ofrecer los mismos servicios no sería la decisión acertada. Analizando la sección de mercado en la que se puede situar esta idea, observamos que ninguna de las herramientas y empresas que están actualmente en el mercado ofrecen una solución completa, desde el diseño hasta la posibilidad de revisar la aplicación y modificarla usando herramientas más simples.

Además, esta herramienta no solo sirve para usuarios, es decir con una estrategia de negocio B2C[3], sino también para una estrategia de negocio B2B[4], donde las empresas que ya tienen clientes que necesitan del desarrollo de aplicaciones móviles, pueden hacer uso

[3] El acrónimo B2C se refiere a un término anglosajón que quiere decir "Business to Customers", o lo que viene a decir, comercio directo entre empresa y consumidor.

[4] El acrónimo B2B se refiere a un término anglosajón que quiere decir "Business to Business", o lo que viene a decir, comercio entre empresas.

de los servicios que esta proporciona para, de esta forma, proveer al cliente de un prototipo en cuestión de horas y que el cliente pueda validar que es la aplicación que necesita. También estas empresas pueden desarrollar diferentes modelos o plantillas, en los que solo hace falta introducir la información, por ejemplo un modelo para hoteles o para restaurantes.

De esta manera tendríamos dos modelos diferentes de negocios, B2C y B2B, aunque ambos son basados en modelos de negocios tipo SaaS.

8.1. Modelo B2C.

El modelo de negocio para B2C, se basaría en un modelo de subscripciones basado en diferentes segmentos, donde el usuario paga mensualmente o mediante una subscripción a varios meses para el segmento que quiere usar. Estos segmentos se separan en cuatro:

- Muy bajo coste. Para los clientes que buscan una solución muy sencilla y están dispuestos a pagar un coste muy bajo o nulo. Este segmento es necesario sobre todo para atraer a los usuarios a usar los servicios.
- Bajo coste o básico. Estos clientes buscan algo que puedan personalizar, pero no pueden permitirse pagar un alto coste, dado que su aplicación está diseñada normalmente para promover una actividad poco comercial.
- Avanzado. Son usuarios que quieren pagar por tener servicios de desarrollo de aplicaciones. Normalmente tienen conocimientos IT, pero no son capaces de desarrollar aplicaciones del nivel que esta herramienta puede diseñar. Quieren tener el control total y poder hacer cuantas más actualizaciones de las aplicaciones como quieran.
- Premium. Usuarios que quieren tener todas las funcionalidades en su mano y no tener que preocuparse por nada de la aplicación. También, como están pagando unos servicios más caros, esperan recibir ayuda por parte de la empresa para desarrollar su negocio. Son usuarios que se parecen a los usuarios que se esperan en el sector B2B, pero que en realidad no necesitan tanto.

Estas cuatro sectores tendrán que tener diferentes funcionalidades y por tanto diferentes precios para los servicios. La siguiente tabla representa la idea inicial de como dividir las funcionalidades y en que rango de precio se situaría el servicio.

Tabla de funcionalidades y precios				
Descripción	**Ultra bajo coste**	**Básico**	**Avanzado**	**Premium**
Creación de aplicación	√	√	√	√
Creación por vistas	5 vistas	10 vistas	Ilimitado	Ilimitado
Publicación	*	√	√	√
Edición post publicación		4 veces	√	√
Logo propio		*	√	√
Edición del logo post publicación			*	√
Uso de árbol para control			√	√
Edición vía árbol			√	√
Posibilidad de crear plantillas				√
Creación de múltiples aplicaciones			Max 3	Max 7
Publicidad	Si	Si	Si	No
Soporte				√
Precio	Gratuito	3€/mes	10€/mes	25€/mes

Tabla 21 - Matriz de funcionalidades para B2C

Esta sería una tabla orientativa para la distribución de los diferentes servicios. Las funcionalidades que en algún segmento tiene el icono *, indica que este servicio está disponible, pero tiene un coste extra.

8.2. Modelo B2B.

El modelo de negocio para B2B es totalmente distinto, dado que los clientes son empresas que van a distribuir los servicios con otros clientes, por lo que estas empresas tienen que tener margen de beneficio.

El modelo propuesto para estos usuarios es único, y todas las empresas tendrían en un principio los mismos servicios. En este caso, los servicios son únicos y los precios dependerían de la cantidad de usuarios finales que necesitarían usar estos servicios, por tanto dependerá de la capacidad de estas empresas de vender estos servicios que este sistema sea rentable o no.

Por tanto, para aplicar este modelo y atraer al mayor número de proveedores de servicios posibles, el volumen de usuarios que estas empresas traigan serán claves para marcar el precio. También dependerá del tipo de suscripción que quieran para sus usuarios, teniendo en este caso, los tipos Avanzado y Premium, el precio será diferente.

Tabla en función del número de usuarios		
Número de usuarios	**Avanzado**	**Premium**
Hasta 10	10 € / mes	25 € / mes
11 - 25	8,75 € / mes	23 € / mes
26 - 100	7,5 € / mes	20 € / mes
101 - 250	6 € / mes	18 € / mes
251 - 500	5,5 € / mes	16,5 € / mes
Más de 500	5 € / mes	15 € / mes

Tabla 22 - Matriz de precios B2B

Los precios, son precios por usuario, pero cada usuario puede tener tantas aplicaciones como su plan permite, para usuarios Avanzados 3 y para Premium 7.

Estos precios no descartan la posibilidad de poder discutir y acordar precios por suscripciones de largo plazo, como pueden ser anuales o bianuales. Aunque siempre en ese caso se harían por pagos de estas suscripciones por el total y no mensualmente.

8.3. Adaptación de la herramienta

Está claro, que para poder usar esta herramienta para los modelos de negocios mencionados, necesitamos adaptar la herramienta a las necesidades del mercado. Una conclusión que se ha sacado durante el desarrollo del PFC es que las herramientas que se desarrollan en aplicaciones de escritorio o mediante aplicaciones móviles, no son realmente beneficiosas para el negocio B2B, dado que estas necesitarían instalar en sus

usuarios finales la aplicación o hacerla descargar. Para ellos sería mucho más beneficioso usar herramientas basadas en la web.

Con esta lección aprendida, se cambiaría la estrategia y se usaría una herramienta web para que los usuarios pueden hacer exactamente lo mismo que hacen con esta herramienta pero de manera web. Pero este cambio llevaría mucho tiempo de desarrollo. Para acortar el time to market, se aplicarán técnicas de "user story mapping".

Por tanto, para comenzar, se ha decidido que se va a reutilizar los modelos desarrollados, pero que se van a modificar tanto los controladores como las vistas. Del mismo modo, se necesitará añadir ciertos modelos y controladores, para controlar la base de datos de usuarios y para otras funcionalidades, como el diseño de mockup en la web, modificación de árboles en la web, etc.

la siguiente tabla representa el roadmap que se pretendería llevar para el correcto desarrollo de la aplicación. Este roadmap esta priorizado de arriba a abajo y contiene una estimación de complejidad.

Roadmap de Parapp Studio Web	
Funcionalidad	**Complejidad**
Diseño web de la aplicación	10
Adaptación del modelo Balsamiq - Parse.com y viceversa	12
Adaptación del modelo Xmind - Parse.com y viceversa	15
Creación de base de datos interna	10
Creación de gestor de usuarios	5
Creación de gestor de suscripciones	7
Diseño del modelo y controlador de diseño de mockup propio	20
Diseño del modelo y controlador de diseño de árboles propio	35
Diseño web de las vistas de ambos modelos	10
Representación gráfica de estadísticas de uso para los usuarios	15
Gestión de pagos para los usuarios vía web	10

Tabla 23 - Roadmap de futuro diseño

Aunque podrían nombrarse otras muchas funcionalidades, estas son las que son consideradas en este momento. El primer lanzamiento al mercado se podría realizar cuando las primeras 5 funcionalidades estuvieran acabadas.

Teniendo en cuenta la complejidad descrita y la experiencia adquirida durante la realización del PFC se estima que se necesitarían 5 sprints[5], para desarrollar un trabajo de 7 personas / mes.

8.4. Presupuesto para desarrollo

El siguiente apartado define el presupuesto necesario para la realización del software, tanto para los recursos humanos como los materiales y los gastos de comunicación. La siguiente tabla representa todos los detalles.

Presupuesto de Parapp Studio Web	
Descripción	**Precio estimado**
Desarrolladores (2)	9.800 €
Testadores (1)	3.000 €
Alquiler de servidores	300 € / mes
Equipo HW para desarrolladores y testador	4.000 €
Gastos de Marketing y Comunicación	2.000 €
Gastos de creación de empresa	5.000 €
Otros gastos	3.600 €
Total	28.300 €

Tabla 24 - Presupuesto para el desarrollo de Parapp Studio Web

Este presupuesto está basado para lanzar el software al mercado, pero no para mantenerlo posteriormente, dado que dependerá de la respuesta del mercado la continuidad o no.

[5] Sprint es un forma de medir el tiempo en el desarrollo de software dentro del marco de Scrum. 1 sprint equivale normalmente a 2 semanas, pero depende del equipo definir esta medida

9. Conclusiones

Si retomamos los objetivos que tenía este PFC como primordiales para la realización del mismo, podemos ver que con lo explicado en la memoria hemos conseguido alcanzar todos. La siguiente tabla hace un resumen de todos los puntos

Conclusiones	
Objetivo	**Valor añadido obtenido**
Aprendizaje de nuevos lenguajes de programación enfocados en el desarrollo de aplicaciones en entornos Apple, tanto aplicaciones de escritorio como aplicaciones móviles	El proyecto se ha realizado usando el lenguaje de programación de Apple: Objective-C. Aunque actualmente se usa Swift para este tipo de desarrollos, aun Objective-C es una buena forma de programar aplicaciones móviles
Inicio en técnicas de desarrollo como el modelo – vista – presentador u otros patrones de diseño.	Se usaron dos patrones de diseño durante el desarrollo de la arquitectura del proyecto.
Estudio de tendencias actuales empresariales en el desarrollo de aplicaciones y sus procesos internos.	Para llevar la realización del proyecto al día y usar técnicas aprendidas durante las prácticas realizadas y demás, se usó las guías de Scrum y de User Story Mapping para decidir que funciones integrar y cuando integrarlas.
Introducirse en el desarrollo de aplicaciones utilizando Software Kit Developments (SDK) y Frameworks.	Se usaron frameworks y SDK de diferentes compañías para el desarrollo de este proyecto.
Objetivos planteados en el anteproyecto	Todos se han cumplido exitosamente, habiendo extendido incluso los resultados esperados

Tabla 25 - Conclusiones

Dados estos objetivos, se ha conseguido sobre todo poner en práctica durante el desarrollo del proyecto todas las técnicas aprendidas durante el paso de la universidad en el desarrollo de aplicaciones móviles, pero aún más importante y que no se puede plasmar en el proyecto, se usaron técnicas de gestión de proyectos en software usadas en las empresas actualmente para poder desarrollar y validar la idea que se tiene.

Como objetivo primordial de cualquier PFC es validar que la idea que se tiene se puede llevar a cabo y demostrarla. Este PFC tiene una hipótesis al principio, de que es posible analizar Mockup y generarlos por personas de conocimientos bajos en programación y conseguir que estos obtengan una aplicación móvil. Del mismo modo se añaden complementos que pueden ser útiles no solo para este sector, sino para otros sectores.

Por último cabe destacar que este proyecto independiza la aplicación que se desarrolla del modelo de trabajo. Para esta herramienta la arquitectura interna que posea la aplicación destino es indiferente, siempre que esa arquitectura incorpore el mismo modelo de datos.

Es más, como se demuestra, para esta aplicación es totalmente indiferente la plataforma que se utiliza en la aplicación final, por lo que se evita tener que tener diferentes equipos para desarrollar la misma aplicación dos veces, dado que cuando se desarrolla una vez, está disponible en las dos plataformas más usadas en el mercado. De hecho, si se quisiera, se podría desarrollar la aplicación para ser compatible con el IUMA framework en otro sistema operativo, como por ejemplo Windows Phone y se tendrían en un momento acceso a todas las aplicaciones anteriormente desarrolladas.

Con esto, claramente se demuestra que esta aplicación ayuda, tanto a llegar al mercado más rápidamente, reduciendo el time-to-market y a ayudar en la venta de aplicaciones móviles para las empresas que desean incorporar a futuros clientes, pudiéndoles dar la oportunidad de, en el mismo momento, generar un borrador de la aplicación.

10. Resumen en Inglés (English Summary)

This thesis, from now on PFC, tries to solve one unique problem that all current software development companies, especially the ones that are targeted in the mobile app development has: Development time takes too long and the business opportunity can be lost. Also, the PFC has as aim to help people that has low knowledge in the app development to be able to design and release their own mobile app.

As it is explained in this document, there are several companies in the world that already try to solve the problem for the low expertise people, providing them a set of modules that they can use to develop and app. These modules are limited and normally refers to a set of functions. Normally users design the app without seeing the app so they lose the perspective of how is going to be at the end, what makes that process can take too long. Also, these apps, that are in the market, are not useful for the advanced developers, because not only quality, but also possibilities of use are limited and does not provide any information extra to the developer.

In order to solve this problems, the PFC propose to use one tool that combine the use of Mockups and mind maps to provide all useful information to the developers. Using a previous system designed by the IUMA, it is possible to fulfil the database with the information extracted from the mockup or the mind map and obtain using a mobile app with a framework a final app in iOS or Android.

In order to achieve it, was decided to develop one mobile application that contain four different parsers, that parse the information from one format to another and vice versa. This application was developed using the design pattern of MVC. The views and the controllers are standards ones but the model contain all parsers and also all the relations with the database.

This models are split in 3 different ones, one from Balsamiq (where the Mockups are done) to Parse.com (database where the framework extracts the information to build the app) and vice versa. The third one is the Xmind (program used to visualize the mind maps)

to Parse.com with possibility to backwords the work. All these parsers are included in model of the app, what makes it all in one.

It is important to mention, that the structure of the data used, belongs to the previous system designed by the IUMA. It is based in the plugin for Wordpress spider catalogue.

This development made for the PFC concludes that it is possible to provide the tool for clients and for users with no knowledge in one unique framework. All the goals that were pretended to achieve with the project have been achieved, and also new discovers during the process occurred.

Right now, we see that having one unique structure of the data, makes that the development of the apps and the design patterns that are used are totally not depended using this kind of approach. The apps can be designed with any kind of pattern, but they just need to keep the same data structure, what it is actually already independence from the pattern. User just need to fill all the information in the mockup or in the mind map and he will get the app that is looking for, without too much effort.

Also this PFC shows what could be the open market segment where this new tool can be and the estimation of how much time of development is needed in order to achieve a first application that can be tested with real users.

11. Bibliografía

[1] S. F., «España, territorio smartphone,» 30 Enero 2016. [En línea]. Available: http://www.xatakamovil.com/movil-y-sociedad/espana-territorio-smartphone. [Último acceso: 15 Mayo 2016].

[2] Accenture, «5º Estudio de Inversión en Marketing y Publicidad Móvil,» 2012.

[3] Adiante Apps, «Adiante Apps modulos,» Adiante Apps, [En línea]. Available: http://www.adianteapps.com/help/modules. [Último acceso: 2 Junio 2016].

[4] IDC, «IDC - Analyze de Future,» Julio 2015. [En línea]. Available: http://www.idc.com/prodserv/smartphone-os-market-share.jsp. [Último acceso: Julio 2016].

[5] Open Signal, «Open Signal,» Open Signal, Octubre 2014. [En línea]. Available: http://opensignal.com/reports/2014/android-fragmentation/. [Último acceso: Julio 2016].

[6] Contribudores a Wikipedia, «Wikipedia - Modelo-vista-controlador,» Wikipedia, Agosto 2014. [En línea]. Available: https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador. [Último acceso: Julio 2016].

[7] Universidad Complutense de Madrid, «Facultad de Informática - Programación orientada a objetos,» 2008. [En línea]. Available: https://www.fdi.ucm.es/profesor/jpavon/poo/2.14.MVC.pdf. [Último acceso: 2016].

[8] CS193p, Dirección, *Developing iOS 9 Apps with Swift.* [Película]. EEUU: Standford University, 2016.

[9] Oursky, «Make App Icon,» [En línea]. Available: http://makeappicon.com/. [Último acceso: Julio 2016].

[10] J. Velasco, «Dropbox Sync API, integración de Dropbox en aplicaciones móviles,» Febrero 2013. [En línea]. Available: https://hipertextual.com/2013/02/dropbox-sync-api-apps-moviles. [Último acceso: Febrero 2016].

[11] Wikipedia, «Comparison of online backup services,» Agosto 2016. [En línea]. Available: https://en.wikipedia.org/wiki/Comparison_of_online_backup_services. [Último acceso: Agosto 2016].

[12] C. Rebato, « Comparativa de almacenamientos en la nube: cuál elegir y por qué,» 3 Marzo 2015. [En línea]. Available: http://es.gizmodo.com/comparativa-de-almacenamientos-en-la-nube-cual-elegir-1693141994. [Último acceso: Febrero 2016].

[13] Dropbox, «Dropbox - Tutorial for API v1,» [En línea]. Available: https://www.dropbox.com/developers-v1/core/start/ios. [Último acceso: Abril 2015].

[14] Parse.com, «Migración de Parse.com,» Parse.com, Enero 2016. [En línea]. Available: https://parse.com/migration. [Último acceso: Julio 2016].

[15] Gartner, «Gartner Survey Reveals Enterprise Spending on Mobile App Development Remains Low,» Gartner, 21 Junio 2016. [En línea]. Available: http://www.gartner.com/newsroom/id/3353317. [Último acceso: Septiembre 2016].

[16] Apple, «Programming with Objective C,» Apple, 17 Septiembre 2014. [En línea]. Available: https://developer.apple.com/library/content/documentation/Cocoa/Conceptual/ProgrammingWithObjectiveC/Introduction/Introduction.html. [Último acceso: Agosto 2016].

[17] JSON Organization, «JSON Organization,» JSON Organization, [En línea]. Available: http://json.org/example.html. [Último acceso: Agosto 2016].

[18] Wikipedia, «Software as a service,» Wikipedia, 2016. [En línea]. Available: https://en.wikipedia.org/wiki/Software_as_a_service. [Último acceso: Julio 2016].

[19] M. Rouse, «Search Cloud Computing,» Mayo 2016. [En línea]. Available: http://searchcloudcomputing.techtarget.com/definition/Software-as-a-Service. [Último acceso: Junio 2016].

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

BLOQUE SEGUNDO

PLIEGO DE CONDICIONES

1. Lenguajes de programación.

En este apartado se detallan los lenguajes de programación que se usaron para la realización de este proyecto.

1.1. Objective-C

Este es el lenguaje de programación usado para programar aplicaciones móviles en el sistema operativo iOS. [15] Este lenguaje puede ser usado para cualquier propósito y es un lenguaje de programación orientado a objetos. Es un lenguaje que nació a partir del lenguaje C. Usado por Apple para la programación de todos sus dispositivos, tanto Smartphones como ordenadores. Fue diseñado a principio de los '80, por parte de NeXT, sistema del que deriva posteriormente OSX e iOS.

Para programar en este lenguaje es necesario tener la herramienta de desarrollo de Apple, denominada Xcode. Esta herramienta incluye el compilador y el simulador que permite tanto compilar como ejecutar los programas y aplicaciones desarrolladas.

El código de implementación de Objective-C tiene la extensión .m, mientras que sus archivos de cabecera tienen extensión .h, ambos son necesarios para tener una descripción completa de la implementación.

Durante la realización de este proyecto, todos los archivos diseñados por el estudiante, llevan en la cabecera del archivo las letras MB seguido del nombre que se le quiera dar a dicha implementación. Esto se realiza para diferenciar entre los archivos tomados de terceras partes y los realizados para el proyecto.

1.2. Swift

Swift es también un lenguaje de programación diseñado por Apple. Se puede decir, que es la evolución de Objective-C. Está diseñado para ser ejecutado en todos los sistemas diseñados por Apple, desde iOS hasta tvOS, pasando también por OSX y watchOS. Fue diseñado para trabajar con los frameworks de Apple, Cocoa y Cocoa Touch y con la mayor parte de Objective-C.

Fue diseñado en un inicio para resistir mejor el código defectuoso que era posible con Objective-C, es decir más seguro. Otra de las características que definen este lenguaje de programación es que emplea las más nuevas técnicas y teorías de la programación en su concepto, lo que deriva en un lenguaje de programación muy intuitivo y fácil de usar.

Un ejemplo sería como declarar una variable de tipo NSString en Objective-C y como conseguir el mismo resultado pero con Swift. En Objective-C sería

```
NSString *string = @"hello,";
string = [string stringByAppendingString:@" world"];
```

Código 28 - Ejemplo de código en Objective-C

Aunque es bastante intuitivo, con Objective-C, la cosa mejora con Swift:

```
var string = "hello,"
string += " world"
```

Código 29 - Ejemplo de código en Swift

Vemos como la sintaxis es mucho más sencilla. Aunque Swift está disponible desde la versión 6 de Xcode y este proyecto se programó usando la versión 7, se decidió usar Objective-C y no Swift, debido a la familiaridad anterior que se tenía con el lenguaje.

1.3. Otros

Durante la redacción del anteproyecto de este PFC, se describieron otros lenguajes de programación para desarrollar el proyecto. Durante el transcurso del mismo, estos lenguajes fueron descartados, al no acoplarse o no necesitarse para alcanzar los objetivos marcados dentro de este PFC.

Los primeros pasos del proyecto fueron dados con el lenguaje "C" y posteriormente con el lenguaje "C++", pero se observó que se podían conseguir los mismos objetivos usando un lenguaje de programación más moderno y que se adapta mejor al desarrollo de aplicaciones móviles como es Objective-C.

Cabe destacar que se descartó la idea de usar JavaScript dado que no se usó CloudCode de Parse.com y eso supuso el descarte de este lenguaje.

2. Lenguajes de marcado de Texto.

En este apartado se detallan los lenguajes de marcado que se usaron para la realización de este proyecto.

2.1. XML

Extensible Markable Language, es un lenguaje de marcado de texto que fue diseñado para escribir documentos de forma que fuera posible ser leído tanto por humanos como por maquinas. Básicamente, se trata de un lenguaje basado en etiquetas que permiten dar más información que la que un simple texto puede dar.

Existen ciertas reglas que hay que seguir para la correcta redacción de documentos usando este lenguaje.

- Las etiquetas definen a los elementos que se sitúan entre medias. Para abrir una etiqueta se usa el carácter "<" y para cerrarla se usa el carácter ">". Si la etiqueta indica el final de un elemento, incluye el carácter "/" después del carácter "<". Las etiquetas pueden estar constituidas por una o más palabras.
- Los elementos o texto, se definen entre las etiquetas de inicio y final. Puede incluir multitexto o no, pero siempre ha de contener inicio y final.

Existen otros muchos tipos de elementos dentro de XML, pero estos no se usan durante el desarrollo de este PFC.

Un ejemplo de código XML sería el siguiente:

```
<GATO>
  <NOMBRE>Luna</NOMBRE>
  <RAZA>None</RAZA>
  <EDAD>1</EDAD>
  <PROPIETARIO>Mario Bañares</PROPIETARIO>
</Gato>
```

Código 30 -Ejemplo de código XML

Para poder trabajar correctamente con este tipo de lenguaje, se usó la aplicación de escritorio "TextWrangler", que incorpora un parseador de XML, haciendo mucho más sencillo su lectura.

2.2. JSON

JSON es un lenguaje de marcado de texto, o mejor dicho un formato de texto, que se usa para el intercambio de datos. De hecho, JSON y XML tienen muchas similitudes en cómo es su estructura, dado que ambos son jerárquicos. Este lenguaje es mucho más sencillo de usar que los lenguajes XML, donde se incluye mucha información que la mayoría de las veces no es necesaria, pero que se tiene que incluir.

Aunque nació como otro propósito, es ampliamente usado para sustituir el lenguaje XML. Los archivos que son marcados como de tipo JSON, tienen siempre la extensión .json, algo que no ocurre con XML, que aunque es ampliamente usado la extensión .xml, no siempre los archivos que incluyen texto con este formato incluyen la extensión .xml.

Los datos que pueden ser representados usando JSON son muchos, aquí algunos ejemplos.

- Números. Tanto positivos como negativos.
- Strings. Cadenas de caracteres.
- Valores de verdadero o falso.
- Vectores. De cualquiera de los tipos anteriores, sin límite máximo.
- Objetos. Son colecciones de pares, como puede ser un diccionario, tipo nombre : valor.

Un ejemplo de código en JSON sería el siguiente [16] .

```
{
   "glossary": {
      "title": "example glossary",
                  "GlossDiv": {
         "title": "S",
                           "GlossList": {
            "GlossEntry": {
               "ID": "SGML",
                                          "SortAs": "SGML",
                                          "GlossTerm": "Standard Generalized Markup
Language",
```

```
                                    "Acronym": "SGML",
                                    "Abbrev": "ISO 8879:1986",
                                    "GlossDef": {
                "para": "A meta-markup language, used to create markup languages such as DocBook.",
                                        "GlossSeeAlso": ["GML", "XML"]
                },
                                    "GlossSee": "markup"
            }
        }
    }
  }
}
```

Código 31 - Ejemplo de código JSON

3. Programas utilizados.

Los siguientes programas han sido utilizados tanto para desarrollar la presente memoria como para desarrollar el sistema diseñado.

3.1. Xcode

Xcode se trata del entorno de desarrollo de Apple para el desarrollo de aplicaciones para sus dispositivos. Viene de serie en todos los ordenadores fabricados por esta marca y en caso de que no esté dentro de las aplicaciones, puede ser descargado de forma gratuita desde la tienda online de la compañía.

La versión que se usó fue la 7.1, aunque inicialmente se empezó usando la versión 6.

Ilustración 62 - Versión de Xcode

El propio programa incluye el simulador para los diferentes propósitos, tanto los smartphones de la compañía (iPhone) como las tabletas (iPad). Para este proyecto nos hemos centrado en el iPhone 5 y posteriores.

3.2. Balsamiq Mockups

El uso de esta herramienta es ampliamente explicado durante el transcurso de la memoria. A parte de ese uso, esta herramienta se ha usado para diseñar todas las imágenes y esquemas usadas en esta memoria. Para ello se usó la versión de escritorio de este programa.

Muchas de las imágenes que contiene esta memoria son extraídas de diferentes fuentes, y se han vuelto a diseñar para que todas las imágenes compartan una misma simetría, algo que se considera muy importante para facilitar la lectura del proyecto. Del mismo modo, todas las capturas de pantalla que incluyen un cuadro para remarcar alguna sección en concreto también fueron diseñadas con esta aplicación.

Se usó la versión de escritorio por facilitar exportar e importar y porque con la versión de prueba de 1 mes, fue suficiente para desarrollar todas las imágenes del proyecto.

3.3. Parse.com

La versión web de Parse.com se ha usado durante todo el proyecto como se ha explicado anteriormente. La cuenta que se usó fue facilitada por los tutores del proyecto. Su uso se limitó al uso dado en el apartado técnico del PFC.

Para poder usar este proyecto, hace falta tener una cuenta en Parse.com o en el servidor propio del IUMA donde se están migrando las aplicaciones. Esta migración está sucediendo debido a que Parse.com va a dejar de ofrecer sus servicios y se necesita una restructuración y una migración para poder seguir usando los servicios.

En términos de trabajos necesarios, los cambios necesarios en este PFC son mínimos para que el sistema continúe funcionando.

3.4. Xmind

Al igual que los dos anteriores, este programa se ha usado como herramienta del propio PFC y no como herramienta complementaria. Xmind es un programa que permite la generación de mapas jerárquicos, dando al usuario la posibilidad de organizar ideas, organizaciones, planes, etc., de una forma visual.

Se eligió esta herramienta y no otras debido a que la versión gratuita de no pago, es muy parecida a la de pago, y la interfaz gráfica del programa está muy bien diseñada y pensada para hacer sencillo su manejo por parte de los usuarios.

Existen muchas versiones de Xmind. Durante este proyecto se usó la versión 7.1

3.5. TextWrangler

Este programa es un simple editor de texto, parecido a los editores de texto que traen por defecto los sistemas operativos, pero con una gran ventaja, y es que es capaz de abrir y entender diferentes archivos de texto que contienen código en diferentes lenguajes de programación o de texto enriquecido, como pueden ser HTML, XML o JSON.

De esta forma, y pudiendo tener bien organizado, sobre todo con diferentes colores para diferentes secciones o elementos, la organización y analizado de los mockup fue mucho más rápida y sencilla. Del mismo modo, se usó el programa para analizar los archivos XML generados por Freemind y los JSON generados por el generador del IUMA.

Se trata de una herramienta de software libre, por lo que lo hace aún más atractivo para seguir usándose en el futuro. La versión utilizada fue la 5.0.

3.6. Microsoft Word

Microsoft Word es una de las herramientas más populares para escribir documentos. Su uso se limitó a la memoria del proyecto final de carrera.

4. Estrategias y frameworks de desarrollo empleados

Uno de los principales problemas que se ha tenido durante la realización de este PFC ha sido que todo el trabajo se ha realizado a distancia, dado que el proyectando se encuentra trabajando en el extranjero durante toda la duración del proyecto.

Debido a este hándicap, y para poder llevar un control y un buen paso para la realización del proyecto se decidieron aplicar técnicas de desarrollo de software, así como diferentes frameworks que pudieran hacer más sencillo el control del proyecto.

Actualmente, el framework más usado para el desarrollo de software se denomina Agile. Este describe una serie de principios para el desarrollo de software, basados en la colaboración en equipo y equipos de desarrollo con capacidades cruzadas, es decir que el equipo se compone de miembros que pueden realizar todo el trabajo. Aunque Agile indica que no se puede hacer Agile con un solo miembro, se decidió tomar las buenas prácticas que tiene para poder llevar un control y validar algunas de las ideas que se tienen como referencia en este framework.

Agile posee multitud de extensiones o de diferentes maneras de aplicarlos. Debido a la experiencia laboral, se decidió aplicar la misma que se aplica en el puesto de trabajo, Scrum. Scrum está pensado para desarrollar Software y poder validar la idea lo más rápido posible, de la misma manera que poder desarrollar y proporcionar funcionalidades a los usuarios de la manera más rápida posible.

Para ello divide los periodos de trabajo en periodos de no menos de dos semanas y no más de un mes. Incorpora diferentes reuniones, que no se han llevado a cabo en el PFC, pero que permiten mejorar la comunicación del equipo. Existen varios términos que se necesitan entender para poder comprender como se aplicó Scrum para el desarrollo de este proyecto.

- Stories. Son el corazón de Scrum. Se trata de elementos individuales, minimizados al máximo para desarrollar una funcionalidad o parte de ella. Esta funcionalidad debe ser suficientemente pequeña para caber en el desarrollo de unos cuantos días. Un story puede ser por ejemplo enviar un mail, escribir un mail.

- Epic. Son el conjunto de stories que conforman una funcionalidad. Si tomamos el ejemplo anterior, un epic sería la funcionalidad de envío de emails, que se componen de escribirlo y de poder enviarlo.

Por tanto, atendiendo a esta definición, el proyecto se dividió en 4 grandes epics, que serían los de los 3 módulos y la aplicación. Estos a su vez se dividieron en pequeñas funcionalidades de duración no más de 16 horas de trabajo. Esto se debe a que el proyectando no podía dedicar más de 16 horas semanales al proyecto.

Una vez divididas todas las funcionalidades, al principio de cada semana se decidía que trabajos había que coger y que trabajos se dejaban para la siguiente semana, así como analizar durante unas horas si los trabajos de las semanas anteriores estaban acabados o no. Siempre se cogían trabajos de diferentes epics, de forma que la aplicación y los módulos crecían de manera constante.

La ventaja más grande de poder llevar a cabo esto es que, como se tenían estimaciones aproximadas de cuando iban a durar los trabajos, algunas de las funcionalidades que se querían desarrollar se dejaron fuera del alcance de este proyecto, debido a que se podía convertir en un proyecto demasiado largo. Una de las premisas que tiene scrum, es que lo primero es poder entregar a los clientes y usuarios la aplicación que podría ir creciendo en el futuro y que los usuarios pudieran notar las mejoras e ir valorando el crecimiento de la aplicación.

5. Manual de Usuario

La siguiente guía representa el manual de usuario para el correcto manejo del sistema. Para poder usar el sistema, el usuario debe tener creada una cuenta en varias aplicaciones y tener descargados los programas adecuados. Estos son los siguientes:

- Parse.com. Deberá tener una cuenta de usuario configurada como se indica en este manual.
- Balsamiq.com. Deberá tener una cuenta para poder editar los mockup necesarios.

- Xmind. Deberá tener instalado el programa en el ordenador que desee usar para poder editar los árboles.
- Dropbox. Se deberá tener una cuenta para poder sincronizar los archivos con esta base de datos.

De la misma manera, para poder usar la herramienta, se necesita disponer de:

- Un Smartphone con sistema operativo iOS 8.0 o posterior y con la herramienta instalada.
- Un ordenador con cualquier sistema operativo para poder diseñar la aplicación.

Una vez se tienen todas las herramientas se necesita configurar correctamente cada una de ellas.

5.1. Instalación y preparación de Parse.com

Una vez que se tiene creada la cuenta en Parse hay que seguir los siguientes pasos:

a. Abrir el dashboard y crear una nueva aplicación. Configurar los parámetros requeridos durante la creación de la nueva aplicación.

b. Crear dos clases con los siguientes nombres: wp_spidercatalog_product_categories y wp_spidercatalog_products.

Una vez creadas, hay que apuntar las claves necesarias para poder sincronizar la herramienta con la aplicación, así como la futura aplicación que se creará con Parapp Studio. Para sincronizar Parapp Studio con Parse se necesitan las siguientes claves: Parse ID y REST API key. Estas claves han de ser introducidas en la pantalla de configuración de la aplicación.

5.2. Uso de Balsamiq

Para usar Balsamiq tenemos que empezar por crear un nuevo proyecto y proporcionarle un nombre que deberá ser igual al nombre que queremos usar para la aplicación. Una vez que tenemos creado nuestro proyecto, tenemos que ir a edición para poder empezar a diseñar nuestra aplicación.

Una vez en sección de edición, podemos empezar por importar las vistas básicas que queremos usar desde la base de datos que se proporciona. Para importar correctamente, tenemos que abrir el código XML que queremos importar y copiarlo a nuestro portapapeles. Una vez copiado, vamos a Balsamiq y presionamos en el botón de importar XML en el menú principal sobre la opción Proyecto. Pegamos en la nueva ventana que nos aparece y le damos a importar. Automáticamente nos aparece en el editor el mockup que hemos seleccionado y podremos editarlo.

Una vez editado, podemos guardarlo y exportarlo. Presionamos el botón de exportar XML y automáticamente tendremos copiado el código. Abrimos un editor de texto y pegamos. Para guardar el código, tenemos que guardar el archivo con extensión XML y con la estructura explicada en el apartado 5.5 de este documento.

Es importante remarcar que cada pantalla debe estar en un nuevo mockup, para que la herramienta funcione correctamente.

De la misma manera, la sección 5 explica y define como deben ser editados los Mockup para poder construir la aplicación de la manera correcta. Una vez que todos los Mockup están diseñados, tenemos que introducirlos en la carpeta de Dropbox que queremos usar para sincronizar los archivos con Parapp Studio.

Una vez en Parapp Studio, introducimos la carpeta origen y la capeta destino en las opciones, y presionamos sincronizar con Dropbox. Posteriormente nos dirigimos al icono de Balsamiq y presionamos la opción destino que queremos.

5.3. Uso de Xmind

Xmind es una aplicación de escritorio y normalmente es usada para examinar cómo se han analizado los mockup o para analizar otras aplicaciones y actualizarlas. Por tanto para ello, debemos usar la aplicación de Parapp Studio para generar el archivo con los datos desde Parse y usarlos en Xmind.

Una vez tenemos el archivo, que tenemos que descargar desde la carpeta de Dropbox que indicamos en Parapp Studio, abrimos el programa de Xmind y nos dirigimos a Archivo -> Importar. Seleccionamos que queremos importar desde un archivo Freemind y seleccionamos el archivo en el explorador de archivos que nos aparece y le damos a

ejecutar. Dependiendo de la capacidad del ordenador y del tamaño de la aplicación, esta operación puede tardar varios minutos.

Una vez tenemos el árbol podemos expandirlo o contraerlo usando las funcionalidades de Xmind. Del mismo modo podemos mover nodos, añadir o eliminar e incluso modificar el contenido.

Una vez acabado el análisis, podemos exportar el trabajo a un archivo Freemind y subir este archivo a Dropbox. Indicamos en Parapp Studio donde hemos introducido este archivo y además donde queremos recoger nuestros archivos .CSV. Una vez tenemos toda la información, podemos ir a Parapp Studio y sincronizar con Parse.com, obtendremos los archivos CSV que podemos subir manualmente a Parse.com

5.4. Uso de Parapp Studio

Parapp Studio debe estar instalada en un Smartphone de Apple con una versión de iOS 8.0 o superior. Una vez instalada podemos configurar la aplicación lo primero de todo y una vez configurada, podemos usar las funcionalidades explicadas durante el desarrollo de esta memoria, para conseguir cualquiera de los resultados que necesitamos.

Es importante recordar que el usuario debe configurar la aplicación correctamente para su correcto funcionamiento, dirigiéndose a la pantalla de sincronización.

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

BLOQUE TERCERO
PRESUPUESTO

1. Presupuesto

El siguiente capítulo presenta el presupuesto estimado para la realización de este proyecto. Este presupuesto incluye el estudio inicial de la solución, el tiempo de desarrollo empleado, herramientas utilizadas y redacción de la memoria.

1.1. Recursos Humanos

Según el Colegio Oficial de Ingenieros de Telecomunicación, en adelante COIT, se han eliminado los baremos orientativos por el Ministerio de Economía y Hacienda, siguiendo directivas europeas. Debido a esto, los honorarios de los ingenieros por mano de obra son libres.

Para tomar de forma orientativa los precios, se toma como referencia el precio de venta de desarrolladores usados por la empresa en la que trabaja el proyectando, que son de 50€ por hora durante días laborales, mientras que el precio por hora durante jornadas no laborales, véase festivos o fines de semana es de 65€. De la misma manera, y siguiendo la misma referencia que anteriormente, se aplica un coeficiente dependiendo del número de horas totales usadas. Este coeficiente se detalla en la siguiente tabla:

Coeficiente de Ajuste	
Total de Horas trabajadas	**Coeficiente**
Hasta 100 h	1
100 h - 200 h	0.9
200 h - 400 h	0.8
400 h - 800 h	0.7
800 - 1600 h	0.6
Mas de 1600 h	0.5

Tabla 26 - Coeficiente de Ajuste del presupuesto

La duración del proyecto ha sido de 2 años, dedicando una media de 8 horas en horario laboral a la semana y 8 horas en horario extra laboral a la semana. Por ello, 832 horas laborales y 832 horas extra laborales. Aplicando la fórmula explicada anteriormente, obtenemos que:

$$\boldsymbol{T = ((832 * 50) + (832 * 65)) * 0.5 = 47.840\ €}$$

Por tanto, los honorarios por mano de obra ascienden a un total de **cuarenta y siete mil ochocientos cuarenta euros**.

1.2. Recursos Materiales

En los recursos materiales se tiene en cuenta la amortización de inmovilizados que se usan durante la realización del proyecto, ya sean elementos o materiales. Cada uno de estos elementos tiene un periodo de vida útil que normalmente es superior al del desarrollo del proyecto. Para calcular esta amortización se va a usar la siguiente fórmula:

$$A = \frac{(P - V_R)}{U}$$

Donde:

- A indica la amortización anual.
- P indica el precio que se pagó por su adquisición.
- V_R es el valor residual que se estima tendrá la herramienta al final de su vida útil. Tomaremos como referencia un 10% de P
- U es la vida útil de este elemento tomado en años.

Todos los recursos usados en este proyecto han sido materiales, dado que los elementos de software que se han usado han sido de software libre, versiones gratuitas de programas de pago o con licencia para estudiantes.

Los elementos hardware usados durante el desarrollo de este PFC han sido, dos ordenadores Macbook, uno Macbook Pro y otro Macbook Air y dos iPhone, uno iPhone 4S y otro iPhone 5S.

Para los elementos materiales o de hardware, se toma como periodo de amortización los 5 años.

La siguiente tabla los valores usados para el cálculo de todas las variables necesarias para calcular la amortización. Se ha calculado primero la amortización de cada elemento y posteriormente se han sumado todas las amortizaciones para calcular la amortización total.

Amortización						
Descripción	P	V_R	U	A	TA	Total
Macbook Pro 2013	1100 €	110 €	5	198 €	2	396 €
Macbook Air 2015	999 €	99.9 €	5	179.82 €	2	359.64 €
iPhone 4S 2012	300 €	30 €	5	54 €	2	108 €
iPhone 5S 2016	550 €	55 €	5	99 €	2	198 €

Tabla 27 - Amortización

Por tanto, el coste total de equipos materiales es de 1.061,64 €, es decir, **mil sesenta y un euro con sesenta y cuatro céntimos**.

1.3. Presupuesto total

Para el cálculo del presupuesto total, hemos de tener en cuenta los recursos materiales y los recursos humanos y aplicar las correspondientes tasas e impuestos de la Comunidad Autónoma de Canaria, conocidos como I.G.I.C y que actualmente asciende al 7% del total.

La siguiente tabla representa los cálculos y el precio total:

Presupuesto total	
Recursos Humanos	47.840 €
Recursos Materiales	1.061,64 €
Total antes de impuestos	**48.901,64 €**
I.G.I.C. (7%)	3.423,11 €
TOTAL	**52.324,75 €**

Tabla 28 - Presupuesto Total

Por tanto, el presupuesto total de este proyecto final de carrera es de **cincuenta y dos mil trescientos veinticuatro euros y setenta y cinco céntimos.**

UNIVERSIDAD DE LAS PALMAS
DE GRAN CANARIA

BLOQUE CUARTO

ANEXOS

Índice de Ilustraciones

Índice de Tablas

Índice de Código