laSalle

UNIVERSITAT RAMON LLULL

Escola Tècnica Superior d'Enginyeria La Salle

Treball Final de Màster

Màster Universitari en Enginyeria de Telecomunicació

iKlubbers: diseño e implementación de una aplicación iPhone (Vol. 1)

Alumne
Toni Gómez Maldonado

Professor Ponent
Emiliano Labrador Ruiz de la Hermosa
Oscar García Pañella

ACTA DE L'EXAMEN DEL TREBALL FI DE CARRERA

Reunit el Tribunal qualificador en el dia de la data, l'alumne

D. Toni Gómez

va exposar el seu Treball de Fi de Carrera, el qual va tractar sobre el tema següent:

iKlubbers: diseño e implementación de una aplicación iPhone (Vol. 1)

Acabada l'exposició i contestades per part de l'alumne les objeccions formulades pels Srs. membres del tribunal, aquest valorà l'esmentat Treball amb la qualificació de

Barcelona,

VOCAL DEL TRIBUNAL VOCAL DEL TRIBUNAL

PRESIDENT DEL TRIBUNAL

"La gente piensa que enfocarse significa decir sí a aquello en lo que te enfocas, pero no es así. Significa decir **no** a otras cientos de ideas buenas que hay."

Steve Jobs

ABSTRACT 

Este trabajo consiste en la creación de una aplicación para el famoso dispositivo móvil de *Apple*, *iPhone*. Además del reto que supone programar para esta tecnología (tiene poco más de 3 años de vida), el proyecto abarca mucho más, debemos hacer una campaña de *marketing,* apoyada por videos promocionales, web, póster, etc. También nos hemos visto afectados por el hecho de cambiar la metodología de trabajo que se llevaba realizando hasta ahora en ejercicios de este tipo en la universidad, ya que ***iKlubbers*** ha visto la luz gracias a 5 personas, y no una como venía siendo habitual.

A continuación puede leer el resultado de 5 meses de trabajo por parte de mi equipo, centrándonos en mi parte del proyecto, para llegar al objetivo de crear una aplicación para *iPhone* profesional y competitiva dentro del mercado actual.

ABSTRACT 

Aquest treball consisteix en la creació d'una aplicación per el més que conegut dispositiu móvil d'*Apple*, *iPhone*. A més del repte que suposa programar per aquesta tecnología (te poc més de 3 anys de vida), el projecte arriba més enllà, hem de fer una campanya de *marketing*, recolçada per videos promocionals, web, póster, etc. També ens hem vist afectats pel fet de cambiar la metodología de treball que es portava realitzant fins ara en exercicis d'aquest tipus a la universitat, ja que ***iKlubbers*** ha vist la llum gracies a 5 persones, i no d'una sola com era habitual.

A continuación podrá llegir el resultat de 5 de treball per part de l'equip, fixant-nos en el meu treball, per arribar a l'objectiu de crear una aplicación per *iPhone* profesional i competitiva dins el mercat actual.

ABSTRACT 

This work consists in the creation of an application for the famous *Apple* mobile device, *iPhone*. Besides the challenge of programming for this technology (a little over 3 years of life), the project encompasses much more, we should do a marketing campaign, supported by promotional videos, web, posters, etc. We have also been affected by the act of changing the methodology of work that had made so far in exercises of this kind at the university, as ***iKlubbers*** has been created thanks to 5 people, and not one as had been being usual.

Here is the result of five months of the team work, focusing on my work, to reach the goal of creating a professional and competitive iPhone.

RESUMEN

Este proyecto es el punto final al Máster universitario en creación, diseño e ingeniería Multimedia. Este trabajo final de máster (*TFM* a partir de ahora) se distingue en varios puntos de los trabajos finales que se venían entregando hasta ahora.

El punto más destacable, es que ahora dicho proyecto no se realiza de manera individual, sino que la responsabilidad de su elaboración reside en un grupo de personas (5 en nuestro caso) las cuales disponen de diferentes roles dentro del grupo (2 programadores, 1 diseñador, 1 Ingeniero de Software y 1 jefe de proyecto. En mi caso, ejerzo el rol de *Programador*).

Otro de los cambios más significativos del *TFM* es el hecho de desarrollar nuestro trabajo para una empresa real, que lo venderá y sacará beneficios de él, es decir, como grupo tenemos un cliente. Así pues, se puede decir que entre todos los compañeros formamos una *microempresa*.

Estas 2 modificaciones (las más significativas) nos abastecen de una experiencia *pre-laboral* muy similar al ámbito que nos encontraremos al salir de la universidad:

- Tenemos fechas de entrega, tanto con el cliente como con los tutores.
- Debemos organizar nuestro trabajo y aprender a desarrollar en paralelo.
- Debemos ser conscientes de que nuestro trabajo repercute en el de los demás.
- Tenemos responsabilidades, no sólo con nuestra idea académica, sino con el cliente, por lo que debemos aprender a negociar con élt matices sobre nuestro trabajo.

La idea de ***iKlubbers*** vino precisamente desde nuestro cliente *Mobivery*, el cual nos propuso realizar una aplicación de temática nocturna. El concepto surgió de un contacto de esta empresa, el cual tiene numerosos clientes relacionados con el ocio nocturno (locales, discotecas, bares, etc.) y deseaba asociarlos de alguna manera a *iPhone*.

De este modo nació ***iKlubbers***, *software* instalable en cualquier dispositivo *iPhone* que permite al usuario saber que clubs nocturnos están cerca de él en un determinado momento, y también le da la posibilidad de apuntarse a sus listas VIP con la finalidad de entrar gratis en ellos, entre otras muchas cosas.

En las siguientes páginas veremos todas sus funcionalidades más profundamente, como han sido pensadas y como finalmente se han implementado.

Hace falta decir, que dentro del grupo tuvimos muchísimas ideas relacionadas con la propuesta de *Mobivery*, pero debido al tiempo no hemos podido desarrollarlas todas y hemos decidido dejar algunas para un versión posterior, ***iklubbers 2.0***.

AGRADECIMIENTOS

Agradecer por su ponencia y colaboración a Oscar García Pañella y Emiliano Labrador Ruiz de la Hermosa, quienes nos han guiado y enseñado durante la elaboración de todo este trabajo.

Agradecer también a todos los empleados de *Mobivery* por su aportación a problemas puntuales tanto de implementación como de planteamiento.

iKlubbers: Diseño e implementación de una aplicación iPhone - Parte iPhone Vol.I

ÍNDICE

Capítulo 1:

Introducción

1.1 Marco

La carrera por desarrollar el mejor móvil empezó hace mucho tiempo, los principales avances eran de capacidad y forma. La principal innovación era poder tener música y mandar mensajes con imágenes y sonidos reales.

Pero en Enero de 2007 Apple anunció algo que llamaba simplemente *iPhone*. Cuando lo mostró por primera vez, nos dejó a todos impresionados, primero por su espectacular diseño, pero lo que de verdad desconcertó fue el hecho de que carecía de teclado físico.

Figura 1.1.1: iPhone

Un año después surge la *AppStore,* un modo de negocio que fue un rotundo éxito, los programadores mandan su aplicación a la tienda y los usuarios la pueden comprar fácilmente a precios muy bajos. Cabe señalar que la tienda de aplicaciones fue una idea de la comunidad hacker y después *Apple* adoptó el concepto.

Por todos estos cambios introducidos en el sector móvil, hoy en día el dispositivo *multitouch* es el líder indiscutible tanto en tecnología como en usabilidad y es tomado como una referencia por todas las compañías de telefonía móvil.

Cuando algo tiene mucho éxito, la competencia siempre lo copia, después del *iPhone,* ahora la mayoría de móviles cuentan con pantalla táctil, intentan crear un sistema operativo tan bueno como el de *Apple* y están sacando sus propias tiendas de aplicaciones.

Gracias al *iPhone* el mercado de móviles dio un paso de gigante.

1.2 Estado del arte

En la actualidad hay muchas y muy variadas aplicaciones de localización de lugares, tanto de locales de ocio nocturno, como de restaurantes, cines, etc. Pero *iKlubbers* es especial, goza de algo que no tiene ninguna de ellas, el contacto directo con dichos locales y la posibilidad de negociar con ellos.

Gracias a esto, nuestra aplicación ofrece al usuario la posibilidad de salir en una noche de ocio sin tener que pagar por la entrada del local, simplemente usando el sistema desarrollado por nuestro equipo. Aún y teniendo algo que nos diferencia del resto, debemos evaluar la competencia y saber cuáles son sus puntos fuertes y débiles, para poder tomar ventaja de ello y mejorar nuestro software.

Estas son algunas de las más importantes:

Lonely Planet Barcelona City Guide

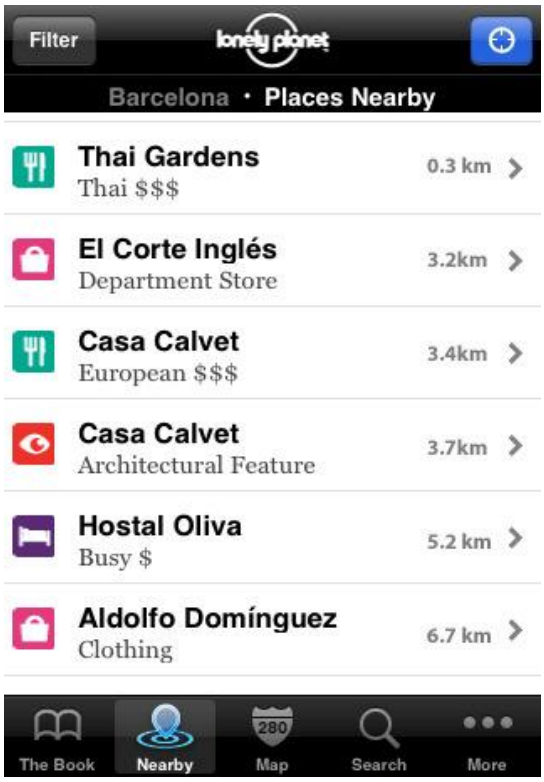


Figura 1.2.1: Lonely Planet

- Geolocalización
- Variedad de localizaciones
- Adaptación para iPhone
- También es una guía

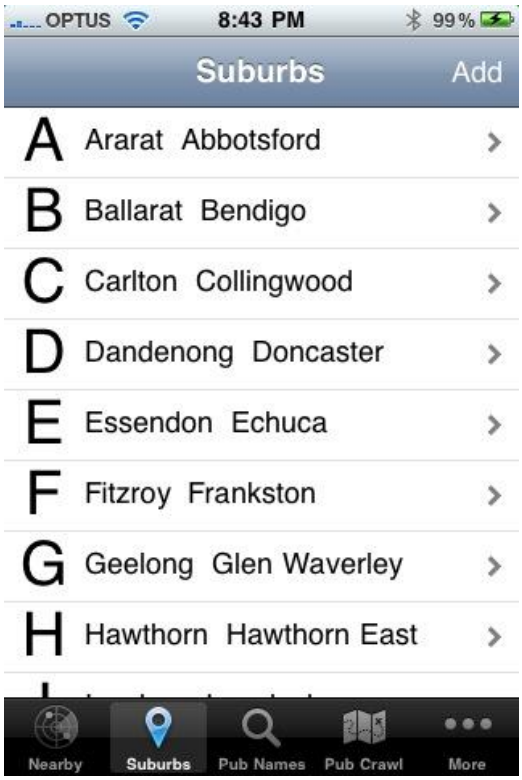
iKlubbers: Diseño e implementación de una aplicación iPhone - Parte iPhone Vol.I

Bliquo

- Geolocalización
- Variedad de localizaciones
- Añadir localizaciones

Figura 1.2.2: Bliquo

Pub Crawl

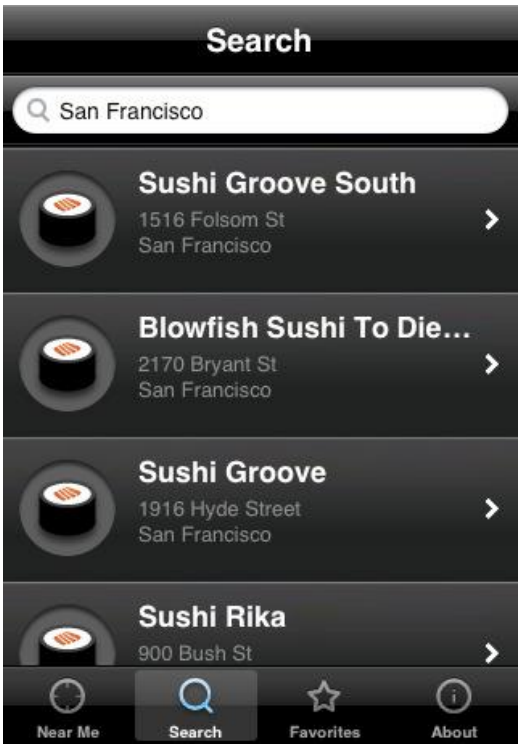


- Geolocalización

Figura 1.2.3: Pub Crawl

iKlubbers: Diseño e implementación de una aplicación iPhone - Parte iPhone Vol.I

iSushi



- Geolocalización
- Sólo restaurantes japoneses

Figura 1.2.4: iSushi

Beeloop

- Geolocalización
- Guía de viaje

Figura 1.2.5: Beeloop

Pasando muy por encima de las funcionalidades de cada aplicación (las más significativas del mercado actual) vemos que todas ofrecen en gran medida las mismas opciones.

La *geolocalización* es básica en este tipo de producto, es decir, el *iPhone* debe ser capaz de situarnos en un mapa y decirnos que opciones tenemos a nuestro alrededor, este recurso es el que siempre se da por hecho en cualquier aplicación de esta índole.

La más famosa entre todas es **bliquo**, que está instalada en la mayoría de dispositivos móviles de *Apple,* debido a su gran popularidad entre la comunidad y a su carácter gratuito. Esta aplicación nos permite encontrar locales a nuestro alrededor (tanto discotecas, como restaurantes, como cafeterías, etc.) y además nos aporta el hecho de poder realizar reservas en los mismos a partir de *iPhone*. Este es un punto que se acerca mucho al nivel de exclusividad que intentamos ofrecer con *iKlubbers*, pero que a su vez tiene una gran diferencia. Mientras que con nuestra aplicación podemos entrar gratis en los pubs de ocio nocturno, con esta solo nos podemos reservar un sitio en restaurantes y demás locales, pero tendremos que pagar igualmente.

Este proyecto es nuestro máximo competidor, ya que los demás, ofrecen cosas no muy similares a las nuestras y con poca flexibilidad. Por ejemplo **iSushi** solamente nos permitirá encontrar restaurantes de etnia japonesa. Aunque su estructura es muy similar a la nuestra, el objetivo de cliente al que va dirigido no es el mismo que el nuestro, aunque pueda coincidir en algunos sectores de nuestro público, la finalidad de uso de ambas aplicaciones es diferente.

Beeloop y **Lonely Planet** nos ofrecen, aparte de localizarnos en un mapa y mostrarnos los lugares de nuestro interés que están alrededor nuestro, la posibilidad de consultar guías sobre los mismos. Este hecho puede

afectarnos en cierta manera, ya que *iKlubbers* tiene muy en cuenta el sector turístico, puesto que nos damos cuenta que mucha de la gente que use nuestro producto lo hará porque es nuevo en la ciudad y no sabe a dónde ir. Aún y sabiendo esto, entendemos que el ocio, aunque sea de turistas no choca directamente con el turismo habitual de las diferentes ciudades que soportará nuestro proyecto. Por lo tanto no significarían una competencia tan directa.

Por último, y con un nombre bastante aclarador, **Pub Crawl** es una aplicación muy similar a la nuestra y que por lo tanto se entromete de lleno en nuestro sector, pero como hemos comentado antes, disponemos del arma de la 'gratuidad' que ellos no gozan, esto nos da una ventaja enorme.

1.3 Descripción del problema

El principal objetivo planteado, fue el de realizar una aplicación usable y clara, que a la vez que nos aportaba valores académicos, pudiese ser competitiva a nivel profesional.

La aplicación debía ser soportada por *iPhone*, esto era requisito indispensable, no podía ser para otro dispositivo, debido al carácter de nuestro cliente, relacionado estrechamente con esta tecnología.

Debía resolver el problema que tiene mucha gente cuando se reúnen con amigos y les surge la pregunta *¿A dónde vamos hoy?* Así que de manera clara teníamos que solucionar esta cuestión de un modo muy sencillo para el usuario. Además, debíamos dotar a la aplicación de cierto aire de exclusividad en comparación con otras herramientas que ofrecen servicios similares.

Además, teníamos que buscar una manera de mostrar distintos locales de forma preferente, donde los clubs que pagasen una cantidad extra saldrían promocionados de manera especial. Las demás opciones de *iKlubbers* fueron discutidas por los miembros del equipo para dotar a nuestro proyecto de una entidad considerable y hacerlo potente y competitivo.

1.4 Solución propuesta

Con el fin de solucionar el problema principal, se optó por 5 funcionalidades básicas que harían de la aplicación una herramienta fundamental en los dispositivos móviles de todos aquellos aficionados al ocio nocturno.

Figura 1.4.1: Tab Bar iKlubbers

Destacados: La primera opción del menú nos mostrará de una manera diferente a todos aquellos locales que, por pagar más, deseen ser mostrados en lugares preferentes de la aplicación.

Clubs: Desde aquí podremos ver todos los clubs que están dados de alta en el sistema, los podremos identificar situándolos a través de un mapa o a mediante una lista. Los locales que veamos serán el resultado a una búsqueda realizada mediante varios filtros (como se verá más adelante). Además, a partir de la información detallada de cada club, podremos apuntarnos a su lista VIP.

Favoritos: Tendremos una lista con todos los clubs favoritos de cada usuario. Además, cada vez que se acceda a ella, *iKlubbers* mandará un mensaje a nuestro servidor, de tal manera que podamos sacar estadísticas de las preferencias de cada persona.

Buscar: A parte de ver los clubes en un mapa o en una lista, los podremos buscar por nombre, algo imprescindible en cualquier aplicación.

Pases: Opción exclusiva para aquellas veces que nos apuntemos a una lista VIP. Una vez en ella, podremos consultar en este aparatado nuestro número de referencia para poder entrar al local sin tener que desembolsar ni un solo euro.

Además de estas funciones básicas se estan implementando 2 más, llamadas *Shake it!* y *Realidad aumentada*. La primera nos permitirá elegir un lugar de ocio nocturno de manera aleatoria, simplemente agitando el *iPhone* de manera brusca y bajo los parámetros de búsqueda que hayamos definido.

La segunda nos aportará información sobre lo que estemos visualizando en la cámara del dispositivo, superponiendo etiquetas encima de la misma. Estas etiquetas serán creadas por los propios usuarios de *iKlubbers*.

1.5 Perspectiva general del proyecto

Fundamentos Teóricos

Antes de Empezar

BrainStorming

Se explica la técnica de propuestas de ideas en grupo llamada BrainStroming

Scrum

Explica el método de trabajo en grupo basado en Scrum.

Lenguaje

HTML y PHP

Hace referencia a los lenguajes usados para programar la parte web del proyecto.

Objective C

Explica los fundamentos del lenguaje de programación para aplicaciones *iPhone/iPod touch*.

Herramientas de programación

WAMP

Hace referencia a los lenguajes usados para programar la parte web del proyecto.

XCode

Explica los fundamentos del lenguaje de programación para aplicaciones *iPhone/iPod touch*.

Interface Builder

Hace referencia a los lenguajes usados para programar la parte web del proyecto.

iPhone Simulator

Explica los fundamentos del lenguaje de programación para aplicaciones *iPhone/iPod touch*.

iKlubbers: Diseño e implementación de una aplicación iPhone - Parte iPhone Vol.I

Parte Práctica

Definición de la aplicación

Explicación de que es *iKlubbers* a nivel de programación

iKlubbers

Introducción

Introducción a la temática del tema a tratar y a las observaciones que se plantean.

Destacados

Explicación de la programación de dicha pantalla de la aplicación.

Clubs

Explicación de la programación de dicha pantalla de la aplicación.

Buscar

Explicación de la programación de dicha pantalla de la aplicación.

Web

Introducción

Introducción sobre que debe contener la web y la programación de las primeras páginas.

Usuario

Explicación de las pantallas de este sector de la parte 'usuario' de la web.

Clubs

Explicación de las pantallas de este sector de la parte 'club' de la web.

Resultados

Muestra una galería de imágenes y capturas con los resultados finales del trabajo.

Conclusiones y Líneas de Futuro

Comenta las conclusiones del trabajo y cita posibles mejoras u optimizaciones.

Fuentes Documentales

Enlaces y referencias a las fuentes documentales consultadas.

Capítulo 2:

Fundamentos teóricos

2.1 Antes de empezar

2.1.1 BRAINSTORMING

La **lluvia de ideas** (en inglés *brainstorming*), también denominada tormenta de ideas, es una herramienta de trabajo grupal que facilita el surgimiento de nuevas ideas sobre un tema o problema determinado. La lluvia de ideas es una técnica de grupo para generar ideas originales en un ambiente relajado.

La principal regla del método es aplazar el juicio, ya que en un principio toda idea es válida y ninguna debe ser rechazada. Habitualmente, en una reunión para resolución de problemas, muchas ideas tal vez aprovechables mueren precozmente ante una observación "juiciosa" sobre su inutilidad o carácter disparatado. De ese modo se impide que las ideas generen, por analogía, más ideas, y además se inhibe la creatividad de los participantes. En un brainstorming se busca tácticamente la cantidad sin pretensiones de calidad y se valora la originalidad. Cualquier persona del grupo puede aportar cualquier idea de cualquier índole, la cual crea conveniente para el caso tratado. Un análisis ulterior explota estratégicamente la validez cualitativa de lo producido con esta técnica.

2.1.2 SCRUM

Scrum es un modelo de referencia que define un conjunto de prácticas y roles, y que puede tomarse como punto de partida para definir el proceso de desarrollo que se ejecutará durante un proyecto. Los roles principales en Scrum son el *ScrumMaster*, que mantiene los procesos y trabaja de forma similar al director de proyecto, el *ProductOwner*, que representa a los *stakeholders* (clientes externos o internos), y el *Team* que incluye a los desarrolladores.

Durante cada *sprint*, un periodo entre 15 y 30 días (la magnitud es definida por el equipo, 2 semanas en nuestro caso), el equipo crea un incremento de software *potencialmente entregable* (utilizable). El conjunto de características que forma parte de cada sprint viene del *Product Backlog*, que es un conjunto de requisitos de alto nivel priorizados que definen el trabajo a realizar. Los elementos del *Product Backlog* que forman parte del sprint se determinan durante la reunión de *Sprint Planning*. Durante esta reunión, el *Product Owner* identifica los elementos del *Product Backlog* que quiere ver completados y los hace del conocimiento del equipo. Entonces, el equipo determina la cantidad de ese trabajo que puede comprometerse a completar durante el siguiente sprint. Durante el sprint, nadie puede cambiar el Sprint Backlog, lo que significa que los requisitos están congelados durante el sprint.

Scrum permite la creación de equipos auto-organizados impulsando la co-localización de todos los miembros del equipo, y la comunicación verbal entre todos los miembros y disciplinas involucrados en el proyecto.

Un principio clave de Scrum es el reconocimiento de que durante un proyecto los clientes pueden cambiar de idea sobre lo que quieren y necesitan (a menudo llamado *requirements churn*), y que los desafíos impredecibles no pueden ser fácilmente enfrentados de una forma predictiva y planificada. Por lo tanto, Scrum adopta una aproximación

pragmática, aceptando que el problema no puede ser completamente entendido o definido, y centrándose en maximizar la capacidad del equipo de entregar rápidamente y responder a requisitos emergentes.

Existen varias implementaciones de sistemas para gestionar el proceso de Scrum, que van desde notas amarillas "post-it" y pizarras hasta paquetes de software. Una de las mayores ventajas de Scrum es que es muy fácil de aprender, y requiere muy poco esfuerzo para comenzarse a utilizar.

2.2 Lenguajes

2.2.1 HTML y PHP

Como se ha comentado en apartados anteriores, teníamos que apoyar nuestra aplicación en otros formatos (dejando de lado *iPhone)* para enriquecer la experiencia *iKlubbers*. Se decidió desarrollar una web para esta finalidad, pero ya que no era el objetivo del proyecto ni dónde más debíamos enfocarnos, decidimos utilizar lenguajes que ya conocíamos, dejando de lado otros que seguramente nos hubiesen dado mucho más juego (como *flash)*.

Así pues, se implemento una página en *html* y *php*. *¿Por qué estos 2 lenguajes y no sólo html?* La elección de *html* nos permite colgar nuestra web en la red, y con *php* podemos hacer que esta web soporte varios idomas.

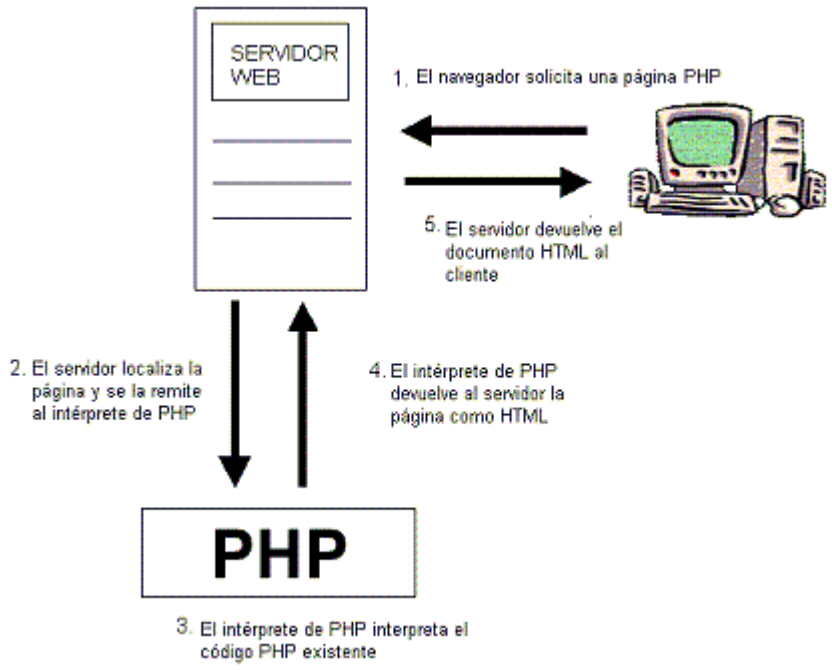


Figura 2.2.1.1: Diagrama PHP

En el diagrama no vemos por ningún lado *HTML* ya que no necesita ser interpretado en servidor, éste es un lenguaje que interpreta el cliente (el ordenador del usuario que está conectado a la red) mediante su navegador.

El lenguaje HTML puede ser creado y editado con cualquier editor de textos básico, como puede ser Gedit en Linux, el Bloc de Notas de Windows, o cualquier otro editor que admita texto sin formato como GNU Emacs, Microsoft Wordpad, TextPad, Vim, Notepad++ (este último es el que hemos usado), entre otros.

HTML utiliza etiquetas o marcas, que consisten en breves instrucciones de comienzo y final, mediante las cuales se determina la forma en la que debe aparecer en su navegador el texto, así como también las imágenes y los demás elementos, en la pantalla del ordenador.

Toda etiqueta se identifica porque está encerrada entre los signos menor que y mayor que (<>), y algunas tienen atributos que pueden tomar algún valor. En general las etiquetas se aplicarán de dos formas especiales:

- Se abren y se cierran, como por ejemplo: **negrita** que se vería en su navegador web como negrita.
- No pueden abrirse y cerrarse, como <hr> que se vería en su navegador web como una línea horizontal.
- Otras que pueden abrirse y cerrarse, como por ejemplo <p>.
- Las etiquetas básicas o mínimas son:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
<html lang="es">
 <head>
    <title>Ejemplo</title>
 </head>
  <body>
    <p>ejemplo</p>
  </body>
 </html>
```

Por otra parte, PHP es un lenguaje sencillo, de sintaxis cómoda y similar a la de otros lenguajes como C o C++, es rápido a pesar de ser interpretado, multiplataforma y dispone de una gran cantidad de librerías que facilitan muchísimo el desarrollo de las aplicaciones.

Con estas siglas nos referimos a un lenguaje de programación que está muy orientado al desarrollo de aplicaciones web. Cuando pedimos a nuestro servidor web una página PHP, que no es más que un programa PHP que genera HTML, antes de enviar dicha página al cliente se la pasa al intérprete de PHP. Este la interpreta y es el resultado de esta interpretación del programa PHP, contenido en la página PHP, lo que termina llegando al cliente. Supongamos que el contenido de una página web que reside en el servidor, y cuyo nombre es "intro.php", tiene el siguiente contenido:

<? echo "<h1>Mensaje desde PHP</h1>";?>

Cuando un navegador le pida al servidor web la página "intro.php", el servidor web va a darse cuenta, por la extensión ".php", de que esta página ha de enviarse primero al intérprete de PHP. Este recibe el contenido de la página y como resultado de esta ejecución (interpretación) genera una página HTML, que es la que envía al cliente. E una página PHP se puede mezclar HTML y PHP (como hemos hecho nostros), algo muy flexible pero que hay que manejar con cuidado ya que puede llevar a confusiones

PHP es un lenguaje basado en herramientas con licencia de software libre, es decir, no hay que pagar ni licencias, ni estamos limitados en su distribución y, podemos ampliarlo con nuevas funcionalidades si así lo quisiéramos. Respecto a su licencia, en la versión PHP 3.0 era GPL, pasando a ser modificada en su versión 4.0, por la incorporación de Zend, un nuevo interprete de PHP mucho más rápido que el anterior de PHP.

iKlubbers: Diseño e implementación de una aplicación iPhone - Parte iPhone Vol.I

2.2.2 Objective C

Objective-C es un lenguaje de programación orientado a objetos creado como un superconjunto de *C*. Originalmente fue creado por Brad Cox y la corporación StepStone en 1980. En 1988 fue adoptado como lenguaje de programación de *NEXTSTEP* y en 1922 fue liberado bajo licencia GPL para el compilador GCC. Actualmente se usa como lenguaje principal de programación en *Mac OS* X y *GNUstep*.

Objective-C requiere que la interfaz e implementación de una clase estén en bloques de código separados. Por convención, la interfaz es puesta en un archivo cabecera y la implementación en un archivo de código; los archivos cabecera, que normalmente poseen el sufijo .h, son similares a los archivos cabeceras de *C*; los archivos de implementación (método), que normalmente poseen el sufijo .m, pueden ser muy similares a los archivos de código de *C*.

La interfaz de la clase es usualmente definida en el archivo cabecera. Una convención común consiste en nombrar al archivo cabecera con el mismo nombre de la clase. La interfaz para la clase Clase debería, así, ser encontrada en el archivo Clase.h.

La declaración de la interfaz de la forma:

```
@interface classname : superclassname {
    // instance variables
}

+classMethod1;
+(return_type)classMethod2;
+(return_type)classMethod3:(param1_type)parameter_varName;

-(return_type)instanceMethod1:(param1_type)param1_varName
:(param2_type)param2_varName;
-(return_type)instanceMethod2WithParameter:(param1_type)param1_varName
andOtherParameter:(param2_type)param2_varName;

@end
```

iKlubbers: Diseño e implementación de una aplicación iPhone - Parte iPhone Vol.I

La interfaz únicamente declara los métodos de la clase pero no su implementación en sí; el código real está escrito en la implementación. Los archivos de implementación (métodos) normalmente poseen la extensión *.m*.

```
@implementation classname
   +classMethod {
       // implementation
   }
   -instanceMethod {
       // implementation
   }
@end
```

Visto como se implementa una clase en *Objective – C,* es obvia la similitud con el lenguaje *C* (ya que es un superconjunto de este). Este hecho nos ha facilitado mucho el trabajo, ya que este lenguaje era conocido por nosotros, y aunque hay muchas diferencias, las similitudes también son grandes.

2.3 Herramientas de programación

2.3.1 WAMP

WAMP es el acrónimo usado para describir un sistema de infraestructura de internet que usa las siguientes herramientas:

- Windows, como sistema operativo;
- Apache, como servidor web;
- MySQL, como gestor de bases de datos;
- PHP (generalmente), Perl, o Python, como lenguajes de programación.

El uso de un WAMP permite servir páginas html a internet, además de poder gestionar datos en ellas, al mismo tiempo un WAMP, proporciona lenguajes de programación para desarrollar aplicaciones web.

Por estos motivos decidí usar *WAMP*, para poder implementar de una manera sencilla todos los requerimientos que nos exigía la web.

2.3.2 XCODE

Xcode es el entorno de desarrollo integrado (IDE, en sus siglas en inglés) de Apple Inc. y se suministra gratuitamente junto con *Mac OS X*. *Xcode* trabaja conjuntamente con *Interface Builder*, una herencia de NeXT, una herramienta gráfica para la creación de interfaces de usuario.

Xcode incluye la colección de compiladores del proyecto GNU (GCC), y puede compilar código C, C++, Objective-C, Objective-C++, Java y AppleScript mediante una amplia gama de modelos de programación, incluyendo, pero no limitado a Cocoa, Carbón y Java. Otras compañías han añadido soporte para GNU Pascal, Free Pascal, Ada y Perl.

Entre las características más apreciadas de Xcode está la tecnología para distribuir el proceso de construcción a partir de código fuente entre varios ordenadores, utilizando Bonjour.

Este IDE es el que hemos usado para desarrollar nuestra aplicación para *iPhone*. Hay que decir que no es necesario implementar el código en *XCode* pero sí lo es para poder compilarlo y generarlo para ejecutar en un dispositivo móvil de *Apple*.

2.3.3 INTERFACE BUILDER

Interface Builder es la aplicación del entorno de programación para Mac OS X e iPhone OS que usamos para hacer interfaces gráficas de usuario. Es una aplicación muy potente, como podemos comprobar en detalles tan pequeños como el uso inteligente de las guías para colocar componentes. La programación en *Cocoa* es un tanto peculiar en cuanto a la unión de eventos y elementos de la interfaz gráfica. En otros entornos, como Visual Studio, para colocar una acción a un elemento de la interfaz hacemos doble click y se nos dirige directamente a la porción de código a escribir para ese elemento.

Sin embargo, el paradigma de programación que sigue Cocoa nos hace enlazar eventos de elementos de la interfaz gráfica a objetos (implementaciones en código). De esta interpretación de las interfaces aparecen los outlets (IBOutlet) y las acciones (IBActions).

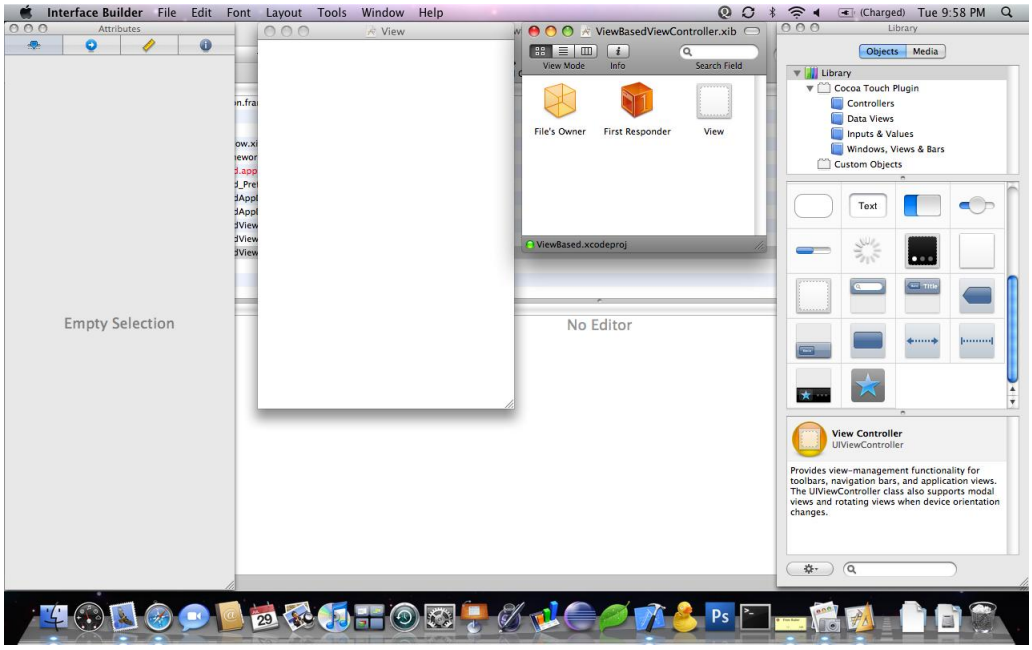


Figura 2.3.3.1: Interface Builder

2.3.4 IPHONE SIMULATOR

El SDK que nos proporciona *Apple* tiene como última herramienta el *iPhone Simulator*. Este software nos permite ejecutar nuestras aplicaciones directamente en nuestro *MAC*, sin tener que generar y compilar para volcarla en nuestro *iPhone*.

Figura 2.3.4.1: iPhone Simulator

De esta manera se agiliza muchísimo la etapa de programación del proyecto, ya que con una simple combinación de teclas, podemos probar en un simulador lo que realiza nuestro resultado final, y compararlo con lo que debería hacer.

Capítulo 3:

Parte Práctica

3.1 Definición de la aplicación

3.1.1 DEFINICIÓN DE LA APLICACIÓN

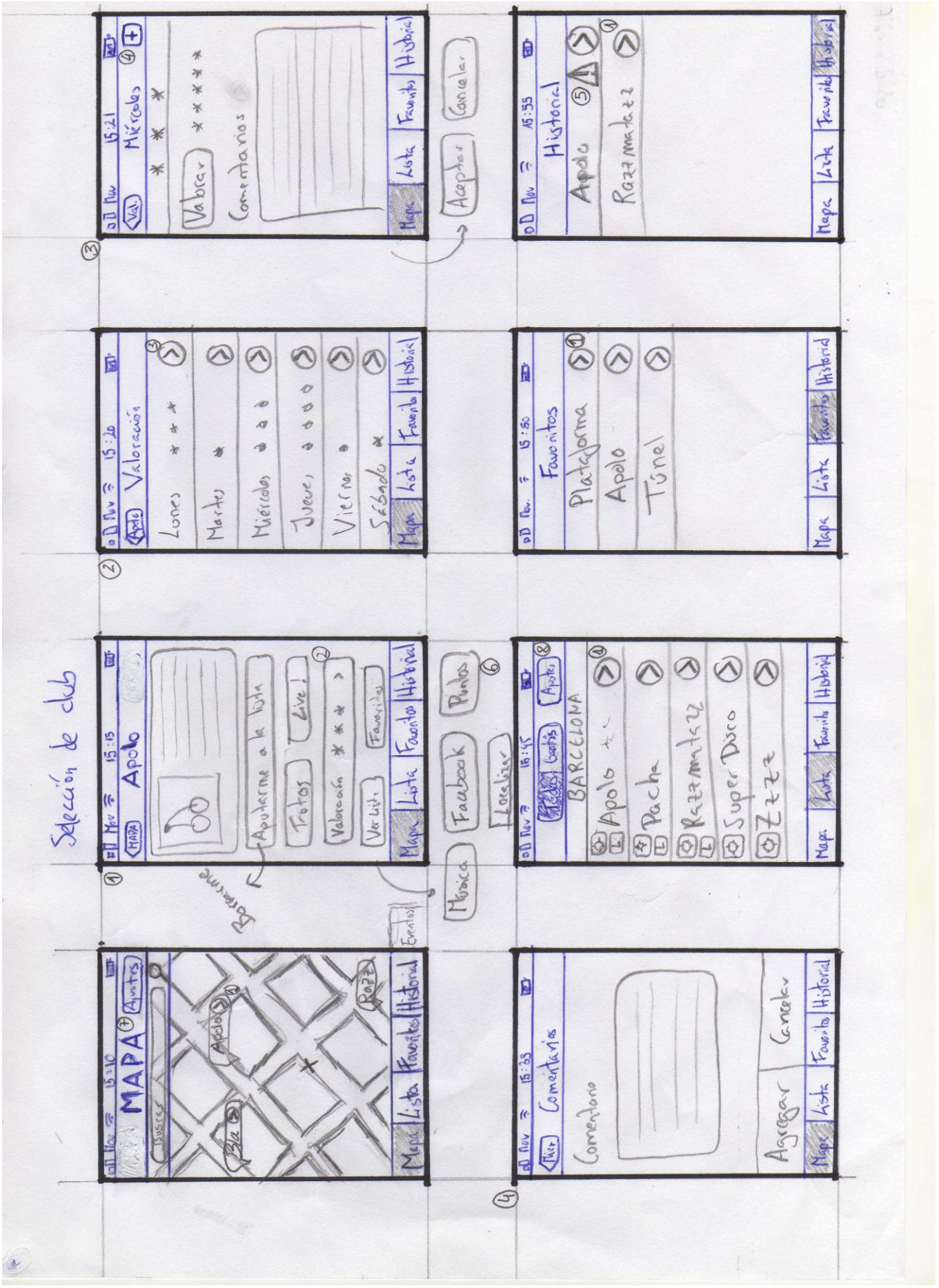
Como en todo proyecto que se precie, antes de empezar a programar cualquier cosa se tienen que sentar unas bases para garantizar que no se realiza más trabajo del debido y que el trabajo que se realiza es el correcto. Para hacer que nuestras horas de programación fueran más efectivas, tuvimos que reunirnos en numerosas ocasiones y definir que funcionalidades debía tener *iKlubbers* y cuales debíamos obviar.

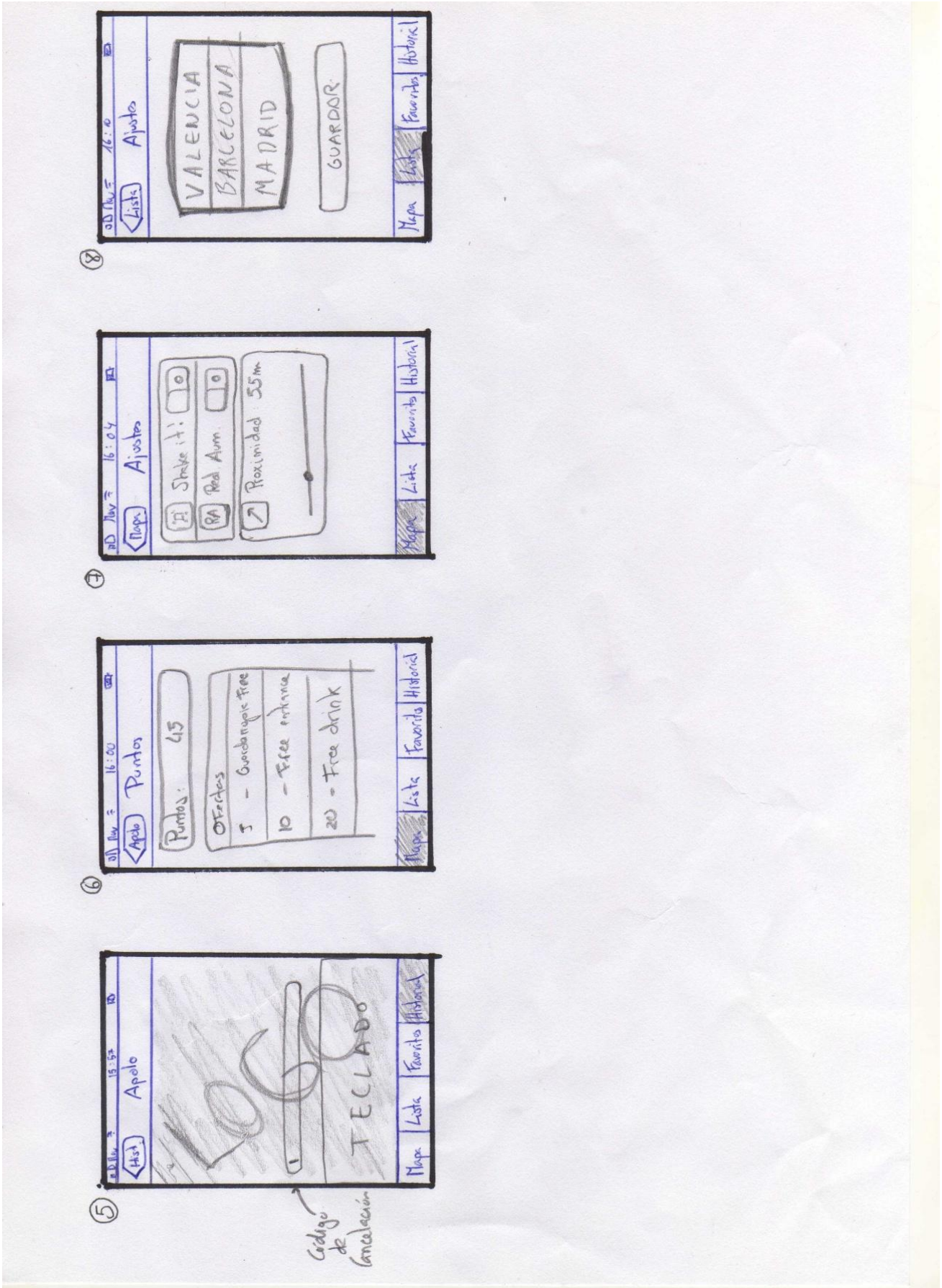
Pero antes, para definir dichas funcionalidades tuvimos que reunirnos los componentes del grupo y realizar un *Brainstorming*. De esta reunión salieron ideas descabelladas e ideas muy buenas. De la combinación de todas ellas salió la aplicación que hoy presentamos.

Una vez tuvimos todas las opciones posibles, y las filtramos para tener un proyecto acotado y realista, debíamos plasmarlas de un modo que el usuario pudiese apreciarlas de una forma fácil dentro de *iPhone*. Por esta razón, cada miembro del grupo hizo su propuesta de 'aplicación', y del total de ellas surgió la idea final.

La que vemos a continuación, es mi propuesta:

iKlubbers: Diseño e implementación de una aplicación iPhone - Parte iPhone Vol.I




Apolo
Puntos
Ajustes
Lista
Favorito
Historial
Mapa
TECLADO
Código de cancelación
Puntos: 415
Ofertas
1 - Guardaespalda free
10 - Free entrada
20 - Free drink
Shake it!
Red Aum.
Proximidad 55m
VALENCIA
BARCELONA
MADRID
GUARDAR

Como vemos en los bocetos, la idea es realizar una aplicación con un menú principal que muestra de manera rápida los contenidos de: *Mapa, Lista, Favoritos e Historial.*

De esta manera he intentado priorizar las opciones que a mi parecer serán las más usadas por los clientes. Desde todas las pantallas de este menú, podremos ver la información asociada a un club, tocando sobre él en la pantalla, ya que de una forma u otra (ya sea en modo lista o en modo mapa) el resultado que obtendremos siempre será un conjunto de locales seleccionados bajo un cierto criterio criterio.

Pero como es lógico, esto solamente es el primer boceto por mi parte, y hay muchas cosas que no son correctas (a nivel lógico para *iPhone)*, como poner los apartados de mapa y lista separados, ya que muestran la misma información pero de distintas formas. O no introducir la opción de buscador.

Para realizar este esquema, se tuvo en cuenta las características que nuestra aplicación debía ofrecer e intenté plasmarlas en papel de modo que en *iPhone* fueran lo más usables posible, hecho que distingue a *Apple* sobre todas las demás compañías, a nivel de telefonía móvil.

3.2 iKlubbers

3.2.1 INTRODUCCIÓN

Antes de empezar con la programación propia de cada parte de la aplicación, tuvimos que diseñar la estructura de la misma para poder generar las transiciones y efectos que deseábamos.

Para empezar, y debido a que la aplicación requería del elemento *TabBar*, añadimos como elemento base un *Tab Bar Controller*, donde se apoya toda la aplicación. Gracias a esto, solamente debemos enlazar nuestras vistas de cada pantalla con un ítem del menú, y la programación de navegación nos viene proporcionada automáticamente por *XCode* e *Interface Builder*.

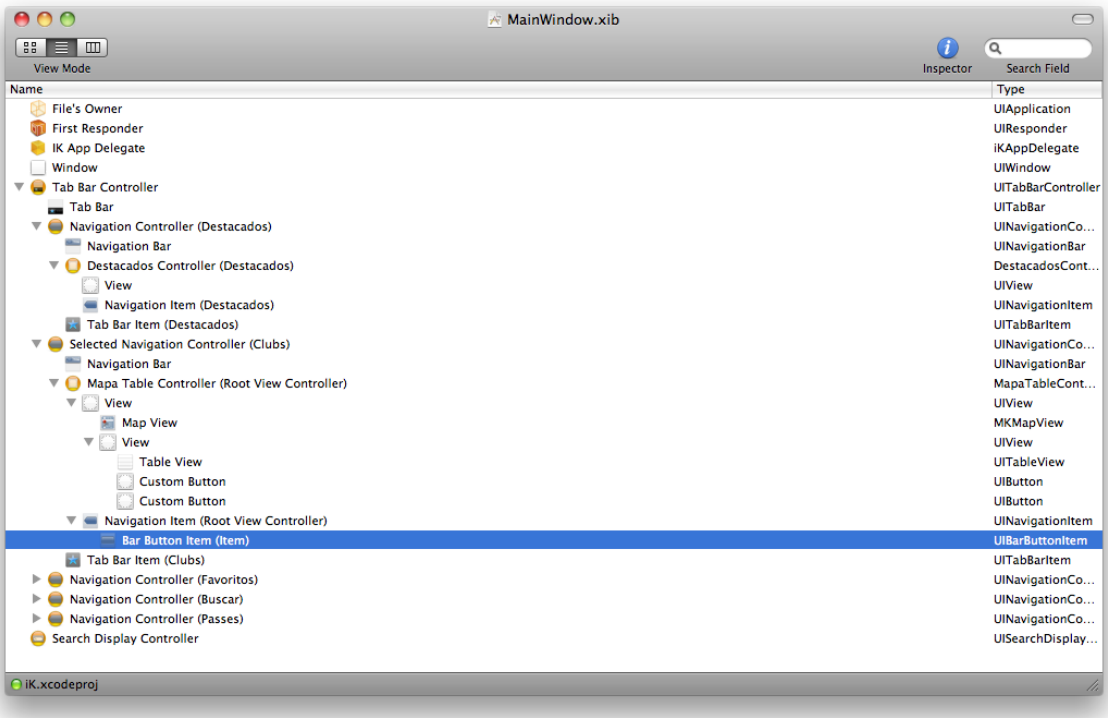


Figura 3.2.1: Estructura de la aplicación

Dentro del *Tab Bar Controller* añadimos un *navigation controller* por cada pantalla de la aplicación, para implementar la barra de navegación superior. Este será un código que también es proporcionado de manera automática por el IDE. El trabajo que realiza exactamente este elemento, es el de poder navegar por las pantallas que nos ofrece un mismo ítem de la *Tab Bar*.

Figura 3.2.1.2: Ejemplo de navBar

Una vez insertados estos 2 elementos desde el *Inteface Builder*, ya tenemos establecida la base de nuestra aplicación, y su navegación. De tal manera que ya podemos empezar a programar las diferentes vistas que formarán *iKlubbers*.

3.2.2 DESTACADOS

Como ya hemos comentado en varias ocasiones en este documento, *iKlubbers* no es solamente un trabajo académico, también es un proyecto que debe dar beneficios a la empresa que lo vende *(Mobivery)*.

Es el caso de *Destacados*. Una idea que no fue concebida por el grupo, sino que nos fue impuesta por el cliente para poder vender el producto con más facilidad. El argumento principal de esta pantalla es el de dotar a los clubs que paguen una cantidad extra, de un lugar privilegiado dentro del programa.

El funcionamiento es simple, cada vez que se acceda a *iKlubbers,* la primera pantalla que se visualizará será esta. Lo que veremos, serán los 'flyers' que proporcionen las discotecas para ese día, y mediante un motivo giratorio, podremos ver todos esos clubs que tienen el privilegio de salir en un lugar donde otros no pueden estar. Tocando la imagen de cada club, también podremos ir a su información asociada.

Para empezar, debemos aplicar una imagen a la *navBar*, este código será el mismo para todas las pantallas de la aplicación, y no lo volveremos a ver.

```
-(void)drawRect:(CGRect)rect{
      UIImage *navImage = [UIImage imageNamed:@"nav bar.png"];
      [navImage drawInRect:CGRectMake(0,0,320, 44)];
}
```

De esta manera le decimos que queremos que nos muestre la imagen "*nav bar.png*" con las medidas determinadas. Una vez ajustada la vista de la navegación, seleccionamos también un fondo de pantalla, y ajustamos los tipos de letra.

iKlubbers: Diseño e implementación de una aplicación iPhone - Parte iPhone Vol.I

Esta función pertenece a la clase *DestacadosController*, que es la clase que está asociada a la vista de *Destacados* en el *Interface Builder*.

```
#import <UIKit/UIKit.h>
#import "DestacadosRotatorViewController.h"

@interface DestacadosController : UIViewController {

      IBOutlet UIView *contentView_;
      DestacadosRotatorViewController *rotatorViewController_;

}

-(void) pushViewController:(UIViewController *) viewController;

@end
```

Si observamos el *.h* de esta clase, vemos que tiene un elemento que hace referencia a otra clase, llamada DestacadosRotatorViewController. Dentro de esta última clase será donde desempeñemos las funcionalidades de este apartado, a excepción de la acción de pulsar una imagen e ir a ver su información asociada. Esta acción se realizará mediante la función pushViewControlller:

```
-(void) pushViewController:(UIViewController *) viewController{

      [self.navigationController pushViewController:viewController
animated:YES];
      [viewController release];
}
```

Lo que haremos en esta clase, es cargar todos los clubs que deban salir en destacados cuando la vista se inicialice.

```
[super loadView];
[self cargarDestacados];
```

Una vez cargados los clubs inicializamos el contador de 'cartas' que tendremos sobre los 'flyers' de cada discoteca.

iKlubbers: Diseño e implementación de una aplicación iPhone - Parte iPhone Vol.I

```
destacadosCount_ = [[UILabel alloc] initWithFrame:CGRectMake(100, 10,
80, 20)];
destacadosCount_.textAlignment=UITextAlignmentRight;
destacadosCount_.textColor=[UIColor whiteColor];
destacadosCount_.backgroundColor=[UIColor clearColor];
destacadosCount_.font=[UIFontfontWithName:@"Helvetica-Bold"size:18.0];
```

De este modo estamos posicionando el contador y le damos un estilo propio.

La carga de los locales destacados dentro de la aplicación viene dada por el servidor, el cual nos retornara una lista con todos los elementos que necesitamos para, en caso de que toquen un club, poder mostrar su información asociada.

Esta clase, implementa otra llamada MVYRotatorViewController, que es una librería encargada de realizar el efecto de giro de las cartas de cada club, esta librería no ha sido desarrollada por no sotros, sino que se nos ha proporcionado a través de *Mobivery*.

Pero para hacer que funcione correctamente se debe implementar el método viewControllerForPage dentro de nuestra clase:

```
UIViewController *vc = [[UIViewController alloc] init];
       NSDictionary *oDestacado = [destacados_ objectAtIndex:page];
       UIButton *btn = [[UIButton
buttonWithType:UIButtonTypeRoundedRect] retain];
       btn.frame = CGRectMake(20, 60, 220, 260);
       [btn setBackgroundImage:[UIImage imageNamed:[oDestacado
objectForKey:@"flyer"]] forState:UIControlStateNormal] ;
       [btn addTarget:self action:@selector(myButtonClick:)
forControlEvents:(UIControlEvents)UIControlEventTouchDown];
      //Asignamos al botÃ³n el identificador del club
      int idClub = [[oDestacado objectForKey:@"id"] intValue];
       [btn setTag:idClub];

       [vc.view addSubview:btn];
       return [vc autorelease];
```

Con las 2 primeras líneas estamos volcando el club que toca pintar en pantalla en una variable local, seguidamente creamos un botón, al cual

le asociaremos una imagen de fondo (el "flyer" de la discoteca). ¿Por qué un botón? Porqué de esta manera podemos controlar el 'click' dentro de la imagen sin tener que llegar a implementar un UIViewController (un poco más complicado de hacer).

De tal manera, que a la propiedad 'tag' del botón le asocio el identificador del club, y además también le asocio la acción que debe realizar cuando reciba un evento 'touch inside' (cuando sea presionado). La función a realizar es la siguiente:

```
-  (void)myButtonClick:(id)sender {

      InfoClubViewController *inf = [[InfoClubViewController alloc]
initWithNibName:@"InfoClubViewController" bundle:nil];
      inf.inf=[destacados_ objectAtIndex:[sender tag]];
      [destacadosController_ pushViewController:inf];

}
```

De esta forma, llamo a la clase 'InfoClubViewController', la cual mostrará la información del club que yo le porporcione (gracias al identificador que hemos guardado en una de las propiedas del botón). Una vez inicializada la vista, llamo a la función 'pushViewControlle' de la superclase, ya definida anteriormente.

La clase 'MVYRotatorViewController' también requiere que se implementen 2 funciones con tal de tener la información necesaria para realizar el efecto de rotado, estas 2 funciones son las siguientes:

```
- (NSInteger) numberOfPages{
      return [destacados_ count];
}
- (CGSize) sizeOfPages{
      return CGSizeMake(251,300);
}
```

Con la primera, estamos determinando el número de 'cartas' que tendrá nuestra animación, y con la segunda ajustamos sus medidas.

iKlubbers: Diseño e implementación de una aplicación iPhone - Parte iPhone Vol.I

También debemos especificar una función para el pintado de las imágenes que se verán en ese instante (en nuestro caso solo podremos ver 3 a la vez). Esto se ha realizado con la función que sigue a continuación:

```
- (void) drawCards:(CGFloat)offset {
      [super drawCards:offset];

      int page = (int) (offset+0.5) % ((int) numberOfPages_) + 1;
      destacadosCount_.text=[NSString stringWithFormat:@"%d / %d",
page, (int) numberOfPages_];
}
```

Con esta función no solo determinamos la página a pintar sino que también estamos mostrando la imagen en la que estamos y de cuantas disponemos.

Figura 3.2.2.1: Destacados

3.2.3 CLUBS

En esta funcionalidad, se muestran aquellos clubs que se encuentran cercanos al usuario. A través de unos ajustes se podrá definir la distancia donde buscar clubs, ver solo a los clubs donde se pueda entrar gratis, y el tipo de música que se prefiere.

Para ayudar al usuario, se han propuesto dos formas diferentes de mostrar la información, una mediante el mapa, y otra mediante una tabla. En las dos se muestra la misma información, la diferencia es que en una, la tabla, se muestran los clubs más cercanos ordenados por proximidad al usuario, o por valoración, y en la vista de mapa se ve la ubicación del usuario más la situación de los clubs, de esta forma, el usuario podrá escoger como ver la información.

Para poder utilizar esta funcionalidad, para que en la misma vista se puedan ver la lista y el mapa, se añadió un botón que al clicar permite cambiar la vista rotándola.

```
[UIView beginAnimations:nil context:nil];

[UIView setAnimationDuration:1.0];

[UIView setAnimationTransition:UIViewAnimationTransitionFlipFromRight

forView:[self view] cache:YES];

[[self view] exchangeSubviewAtIndex:1 withSubviewAtIndex:0];

[UIView commitAnimations];
```

Se ponen las dos vistas bajo la misma, y luego, con el código de arriba, podemos realizar la rotación.

iKlubbers: Diseño e implementación de una aplicación iPhone - Parte iPhone Vol.I

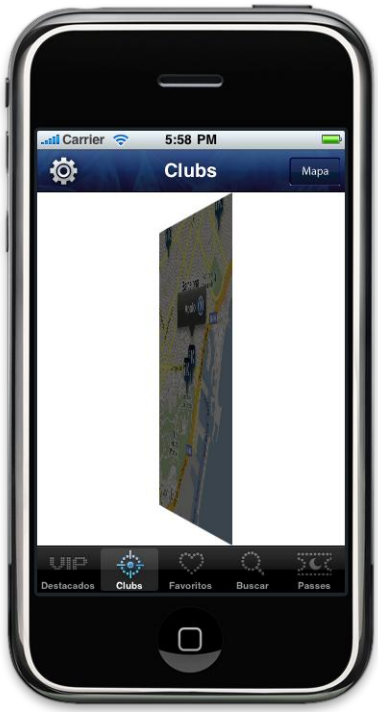


Figura 3.2.3.1: Rotación entre mapa y lista

Tabla

Para poder visualizar los clubs que hay cercanos al usuario, se muestra una lista con ellos ordenados por proximidad o por la valoración de los otros usuarios, según el usuario crea conveniente.

Para el desarrollo de esta funcionalidad, se ha creado un .plist (fichero XML) con clubs de prueba y con datos reales, tales como el nombre, logotipo, y posición.

Para cargar los clubs en la aplicación se utiliza un NSMutableArray. Esta estructura es un array modificable de objetos, con lo que además de tener las características de un array, tiene métodos de inserción y borrado de elementos

```
NSString *fileName = @"llista.plist";

NSString *filePath = [[NSBundle mainBundle] pathForResource:fileName
ofType:nil];
```

iKlubbers: Diseño e implementación de una aplicación iPhone - Parte iPhone Vol.I

```
tableList= [[NSMutableArray alloc] initWithContentsOfFile:filePath];
```

Con este código, primero definimos el nombre del fichero, luego él busca la ruta para llegar al fichero antes descrito, y finalmente lo abre y lo carga todo a la variable tableList. La variable tableList es una variable global, así podemos acceder a ella desde toda la clase.

Para poder programar la tabla, se tienen que programar una serie de funciones.

```
- (NSInteger)tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section {
```

En este método se indican cuantas filas tendrá la tabla. En ella retornamos directamente el número de objetos que tiene el array antes declarado.

```
- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath {
```

En este, se define cada celda de la tabla como es. La función da el número de la celda que se tiene que implementar, hasta llegar al máximo establecido por la función numberOfRowsInSection.

```
UITableViewCell *cell = [tableView
dequeueReusableCellWithIdentifier:kCellIdentifier];

if (cell == nil) {

cell = [[[UITableViewCell alloc]
initWithStyle:UITableViewCellStyleDefault
reuseIdentifier:kCellIdentifier] autorelease];

cell.accessoryType = UITableViewCellAccessoryDisclosureIndicator;

}
```

Primero definimos una variable del tipo celda para rellenar con la información, pedimos memoria y la inicializamos. El atributo accessoryType define que tipo de accessory se tiene que poner. Un accessory es un icono, que se sitúa a la derecha de las celdas de una tabla como control.

iKlubbers: Diseño e implementación de una aplicación iPhone - Parte iPhone Vol.I

```
NSDictionary* item = [listContent
objectAtIndex:indexPath.row];cell.textLabel.text = [item
objectForKey:kItemTitleKey];

return cell;
```

Luego cargamos el texto que queremos poner en la celda y se lo atribuimos. Finalmente devolvemos la celda misma.

Con esto, tenemos definido que texto va en cada celda, pero si queremos añadir más cosas como un icono, o un segundo texto, se tiene que crear una clase nueva que sea del tipo UITableCellView, y en el Interface Builer se crea la misma con los elementos necesarios. Finalmente, se le dice con código de la siguiente forma.

```
// Customize the appearance of table view cells.
- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    static NSString *CellIdentifier = @"CustomViewCell";

      ClubViewCell *cell = (ClubViewCell *)[tableView
dequeueReusableCellWithIdentifier:CellIdentifier];
      if(cell == nil){
            [[NSBundle mainBundle] loadNibNamed:@"ClubViewCell"
owner:self options:nil];
            cell = tmpCell;
            self.tmpCell = nil;
      }
      NSDictionary *inf=[[tableList
objectAtIndex:indexPath.row]retain];

      //name
      cell.name.text=[inf objectForKey:@"club"];
      [cell.name setFont:[UIFont fontWithName:@"Helvetica-Bold"
size:16.0]];
      cell.name.textColor = [UIColor colorWithRed:0.07 green:0.03
blue:0.14 alpha:1];

      //valoration
      cell.val.image=[UIImage imageNamed:[@"E"
stringByAppendingString:[[[inf objectForKey:@"valoracio"] stringValue]
stringByAppendingString:@".png"]]];

      //distance
      cell.dist.text=[[NSString stringWithFormat:@"%.2f",[self
dist:indexPath.row andPosx:0 andPosy:0]]
stringByAppendingString:@"m"];
      [cell.dist setFont:[UIFont fontWithName:@"Helvetica-Bold"
size:12.0]];
      cell.dist.textColor = [UIColor colorWithRed:1 green:0 blue:0
alpha:1];
```

iKlubbers: Diseño e implementación de una aplicación iPhone - Parte iPhone Vol.I

```
      //imagen free
      if([[inf objectForKey:@"free"] intValue]) cell.free.image =
[UIImage imageNamed:@"iconoFree.png"];
      else cell.free.image = nil;

      //logo
      cell.imageView.image=[UIImage imageNamed:[inf
objectForKey:@"photo"]];//[UIImage imageNamed:@"iphone.png"];
      [inf release];

      //accessoryIndicator
      cell.accessory.image = [UIImage imageNamed:@"ControlPag.png"];

    return cell;

      }
```

De esta forma podemos añadir la estructura que queramos i personalizar las tablas a nuestro gusto.

```
-(void)tableView:(UITableView *)tableView
didSelectRowAtIndexPath:(NSIndexPath *)indexPath {
```

Finalmente, este método, indica qué hacer una vez alguien pulsa una celda de la tabla. En nuestro caso, llevamos al usuario a otra pantalla.

```
InfoClubViewController *inf = [[InfoClubViewController alloc]
initWithNibName:@"InfoClubViewController" bundle:nil];

inf.inf=[tableList objectAtIndex:indexPath.row];

inf.navbar=[self navigationItem];

[self.navigationController pushViewController:inf animated:YES];

 [inf release];
```

Para abrir otra vista en el iPhone, declaramos una variable del tipo que queremos, y le añadimos la información que necesita. Con la instrucción pushViewController, se indica que se abrirá una nueva vista. Además, con las variables que le pasamos, le decimos que se quiere animada, con lo que la nueva vista entrará por la derecha deslizándose.

Con esto ya tenemos la tabla lista, el último paso es enlazar con el Interface Builder. Abrimos el archivo que contiene la interfície y añadimos

una tabla en la ventana. En la ventana de Connections Inspector, enlazamos los Outlets DataSource y delegate, de la tabla con el controlador, es decir, con la clase que hemos creado antes. Haciendo esto, le indicamos dónde puede ir a buscar los métodos que van a hacer funcionar la tabla.

Una vez está la tabla llena, se tiene que ordenar, pues especificamos que dicha tabla estaría ordenada por proximidad al usuario o por valoración. Para ordenarlo, utilizamos el algoritmo bubble, que se basa en mirar cada elemento de la lista, i comparando cada uno con su adjunto, i cambiándolos si es necesario. Una vez hemos ordenado el array, se tiene que actualizar la tabla, que se hace mediante el comando reloadData, método de la clase UITableView.

Figura 3.2.3.2: Lista de clubs

Mapa

Como hemos dicho anteriormente, en esta vista podemos obtener los mismos locales, pero vistos en sus respectivos lugares y relativos a nuestra posición actual. Lo primero que debemos hacer, es arrastrar una vista de mapa del *Interface Builder* a nuestra vista, una vez hecho esto, enlazamos el mapa con un IBOutlet que habremos creado anteriormente del tipo *IBOutlet MKMapView *myMapView*. Una vez hecho esto, para mostrar la posición actual del usuario, sólo debemos marcar la casilla del *Interface Builder* para habilitar esta opción *Show current user location*.

Lo primero que hacemos en esta vista, es cargar los locales a partir del mismo .plist que usamos para completar la tabla anterior.

```
- (void) loadClubsLocation {
      // Pcargamos los datos de los clubs que nos vienen del plist
      for (NSDictionary *club in tableList) {
            //Creamos una variable de localizaciÃ³n
            CLLocationCoordinate2D location;
            //Hacemos un cast de la longitud y la latitud
            location.latitude = [[club objectForKey:@"latitude"]
doubleValue];

            location.longitude = [[club objectForKey:@"longitud"]
doubleValue];
```

Guardamos los datos de posición de cada local para luego poder pintar la 'burbuja' asociada a este, que se verá cuando toquemos el icono del club en el mapa.

```
//Creamos la Annotation
            AddressAnnotation *myAdressAnnotation = [[AddressAnnotation
alloc] initWithCoordinate:location];
            myAdressAnnotation.Title = [club objectForKey:@"club"];
            //Agregamos la Annotation al mapa
            [myMapView addAnnotation:myAdressAnnotation];
            //Liberamos los recursos de la Annotation una vez que la
hemos insertado en el mapa
            [myAdressAnnotation release];
      }
}
```

iKlubbers: Diseño e implementación de una aplicación iPhone - Parte iPhone Vol.I

Como vemos en el código anterior, para insertar esta 'burbuja' debemos crear un objeto AddressAnnotation y añadirlo al mapa, iniciándolo con las coordenadas que hemos obtenido anteriormente de la lista de locales.

Figura 3.2.3.3: Mapa de clubs

Una vez insertadas todas las 'Annotations' las pintamos en el mapa dentro de la función

```
- (MKAnnotationView *) mapView:(MKMapView *)mapView
viewForAnnotation:(id <MKAnnotation>) annotation{
```

De tal manera, que diferenciamos las posiciones que pertenecen a un club de la posición actual del usuario. Si la posición es del local,

pintaremos una chincheta diseña por nosotros y asociaremos un botón para poder acceder a la información del club.

```
//Añadimos una imagen custom en lugar de la chincheta
            [annView setImage:[UIImage imageNamed:@"chincheta.png"]];
            UIButton *button = [UIButton
buttonWithType:UIButtonTypeDetailDisclosure];
            [annView setRightCalloutAccessoryView:button];
            //Añadimos el boton hacia la vista detalle DISCLOUSURE
            annView.canShowCallout = YES;
```

Sino, pintaremos simplemente una chincheta roja, que viene por defecto en este tipo de objeto (MapKit)

```
MKPinAnnotationView *pin = [[[MKPinAnnotationView alloc]
initWithAnnotation:annotation reuseIdentifier:defaultPinID]
autorelease];
pin.pinColor = MKPinAnnotationColorRed;
annView = pin;
```

Figura 3.2.3.4: Annotation

Ajustes

Como se ha comentado anteriormente, en esta parte de la aplicación podemos encontrar una pantalla de ajustes, que nos permitirá ejecutar un filtro determinado para nuestra búsqueda de locales, dependiendo de distancia, votaciones y tipo de música.

Para acceder al tipo de música, tendremos otra vista para poder seleccionar la que queramos.

```
- (IBAction) goType:(id)sender{
      tiposViewController *tipos = [[tiposViewController alloc]
initWithNibName:@"tiposViewController" bundle:nil];
      [self.navigationController pushViewController:tipos
animated:YES];
      [tipos release];
      }
```

Luego agregamos un *Slider* y un botón booleano para identificar si queremos ordenar por proximidad o no. En el *Slider* también tendremos un texto que nos dirá la distancia que queremos aplicar como límite a nuestra búsqueda por cercanía.

```
- (IBAction)sliderChange:(id)sender{
      prox.text = [NSString stringWithFormat:@"%.2f",[slider value]];
      }
```

3.2.4 BUSCAR

Como cualquier aplicación que gestiona datos de forma masiva, y entendiendo que este es un proyecto para buscar lugares de ocio nocturno, un lugar específico para buscar locales era obligatorio. De esta manera se ha implementado el buscador que *iPhone* ofrece por defecto cuando se desea encontrar algo específico.

Éste buscador, va a hacer una petición por cada acción que haga el usuario en la pantalla, es decir, por cada pulsación de tecla del teclado, va a hacer una petición al servidor. Si al hacer la nueva petición, la anterior todavía no se ha terminado, ésta se va a cancelar, manteniendo viva la última petición que se ha hecho. Luego, desde servidor se devolverá en primer lugar el club que contenga el nombre exacto escrito y a continuación clubes que tengan un nombre similar.

Mirando el código, podemos ver 2 listas diferentes, una cuando estamos directamente buscando en la caja de búsqueda del *gadget* y otra, cuando no estamos tocando nada.

```
- (NSInteger)tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section
{
      /*
       If the requesting table view is the search display controller's
table view, return the count of
     the filtered list, otherwise return the count of the main list.
       */
      if (tableView ==
self.searchDisplayController.searchResultsTableView)
      {
        return [self.filteredListContent count];
    }
      else
      {
        return [self.listContent count];
    }
}
```

De esta manera sabemos que lista debemos mostrar, la resultado de aplicar los filtros indicados, o sin ellos. Una vez hecho este filtro, pintaremos la tabla también siguiendo estas directrices de la siguiente forma.

```
- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
      static NSString *kCellID = @"cellID";

      UITableViewCell *cell = [tableView
dequeueReusableCellWithIdentifier:kCellID];
      if (cell == nil)
      {
            cell = [[[UITableViewCell alloc]
initWithStyle:UITableViewCellStyleDefault reuseIdentifier:kCellID]
autorelease];
            cell.accessoryType =
UITableViewCellAccessoryDisclosureIndicator;
      }

      /*
       If the requesting table view is the search display controller's
table view, configure the cell using the filtered content, otherwise
use the main list.
       */
      NSDictionary *product;
      if (tableView ==
self.searchDisplayController.searchResultsTableView)
      {
        product = [self.filteredListContent
objectAtIndex:indexPath.row];
    }
      else
      {
        product = [self.listContent objectAtIndex:indexPath.row];
    }

      cell.textLabel.text = [product objectForKey:@"club"];
      return cell;
}
```

Una vez tenemos el resultado en la tabla, simplemente hacemos que se dirija a la información del club si tocamos sobre el ítem (como hemos hecho en la tabla anterior).

```
InfoClubViewController *inf = [[InfoClubViewController alloc]
initWithNibName:@"InfoClubViewController" bundle:nil];
      inf.inf= product;
      [self.navigationController pushViewController:inf animated:YES];
      [[tableView cellForRowAtIndexPath:indexPath] setSelected:NO
animated:YES];
      [inf release];
```

Figura 3.2.4.1: Estados de la pantalla Buscar

3.3 Web

3.3.1 INTRODUCCIÓN

Como se ha comentado en apartados anteriores, la idea de hacer una página web es simple, apoyar en otras plataformas a la aplicación y de esta manera enriquecer la experiencia *iKlubbers.*

La página se basa en 2 ramas, por un lado la parte dedicada al usuario, la persona que tiene o tendrá la aplicación descarga en su teléfono, y por otro, la parte dedicada al club que quiere ser visible en nuestra aplicación. Esto se refleja en un *.php* con la apariencia que vemos a continuación:

Figura 3.3.1.1: Web

Pero antes de acceder a esta página, hemos tenido que pasar por una mucha más simpe, dónde hemos elegido el idioma en el que visualizaremos el contenido de la web (Castellano o Inglés).

El funcionamiento del website es sencillo, pasando por encima de la palabra "club" o "usuario" veremos información asociada a estas páginas.

Y haciendo 'click' en ellas nos iremos al apartado de la web correspondiente.

Para saber qué idioma hemos elegido en la pantalla anterior, evaluaremos la URL por la cual hemos llegado hasta aquí, en dónde recibiremos un parámetro cómo el siguiente:
www.iklubbers.es/clubuser/clubuser.php?lang=es.

El parámetro *lang=es (castellano) o lang=en (inglés) nos determina el idioma de la web. Para ello, dentro del código de la web se ejecuta la siguiente línea:*

```
<?php
echo retornaLiteral($_GET['lang']  , "indice", "titulo");
?>
```

Si analizamos el código, vemos que llamamos a la función *retornaLiteral*, la cual nos devolverá el texto del idioma, la página ("indice") y el apartado ("titulo") en el que estemos. Esta función, implementada en *php* se alimenta de un *xml* dónde encontramos todos los textos de la web en ambos idiomas y que sigue la siguiente estructura:

```
<?xml version='1.0'?>
<idiomas>
  <web lang="castellano">
      <pagina name="indice">
            <titulo>Titulo</titulo>
      <descripcion>Descripcion</descripcion>
      </pagina>
  </web>
  <web lang="ingles">
    <pagina name="indice">
            <titulo>Tittle</titulo>
      <descripcion>Description</descripcion>
      </pagina>
  </web>
</idiomas>
```

Además de estas características, se han implementado otras no tan funcionales pero un poco más gráficas. Si pasamos por encima de las

palabras que vemos en la pantalla, además de mostrar información, lo hará de una manera muy *amigable*. El foco de la otra palabra se apagará y saldrá nuestra descripción mediante un degradado (es decir, se mostrará invisible y poco a poco la iremos viendo hasta su estado final).

Para implementar estos efectos se ha usado *JQuery* de la siguiente manera:

```
<script src="http://code.jquery.com/jquery-latest.js"></script>
.
.
.
$('#botonClub').mouseover(function() {
  $('#focoUsuario').fadeOut('slow', function() {});
  $('#infoImgClub').fadeIn('slow', function() {});
  $('#infoClub').fadeIn('slow', function() {});
});
```

Si observamos el código, vemos que para poder usar *JGquery* solamente hace falta incluir la librería, que tendremos que tener descargada en nuestro servidor. La manera que se ha usado es la siguiente, cuando el ratón pasa por encima del objeto con *id=botonClub* los objetos definidos dentro de la función hará un *fadeIn (de visible a invisible)* o *fadeOut (de invisible a visible)*.

Con el ejemplo anterior vemos que pasando por encima de la palabra 'Club' la imagen de foco de '*usuario'* desaparece y aparece su información asociada (previamente extraída del *xml*) y la imagen que hay por debajo de esta.

Figura 3.3.1.2: Ratón sobre *usuario*

Además de esto, la página también cuenta con un *css* propio (como cualquier otra) para poder dotar a cada elemento de un estilo gráfico propio y una posición en la página web adecuada a las demandas de la diseñadora del grupo. También contamos con archivos *javascript* que nos ayudan a mostrar la página dependiendo de la resolución de pantalla del usuario y de su navegador.

3.3.2 USUARIO

Una vez estamos en la pantalla dedica al usuario de *iKlubbers,* encontraremos un menú con varias opciones en la parte superior.

```
   <div id="menu" style="position: absolute">
   <a href="www.google.es">club</a><b><a id="subMenu1"
href="iKlubbers.php">¿qué es iKlubbers?</a></b><br />
   <b><a href="iKlubbers.php">usuario</a></b><a id="subMenu2"
href="funcionalidades.php">funcionalidades</a><br />
   <a id="subMenu3" href="aboutus.php">about us</a>
   </div>
```

Y en la parte inferior se nos cargará la información del menú que hayamos seleccionado.

```
   <div id="panel" style="position: absolute">
   <div id="izquierda">
   <div id="titulopanel">
   <img src="../img/User/tituloIklubbers.png">
   </div>
   <div id="infopanel">
   <p>
   <?php
   echo retornaLiteral($_GET['lang']  , "iklubbers", "descripcion");
   ?>
   </p>
   </div>
   </div>
   <div id="derecha">
   <img src="../img/User/iPhone.png">
   </div>

   </div>
```

iKlubbers: Diseño e implementación de una aplicación iPhone - Parte iPhone Vol.I

Además, arriba a la derecha podremos cambiar el idioma de la web a inglés o castellano.

```
   <div id="idiomas">
   <a href="iKlubbers.php?lang=es">castellano</a> | <a
href="iKlubbers.php?lang=en">english</a>
   </div>
```

Figura 3.3.2.1: Pantalla de *usuario*

3.3.3 CLUB

En esta parte de la web, el club interesado podrá consultar la manera y condiciones para poder posicionarse en nuestra aplicación. La idea principal fue la de dotar a nuestro servidor con un *web service*, de tal manera que cada discoteca pudiese gestionar su contenido en *iKlubbers*. Pero una vez analizados el tiempo y la dificultad de tal desarrollo se desestimó la idea para poder centrarnos mucho más en la aplicación en sí.

Así pues, en esta pantalla veremos nuestros datos de contacto y las opciones que tiene un club para aparecer en nuestra aplicación móvil.

Capítulo 4:

Conclusiones y líneas de futuro

Después de mucho programar con *XCode* en *Objective C*, entiendes porqué este es un superconjunto de *C*, las similitudes entre estos dos lenguajes son enormes, y eso ha facilitado mucho la programación de esta aplicación, pero también se nota muchísimo que la tecnología que se aplica para *iPhone* está todavía en 'pañales' (hay que recordar que este dispositivo lleva poco más de 3 años en el mercado).

Y esto último se advierte debido a las grandes dificultades que un programador tiene para realizar tareas que, a priori, parece que deberían realizarse de una manera muy sencilla. Como por ejemplo, los problemas que se han tenido para realizar la rotación entre la vista de mapa y lista del apartado de clubs, ya que la primera implementanción hacia rotar toda la pantalla (navBar y tabBar incluídas).

A parte de problemas de programación generales, también hay muchos más particulares, como por ejemplo con la herramienta de *MapKit*. Resultó muy complicado hacer que una chincheta siguiese pintando la imagen personalizada que se le había asignado y no cambiase a la chincheta por defecto una vez clicada. O que el botón una burbuja (Annotation) llame a otra pantalla una vez ha sido tocada.

Aún y con estos problemas, que muestran, como hemos dicho antes, la precariedad de esta tecnología, se advierte un gran futuro para este tipo de aplicaciones, por 2 motivos principales. El primero, y posiblemente el más importante, es la gran aceptación de *AppStore* que permite a los desarrolladores vender sus productos de una manera rápida y sencilla. Esto hace que cada vez más gente decida entrar en este lenguaje de programación.

El segundo, es la constante evolución que tienen tanto *XCode* como *Interface Builder*, los cuales ya se parecen bien poco a las primeras versiones que surgieron. También evoluciona de manera muy sólida y

rápida las librerías proporcionadas por estos programas y los accesos que ofrecen. Por ejemplo, hasta hace poco era imposible acceder a funciones que tocasen la cámara del dispositivo, y si se hacía, se tenía que hacer a muy bajo nivel y de una manera muy costosa. Hoy ya se proporcionan las librerías adecuadas para poder 'jugar' con la cámara de un *iPhone* o un *iPod Touch*.

A grandes rasgos diría que hay muchos problemas todavía, pero que se están solucionando de una muy buena manera y que el futuro es prometedor.

También se nos han quedado varias cosas por hacer, debido a la falta de tiempo para hacerlas. En una versión futura, se debería integrar seguridad en la parte del servidor, para poder enlazar directamente la web con el mantenimiento de un club.

Dentro de la aplicación, también se debería implementar video en *live streamming*, es decir, video en vivo y en directo de lo que sucede en los locales. Y otra de las características que nos han quedado por desarrollar ha sido la de enlazar *iKlubbers* con *Facebook*.

Glosario

Bonjour:

Anteriormente Rendezvous, es una marca comercial de Apple para la implementación de la especificación de la IETF del marco de trabajo Zeroconf, una tecnología de redes de ordenadores usada en el sistema operativo de Apple Mac OS X desde la versión 10.2 en adelante.

Cocoa:

Cconjunto de *frameworks* orientados a objetos que permiten el desarrollo de aplicaciones nativas para Mac OS X. Objective-C es el lenguaje para escribir dichos *frameworks*, aunque también es posible programar en otros lenguajes de programación.

Lista VIP:

Listado de personas de relativa importancia (VIP: very Important people). Normalmente se usa para describir un listado de gente que puede entrar de manera preferente en diferentes locales.

Multitouch:

Nombre con el que se conoce a una técnica de interacción persona-computador y al hardware que la implementa. La tecnología multitáctil consiste en una pantalla táctil o touchpad que reconoce simultáneamente múltiples puntos de contacto, así como el software asociado a esta que permite interpretar dichas interacciones simultáneas.

navBar:

Diminutivo asociado por *XCode* para referenciar una *navigation Bar*, es decir, una barra de navegación, que se sitúa siempre en la parte superior de la pantalla y permite que nos desplacemos hacia atrás o hacia delante entre diferentes vistas de un mismo menú.

navBarController:

Controlador usado en la programación para una *navBar.*

Planning:

Conjunto de técnicas para conseguir el máximo aprovechamiento de los medios de producción de que dispone una empresa.

Post-it:

Pequeña hoja de notas que tiene una tira adhesiva en uno de sus extremos.

Product Backlog:

Documento de alto nivel para todo el proyecto. Contiene descripciones genéricas de todos los requerimientos, funcionalidades deseables, etc. priorizadas según su valor para el negocio (*business value*).

Product Owner:

Persona encargada de representar al equipo en el exterior, es decir, ningún cliente puede hablar directamente con el equipo, siempre debe hablar con el *Product Owner* y este se encargará de transmitir lo que crea necesario al resto del grupo.

ScrumMaster:

El *Scrum* es facilitado por un *ScrumMaster*, cuyo trabajo primario es eliminar los obstáculos que impiden que el equipo alcance el objetivo del sprint. El *ScrumMaster* no es el líder del equipo (porque ellos se auto-organizan), sino que actúa como una protección entre el equipo y cualquier influencia que le distraiga. El ScrumMaster se asegura de que el proceso Scrum se utiliza como es debido. El ScrumMaster es el que hace que las reglas se cumplan.

SuperConjunto:

Un superconjunto es un conjunto que incluye todos los elementos (y posiblemente más) de otro conjunto.

tabBar:

Situada siempre en la parte inferior de la pantalla, es una franja que contiene como máximo 5 botones para navegar por las diferentes opciones de la aplicación.

tabBar Controller:

Controlador usado en la programación para una *tabBar*.

Web Service:

Conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones. Distintas aplicaciones de software desarrolladas en lenguajes de programación diferentes, y ejecutadas sobre cualquier plataforma, pueden utilizar los servicios web para intercambiar datos en redes de ordenadores como Internet. La interoperabilidad se consigue mediante la adopción de estándares abiertos.

Fuentes documentales

iKlubbers: Diseño e implementación de una aplicación iPhone - Parte iPhone Vol.I

WEBGRAFÍA

http://developer.apple.com

http://www.iphonedevsdk.com

http://stackoverflow.com/

http://www.sourcecodeonline.com

http://forums.macrumors.com

http://es.wikipedia.org

VIDEOS TUTORIALES

http://www.youtube.com/watch?v=LRtimf3L8uA

http://www.youtube.com/watch?v=LtJI799MFA0&feature=player_embedded

http://www.youtube.com/watch?v=G4bbdZUG0oQ&feature=player_embedded

http://www.youtube.com/watch?v=LtJI799MFA0&feature=player_embedded

FIGURAS

iKlubbers: Diseño e implementación de una aplicación iPhone - Parte iPhone Vol.I

Figura 3.2.1.2

http://z.about.com/d/email/1/0/i/T/3/mail_undisclosed_recipients_iphone_mail_1.jpg

NOTA:

Las figuras incluidas en el trabajo y que no se ven reflejadas en esta sección son de elaboración propia.