UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

PROYECTO FIN DE CARRERA

CREACIÓN DE EMPRESA DEDICADA A LA INVENCIÓN Y DESARROLLO DE APLICACIONES PARA DISPOSITIVOS MÓVILES

Ingeniería de Telecomunicación

Jesús Ruiz Quijano
Marzo 2014

CREACIÓN DE EMPRESA DEDICADA A LA INVENCIÓN Y DESARROLLO DE APLICACIONES PARA DISPOSITIVOS MÓVILES

AUTOR: Jesús Ruiz Quijano
TUTOR: Germán Montoro Manrique

Dpto. de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Marzo 2014

Resumen y Palabras Clave

Resumen

Este proyecto es propuesto debido al gran incremento de oportunidades relacionadas con las aplicaciones móviles que han aparecido en el mundo laboral y las necesidades que surgen debido a este incremento. Para aprovechar estas oportunidades, se ha planteado crear una aplicación móvil y aprovecharla para crear una empresa con un carácter emprendedor.

Esta aplicación va a gestionar eventos entre los usuarios registrados a la vez que se permite la comunicación entre ellos a través de un chat interno. Para desarrollarla, se ha elegido la plataforma *iOS*, ya que posee un gran entorno de desarrollo, muy intuitivo, que facilita su uso en los primeros momentos.

Una vez desarrollada, se hará un estudio de los principales métodos para monetizar el desarrollo de una aplicación, al igual que los distintos métodos de financiación de las empresas con carácter emprendedor —*startups*—

Palabras Clave

SDK, *Smartphone*, *API*, *AppStore, Startup*

Abstract

This project is proposed due to the huge increase in opportunities related to mobile applications that have appeared in the workplace and the needs that arise due to this increase. To take advantage of these opportunities, it has been proposed to create a mobile application and use it to create a company with an entrepreneurial spirit.

This application will manage events among registered users and it will communicate them through an internal chat. For developing it, the iOS platform has been chosen, as it has a great development environment, very intuitive and easy to use at the beggining.

Once developed, a research of the main methods will show how to monetize the development of an application, as well as some methods of financing companies with entrepreneurial character —*startups*—.

Key Words

SDK, *Smartphone*, *API*, *AppStore*, *Startup*

Agradecimientos

En primer lugar me gustaría agradecer a mi tutor Germán, por la dedicación y el esfuerzo que ha mostrado a lo largo de este extenso proyecto. Siempre has demostrado interés y predisposición cuando me ha hecho falta, y es algo que me ha ayudado a llegar al punto donde ahora estoy. Muchas gracias.

Me gustará recordar a todos mis amigos de la carrera, porque no podría haber llegado aquí si no fuera por su ayuda. Por su apoyo en los momentos más difíciles y su compañía en los fáciles. Cubero, Muse, Chema, Jaime, Marta, Chuso, Javi, María, Gonzalo, David, Nerea, Pedro, Unai, Miguel, Borja… Sé que es arriesgado enumerar, espero no dejarme ninguno. Gracias a todos por estos años y sobre todo por los que nos quedan.

Siempre es más fácil hacer las cosas cuando tienes amigos de toda la vida que te apoyan y valoran tu trabajo. Que saben reconocer la dificultad y el esfuerzo que cuesta conseguir esta carrera y le dan siempre esa importancia. Querría mencionar en especial a Basti en este sentido. Y al igual que él, a todos mis amigos de toda la vida, con los que he compartido tantos momentos y con los que comienzo una nueva etapa de mi vida. Espero que sea una de tantas. ¡Gracias chavales!

También querría agradecer el apoyo de la gente que ha aparecido en la última etapa de mi vida, la cual ha resultado fundamental para mi proyecto. Gracias a sus constantes ánimos he podido conseguir la fuerza necesaria para acabar lo que empecé. En especial a Raúl, que ha tenido la actitud insistente que me hacía falta para acabar. ¡Gracias man!

Quiero mencionar a mi familia. A todos los que se han preocupado durante todo este tiempo de mi proyecto, dándome la perspectiva que necesitaba para verlo todo con la claridad precisa. Mi abuelo y mis abuelas, que tantos rosarios han invertido en mí. A mis primos y tíos. Y en especial a mis padrinos. Que son de las personas que más quiero y de las que he recibido más apoyo. Sin ellos no habría podido acabar esto. En un sentido más literal incluso de lo que parece.

Y por último, sería imposible cerrar esto sin agradecer más que a cualquier otra persona, a las tres que han estado a mi lado en todo momento. A los que han sufrido todas mis desesperaciones y han celebrado todas mis alegrías como si fueran suyas. Los que me han hecho sentir tan orgulloso de mi mismo casi tanto como lo estoy de ellos. Gracias a los cuales hoy puedo decir que soy ingeniero. A mi hermana y mis padres. Sin vosotros nada de esto sería posible. Nunca podré agradecéroslo lo suficiente. Gracias de corazón.

GRACIAS A TODOS.

~~Jesús~~

El Ingeniero Superior de Telecomunicación Jesús Ruiz Quijano :P

Índice general

Índice de figuras

1.Introducción

En la actualidad, los teléfonos móviles inteligentes o *smartphones* están copando el mercado de la telefonía móvil, gracias al gran aumento de las tecnologías y productos desarrollados para ellos.

El mercado de las aplicaciones para dispositivos móviles es uno de los más atractivos a día de hoy, ya que ofrece un alcance con gran potencial, sin mostrar apenas barreras de entrada. Esto permite a los desarrolladores, con un único producto, poder acceder a millones de usuarios de todo el mundo a través de una plataforma global. Este contexto se muestra muy favorable para actitudes emprendedoras, ya que la entrada a este mercado no requiere unas inversiones iniciales elevadas y ofrece unas posibilidades de éxito esperanzadoras, si se parte de una buena base.

Por otro lado, en los últimos años se están fomentando acciones con este carácter emprendedor, intentando impulsar este tipo de actitudes. Así, programas como Plan de Emprendedores Comunidad de Madrid [1], Madrid Emprende [2] o Línea ICO Empresas y Emprendedores 2013 [3] entre otros, apoyan a las nuevas empresas tecnológicas con una labor tanto económica como de asesoramiento.

1.1. Motivación

La motivación principal de este proyecto viene del interés del estudiante por la programación en aplicaciones móviles, un campo con una enorme proyección y el cual requerirá en un futuro próximo profesionales con las capacidades adecuadas. Dado este interés, la posibilidad de emprender a través de la creación de una empresa es altamente recomendable en este mercado, puesto que presenta unas barreras de entrada muy reducidas y ofrece la posibilidad de grandes beneficios si se tiene un buen producto.

1.2. Objetivos

El objetivo de este proyecto es la creación de una empresa de carácter tecnológico para el desarrollo de aplicaciones móviles. Dentro de este entorno, este proyecto se ha centrado en la implementación de *HiUp*, una aplicación para la gestión de eventos.

Esta aplicación permitirá al usuario organizar eventos de cualquier tipo —laboral, social, personal, de ocio...— e invitar a estas reuniones a otros usuarios registrados en la aplicación. Además, se implementará un chat interno que permita comunicarse a los usuarios de modo que no sea necesario recurrir a aplicaciones externas. Para ello, se hará un estudio de las diferentes alternativas de entornos de desarrollo que existen, al igual que de aplicaciones de la misma temática.

Una vez desarrollada, se hará un estudio sobre las diferentes posibilidades de ingresos que tendrá la aplicación y cuáles de ellas resultan más beneficiosas. Con esta información se elaborará un plan de acción que considere diferentes alternativas para poder emprender la creación de la empresa.

1.3. Organización de la memoria

En el primer capítulo se presenta el proyecto, la motivación y los objetivos del mismo.

En el segundo capítulo se hace un estudio sobre el estado del arte de los campos asociados a este proyecto: plataformas móviles existentes, productos de mercado similares y el mercado de las aplicaciones móviles.

En el tercer capítulo se hace una descripción de los diferentes módulos que componen la aplicación y la interrelación que existe entre ellos. Además, se explican diferentes conceptos que van a aplicarse durante el desarrollo del proyecto.

El cuarto capítulo trata sobre la fase de implementación y desarrollo de la aplicación móvil, detallando todas las alternativas que ofrece la aplicación al usuario.

En el quinto capítulo se plantea la viabilidad económica, estudiando distintos modelos de financiación para empresas como la desarrollada en este proyecto.

2.Estado del arte

Dentro del mundo de las aplicaciones móviles —*apps*—, son cada vez más numerosas aquellas que se enfocan desde un punto de vista social, intentando conectar al mayor número posible de usuarios. El aumento de este tipo de *apps* se debe, en su gran mayoría, al *boom* que han tenido las redes sociales a nivel global.

«El éxito de las redes sociales ha sido fulminante. Los expertos no encuentran ningún otro producto que haya recibido una acogida tan veloz y masiva.» [4]

Muchas de estas aplicaciones, entre sus funcionalidades, ofrecen la posibilidad de organizar eventos con todos los «amigos» con los que el usuario está conectado. Pero se trata de una función parcial de la aplicación y no un propósito concreto de la misma. Es por ello que este tema —la organización de eventos— queda difuminado entre un conjunto de opciones demasiado amplio.

En este capítulo van a verse algunos ejemplos representativos de este tipo de aplicaciones, al igual que las distintas plataformas en las que se encuentran desarrolladas y los requisitos para la implementación en cada una de ellas. Una vez estudiado los distintos casos, se presentará una conclusión argumentada de la elección hecha y la manera de proceder.

2.1. Plataformas móviles

Existen múltiples plataformas para el desarrollo de aplicaciones móviles en la actualidad. Éstas están diseñadas con el propósito de aprovechar al máximo los recursos de los dispositivos portátiles, a la vez que favorecer la interacción del usuario con el terminal. A continuación se van a mostrar las principales plataformas que conviven a día de hoy en los equipos móviles.

2.1.1. BlackBerry

Plataforma desarrollada por la compañía canadiense del mismo nombre, antes conocida como RIM —*Research In Motion*—. Esta plataforma se caracteriza por sus terminales con teclado físico y correo integrado desde su primer dispositivo, aunque algunos de los últimos modelos ya usan el teclado táctil en la pantalla del teléfono.

Enfocada siempre al ámbito laboral, esta plataforma permite además la navegación web y la descarga de archivos, al igual que la mensajería entre terminales de la compañía a través de *BlackBerry Messenger*, una aplicación de mensajería instantánea exclusiva para todos los teléfonos inteligentes de la marca *BlackBerry*.

Esta compañía se caracteriza por comercializar terminales de bajo coste, cubriendo la parte del mercado que los teléfonos de alta gama no pueden cubrir. En la actualidad se está actualizando el sistema operativo para adaptarlo a terminales táctiles y *tablets*, migrando todo el sistema. Es el caso de los últimos terminales sacados a mercado, que corren sobre la plataforma *BlackBerry* 10.

Pese a que en un principio fue capaz de disputar con sus competidores e incluso liderar las cuotas de mercado de los sistemas operativos para dispositivos móviles, lo cierto es que a día de hoy, la compañía canadiense no puede competir con las dos plataformas más utilizadas —*iOS* y *Android*—.

2.1.2. *Windows Phone*

Ha sido el último sistema operativo en salir al mercado de los dispositivos móviles. Se trata de la plataforma que sustituye a *Windows Mobile*, aunque es incompatible con ella. Su primera versión, *Windows Phone 7*, salió al mercado en 2010 y es incompatible, a su vez, con la que existe actualmente, *Windows Phone 8*.

Todos estos problemas de incompatibilidad junto con la tardía llegada al mercado hacen que tanto usuarios como desarrolladores sean reticentes a un cambio, comportamiento que se ve reflejado en el número de aplicaciones disponibles, que es claramente menor —cerca de 150.000 frente a las 800.000 tanto de *iOS* como de *Android*—, como se muestra en la figura 2.1.

Aun así, se trata de un entorno actualizado y robusto, con un sistema de desarrollo intuitivo gracias al alto número de *APIs* que se han introducido. Además, aunque cualquier fabricante puede desarrollar un teléfono con este sistema operativo, *Windows* exige que el hardware cumpla unos requisitos mínimos para que la experiencia del usuario sea similar en todos los terminales con *Windows Phone*. Esto asegura que los dispositivos con esta plataforma garanticen un buen rendimiento y que el desarrollador pueda contar con unos buenos recursos.

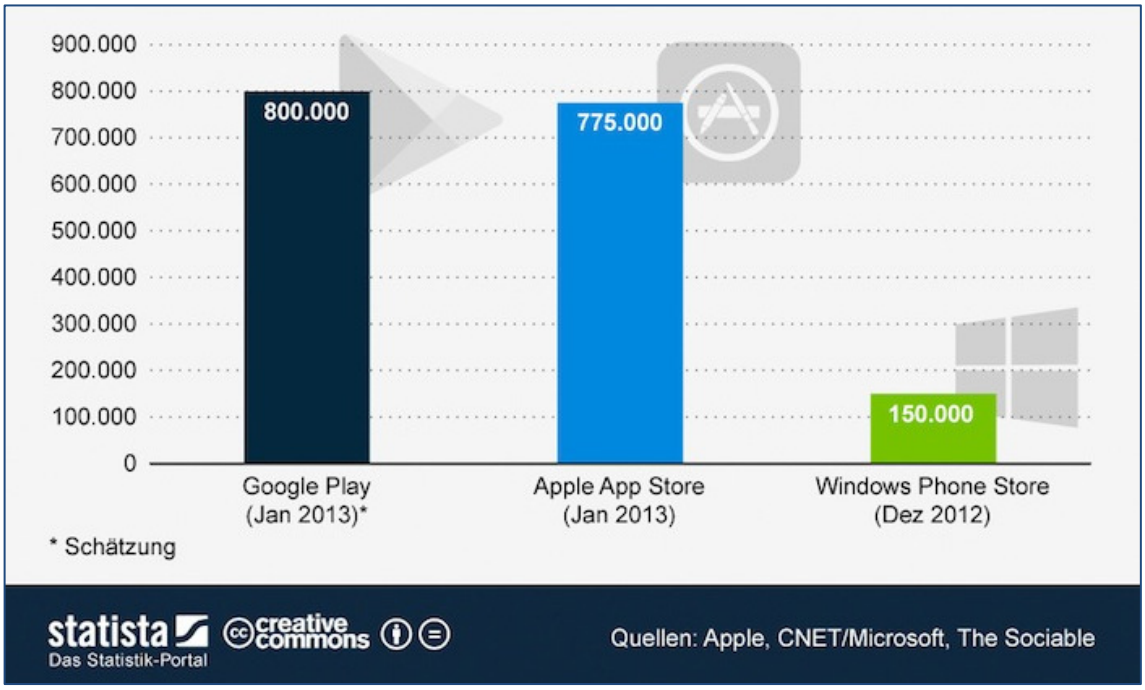


Figura 2.1 Número de aplicaciones disponibles en Google Play, App Store y Windows Phone Store. (20)

2.1.3. Android

Sin duda, se trata de una de las plataformas con mayor éxito y más extendidas hasta la fecha. Este sistema operativo fue desarrollado por la compañía *Android Inc.*, respaldada económicamente por la empresa *Google*, que más tarde la compró. En 2008 se sacó al mercado el primer terminal con este sistema operativo.

Se trata de un sistema basado en Linux, que se encuentra bajo la licencia Apache, una licencia libre y de código abierto. Es por ello que los desarrolladores tienen una tendencia a utilizar este sistema operativo, ya que además, el *SDK* tiene un precio muy asequible —una única cuota de 25 dólares—. Todas estas libertades han provocado que muchos de los fabricantes se hayan decantado por *Android*, con terminales de todas las gamas y prestaciones, disparando así el volumen de ventas de dispositivos con este sistema operativo.

Este *SDK* incluye un conjunto de *APIs* que permiten el uso de los recursos disponibles en el teléfono, como las conexiones inalámbricas —*Wi-Fi*, *Bluetooth*, 3G…—, acelerómetro, GPS, cámara… Además existe la posibilidad de realizar gráficos optimizados en 2D y 3D con la librería *OpenGL*. Todas estas prestaciones están recogidas en Eclipse, un entorno de desarrollo el cual incluye también un simulador del dispositivo, herramientas para la depuración, perfiles de memoria y rendimiento…

Pero todas esas ventajas proporcionadas por la libertad en el desarrollo suponen también ciertas limitaciones. Al existir tantos dispositivos con prestaciones distintas, el rendimiento de la aplicación puede variar mucho entre unos y otros, siendo responsabilidad del desarrollador adaptar el funcionamiento de la aplicación para el mayor número posible de modelos. Esta tarea resulta difícil y en muchos casos imposible de conseguir debido a incompatibilidades o prestaciones distintas entre los terminales.

2.1.4. iOS

Si existe un competidor de *Android* en el mercado de las plataformas para dispositivos móviles, ese es *iOS*. *iOS* es el sistema operativo móvil de la compañía *Apple Inc*. Es el sistema operativo usado en los terminales *iPad*, *iPod Touch*, *AppleTV* e *iPhone*, aunque originalmente fue desarrollado para este último.

Asentado sobre una variante del *Mach kernel* de *Mac OS X*, la plataforma se basa en el lenguaje de programación *Objective C* y ofrece un *SDK* que incluye, al igual que su competidor, una gran variedad de *APIs* para la gestión de los recursos de los tres dispositivos —*iPhone*, *iPad* y *iPod Touch*—.

El entorno de desarrollo —*xCode*— recoge todas estas *APIs* de uso, al igual que el emulador de los dispositivos, sistemas de depuración e interfaz gráfica donde se puede gestionar las vistas y los componentes gráficos contenidos en ellas. Y es aquí donde *xCode* se desmarca sobre los demás entornos.

Esta interfaz gráfica ofrece una manera muy intuitiva de realizar conexiones entre el código y los elementos gráficos, facilitando la programación tanto durante el proceso de iniciación en el aprendizaje —acciones sencillas e intuitivas— como en fases más avanzadas de mayor madurez—automatización de procesos—.

Por contra, el precio de esta *SDK* es de 99 dólares al año y es necesario abonar la cuota todos los años si se quiere mantener las aplicaciones en el *AppStore*, aún si ya no se está desarrollando ninguna nueva aplicación

La restricción que tiene este sistema operativo al poder ser instalado únicamente en dispositivos de la marca *Apple*, hace que siempre se pueda obtener el máximo rendimiento de la aplicación para todos los usuarios, sin depender más que de las versiones del propio terminal. Sin embargo, cuando atendemos al aspecto económico, los productos de la marca *Apple* suelen tener un coste bastante elevado, aspecto que suele ser una barrera para los usuarios a la hora de adquirirlos.

2.2. Productos de mercado

En el mercado actual existen aplicaciones orientadas a la organización y gestión de eventos con distintas características. En este apartado se van a analizar aquellas que guardan mayor similitud con *HiUp*.

2.2.1. Doodle

Doodle [5] es un servicio web con más de diez millones de usuarios al mes y muy sencillo de utilizar. Está diseñado para convocar reuniones y fijarlas en un calendario común con el resto de participantes. Para ello, *Doodle* cuenta con una herramienta para organizar encuestas, que permite al resto de usuarios convocados seleccionar la fecha y hora de la reunión. Una vez fijada la cita, esta se puede exportar a otros formatos y clientes de calendario, como *Outlook*, *Google Calendar*, *Exchange* o el *iCal* de *Apple*.

Doodle cuenta con diferentes planes de usuario, desde una opción básica y gratuita que no requiere registro, hasta planes de pago para usuarios individuales y empresas, estos con un mayor número de funciones y con el entorno gráfico personalizado. La aplicación ha sido desarrollada para dar soporte a este servicio web y tiene total compatibilidad con el mismo. La versión para usuarios cuesta 29 dólares al año, mientras que la destinada a empresas se eleva hasta 359 dólares al año, aunque la aplicación para móvil es gratuita. La figura 2.2 muestra la interfaz de la aplicación móvil.

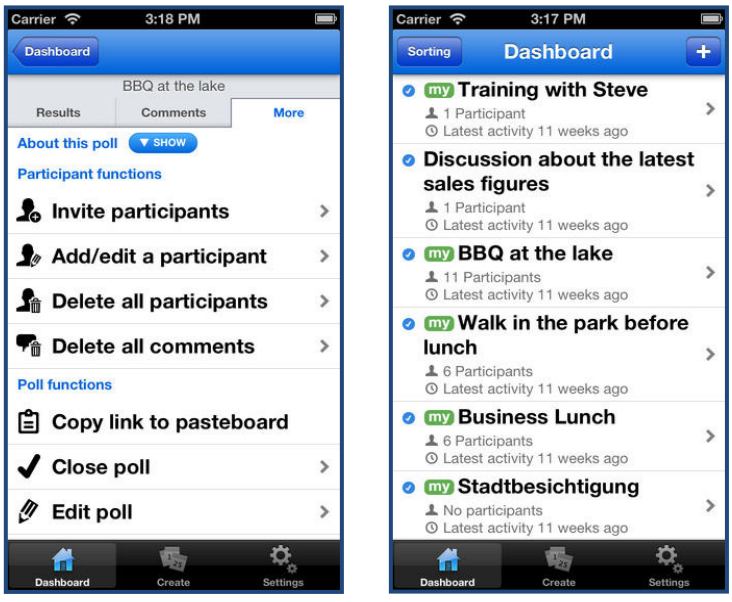


Figura 2.2 Interfaz app móvil Doodle: Easy scheduling (5)

2.2.2. *ScheduleOnce*

SheduleOnce [6] es una herramienta más completa y orientada a profesionales y empresas que precisen una gestión completa de todas las citas y reuniones de trabajo en una única plataforma. Entre otras opciones, *ScheduleOnce* permite integrar diferentes tipos de reuniones y gestionar los invitados de cada una de ellas. Cuenta con una opción básica, donde no es necesario registrarse, y un plan de pago para empresas y otras organizaciones a partir de 29 dólares al año. La página web corporativa anuncia un soporte para telefonía móvil en un futuro cercano. En la figura 2.3 se puede observar la vista que la herramienta web presenta al usuario.

Figura 2.3 Interfaz web ScheduleOnce (6)

2.2.3. *Facebook*

La red social con más usuarios del mundo —cerca de 700 millones de visitas únicas al mes [7] —, también da la oportunidad de organizar eventos a través de su aplicación. *Facebook* [8] permite a los usuarios organizar eventos indicando lugar, hora e invitados, que podría ser toda persona que tenga un perfil registrado en la red. También permite configurar el evento para que cualquier participante pueda invitar a más personas o restringir esta opción únicamente al creador del evento. Aparte, los usuarios pueden comentar cualquier tema relacionado con el evento a través de un espacio —«tablón del evento»— donde todos pueden escribir y leer lo que se ha escrito. La aplicación es totalmente gratuita. La figura 2.4 muestra dos pantallas para la creación de eventos en la aplicación móvil.

Figura 2.4 Interfaz de eventos Facebook app (8)

2.2.4. Mitmi

Mitmi [9] es quizás la aplicación más competitiva con la presentada en el proyecto debido a su parecido. Se trata de una aplicación con muchas similitudes en cuánto a interfaz y funcionalidad se refiere. Está orientada a organizar eventos con usuarios que estén registrados en la aplicación y permite la comunicación entre ellos a través de un chat integrado dentro de la propia aplicación. Y todo esto se enmarca en un entorno de una red social propia entre los usuarios.

Esta aplicación fue descubierta durante el desarrollo del proyecto y fue un duro revés para el mismo. De hecho, este parecido obligó a modificar ciertas partes del mismo para poder conseguir una diferenciación entre ambas aplicaciones y dar un valor añadido a la idea.

Esta diferenciación se ha conseguido gracias al modelo de negocio. *Mitmi* desde sus comienzos orientaba como cliente fundamental del producto y principal origen de los ingresos a los usuarios de la propia aplicación. Obtener un gran número de usuarios y a través de ese volumen, conseguir beneficios. Sin embargo, cómo será explicado más adelante, el modelo de negocio de *HiUp* está enfocado a obtener beneficios gracias a la publicidad otorgada a los locales y comercios dónde van a desarrollarse todas las reuniones. Sí que es cierto que este planteamiento también requiere de un gran volumen de usuarios para atraer a nuevos "clientes", pero no son estrictamente necesarios para poder empezar a obtener ingresos.

2.3. Mercado de aplicaciones móviles

Tras el estudio de las características, pros y contras de los distintos sistemas operativos para dispositivos móviles, es necesario conocer un poco el mercado al que se pretende acceder.

El mercado de las aplicaciones móviles se trata, por su propio carácter, de un mercado distribuido de manera global. Un único producto es capaz de llegar a cualquier lugar del planeta gracias a cada una de las plataformas de venta desarrolladas por los sistemas operativos. Es por ello que el estudio debe ser contemplado de esta manera, teniendo en cuenta el comportamiento de los usuarios desde un punto de vista general.

La empresa para el desarrollo de aplicaciones móviles *Flurry Analytics* [10], ha realizado un estudio donde se analizan los 30 países con mayor uso de aplicaciones móviles, teniendo en cuenta para ello las 20.000 aplicaciones más usadas. Se va a realizar un «análisis por grupos» —*cluster analysis*—, agrupando estos 30 países en función de su similitud en el uso de estas aplicaciones, de acuerdo a la figura 2.5.

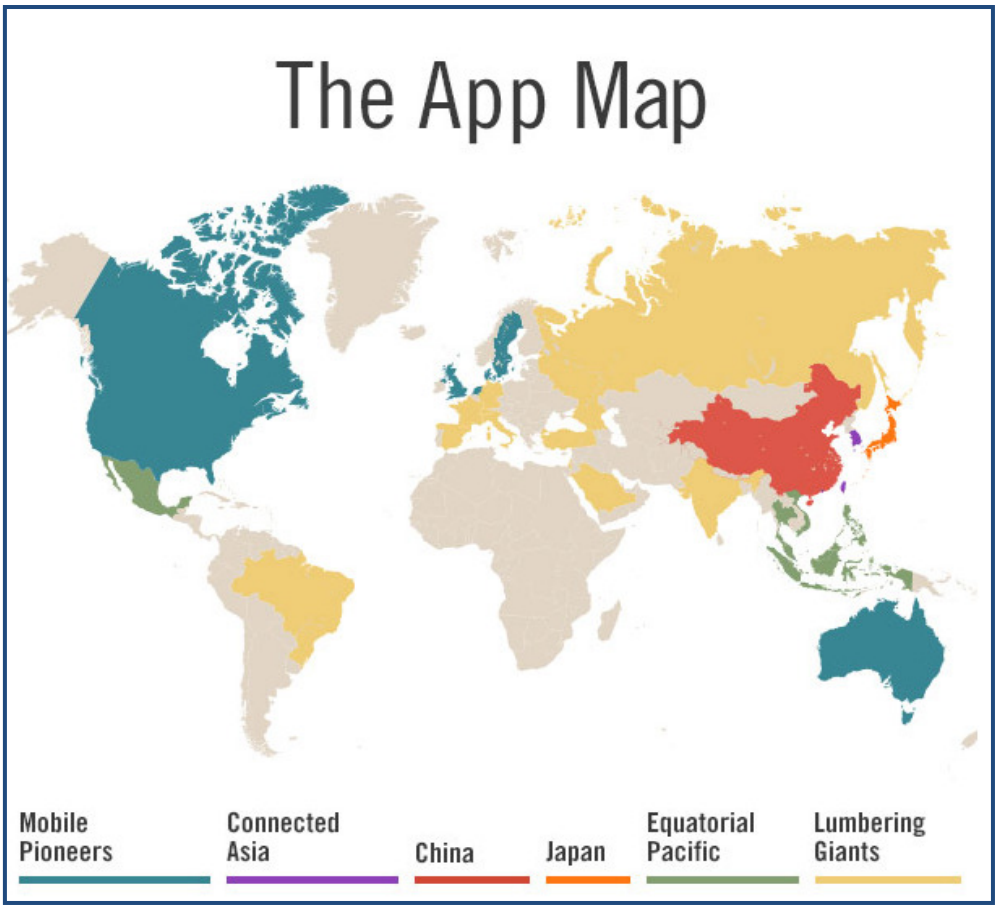


Figura 2.5 División de los países según su uso de aplicaciones (21)

En esta división pueden diferenciarse 6 grupos, concentrados según sus características:

- El primer grupo —*Mobile Pioneers*— son aquellos países que tienden a adoptar rápidamente las novedades tecnológicas en el campo de la telefonía móvil.
- Las dos grandes potencias asiáticas tecnológicamente hablando —China y Japón—, se consideran lo suficientemente importante como para ser dos grupos independientes.
- Por otro lado, *Connected Asia* muestras las zonas de Asia con mayor conexión en telefonía móvil —Corea del Sur, Hong Kong y Taiwán—, después de las dos ya mencionadas.
- Otro grupo está compuesto por varios países del sudeste asiático, a los que se incluye México debido a su similitud en los modelos de uso.
- Por último, el grupo representado con el color amarillo incluye muchos países con características diferentes, pero con un patrón similar entre ellos. Todos se mantienen a la zaga del grupo *Mobile Pioneers* y *Connected Asia*, siguiendo las tendencias que van marcando los pioneros.

Dentro de este informe se puede encontrar el interés que tienen estos países por distintas categorías de aplicaciones. De todos estos datos, existe un especial interés en dos de ellas, en los cuales podría incluirse *HiUp*.

- La primera se trata de aplicaciones de utilidad o productividad, cuyos datos quedan reflejados en la figura 2.6.
- La segunda categoría que podría encajar es aplicaciones «orientadas a un estilo de vida» —*lyfestyle-oriented apps*—, entendiendo como esto aplicaciones que definen el estilo personal y el uso cotidiano de cada usuario. Este segundo dato se muestra en la figura 2.7.

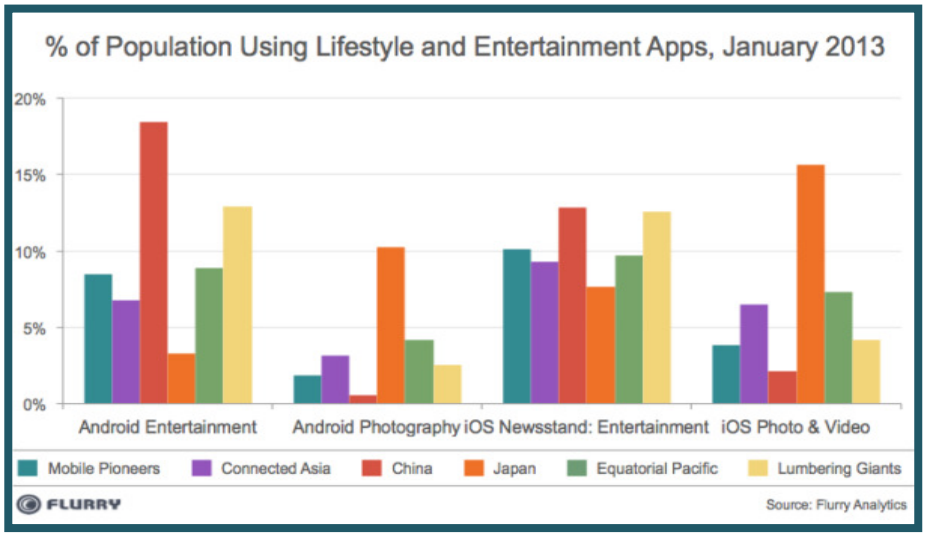


Figura 2.6 Uso porcentual de aplicaciones de «estilo de vida» por población (22)

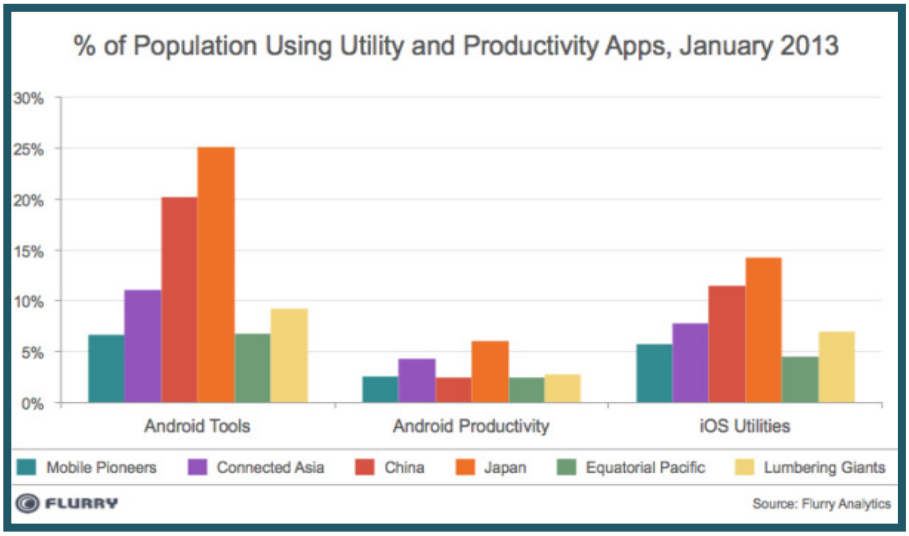


Figura 2.7 Uso porcentual de aplicaciones de productividad por población (23)

Como resultados importantes cabe destacar el mayor uso en este tipo de aplicaciones de las dos grandes potencias asiáticas, en especial Japón, que lidera en casi todas las categorías en todas las plataformas. Además, se observa un uso importante —cercano al 15% de media— de aplicaciones de productividad y utilidad de los grupos estudiados. Por último, se puede apreciar un mayor uso de estas aplicaciones en la plataforma de *Google,* mientras que cuando se trata de aplicaciones orientadas al «estilo de vida», la comparativa entre *Android* e *iOS*, se equipara.

Aparte de un análisis global, también es importante conocer las previsiones de las grandes compañías de la evolución del mercado. Este dato puede resultar crucial a la hora de elegir entre las diferentes opciones que existen.

Para ello se va a recurrir un informe publicado por la revista *Forbes* [11] y desarrollado por la empresa *Gartner* [12], empresa puntera en la consultoría e investigación de las tecnologías de la información.

Según la empresa estadounidense, la plataforma de *Google*, *Android*, domina las ventas de sistemas operativos a nivel mundial con un 60,3% de *share*. Este liderato, según las predicciones, se mantendría hasta el año 2016, aunque se vería reducido hasta el 50,3% en beneficio de *Windows Phone*, la cual aumentaría en casi un 600% su proporción. Pasaría así de un 3,9% en el 2013 a un 23,1% en 2016, arrebatando la segunda posición a *iOS.* El sistema operativo de *Apple* se mantendría muy estable, manteniendo una cuota que rondaría el 22% durante todo el período pronosticado. Las grandes perjudicadas serían *Blackberry*, que caería hasta la mitad su ya mermado porcentaje —de un 4,3% a un 2,4%— y *Symbian* —el sistema operativo de la compañía *Nokia*—, que perdería el 1,2% actual hasta prácticamente desaparecer en el 2016.

Todos estos datos quedan reflejados en la gráfica que muestra la figura 2.8.

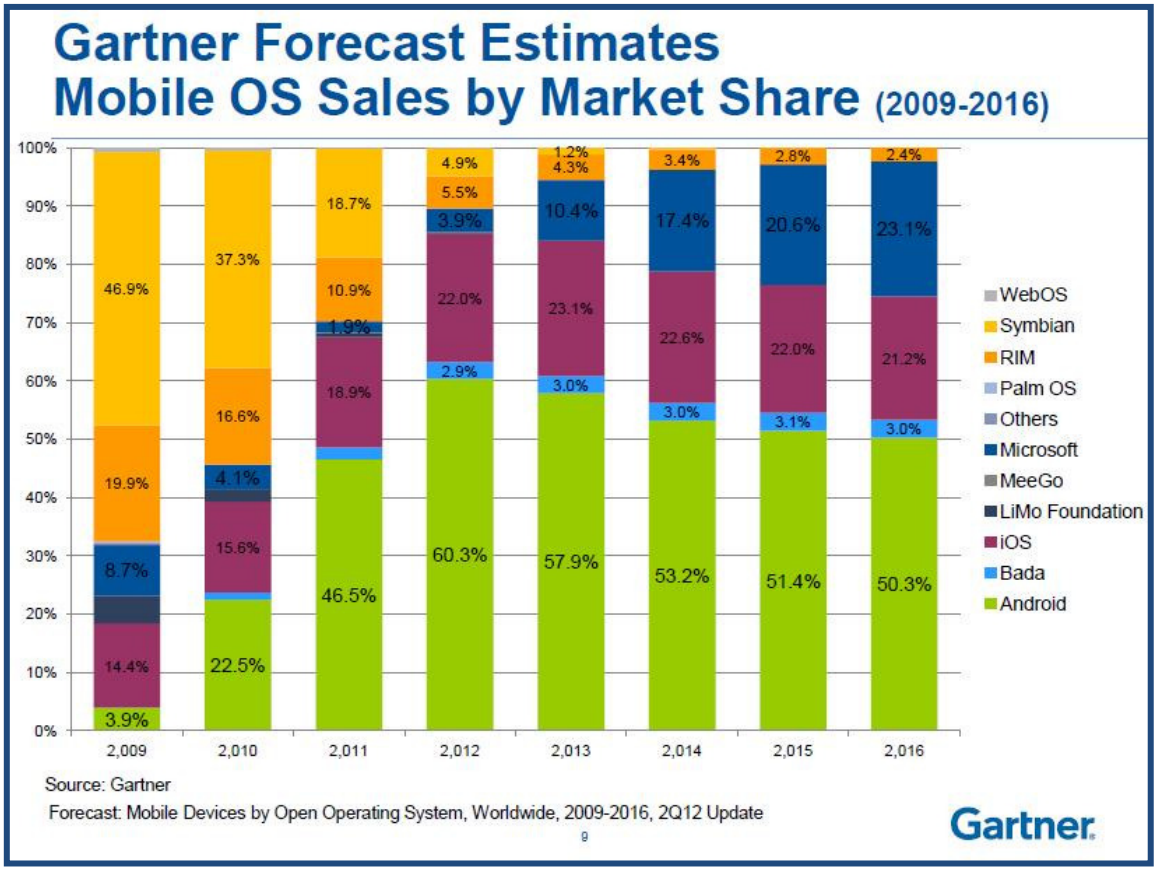


Figura 2.8 Predicción Market share sistemas operativos móviles (24)

2.4. Conclusiones

Como puede observarse en el estudio realizado en los apartados anteriores, existen tres sistemas operativos que cualquier desarrollador debería plantearse a la hora de comenzar una aplicación móvil. *Windows Phone*, *iOS* y *Android*. El resto de plataformas no se muestran nada favorables para el progreso de la aplicación, aunque no deberían descartarse para un futuro, ya que cualquier mercado, y en especial el mercado de la tecnología, puede sufrir grandes fluctuaciones en periodos cortos de tiempo.

El primero de ellos —*Windows Phone*— ofrece grandes perspectivas y será muy conveniente tenerlo en cuenta para futuras versiones, pero a día de hoy no sería la mejor elección de los tres. El sistema operativo empieza a ser consistente con las compatibilidades entre versiones y está haciendo grandes esfuerzos en motivar a desarrolladores y usuarios a realizar un cambio, pero en el momento de la elección generaba una gran cantidad de dudas y no fue planteado.

Teniendo en cuenta esto, la elección queda entre los dos sistemas operativos, *Android* e *iOS*.

Queda claro que la plataforma de *Google* es, en volumen de usuarios, la número uno. Esta posición la ha conseguido gracias a su diversificación y la adaptabilidad a los múltiples perfiles de usuario. *Android* puede encontrarse tanto en dispositivos con modestas prestaciones de bajo coste como en móviles de la más alta gama, pasando por *tablets* de cualquier modelo y siempre manteniendo una política razonable de precios.

Por otro lado, *iOS* ha sabido ganarse un sector notable del mercado y conservar su fidelidad. Ofrece un único producto — *iPhone*—, de altas prestaciones y una gran robustez, aprovechando al máximo los recursos del mismo. El resultado es un comportamiento fluido y unas aplicaciones que se ajustan perfectamente al terminal, gracias en parte al entorno de desarrollo y las propias «Guías para desarrollo» creados por *Apple*. Además, permite la adaptación a los distintos dispositivos de la compañía —*iPad*, *iPod Touch* e *iPhone*— con apenas modificación en el código fuente.

En cuanto a la programación en ambas plataformas, las dos proveen a los desarrolladores entornos gráficos para la creación de las aplicaciones, gran cantidad de *APIs* que permiten aprovechar al máximo las prestaciones de los terminales y numerosa documentación que detalla el funcionamiento básico para el desarrollo.

Es por ello que ambas plataformas resultan bastante parejas y suficientemente completas cuando se trata de un caso de iniciación, como es este proyecto. Por lo tanto, prácticamente queda a la elección del propio desarrollador cuál de las dos escoger.

Se optó por el entorno *iOS* para este proyecto por un motivo ligeramente diferenciador. En el momento de la decisión, Apple poseía una interfaz gráfica bastante intuitiva que le distinguía de su competidor. Este modelo de entorno de desarrollo facilita la programación y la automatización de las acciones básicas, sobre todo en los primeros pasos, permitiendo la unión de fragmentos de código con componentes gráficos de un modo totalmente visual. Aparte, existía ya un interés previo en la programación en este sistema.

Es por ello que se ha elegido la plataforma de *Apple* para el desarrollo de este proyecto.

Una vez elegida la plataforma de desarrollo, el siguiente punto que se debe abordar es la temática de la aplicación. En el momento de la elección, existían aplicaciones para se encargaban de la gestión de eventos, pero todas lo hacían de un modo incompleto. Alguna de ellas únicamente permitía fijar el lugar y el día, como es el caso de *Doodle*. En otras más completas, la gestión era un servicio añadido, perdiéndose entre un conjunto de funcionalidades mucho mayor y resultando poco intuitivo para el usuario. Aparte, la mayoría de ellas no permitía la comunicación entre los usuarios, hecho que se antoja fundamental para una aplicación móvil para gestión de eventos propiamente dicha.

Por lo tanto, y puesto que con los objetivos ya definidos se trataba de una aplicación bastante completa para un Proyecto de Fin de Carrera, se consideró adecuado desarrollar una aplicación de esas características.

Por último, desde el punto de vista empresarial, el mercado de las aplicaciones móviles se presenta idóneo para una empresa nueva de carácter tecnológico. Se trata, como ya se ha mencionado, de un mercado global con unas barreras de entrada casi despreciables, perfecto para una empresa con poco capital inicial. Además, las previsiones del mercado laboral son muy favorables para los desarrolladores de aplicaciones móviles, por lo que aprovechar el Proyecto de Fin de Carrera para obtener experiencia en este campo era muy interesante para el futuro de la posible empresa.

3.Diseño del sistema

HiUp es una aplicación para dispositivos móviles con la que se podrá organizar de una manera **rápida, intuitiva y sencilla** cualquier tipo de evento que el usuario desee. Además, de forma complementaria, la aplicación permitirá la comunicación entre sus usuarios gracias a un chat interno que también ha sido implementado en el desarrollo de este proyecto.

Para mejorar la rapidez y sencillez en el uso de la aplicación, la interfaz va a permitir al usuario poder acceder directamente a cualquiera de las cinco funciones en las que se divide la aplicación desde la pantalla inicial. Esta división sigue estos mismos criterios; se muestran a primera vista el mayor número de campos para facilitar el acceso, procurando no recargar en exceso la interfaz. De este modo, la aplicación queda organizada en:

- *Groups*: En este campo, el usuario podrá definir grupos de contactos que se usarán para simplificar el proceso a la hora de añadir gente, tanto a los eventos como a los chats.
- *Events*: En el segundo apartado, podrán verse todos los eventos a los que el usuario ha sido invitado. Estos eventos estarán organizados cronológicamente para facilitar una rápida visualización.
- *New event*: Es la función más importante y sobre la que gira toda la aplicación. Será aquí donde se definan todos los detalles de la reunión, tales como la fecha, el lugar, los invitados…
- *Chats*: Esta sección recoge todas las conversaciones del usuario con los contactos que tengan la aplicación ya instalada.
- *Ajustes*: Por último, el usuario podrá personalizar algunos parámetros, al igual que salir de la aplicación desde esta pestaña.

3.1. Requisitos previos

3.1.1. Modelo-Vista-Controlador (MVC)

Una aplicación móvil es un claro reflejo del patrón de diseño software **Modelo-Vista-Controlador** —MVC— [13]. Este patrón divide cualquier sistema o aplicación interactiva en tres áreas diferenciadas: procesamiento, salidas y entradas.

El **modelo** se ocupa de proteger y aislar toda la información importante, separando los datos que el usuario se preocupa de preservar de aquellos que son temporales o pueden ser regenerados por los modelos. A su vez, debe proveer métodos eficientes para consultar y modificar sus datos con seguridad, y es responsable de asegurar la integridad y la consistencia de los mismos. Es muy importante que estos métodos para extraer y alterar los datos estén bien definidos y sean respetados.

La **vista** es la cara pública de la aplicación. Determina lo que el usuario va a percibir de la aplicación y cómo va a interaccionar con ella. Su principal misión es comunicar los datos tan efectivamente como sea posible al usuario, al igual que transmitir el carácter interactivo que posee la propia interfaz. Una vista se ocupa de todo lo relacionado con la parte estética del sistema —imágenes, botones, estilos, plantillas…—. También puede indicar qué elementos van a interactuar con el usuario, pero todo el procesado de la información debe llevarlo a cabo el propio controlador de la vista. Una vista únicamente debe preocuparse de cómo se presenta la interfaz y los datos en ella.

El **controlador** es el responsable de procesar toda la información introducida por el usuario en la vista y actuar en consecuencia. Se encarga de unir vista y modelo y adaptarlos para cubrir las necesidades del usuario. Los límites de la función del controlador no están tan claros como en el caso de la vista o el modelo, pero tienden a tener una estructura bien definida por la propia funcionalidad del sistema.

En un caso sencillo, el controlador obtiene la información del modelo, la pre-procesa y envía a la vista para que sea mostrada. Una vez que el usuario interactúa con la vista, el controlador valida esas entradas, modifica el modelo en consecuencia y actualiza la vista. Estos pasos quedan mostrados de un modo esquemático en la figura 3.1.

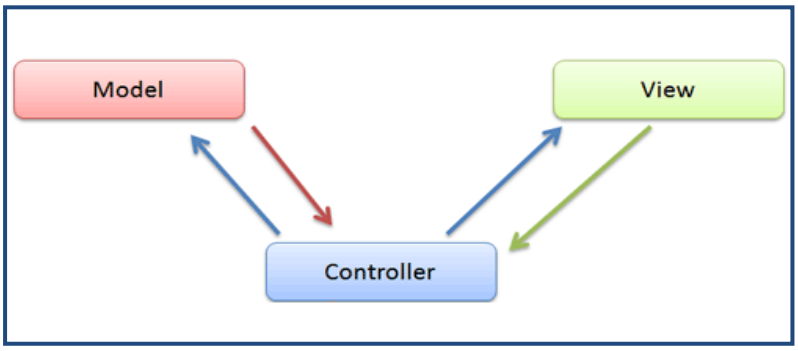


Figura 3.1 Esquema Modelo Vista Controlador (25)

3.1.2. Parse

Los usuarios de *HiUp* van a estar en constante comunicación, tanto en el chat como en la parte relacionada con los eventos. Esta interacción va a realizarse a través de un sistema **Cliente/Servidor**. [14]

Desde un punto de vista funcional, se puede definir un modelo Cliente/Servidor como una arquitectura distribuida que permite a los usuarios finales —clientes— obtener acceso a la información almacenada —servidores— de forma transparente, aún en entornos multiplataforma. Algunas de las características más representativas son:

- Recursos compartidos: Muchos clientes utilizan los mismos servidores y, a través de ellos, comparten tanto recursos lógicos como físicos.
- Protocolos asimétricos: Los clientes inician comunicaciones. Los servidores esperan su establecimiento pasivamente.
- Transparencia de localización física: Ninguno de los dos elementos de la comunicación tiene porqué saber dónde se encuentra situado el recurso que se va a utilizar.
- Encapsulamiento de servicios: Los detalles de implementación de un servicio son transparentes al cliente.
- Escalabilidad horizontal —añadir clientes— y vertical —ampliar potencia de los servidores—.

Hoy en día, esta tecnología está evolucionando, modificando estos servidores de almacenamiento local por servidores remotos. Este entorno es lo que comúnmente es conocido como «la Nube» —*the Cloud*—. El concepto no se refiere nada más que al uso de servidores con posibilidad de conectarse a Internet, los cuales son accesibles desde cualquier punto en el que exista una conexión a la Red.

Parse [15] se define como una plataforma de apoyo para aplicaciones móviles desarrollada completamente para su uso en la Nube. Provee un soporte para la mayoría de sistemas operativos móviles—*iOS*, *Android*, *JavaScript*, *Windows Phone 8*…—, a través de *APIs* desarrolladas específicamente para cada uno de los sistemas.

Parse ofrece al desarrollador un paquete con múltiples servicios —seguridad, almacenamiento, integración en redes sociales, notificaciones…— que simplifican y agilizan la implementación de aplicaciones móviles. Debido a todas estas facilidades, se ha elegido esta plataforma como servidor para el desarrollo de *HiUp*. El funcionamiento y la interacción con la misma serán detallados durante el capítulo de implementación.

3.1.3. La interfaz

El éxito o fracaso de una aplicación se debe, en gran parte, a la facilidad que se ofrezca al usuario a la hora de usarla y el atractivo de lo que está viendo. Una aplicación con una buena funcionalidad no tiene apenas valor si el usuario no es capaz de aprovecharla al máximo. Es por ello que la interfaz que se ha de presentar debe ser **clara, intuitiva y que saque el máximo partido a las prestaciones** tanto del terminal como de la propia aplicación.

Apple es consciente de esta realidad y se preocupa mucho por ello, ofreciendo un amplio soporte tanto documental como con herramientas adecuadas dentro del propio entorno de desarrollo —*xCode*—. Es tal la preocupación, que incluso ha creado el documento «*Guidelines Human Interface*», donde presenta las directrices para crear una correcta interfaz en el sistema *iOS*.

Una de las pautas a seguir para conseguir una buena interfaz es la claridad y simplicidad de la misma. Para ello es necesario que exista una correcta separación de funciones y que esta separación quede latente en el diseño.

La navegación entre estas funciones debe ser rápida y sencilla aunque, al mismo tiempo, debe haber una diferenciación suficiente entre ellas para saber que se están realizando acciones diferentes.

En una aplicación para *iOS* existen cuatro sistemas estándar de navegación propios del sistema: «*Navigation Bar*», «*Tab Bar*», «*Collection*» y «*Pages*».

- *Navigation Bar:* Es un sistema de navegación que se utiliza para elementos con un comportamiento jerárquico. Se trata de una barra en la parte superior de la pantalla donde se muestra el título de la vista y un botón que indica la vista superior en el orden de jerarquía. Se puede observar un ejemplo en la figura 3.2.

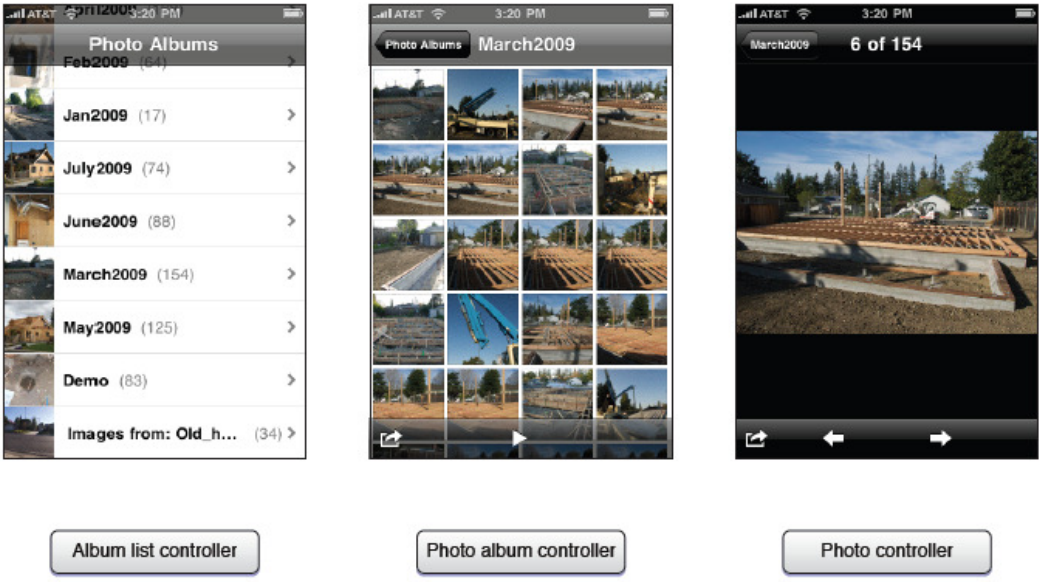


Figura 3.2 Ejemplo Navigation Bar Controller (26)

- *Tab Bar:* Este sistema es usado por elementos que no tienen una relación directa o se trata de una relación horizontal. En este caso, la barra se encuentra en el inferior y se divide en varias pestañas para poder navegar entre ellas. Aparece un ejemplo en la figura 3.3.

Figura 3.3 Ejemplo Tab Bar Controller (27)

- *Collection* y *Pages:* Estos dos últimos a pesar de que son usados para ello, no pueden considerarse puramente sistemas de navegación. Ambos se utilizan más para mostrar colecciones de objetos de una manera ordenada y estética. Se muestran dos ejemplos de aplicaciones que implementan estos sistemas en la figura 3.4.

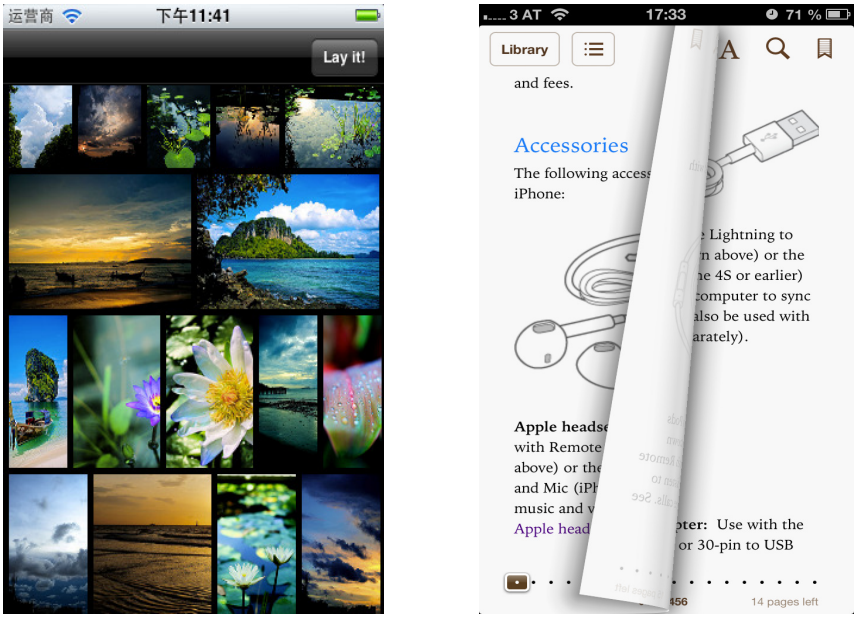


Figura 3.4 Ejemplo Collection (izda) y Pages (dcha.) (28)

Como ya se ha comentado, la interfaz es una de las partes más importantes de una aplicación y puede utilizarse como medio de diferenciación del resto de aplicaciones.

Para esta primera versión, se va a utilizar una interfaz genérica para la vista principal, usando una *Tab Bar* con cinco pestañas, una por cada función de la aplicación. Aparte, el sistema de navegación por cada una de estas funciones se gestionará a través de diferentes *Nav Bar*, la mayoría de ellas con transiciones estándar. Una vez dentro de cada vista, la apariencia de la misma y el comportamiento de los distintos elementos contenidos en ella han sido desarrollados específicamente para este proyecto.

Sin embargo, para conseguir una diferenciación con las *apps* competidoras, es importante separarse de los estándares marcados, sin perder de vista los objetivos marcados —sencillez, claridad y aprovechamiento de las prestaciones—. Para ello es muy recomendable crear una interfaz personalizada y atractiva que sea capaz de dar una posición diferenciadora con respecto a otras aplicaciones del mercado y que responda eficientemente a las funcionalidades que requiere la aplicación.

Por ello, pese que para esta primera versión no va a ser implementado, será muy conveniente tener en cuenta esta alternativa como una posible mejora en el futuro.

3.2. Elementos gráficos

Dentro del entorno de desarrollo, existe una gran variedad de elementos gráficos con los que puede conseguirse una incontable cantidad de interfaces que ofrecer al usuario. Sin embargo, varios de ellos tienen un especial interés para el desarrollo de este proyecto debido a su reiterado uso. Este apartado pretende dar una breve explicación de las características y principales usos dentro de la aplicación de alguno de ellos.

3.2.1. UIView

La clase *UIView* define un área rectangular en la pantalla y los métodos para la gestión del contenido en esa zona. Durante la ejecución del programa, un objeto *UIVIew* se encarga de cualquier contenido dentro de su área al igual que de las interacciones con ese contenido. Es la clase de la que derivan todas las demás. Dependiendo de los métodos implementados, estas subclases pueden ir desde simples botones a complejas tablas.

En el proyecto se han utilizado principalmente, junto con el objeto *UITableView*, como contenedores de todas las interfaces que se presentan al usuario. Gracias a su capacidad para agrupar varios elementos, *UIView* ha facilitado mucho el trabajo en la parte más dinámica de la aplicación, como son los movimientos y efectos visuales.

3.2.2. UITableView

El elemento gráfico *UITableView* es una subclase del objeto *UIView* que permite mostrar de una manera ordenada un conjunto de objetos que se encuentras almacenados en un contenedor genérico —*array*, diccionario, cola…—. Estos elementos se ordenan en celdas dentro de una tabla que tiene una o más secciones. Esta tabla es totalmente parametrizable, definiendo desde el alto de las celdas hasta el color del fondo o si se trata de una tabla estática —fija el número de celdas durante el proceso de desarrollo— o dinámica —puede modificarse durante la ejecución del programa—. Además, este elemento ofrece un conjunto de acciones gráficas muy reconocibles para un usuario de *iPhone* como son borrar a partir del botón «*Edit*» o acceder a una información más detallada de algún elemento de la tabla a través de la flecha que se muestra a la derecha de cada celda — cuyo nombre técnico es *disclosure indicator* —.

Dentro del proyecto, este elemento ha sido fundamental a la hora de mostrar la mayoría de los datos que se usan. De hecho, existen algunas pantallas que únicamente presentan un *UITableViewController* con la información correspondiente, como es el caso de la vista «*Groups*» o «*Chats*».

3.2.3. *UIImageView*

UIImageView es una subclase de *UIView* que proporciona un contenedor en el que mostrar una sola imagen o una animación de una serie de imágenes.

Cuando un objeto *UIImageView* muestra una de sus imágenes, el comportamiento real se basa en las propiedades de la imagen y la vista, aunque dependiendo de cómo se configure, la imagen pude ser ajustada de distintas maneras. Del mismo modo, este elemento permite configurar valores como la frecuencia o la duración en el caso de que contenga una animación.

Todas las imágenes que aparecen en *HiUp* están mostradas a través de un *UIImageView*. En algunas ocasiones, gracias a este elemento y algunos ajustes en código, se pueden conseguir efectos como los que podrán observarse en la pantalla de registro, mostrada en el capítulo 4.1.

3.2.4. *UIButton & UITextField*

UIButton y *UITextField* son dos de las subclases más básicas de *UIView*, pero a su vez de las más utilizadas.

La primera se trata de una vista que emula a un botón, como su propio nombre indica, y es capaz de realizar una acción cuando el usuario pulsa sobre ella. Además, el aspecto de esta vista es totalmente configurable, lo que permite al desarrollador poder crear de una manera casi inmediata cualquier tipo de forma con la que el usuario puede interactuar con la aplicación.

El segundo elemento es *UITextField*. Se trata también de un elemento de entrada de datos a la aplicación, pero en este caso de un modo literal. *UITextField* sirve para introducir información a través del teclado del terminal. La subclase implementa métodos que permite conocer cuando el usuario comienza a escribir, cuando termina o cuando está escribiendo, entre otros. Con esa información, el desarrollador puede permitir o prohibir diferentes acciones según le interese.

Prácticamente toda la interacción entre el usuario y la aplicación se ha implementado a partir de estos dos elementos, ya que gracias a su versatilidad pueden conseguirse efectos muy distintos con os mismos elementos gráficos básicos.

3.3. Funciones de la aplicación

Una vez descritas en la introducción las distintas funciones que se pretenden implementar en la aplicación, se va a hacer un estudio más detallado de cada una de ellas y el motivo por el que se han incluido.

3.3.1. Groups

Cuando el usuario comience a usar *HiUp*, una de las cosas que más va a utilizar son los contactos. Es por ello que, teniendo en cuenta la rapidez y sencillez, se ha creado el concepto de grupos.

Este campo *Groups* está pensado para agilizar al máximo el uso de la aplicación. En concreto a la hora de incluir personas en un evento. El usuario, gracias a este campo, podrá añadir a todas las personas que haya incluidas en un grupo sin más que seleccionarlo. Además, estos grupos podrán editarse o eliminarse en cualquier momento desde la propia pestaña, en donde se mostrarán todos los grupos que el usuario haya creado y que se encuentra en la pantalla inicial.

Cada grupo tendrá un nombre descriptivo y una foto obtenida de la galería del propio *iPhone* que servirá como icono identificativo del grupo. La selección de los contactos podrá hacerse abriendo una única vez la agenda gracias al método de multiselección de contactos incluido en la aplicación. Toda la gestión de esta información será realizada por una vista independiente, desde la cual el usuario podrá editar todos los datos según considere necesario, añadiendo o eliminando usuarios, modificando la imagen del grupo…

Cabe destacar que estos grupos serán almacenados en una base de datos dentro del propio terminal, por lo que cada usuario será responsable de los grupos que cree.

Tanto la vista de grupos creados como la encargada de la gestión de la información se muestran en la figura 3.5.

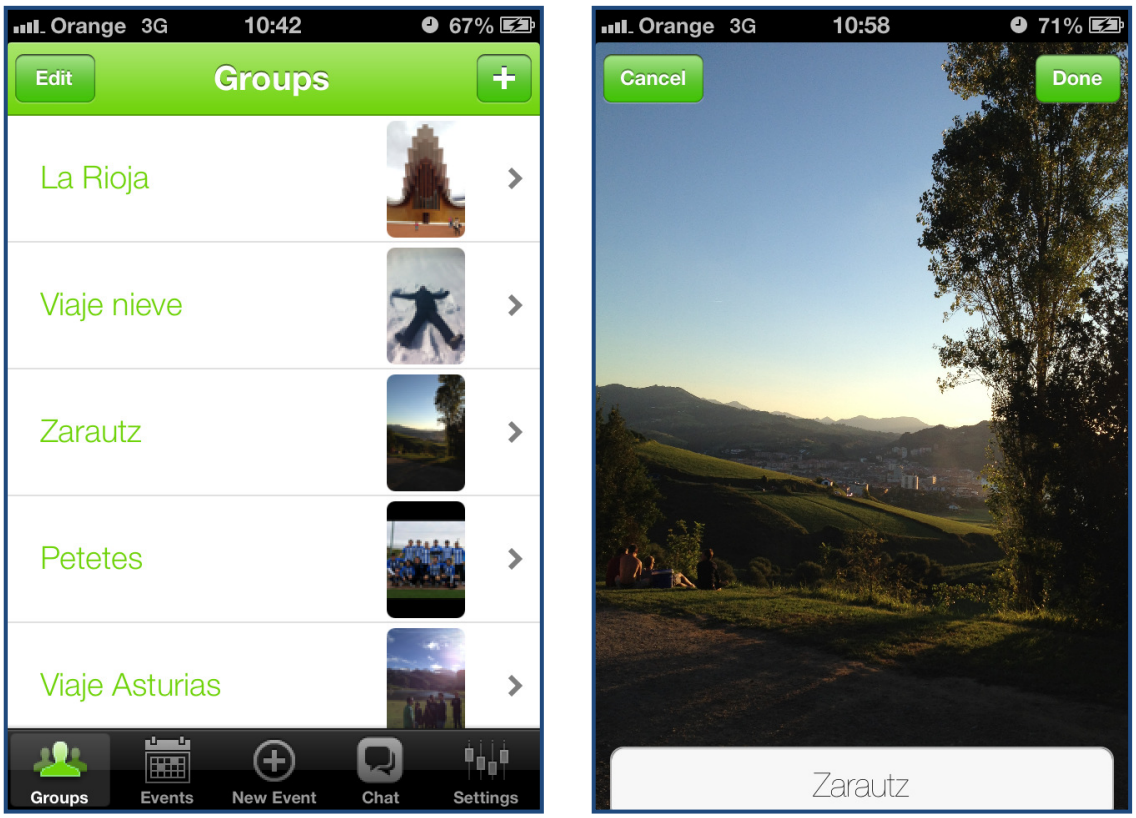


Figura 3.5 Vista Groups (izda.) e Info Group (dcha.)

3.3.2. Events

La función principal de la aplicación es organizar reuniones con amigos, conocidos, familiares o compañeros de trabajo. Y como función principal, es necesario ofrecer en la pantalla inicial todas estas reuniones.

La pestaña de eventos presentará todos los eventos a los cuales el usuario podrá asistir, haya sido creado por él mismo o invitado por otra persona. Estos eventos se mostrarán en orden cronológico. De este modo, el usuario podrá reconocer de una manera rápida y cómoda todas las reuniones que tenga planeadas. Además, mientras el usuario navegue por la tabla, se mostrará un marcador que irá indicando la fecha u hora del evento al que esté señalando.

Para acceder a la información de cualquier evento, no tendrá más que pulsar en la celda correspondiente y se abrirá una nueva pantalla donde podrán modificarse, en el caso de que el usuario tenga permiso, todos los datos que aparecen.

La figura 3.6 muestra la interfaz donde se presentan todos los eventos y la que se presentará cuando se quiera ver a la información de alguno de ellos.

Figura 3.6 Vistas Events (izda.) e Info Event (dcha.)

3.3.3. New Event

La creación de un evento debe permitirse de una manera inmediata. Es la base de la aplicación y como tal debe ser accesible desde la pantalla inicial.

La pestaña *New Event* muestra una pantalla en dónde podrá definirse toda la información del evento. Se especificará el nombre, fecha y hora, lugar, asistentes… También se concretará la imagen que más tarde identificará el evento. Por último, también podrá elegirse el lugar de la reunión, pero en este caso es importante detenerse momentáneamente en las opciones que se permiten.

Como quedará reflejado en el capítulo de viabilidad empresarial, los clientes que potencialmente retribuirán beneficios a la empresa son los regentes de aquellos locales públicos dónde pudieran organizarse los eventos. Debido a esto, es muy importante ofrecer un espacio atractivo dentro de la aplicación donde se muestren, de una manera organizada y clara, estos locales al usuario. Esto se consigue en el apartado *Places*.

En este apartado, se va a realizar una búsqueda de los locales a los que el usuario podría ir desde tres puntos de vista.

- El primero se trata de una búsqueda por cercanía al usuario. El dispositivo detectará la posición del usuario y cargará los resultados de una base de datos, mostrándolos en un mapa. En un principio, esta base de datos será externa a la empresa, aunque podrán filtrarse los datos recibidos para priorizar los requerimientos que tengan los propietarios.

- Una segunda opción de búsqueda permitiría al usuario, a partir de una información, encontrar el resultado que desea y obtener la dirección del local. Esta información podrá ser desde el nombre del local, hasta una dirección aproximada o un alias.

- Por último, una tercera posibilidad sería la búsqueda a partir de una dirección. El usuario introducirá una dirección válida y la aplicación mostrará en el mapa dicha dirección.

Todas estas búsquedas serán indexadas según un criterio que será explicado en el capítulo de implementación. La figura 3.7 muestra dos vistas del apartado *New Event.*

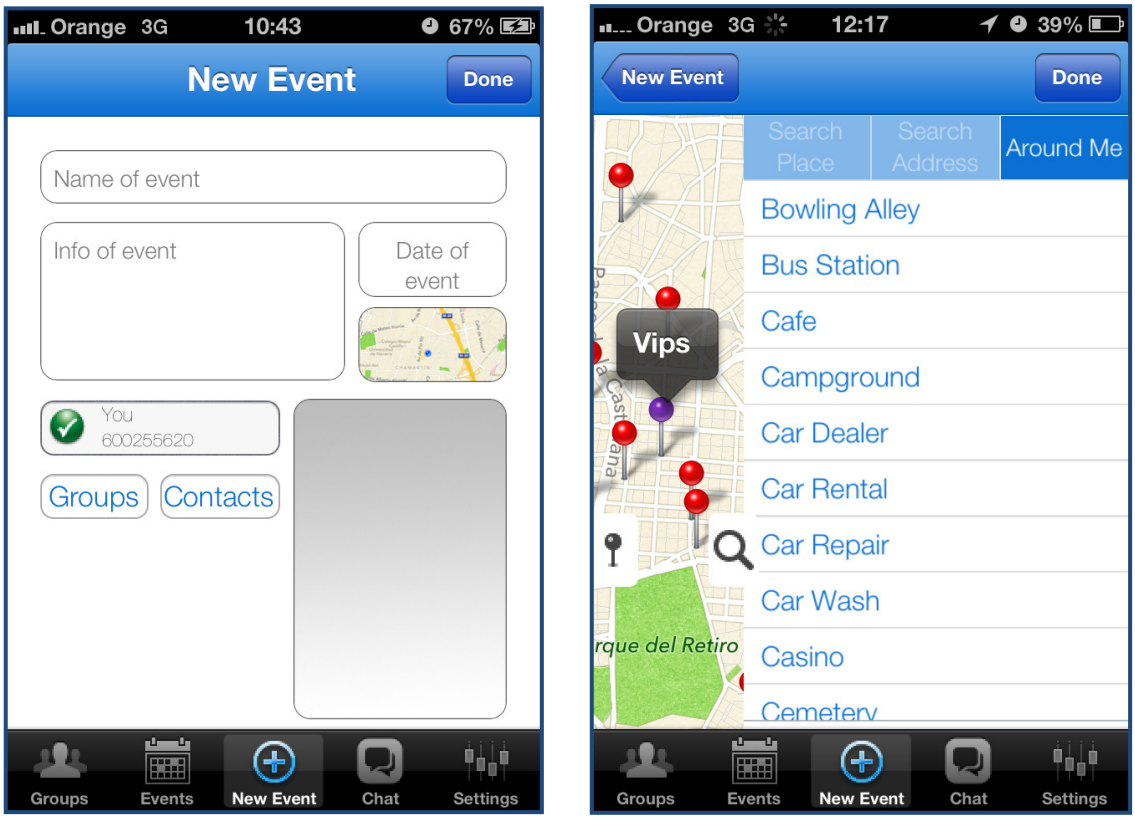


Figura 3.7 Vista New Event (izda.) y Search Place (dcha.)

3.3.4. *Chat*

Aunque la aplicación pretenda centrarse únicamente en los eventos, un complemento casi esencial es un medio con el que poder comunicarse todos los usuarios de la aplicación. De este modo, cualquier asunto referente a una reunión podrá ser comentada de una manera directa y dinámica. Si no se diera esta posibilidad, los usuarios solo podrían comunicarse a través de una aplicación externa, hecho que perjudicaría notablemente a la aplicación.

Es necesario recalcar que la aplicación únicamente permitirá la comunicación entre dos usuarios que tengan la aplicación, por lo que hará falta indicar al usuario cuáles de sus contactos podrán recibir mensajes y cuáles no. Para estos últimos, se dará la posibilidad de invitarles a usar la aplicación a través de un correo electrónico desde la pestaña de *New Event*. En la figura 3.8 se puede apreciar una pantalla con todas las conversaciones del usuario y otra dentro de un chat privado.

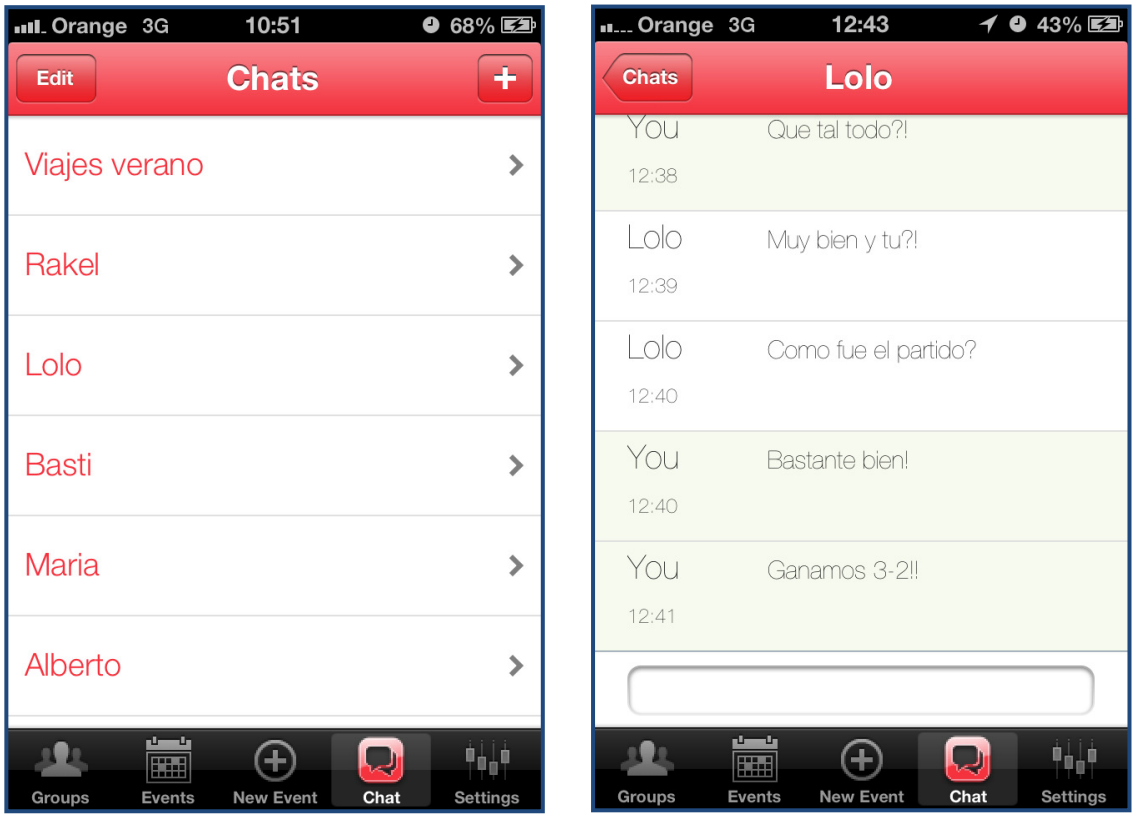


Figura 3.8 Vista Chats (izda.) Chatroom (dcha.)

3.3.5. Ajustes

Siempre es recomendable ofrecer al usuario un espacio en el que poder configurar diferentes parámetros de la aplicación acorde a sus gustos.

En este caso, los valores parametrizables tendrán que ver con el aspecto de la aplicación. El color de fondo o el tipo de transición por defecto de algunas vistas serán algunas de las opciones que se permitirán ajustar en este apartado.

Aprovechando este espacio, se le informará de la versión al usuario y se le dará la posibilidad al usuario de contactar con los desarrolladores para cualquier problema o sugerencia que pudiera tener. La figura 3.9 ofrece una parte de la vista de *Settings.*

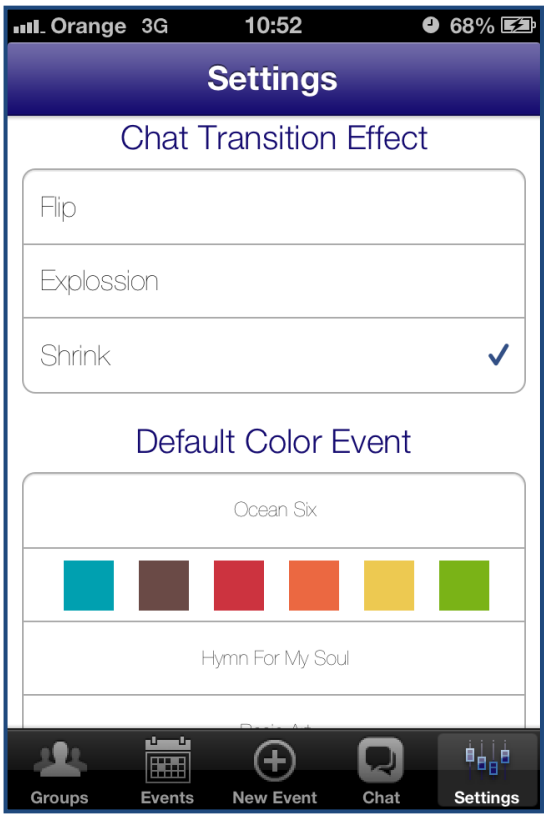


Figura 3.9 Vista Settings

3.4. Relación entre módulos

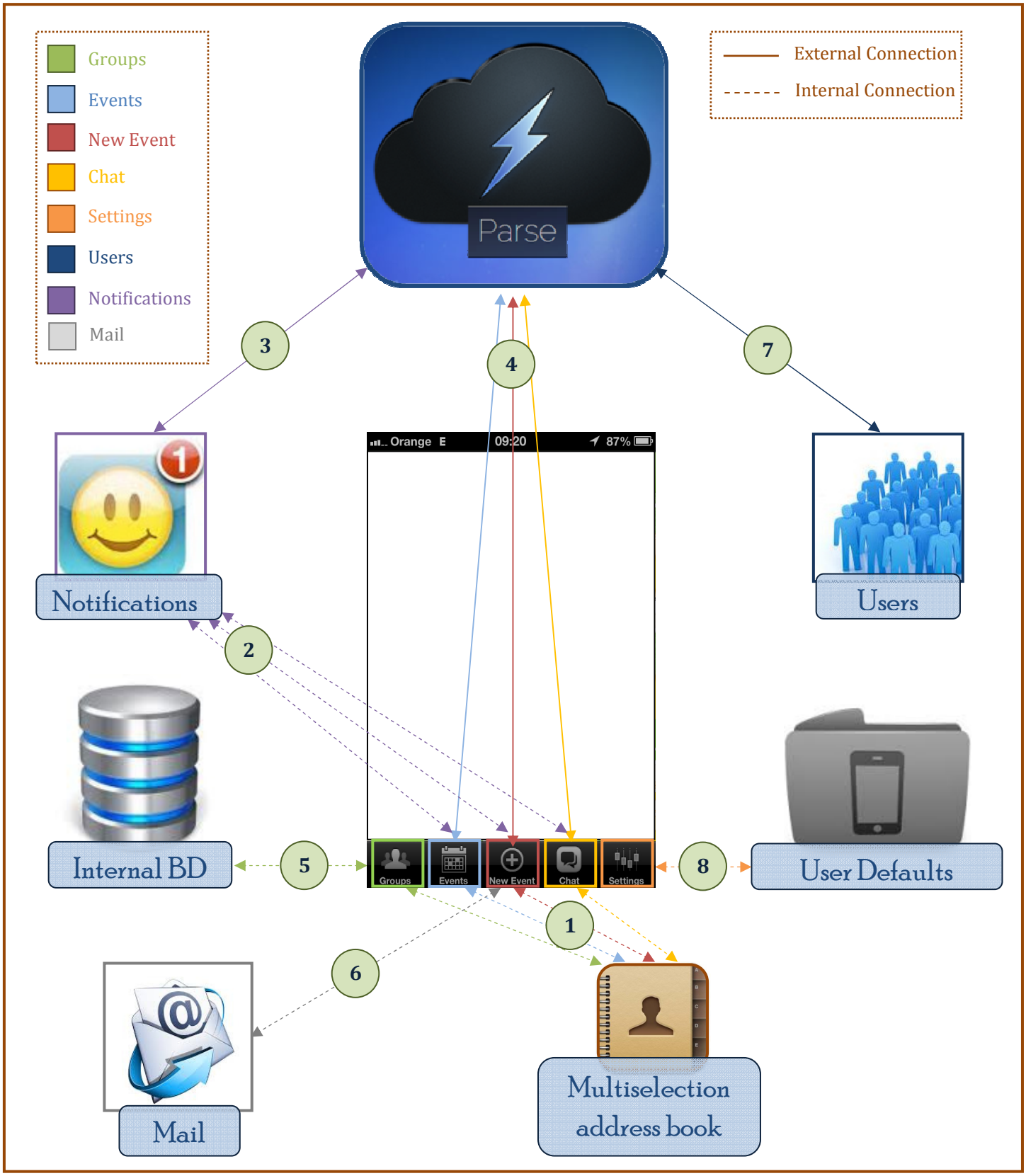


Figura 3.10 Relación intermodular de la aplicación

En el gráfico de la figura 3.10 se muestran los diferentes módulos y elementos que componen la aplicación junto con las relaciones que existen entre ellos numeradas. Cada uno de los cuadros de colores representa un módulo funcional independiente de la aplicación, mientras que los iconos muestran elementos que, sin llegar a ser entidades funcionalmente completas, son fundamentales para una correcta implementación.

Cinco de estos módulos son los ya explicados en el apartado anterior. Son los cinco pilares de la aplicación y conforman el núcleo de la misma. Aparte de éstos, existe un módulo encargado de las notificaciones enviadas por la aplicación, otro responsable del envío de mails a usuarios que no estén registrados y un tercero cuya función es gestionar todo lo relacionado con las incorporaciones y accesos de usuarios.

A continuación se va a explicar cómo se relacionan estos ocho módulos y el motivo de cada una de estas relaciones dibujadas en el gráfico anterior.

La relación más concurrente —número 1— es la que aparece entre los módulos principales —exceptuando el módulo *Settings*— y la agenda multiselección. Cada uno de los cuatro va a necesitar acceder a los contactos del terminal en el uso normal de la aplicación. Mientras el módulo de grupos lo hará para añadir o modificar usuarios de un grupo, los módulos encargados de los eventos recurrirán a la agenda para poder agregar a los invitados y cuando el usuario quiera crear una nueva conversación, podrá hacerlo escogiendo uno o más contactos gracias a esta agenda multiselección.

Cuando el usuario cree un evento nuevo o modifique alguno de los que ya tenía creados, la aplicación deberá registrar estos cambios en «la Nube» y notificar a todos los invitados estos cambios. Esto mismo ocurrirá cuando el usuario cree una conversación y comience a chatear con otros usuarios registrados. Por esto, los tres módulos tienen una relación con el módulo encargado de las notificaciones y otra con el icono que representa la plataforma *Parse*:

La primera de estas relaciones —número 2— indica la transmisión de datos que realizan los tres al módulo de notificaciones. En esos datos se le indica al módulo cual es la información que debe mostrar en esas notificaciones y a que usuarios va dirigida. El módulo de notificaciones, internamente, se encargará de tratar esa información y enviarla a un servidor externo que gestione esas notificaciones. En este caso también se trata de *Parse*, de ahí que aparezca también la relación externa —número 3— entre este módulo y el icono de la plataforma. Todo este proceso de notificaciones se encuentra desarrollado en el apartado 4.8.

La segunda relación —número 4— es directamente con *Parse.* Aparte de las notificaciones, la plataforma permite almacenar información a modo de servidor para que el resto de terminales —clientes— puedan acceder a ella cuando se precise. Por tanto, los tres módulos deberán hacer uso de la *API* de *Parse* para modificar, añadir, eliminar... cualquier información a la que vaya a acceder más de un usuario. La conexión entre los módulos y el icono representan este flujo de información.

En el caso de los grupos, puesto que se trata de una funcionalidad gestionada individualmente por cada usuario, el almacenamiento es algo distinto que en los casos anteriores. En este caso, el usuario va a poder crear grupos que le sirvan para agilizar el proceso de añadir contactos a eventos o crear grupos en el chat. Debido a su naturaleza, no tiene sentido que estos grupos se almacenen en un servidor remoto, ya que va a ser información a la que únicamente acceda el propio usuario. Por esto, el almacenamiento de los grupos va a realizarse en una base de datos interna, desarrollada expresamente para el proyecto, como muestra la relación —número 5— dibujada en el diagrama.

Cuando el usuario cree un nuevo evento, cabe la posibilidad de que haya invitado a contactos de su teléfono que no estén registrados. En este caso, la aplicación va a permitir que el usuario envíe un mail con la información fundamental del evento a todos estos contactos. De esto se va a encargar el módulo de emails. El módulo «*New Event*», cuando reciba que el usuario quiere mandar este email, mandará la información introducida al módulo «*Email»*, el cual se encargará de componer el mensaje con un formato legible y enviarlo a la dirección de correo recibida. Este traspaso de información es el que queda reflejado en el diagrama con la relación —número 6— entre ambos módulos.

Toda la gestión de usuarios va a estar integrada en *Parse*. Cuando un usuario se registre en la aplicación, ésta mandará todos los datos con una estructura especial para usuarios, para que sean almacenados. Del mismo modo, cuando algún usuario trate de acceder a su cuenta, la aplicación contrastará los datos introducidos con los guardados en *Parse* para permitir o rechazar el acceso. Todo este intercambio está plasmado en la relación —número 7— entre el módulo «*Users*» y el icono de *Parse.*

Una vez el usuario se haya registrado en el terminal, la aplicación automáticamente cargará los valores predeterminados del usuario. Estos van ser principalmente los valores que haya seleccionado en la pestaña *«Settings»* de la aplicación. Al igual que con los grupos, va a gestionarse individualmente para cada cuenta, por lo que esta información va a almacenarse internamente en el terminal. Sin embargo, en este caso no es necesario crear una base de datos adicional, ya que la *API* del entorno de desarrollo permite el acceso a una base de datos interna para valores por defecto de la aplicación. Esta base de datos no es válida para los grupos debido a su poca flexibilidad para relacionar distintas entidades, ya que está pensada para almacenar valores aislados, como se trata de los gestionados por la pestaña «*Settings*». Esta última relación —número 8— está dibujada en naranja entre la pestaña y el módulo «*NSUserDefaults»*.

4.Implementación

Como se ha visto en el anterior capítulo, esta aplicación puede separarse en cinco grandes funcionalidades, bien diferenciadas en las cinco pestañas que componen la barra de menú principal. Cada una de ellas está a su vez fraccionada en módulos independientes más específicos para ofrecer una base sólida y eficaz al conjunto de toda la aplicación. A continuación se va a hacer un estudio detallado de cada una de estas secciones, al igual de otras igualmente necesarias para un funcionamiento adecuado, pero que no pueden considerarse funcionalidades propias de la aplicación.

Una buena referencia a cada una de esas secciones es el diagrama de pantallas que se muestra en la figura 4.1.

Figura 4.1. Diagrama de pantallas

En este diagrama pueden observarse todas las vistas a las que el usuario va a poder acceder, al igual que el flujo que hay entre ellas. Cada funcionalidad se relaciona de un modo directo con cada una de las ramas del diagrama, como resaltan los cuadros.

4.1. Log In, Sign In & Forgot Password

HiUp es una aplicación social en la que se va a interactuar con otros usuarios. Por lo tanto, lo primero que debe hacer el usuario para poder utilizar la aplicación es identificarse en el sistema o registrarse en el caso que no lo hubiera hecho ya.

La gestión de los usuarios se va a realizar a través de la plataforma *Parse*. Esta plataforma proporciona altos niveles de seguridad tanto para la transmisión de las contraseñas —transmisión cifrada— como para su posterior almacenamiento, ya que ni el propio administrador de la aplicación puede tener acceso a ellas. De hecho, es la propia plataforma la que se encarga de la gestión de las contraseñas y proporciona una interfaz al programador para cuando el usuario deseara restablecer su contraseña.

Ambas acciones —registro e identificación— van a efectuarse desde la misma vista y para navegar entre las dos opciones, el usuario deberá deslizar un dedo sobre la pantalla, la cual irá mostrando un carrusel de imágenes con situaciones que podrían relacionarse con eventos o reuniones sociales.

4.1.1. Sign In

Muchas veces, un usuario se ve obligado a rellenar interminables formularios con datos personales que, en algunas ocasiones, ni siquiera tienen relevancia para el uso de la propia aplicación. Esto es totalmente contrario a una filosofía de simplicidad y transparencia. Por lo tanto, cuando el usuario desee registrarse, únicamente se le va a pedir la información básica necesaria para un correcto uso de la aplicación.

En el formulario de registro solo será necesario introducir un número de teléfono que servirá como nombre del usuario, una contraseña, la cual deberá repetir para asegurar que la ha escrito correctamente, y un email para poder restablecer la contraseña cuando el usuario la haya olvidado.

Como ya se ha comentado, *Parse* se encarga de la gestión de los usuarios, por lo que ofrece una clase especial para este tipo de entidades. La clase *PFUser*. La aplicación debe encargarse de crear un objeto de este tipo y rellenarlo con la información necesaria para identificar a cada usuario: nombre de usuario, email y contraseña.

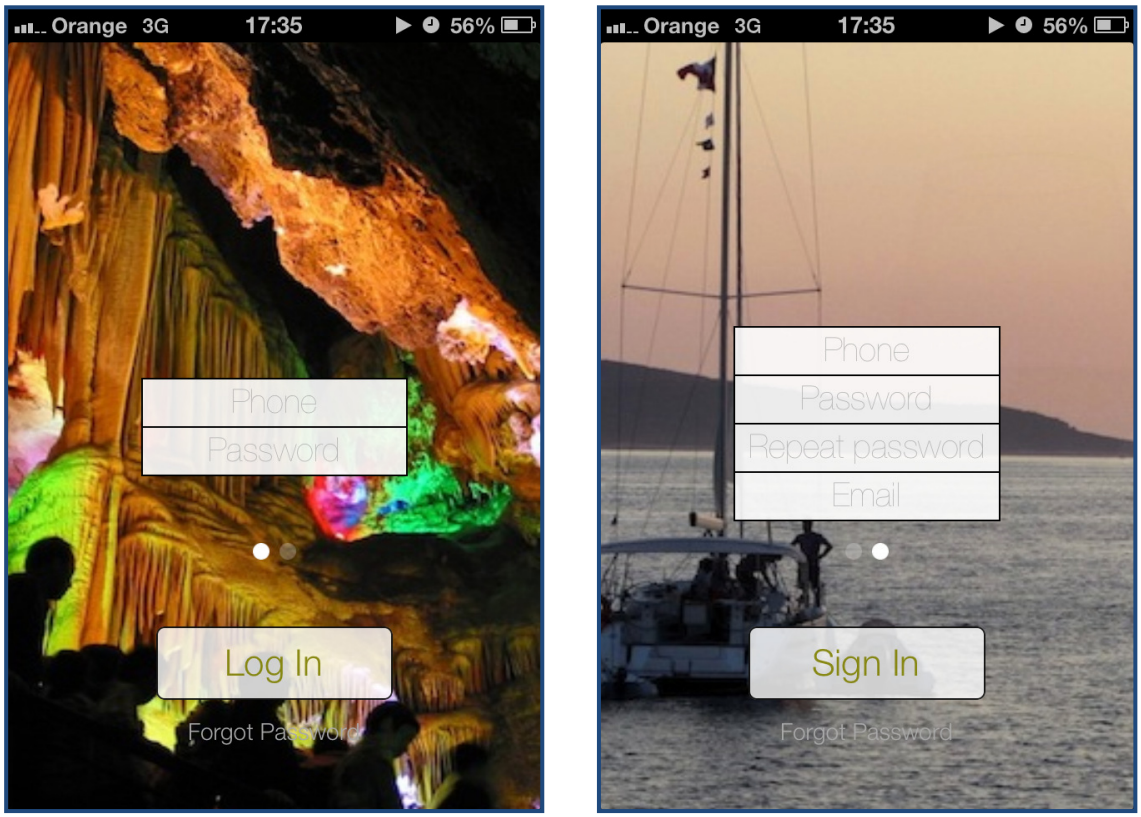
Con estos datos ya completados, la aplicación mandará toda esta información a la plataforma dentro de un objeto *PFUser*. *Parse* comprobará que todos los datos sean correctos y una vez hecha esta comprobación, se asegurará que el usuario no se encuentra ya registrado. Si así fuera, devolverá un mensaje de error que gestionará la propia aplicación, la cual informará que ya existe un usuario con ese nombre.

Durante este proceso de registro puede haber errores del usuario: algún campo se encuentra incompleto, las contraseñas no coinciden, existe algún error de conectividad… La aplicación está protegida frente a estas situaciones y mostrará un mensaje al usuario informando del problema ocurrido antes de enviar los datos. De este modo, se impide que la aplicación mande algún dato que provoque en *Parse* un error no controlado.

4.1.2. Forgot Password

Si el usuario no recordara su contraseña, pulsando sobre el mensaje «*Forgot Password*», la aplicación mostrará un mensaje pidiendo al usuario que escriba el correo electrónico con el que se registró. Una vez introducido correctamente, la aplicación mandará la cadena de texto introducida a través de un método de la *API* proporcionada por *Parse*. Es la propia plataforma quién gestiona internamente todo el proceso para recuperar la contraseña y manda un correo electrónico al mail del usuario. En este correo se le mostrará una página web desde la cual podrá restablecer la contraseña. Con la contraseña ya restablecida, sólo deberá introducirla junto al mail correspondiente en la zona de *Login* para acceder a la aplicación.

En la figura 4.2 pueden observarse las tres pantallas descritas anteriormente.



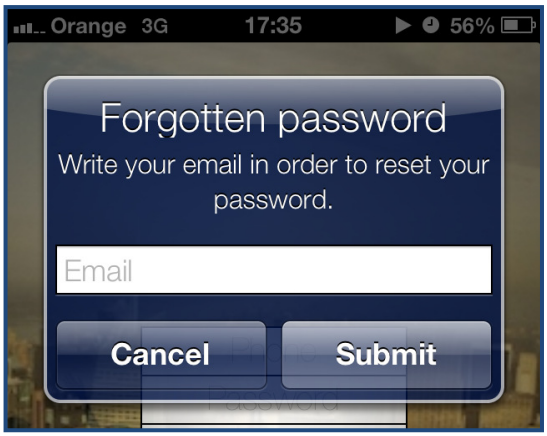


Figura 4.1 Pantalla de Log In (página anterior izda.), Sign In (página anterior dcha.) y mensaje para Forgot Password (página actual)

4.2. New Event

Cuando el usuario ya identificado acceda a la aplicación, la primera pantalla que se muestre será la de creación de un nuevo evento. Puesto que se trata de la base de toda la aplicación, va a situarse en la pestaña central para resaltar su importancia.

En esta pantalla podrá determinarse el nombre del evento, una información que detalle el motivo por el que se organiza y cuándo va a tener lugar. Estos tres campos son requisitos mínimos para poder organizar de un modo adecuado un evento, por lo que si el usuario tratara de crearlo con alguno de estos tres campos incompletos, la aplicación mostraría un mensaje de aviso y requeriría que se rellenara esta información. Estos tres campos van a almacenarse en variables adecuadas para cada uno de ellos. Así, mientras el nombre y la información pueden guardarse en clases de tipo *NSString*, la fecha tiene una clase distinta; *NSDate*. Este tipo de dato permite acciones como comparar fechas o dar distinto formato a la información almacenada. *Parse* no pone ninguna restricción para almacenar una variable del tipo *NSDate*, por lo que todas las fechas gestionadas por la aplicación van a tratarse a través de este tipo de datos. La figura 4.3 muestra la interfaz inicial y uno de los mensajes de error.

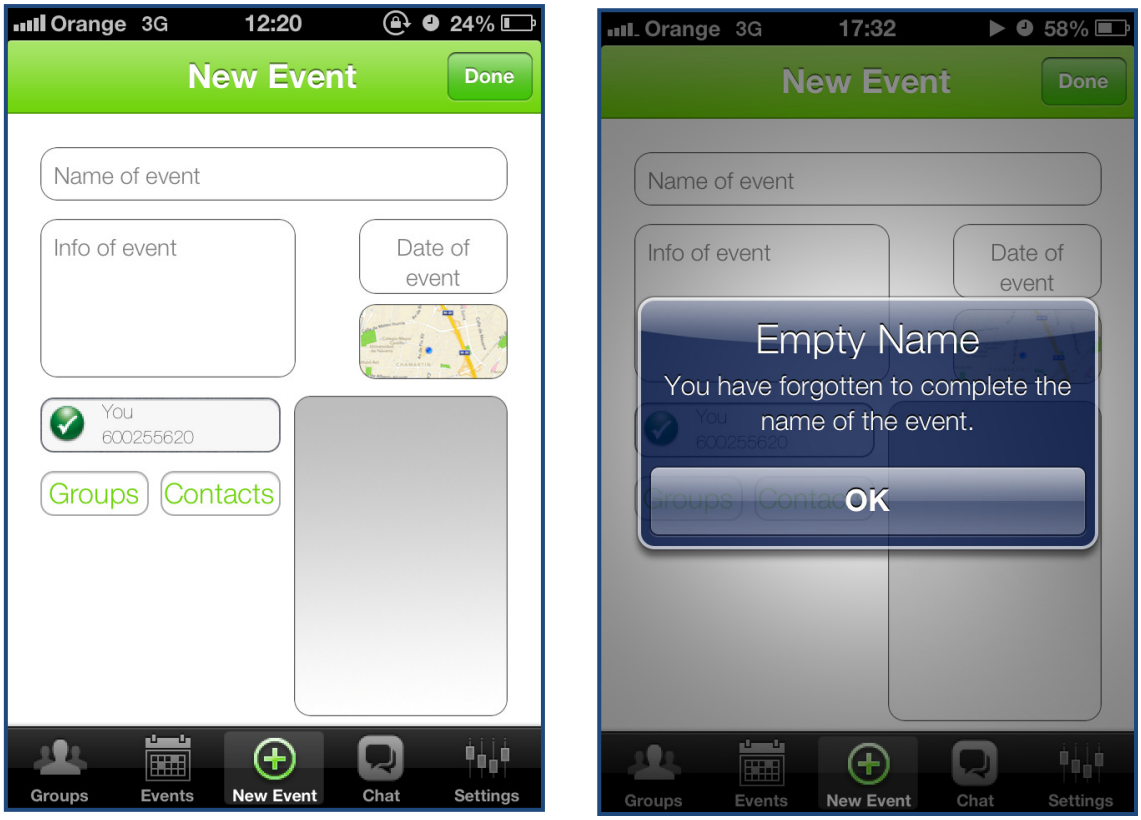


Figura 4.2 Vista principal de New Event (izda.) y mensaje de campo incompleto (dcha.)

4.2.1. Contacts

Tras completar estos tres campos, el creador podrá añadir a las personas que quiera invitar al evento. Por norma general, el creador debería acudir al evento, por lo que la aplicación añadirá automáticamente al usuario dentro del grupo de personas invitadas. Para añadir al resto, existen dos modos distintos de hacerlo:

- El primero es pulsando el botón "*Contacts*". Se mostrará una ventana con todos los contactos almacenados en la agenda En este caso, la aplicación no va a mostrar directamente la lista de contactos nativa de iPhone, ya que ésta no permitía realizar una selección de varios usuarios al mismo tiempo. En cambio, se ha integrado en el proyecto una agenda que permite hacer multiselección, obteniendo los contactos a través de la *API* para acceder a los contactos, proporcionada por el entorno de desarrollo. Puesto que esta agenda va a ser usada por muchos módulos de la aplicación, se le dedicará un apartado más adelante dónde se explicará en profundidad.

- La segunda manera de añadir usuarios es aprovechando la opción de grupos incluida en la aplicación. Si el usuario tuviera algún grupo de personas ya creado, a través del botón "*Groups*" accederá a una ventana donde aparecerán todos los almacenados. Pulsando en cualquiera de ellos, la aplicación añadirá simultáneamente a todos los usuarios del grupo después de mostrarlos. La funcionalidad de *Groups* será descrita con mayor detalle más adelante.

En ambos casos se ha tratado de priorizar la rapidez a la hora de incluir los contactos para conseguir una mejor impresión del usuario.

Cuando los contactos ya estén añadidos, aparecerán en una tabla en la parte inferior izquierda de la pantalla. Puesto que el número de invitados puede variar mucho, se ha optado por implementar una tabla que vaya creciendo dinámicamente con el número de usuarios. Si la tabla superara el límite de la pantalla, la interfaz se ajustaría a la misma permitiendo hacer *scroll* y desplazando los botones consecuentemente para que se mantengan siempre visibles.

En cada una de las celdas aparecerá una imagen indicativa que muestra si el contacto se encuentra registrado o no en la aplicación. Esta información se obtiene desde la plataforma *Parse* cuando se ejecuta la aplicación. Al comenzar, se pide a la plataforma todos los usuarios que se encuentran registrados en la aplicación. Cuando recibe esta petición, la plataforma devuelve en un *array* de objetos *PFUser*, todos los usuarios que tienen una cuenta en la aplicación.

Gestionarlo de esta forma tiene el inconveniente de que haya algún registro nuevo desde que se abrió la aplicación, ya que el usuario no podrá verlo hasta que la vuelva a ejecutar. Sin embargo, esta decisión proporciona grandes ventajas tanto para el usuario como para la aplicación.

Por el lado del usuario, se evita la ralentización que se produce cada vez que se pide la información de usuarios registrados. Y por el lado de la aplicación, se reduce abismalmente el número de consultas al servidor en la nube. Además, esta última ventaja también lo es para el usuario, puesto que reduce considerablemente la cantidad de datos que la aplicación va a consumir.

Si alguno de los usuarios invitados al evento no estuviera registrado, la aplicación informará que no van a quedar vinculados al evento y ofrecerá la posibilidad de mandar un mail automático con la información básica a cada uno de ellos.

En la figura 4.4 se puede ver la tabla extendida más allá del final de la pantalla y el mensaje de error para usuarios no registrados.

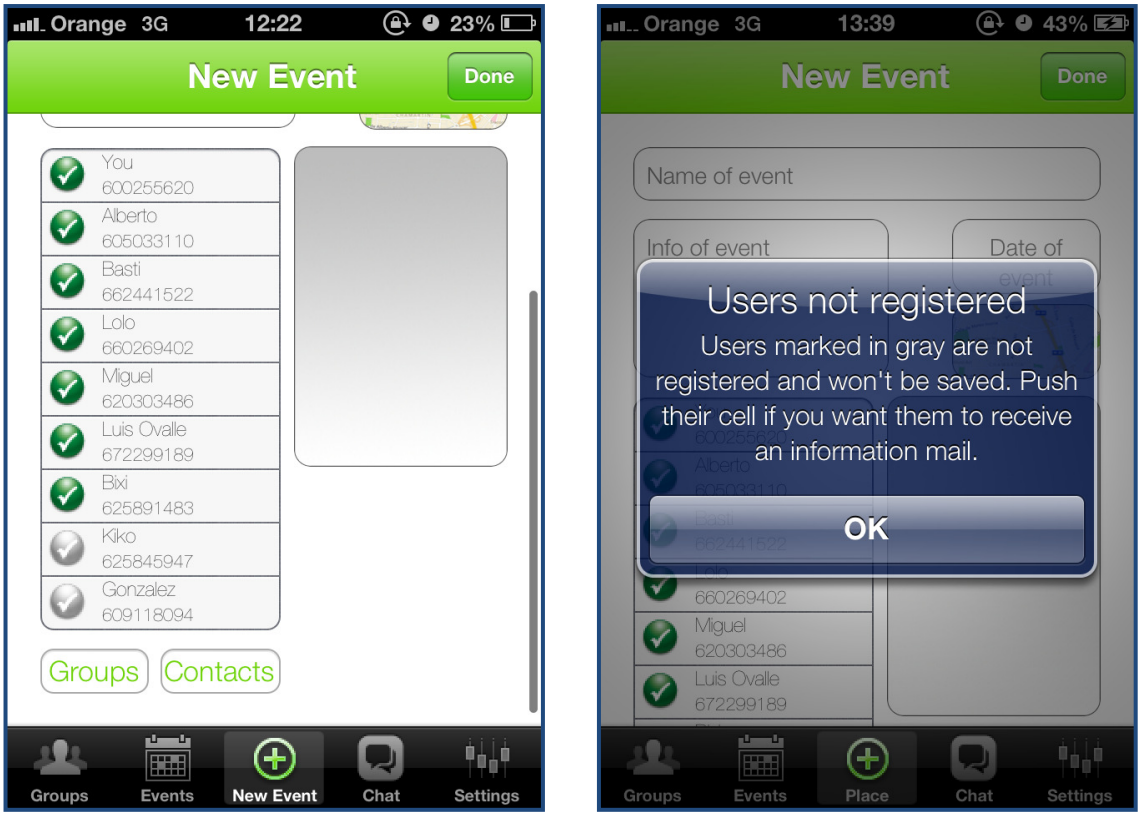


Figura 4.3 Interfaz de New Event flexible a número de usuarios (izda.) y mensaje de usuarios no registrados (dcha.)

Para mandar este email sólo tendrá que pulsar la celda del usuario no invitado y la aplicación mostrará un mensaje pidiendo la dirección de correo electrónico a la que se mandará toda la información del evento. Para ello, los campos obligatorios del evento deben estar ya rellenos.

Por último, si el usuario quisiera eliminar a alguien de la lista de invitados, con un simple deslizamiento en la celda que corresponda, aparecerá un botón para poder eliminarlo. Tanto la tabla como la interfaz se reordenarán para ajustarse al nuevo tamaño. En la figura 4.5 se muestra la interfaz para la eliminación de un invitado y el envío del mail a un usuario no registrado.

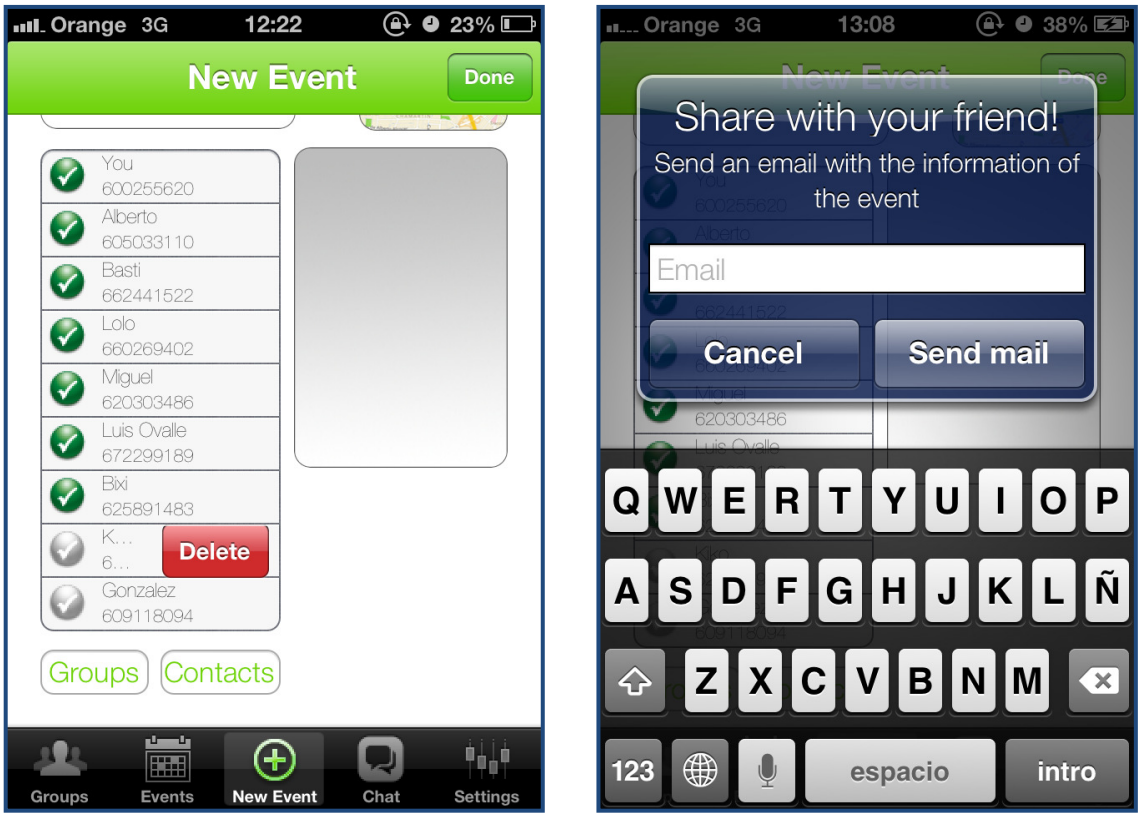


Figura 4.4 Interfaz para eliminación de usuarios de la tabla de invitados (izda.) y mensaje para envío de información del evento a usuarios no registrados (dcha.)

4.2.2. Event Image

Anexo a la tabla de usuarios se encuentra un espacio con fondo gris en el que podrá mostrarse una imagen descriptiva del evento. Pulsando sobre este espacio, se podrá escoger cualquier foto que el usuario tenga ya almacenada. La interfaz para seleccionar imágenes la proporciona directamente una *API* del entorno de desarrollo permitiendo acceder a la galería de fotos del terminal. Una vez abierta, el usuario podrá seleccionar cualquiera pulsando sobre ella. Al pulsar, la *API* retorna una clase del tipo *UIImage*, la cual contiene ya la imagen. El tipo de dato *UIImage*, aparte de almacenar la imagen, provee, entre otros, métodos para escalarla o variar su calidad para aumentar o disminuir el tamaño de la misma.

Parse no permite almacenar variables del tipo *UIImage* directamente, por lo que será necesario convertirlo al tipo *NSData*, que sí que es compatible con la plataforma. *NSData* se trata de un tipo de dato que encapsula distintos objetos que tienen una estructura definida, en un *buffer* de datos continuo sin ninguna estructura concreta. Todas las imágenes que vayan a gestionarse desde la aplicación serán encapsuladas antes en una variable *NSData*, para poder almacenarlo en el servidor remoto.

Es necesario destacar que toda la aplicación esta optimizada para aquellas fotos realizadas con el teléfono en posición vertical, por lo que si la foto no se ajustara a las proporciones de la pantalla del *iPhone* sufriría una distorsión final. Este espacio puede observarse en cualquiera de las figuras mostradas anteriormente.

4.2.3. Places

Lo último que se debe especificar es el lugar donde se va a celebrar la reunión. Puesto que el modelo de negocio se basa en obtener beneficios de los lugares donde se organicen las reuniones, se ha implementado un sistema de búsqueda bastante completo para este último campo.

Este sistema se basa fundamentalmente en cadenas de búsqueda relacionadas con los lugares que se desea encontrar y la geolocalización del usuario en el momento de realizarla.

Todos los lugares y direcciones que se van a obtener proceden de una base de datos que *Google* pone a disposición de desarrolladores de aplicaciones móviles y webs y que es accesible a partir de cualquier dispositivo con una conexión a Internet. Para obtener la información es necesario formular una petición por medio de una *URL* con un formato concreto, que dependerá del tipo de búsqueda que se vaya a hacer.

Cada vez que el usuario quiera realizar una búsqueda, la aplicación creará una clase del tipo *NSURL*, uniendo coherentemente el formato correspondiente para la petición y la cadena de búsqueda introducida por el usuario. *NSURL* es una clase que contiene una dirección web y permite gestionar los datos de respuesta que ofrece esa dirección. Así, cuando el usuario realice alguna consulta, la aplicación, a través de este objeto, recogerá la información devuelta y la gestionará internamente para poder presentarla al usuario una vez formateada del modo adecuado.

El servicio gratuito del servidor de *Google* tiene un límite de 1.000 consultas diarias que puede aumentarse a 100.000 si el desarrollador se identifica con una tarjeta de crédito, sin necesidad de hacer ningún cargo en ella. Este límite es más que suficiente para el propósito de este proyecto.

Para poder acceder a la pantalla de búsqueda, el usuario debe pulsar el icono del mapa en la pestaña de *New Event*, que abrirá una pantalla en la que se mostrará la posición actual del usuario dentro de un mapa. Esta vista contiene dos botones en ambos laterales. Una desplegará una tabla dónde se mostrarán los resultados de la búsqueda, mientras que la otra permitirá introducir los parámetros de búsqueda. La figura 4.6 muestra esta vista principal con ambos botones laterales.

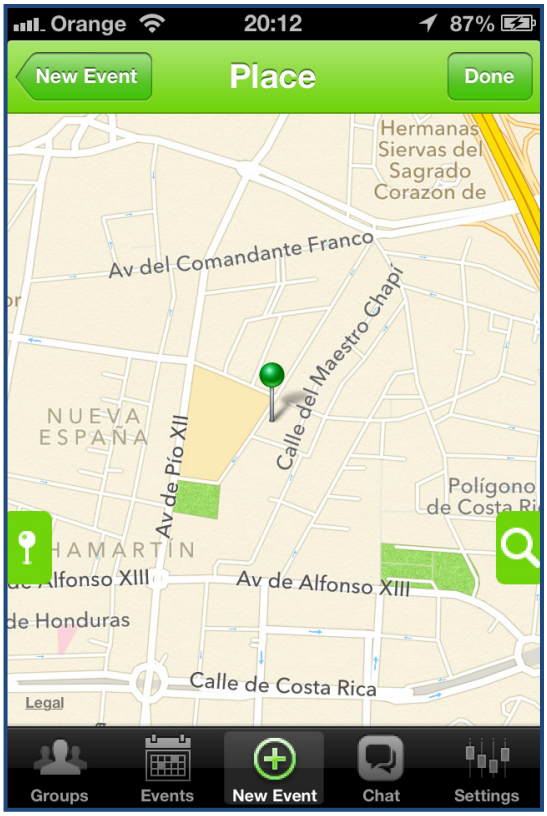


Figura 4.5 Vista principal de interfaz Maps

Si el usuario pulsa el icono de la lupa, el mapa se desplazará hacia la izquierda dejando a la vista la pantalla dónde se podrán realizar las búsquedas. Estas búsquedas podrán ser de tres tipos: *Search Place*, *Search Address* y *Around Me*.

El protocolo para realizar la búsqueda es muy parecido en los tres casos y lo único que cambia es el formato de la *URL* que va a ser tratada. Cada formato será especificado para cada uno de los tipos de búsqueda.

Una vez enviada la solicitud a través de la *URL* correcta, el servidor de *Google*, devolverá los datos requeridos en un formato *JSON*. Éstos serán extraídos y almacenados en un objeto del tipo *NSDictionary* por medio de un procesador de lenguaje *JSON*. La clase *NSDictionary* es una estructura de datos que asocia una clave a un valor almacenado. Así, si se quisiera acceder por ejemplo a la localización del lugar, podría hacerse obteniendo al dato asociado a la clave «*location*».

Con todos los lugares almacenados en su correspondiente estructura *NSDictionary*, se creará un *array* que servirá tanto para localizar todos los lugares en un mapa como para presentar la información en una tabla ordenada. Esta tabla será explicada con más detalle en un apartado más adelante.

Para la gestión de mapas dentro de la aplicación, existe una *API* del entorno de desarrollo llamada *UIMapKit* que se encarga de ubicar todos los puntos que reciba sobre un mapa integrado en la aplicación. Únicamente es necesario indicar la latitud y la longitud del lugar y configurar algunos parámetros para los indicadores dibujados en el mapa.

Esta *API* también se encarga del tratamiento del zoom y el recentrado en el mapa, al igual que de mostrar una información básica cuando se pulsa en cada uno de los iconos dibujados.

Search Place

Con esta primera opción se pretende facilitar al usuario al máximo la búsqueda de un lugar genérico, permitiendo buscarlo por cualquier información que tenga relación con él. Esta información puede ser desde el nombre del lugar, el tipo de local, su dirección, un código postal… Del mismo modo, pueden combinarse varios de estos datos para obtener lugares más precisos. Algunos ejemplos podrían ser desde un lugar tan concreto como «Jardín Botánico de Valencia» hasta uno tan genérico como «Peluquería 28016 Madrid». En esta primera búsqueda, el formato necesario para poder hacer la petición al servidor de Google es el siguiente:

https://maps.googleapis.com/maps/api/place/textsearch/json?parameters

En el campo «*parameters*» se incluirá, entre otros, la cadena de búsqueda que el usuario ha introducido. La figura 4.7 muestra el resultado de la búsqueda «Peluquería 28016 Madrid».

Figura 4.6 Resultado de búsqueda «Peluquería 28016 Madrid» en Search Place

Search Address

En muchos casos, las reuniones puede que no se organicen en un local público o que la búsqueda *Search Place* no de ningún resultado con la información introducida. Para estos casos, la aplicación permite al usuario buscar una dirección exacta. Dependiendo de lo concreta que sea la cadena de búsqueda, *Google* devolverá más o menos coincidencias. Así, si se buscara «Calle Alcalá» se obtendrían más de 40 opciones que coinciden con esa descripción, mientras que si se especifica más con una cadena como «Calle Alcalá 142, Madrid», el resultado es el mostrado en la figura 4.8.

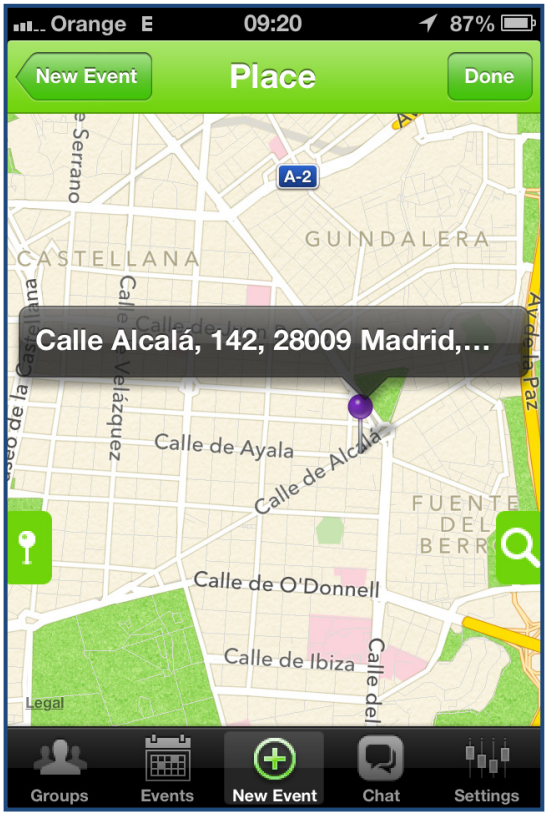


Figura 4.7 Resultado de búsqueda «Calle Alcalá 142, Madrid» en Search Address

Para esta segunda opción, el formato es algo distinto puesto que se trata de la búsqueda de una dirección concreta. Para ello, la *URL* debe mantener el siguiente formato:

http://maps.googleapis.com/maps/api/directions/json?parameters

De nuevo, en el campo «*parameters*» se incluirá como uno de los datos, la dirección que el usuario esté buscando.

Around me

Por último, el usuario podrá buscar lugares que se encuentren alrededor de él. Para poder definir un poco esta búsqueda, la aplicación ofrece una gran variedad de categorías que acotarán los resultados obtenidos. Todas estas categorías también son proporcionadas por *Google*.

El radio de búsqueda alrededor del usuario está definido por el zoom que exista en el mapa. La aplicación siempre va a sondear en la región que se encuentre visible en el momento de hacer la búsqueda. Si el usuario quisiera ver resultados más allá de esa superficie, solo necesitaría reducir el zoom y volver a realizar la misma búsqueda para obtener más opciones.

Para el último tipo de búsqueda, el formato es muy parecido al primer caso, dónde se hacía una búsqueda por lugar:

```
https://maps.googleapis.com/maps/api/place/nearbysearch/output?parameters
```

La diferencia principal está otra vez en el campo «*parameters*». En este campo, aparte de la categoría que el usuario ha seleccionado, es importante indicar la localización desde donde se está realizando la búsqueda y el radio de alcance de la misma. Todos estos datos serán introducidos a través de parámetros en la URL dentro del campo correspondiente. En la figura 4.9 se muestra el resultado de una búsqueda de cajeros automáticos en una región de la zona norte de Madrid, con dos zooms distintos.

Figura 4.8 Resultado de búsqueda de Cajeros Automáticos en Around Me con distintos radios de búsqueda

Tabla de resultados

Una vez obtenido los resultados de la búsqueda, éstos deben ser mostrados en el mapa para que el usuario pueda escoger aquel en el que está interesado. Sin embargo, en muchas ocasiones estos resultados pueden estar muy alejados unos de otros, por lo que van a concentrarse todos en una lista que se mostrará en el lado izquierdo de la pantalla.

Esta lista estará ordenada siempre según el criterio *Rating* aportado por *Google.* Este criterio se basa en las valoraciones que los usuarios del buscador más conocido han hecho sobre los lugares encontrados. De este modo, el primer lugar lo ocupará el resultado que mejor referencia tenga según este parámetro.

Para crear esta lista se va a recurrir a un elemento gráfico del tipo *UITableView*, y el *array* de lugares obtenidos con cualquiera de las tres opciones explicadas en el apartado anterior.

Puesto que el plan de negocio se basa en obtener ingresos a partir de los locales, este criterio de ordenación podría ser modificado más adelante para dar mayor notoriedad a aquellos resultados que proporcionaran beneficios a la empresa, incluyendo anuncios especiales en la propia tabla. De esta forma, estos locales aparecerían siempre a la vista del usuario y motivarían su elección. Del mismo modo, en futuras actualizaciones podrían introducirse nuevos métodos de ordenación según otros criterios.

Para evitar saturar con demasiada información al usuario, la consulta únicamente recogerá los veinte lugares con mejor valoración. Si hubiera más resultados y el usuario quisiera verlos, podrá hacerlo pulsando la celda «*More Places…*». Se mostrarán los siguientes veinte resultados y así sucesivamente hasta que no existan más coincidencias.

Para obtener estos resultados siguientes del servidor de *Google* es necesario utilizar un parámetro opcional que en las búsquedas iniciales no era necesario. Se trata del parámetro «*pagetoken*».

En la respuesta a una consulta, el servidor de *Google* proporciona un dato llamado «*pagetoken».* Se trata de una cadena de caracteres única que identifica la búsqueda y permite continuar con los siguientes resultados en caso de que el desarrollador lo necesite. Para ello, se debe volver a mandar la misma *URL* de búsqueda, pero añadiendo como parámetro opcional el *token* almacenado de la consulta anterior. De este modo, el servidor de *Google* reconocerá que se trata de una continuación de la búsqueda previa y devolverá los resultados inmediatamente siguientes. Este proceso puede repetirse hasta que el servidor devuelva un *token* nulo, hecho que indicará que no hay más opciones coincidentes.

Una vez mostrados los resultados, si el usuario pulsa sobre cualquiera de ellos, la aplicación mostrará la localización del mismo en el mapa junto con una breve información del mismo, resaltándolo frente al resto de opciones. La figura 4.10 muestra esta tabla de resultados con la búsqueda «NH Hotel, Madrid» en la opción *Search Place* y con la búsqueda «Calle Goya, Madrid» en la opción *Search Address*.

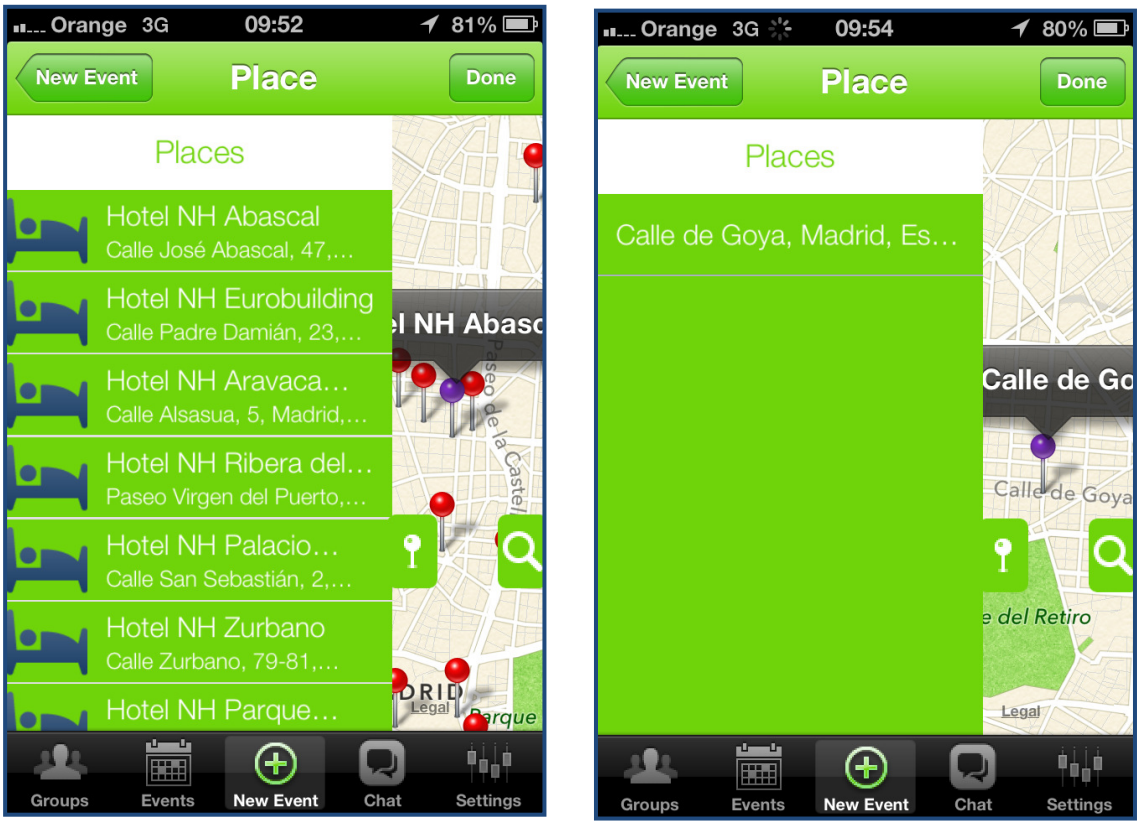


Figura 4.9 Tabla de resultados para búsqueda «NH Hotel Madrid» (izda.) y «Calle Goya, Madrid» (dcha.)

Cuando el usuario haya escogido el lugar, podrá volver a la pantalla *New Event* pulsando el botón «*Done*». La aplicación mostrará la información del lugar elegido y si todo está correcto, se seleccionará la entrada correspondiente del *array* de lugares y se guardará en una variable interna del evento. Puesto que esta entrada del *array* se trata de un objeto *NSDictionary* y *Parse* es totalmente compatible con este tipo de entidades, no será necesario realizar ninguna conversión para poder almacenarlo en la plataforma.

En la figura 4.11 se muestran la interfaz una vez seleccionado un sitio y el mensaje de confirmación al usuario.

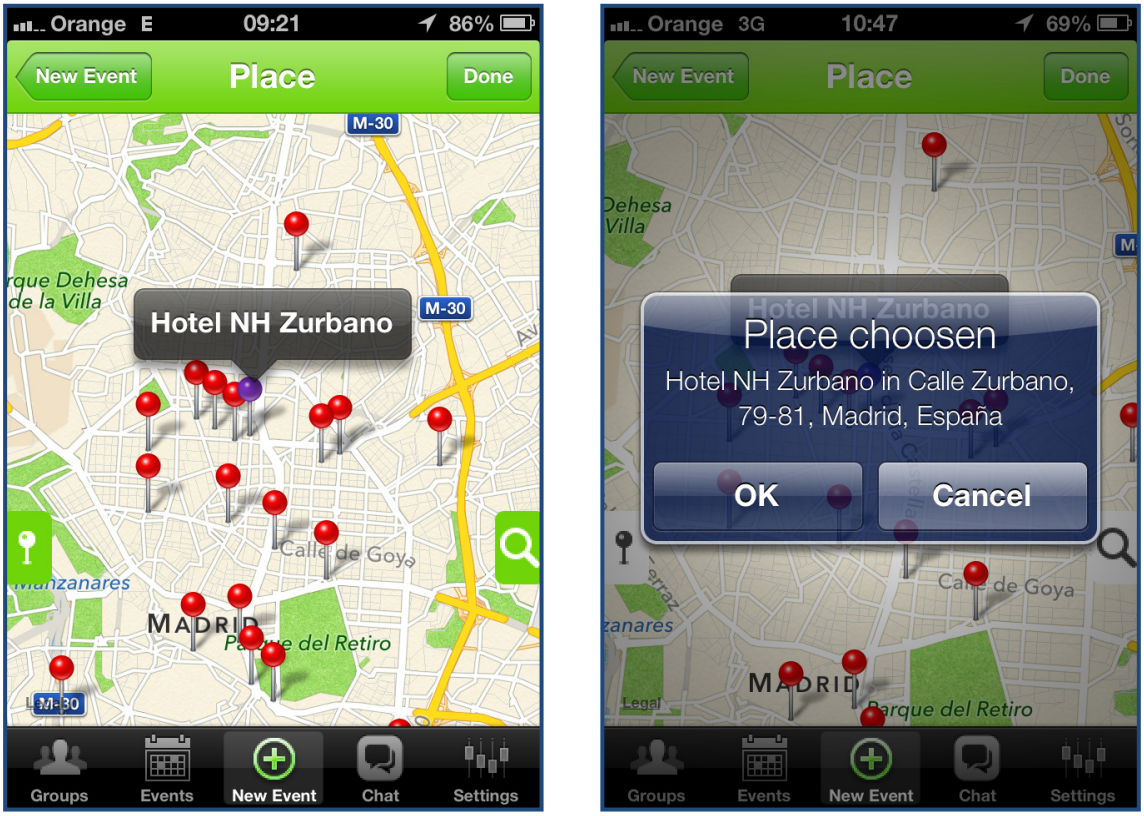


Figura 4.10 Pantalla con el lugar de reunión seleccionado (izda.) y mensaje de confirmación (dcha.)

4.2.4. Registro del evento

Si todos los datos están rellenados correctamente, pulsando de nuevo el botón «*Done*», la aplicación comenzará el proceso para registrar el evento en *Parse*.

Igual que la plataforma provee una clase muy concreta para los usuarios, como es *PFUser,* para el caso de objetos más genéricos existe una clase mucho menos restrictiva donde puede almacenarse cualquier tipo de dato que sea compatible con *Parse.* Se trata de la clase *PFObject.*

La clase *PFObject* es una entidad que no posee un esquema definido y que provee la *API* de *Parse*. El hecho de no tener un esquema fijo permite almacenar cualquier dato en ella a través de un sistema clave-valor, como si de un *NSDictionary* se tratara.

Puesto que todas las clases van a ser del tipo *PFObject*, hace falta diferenciar cada una de las clases. Para ello, en el método constructor del objeto se va a indicar el nombre con el que se identificará cada clase como un parámetro del método. En el caso un nuevo evento, la aplicación creará una clase con el parámetro «*Event* ».

Con el objeto *«newEvent»* ya creado, la *API* de *Parse* provee métodos accesores para poder escribir y leer información de él. En este caso, se almacenará todos los datos que el usuario ha introducido para generarlo. Recopilando toda la información del apartado, un evento será generado con:

- El nombre del evento, almacenando en una clase del tipo *NSString* con la clave «*nameOfEvent»*.
- La información del evento, almacenada en una clase del tipo *NSString* con la clave «*infoOfEvent»*.
- La fecha en la que se va a organizar el evento, almacenada en una clase del tipo *NSDate* con la clave «*dateOfEvent*».
- La imagen del evento, encapsulada en una clase del tipo *NSData* con la clave «*imageOfEvent*».
- La información del lugar dónde va a realizarse el evento, almacenada en una clase del tipo *NSDictionary* con la clave «*eventPlace*».

Todas estas variables van a almacenarse dentro del objeto creado como si de un *NSDictionary* se tratara. Se añade al objeto de la clase *PFObject* que se quiera almacenar cada uno de los anteriores valores. Estos valores se asocian a una clave —del tipo *NSString*— para luego poder recuperarlo correctamente.

Aparte de almacenar objetos directamente, para casi cualquier aplicación con un cierto desarrollo existe la necesidad de hacer una relación entre objetos de distintas clases. En el caso de *Parse*, esta relación se va a hacer tratando a la propia relación como otra clase independiente, llamada *PFRelation.* Un objeto de este tipo es el utilizado por ejemplo, para añadir a todos los usuarios invitados al evento. La relación muchos-a-muchos se modela del siguiente modo.

Una vez que se tiene creado el objeto «*newEvent*», se creará otro objeto del tipo *PFRelation* —*«newRelation»*—, que se igualará a una relación asociada a este evento. Si la relación ya existiera, se usaría «*newRelation»* para modificar la relación entre ambos objetos. Si por el contrario, esa relación no existiera, *Parse* internamente creará una relación automáticamente. A esta relación es necesario añadirle un parámetro —*key*— del tipo *NSString*, para poder diferenciarla, en el caso de que hubiera más de una relación.

Ya creada la relación, es necesario añadirle los usuarios. Esto se puede conseguir iterando en un bucle una acción de inserción en el objeto *«newRelation»* con cada uno de los objetos *PFUser* del *array* «*invitedUsers»*. De este modo, el objeto «*newEvent*» ya tiene almacenada una relación con todos los usuarios invitados que estuvieran registrados.

Si lo que se pretende es hacer una relación uno-a-muchos, la *API* de *Parse* permite conseguir esto en con una única acción. En el caso de un nuevo evento, existe un campo que tiene este tipo de relación. Cuando más adelante se traten los permisos de edición de un evento, será necesario conocer quién es el creador del evento. Por lo tanto, este campo debe ser relleno con un objeto del tipo *PFUser* que identifique al creador.

Para ello, *Parse* ha sobrecargado el método que almacena información en un *PFObject* para poder introducir, aparte de clases del entorno de desarrollo, clases propias de la *API* de la plataforma. Gracias a esto, se puede añadir esta relación del mismo modo que si se añadiera el nombre del evento o la fecha. En este ejemplo, el objeto a almacenar será el que devuelve el método que obtiene el usuario registrado en la aplicación y la clave será *«creatorOfEvent»*.

Por último, el objeto «*newEvent*» también va a almacenar un *array* de clases *NSString* con un formato especial para gestionar las notificaciones. Puesto que este *array* es del tipo *NSMutableArray*, una clase compatible con *Parse*, podrá guardarse del mismo modo que las anteriores variables.

Con todos los datos ya encapsulados dentro del objeto «*newEvent»*, pulsando el botón «*Done*» la aplicación registrará el evento con un método que establecerá la conexión entre *Parse* y el dispositivo. Este método posee dos parámetros que serán rellenados por la plataforma. El primero de ellos es un valor que indica si ha habido éxito o no en el registro de los datos. El segundo es del tipo *NSError* y da una información detallada del error ocurrido si la transacción no se hubiera efectuado correctamente. Es responsabilidad del programador recoger ese error e informar al usuario si fuera pertinente.

Durante el desarrollo de la aplicación, se efectuará en diversas ocasiones el registro de datos en la plataforma. Este registro se va a realizar según se ha explicado en el

apartado, por lo que en futuras entradas de la memoria dónde se mencione, únicamente se hará referencia a este procedimiento.

Si todo ha salido según lo esperado, la aplicación informará al usuario de que el evento ha sido registrado y mostrará automáticamente la pantalla de eventos registrados. Aparte, la aplicación notificará a todos aquellos usuarios registrados que tienen un nuevo evento. La manera de hacerlo será explicada más adelante.

4.3. Events

En esta sección el usuario va a poder consultar todos los eventos a los que está invitado, al igual que modificarlos cuando tenga permisos para hacerlo y rechazar aquellos que ya no le interesen.

Para la interfaz gráfica de esta pestaña se ha recurrido a la sencillez y limpieza que ofrece una vista del tipo *UITableView*, aunque se van a añadir ciertas funcionalidades y elementos gráficos con los que potenciar la funcionalidad sin llegar a sobrecargar visualmente la pantalla.

Cuando el usuario entre en la pestaña «*Events*», la aplicación automáticamente hará una consulta a la plataforma *Parse* pidiendo todos los eventos a los que el usuario registrado está invitado. Para ello, hace falta recurrir al objeto *PFQuery*, definido en la *API* de *Parse.* Este objeto permite realizar llamadas a la plataforma pidiendo una información de la base de datos almacenada. Esta consulta puede precisarse gracias a distintas condiciones que acotan la búsqueda con lo que el desarrollador necesita.

En este caso, se pueden obtener los eventos que tengan como invitado al usuario registrado en el terminal, indicando en el objeto que contiene la consulta —*«newQuery»*— que se quieren únicamente los eventos cuyo campo *«invitedUserToTheEvent»* contenga el usuario registrado.

Una vez definido el objeto, para hacer efectiva la consulta, habrá que implementar una última acción que efectúe la conexión y recoja los datos cuando la plataforma los envíe. Esta acción se realiza mediante un método que incluye dos parámetros los cuales completará la plataforma. El primero de ellos es del tipo *NSArray* y contiene todos los objetos de la clase seleccionada que cumplen las condiciones requeridas en la consulta. El segundo vuelve a ser del tipo *NSError*, para dar información en el caso de que hubiera un fallo en la consulta.

Este procedimiento es el habitual para recuperar información de la plataforma, por lo que cada vez que se vaya a obtener algún dato de *Parse* a lo largo de la memoria, se implementará de este modo, salvo que se especifique lo contrario.

4.3.1. *Color Code*

El primer elemento gráfico que refleja estas diferencias frente a una *UITableView* básica se encuentra en cada una de las celdas de la misma. A la derecha de la celda, cerca del *disclosure indicator*, se muestra un color dentro de un pequeño círculo con un propósito específico.

HiUp pretende ser una aplicación capaz de gestionar cualquier tipo de evento, ya sea laboral, personal, de ocio… Es por ello que, llegado a un punto, el usuario puede tener una gran cantidad de eventos y le cueste identificar cuáles de ellos le interesan en cada momento. Para solucionar este problema se ha creado este código de colores.

Cuando el usuario pulse sobre el color, la aplicación desplegará una nueva vista superpuesta con varias paletas de colores entre las que podrá escoger aquel color que más le guste para clasificar el evento. De este modo, el usuario podrá tener todos los eventos catalogados y ordenados de una forma visual y versátil.

Al permitir escoger entre una paleta de colores tan variada —24 colores organizados en cuatro distintos temas—, se abren múltiples alternativas de clasificación, eliminando la rigidez de otras aplicaciones que sólo permiten catalogar según unos temas predefinidos. Esta funcionalidad queda reflejada en la figura 4.12.

Figura 4.11 Vista principal de la pestaña Events (izda.) e interfaz con la paleta de colores desplegada (dcha.)

Mientras la paleta de colores es mostrada, la circunferencia se ensanchará dinámicamente y la celda será resaltada, deshabilitando el resto de elementos de la vista. De este modo, el usuario no podrá modificar nada que no sea el color asociado a la celda. Para navegar por las distintas paletas disponibles, únicamente deberá deslizar un dedo por encima de la paleta para cambiar los colores. Esta funcionalidad se consigue de un modo sencillo gracias a un *UIGestureRecognizer*. *UIGestureRecognizer* es una clase que debe incluirse en otra que posea interfaz gráfica —cualquier clase que descienda de *UIView* sería válida— y que reacciona ante un gesto concreto realizado sobre la interfaz de la clase contenedora.

En este caso, una vez definido el gesto que se desea reconocer —deslizamiento a la izquierda y a la derecha— y añadido el reconocedor a la vista de la paleta, se puede modificar el contenido de la misma en función de lo que haga el usuario.

En el momento que el usuario escoja el color apropiado, la celda volverá a su posición y tamaño natural, habilitando de nuevo el resto de celdas para que el usuario pueda continuar usando la aplicación con normalidad. Del mismo modo, si el usuario finalmente no quisiera cambiar el color o hubiera pulsado por error el botón, tocando en cualquier lugar de la pantalla, se volvería a la posición inicial.

Puesto que es el propio usuario el que decide cómo gestionar los eventos, el color asociado a cada celda será almacenado en el propio terminal, y no irá asociado al evento, ya que cada invitado podría tratarlo de un modo diferente. Este procedimiento se va a realizar aprovechando zona de memoria que el entorno de desarrollo ofrece para guardar cualquier valor por defecto del usuario.

Para acceder a esta memoria, sólo hay que recurrir al método *[NSUserDefaults standardUserDefaults]*. Este método devolverá una clase con la que se podrá almacenar o recuperar información a partir de sus métodos accesores. Será el entorno de desarrollo quien se encargue de reservar una zona de memoria dentro del terminal que no sea volátil y mantenga la información guardada cuando la aplicación se cierre.

Para el caso de «*Color Code*», la aplicación tendrá que almacenar un color para cada evento que el usuario tenga. Para ello se va a recurrir a una variable del tipo *NSDictionary*, la cual tendrá como clave el identificador único que *Parse* asigna a cada uno de los eventos y como valor un objeto de la clase *UIColor*. *UIColor* es una clase que gestiona un color en el formato RGB —*Red, Green, Blue*—, aunque también acepta formatos del tipo HSB —*Hue, Saturation, Brightness*— e incluso definir una imagen como patrón de color.

NSUserDefaults sólo permite almacenar unos tipos de datos muy concretos —*NSData*, *NSString*, *NSNumber*, *NSDate*, *NSArray*, o *NSDictionary*—. En este caso, puesto que se pretende almacenar un *NSDictionary*, no existe ningún problema, pero será necesario tener esto en cuenta para otros valores predeterminados por el usuario que se quieran guardar.

Todos los eventos tendrán un color por defecto asignado, el cual podrá ser modificado en la pestaña de *Settings*, como será explicada más adelante.

4.3.2. Date Gadget

Para encontrar otro elemento gráfico que diferencia la interfaz de la *UITableView* básica, es necesario hacer *scrolling* en la propia tabla, ya que no puede apreciarse a simple vista.

Como en toda la aplicación, con el desarrollo de la interfaz de eventos se pretende mostrar la mayor información posible evitando en todo momento sobrecargar la interfaz. En el caso de los eventos, una de las informaciones más importantes es la fecha del mismo. En muchas ocasiones el usuario únicamente querrá recordar cuándo va a celebrarse, sin preocuparle cuáles eran los detalles del mismo o quién está invitado.

Para facilitar esta información, cuando el usuario esté desplazándose a través de la tabla, aparecerá un indicador translúcido a la derecha de la pantalla en el que se mostrará esquemáticamente la fecha de cada evento. Este indicador se irá moviendo dinámicamente al mismo ritmo que el usuario se desplace e irá cambiando la fecha según la celda que señale, mostrando siempre la del evento correspondiente. Si el usuario deja de navegar, el indicador desaparecerá dejando de nuevo la interfaz en su aspecto inicial. Este indicador se muestra la figura 4.13.

En el caso de que la fecha del evento sea el mismo día, la aplicación está preparada para modificar el formato e indicar la hora del mismo en lugar del día, ya que al igual que todas las fechas gestionadas por la aplicación, el «*Date Gadget*» se basa en un objeto del tipo *NSDate*. Gracias a su habilidad para dar formato a las fechas, se puede modificar el tipo de información mostrada al usuario en función de la diferencia con el momento actual.

Para conseguir el efecto de aparecer y desaparecer se ha recurrido a la variable *alpha* de la vista que contiene el elemento. Todos los objetos que heredan de la clase *UIView*, poseen la variable *alpha*, que define la transparencia del mismo. Modificándola adecuadamente, puede conseguirse un efecto de desvanecimiento muy atractivo visualmente.

Figura 4.12 Interfaces de Events con Date Gadget visible

Desde un punto de vista funcional, la interfaz permite realizar tres acciones. El usuario podrá consultar y modificar toda la información de cada uno de los eventos, añadir un nuevo evento o eliminar cualquiera de ellos.

Aparte, aunque los eventos se refrescan automáticamente según se van creando, se le da la posibilidad al usuario de realizar esta acción cuando él desee de manera manual. Para ello, debe hacer *scroll* a la parte superior de la pantalla y aparecerá un símbolo indicando que se va a refrescar los eventos. Este gesto es muy utilizado en muchas de las aplicaciones del sistema operativo, por lo que resultará bastante intuitivo para cualquier usuario habitual de *iOS*.

4.3.3. *Creación*

Si el usuario pulsara en el botón con el símbolo más (+), la aplicación volvería a la pestaña de creación de evento y permitiría crear un evento del mismo modo que se explicó en el anterior apartado. Este comportamiento es muy habitual en muchas de las pantallas nativas del sistema operativo. Esto permite al usuario familiarizarse con la *app* de una manera mucho más rápida. En la figura 4.14 se compara una de las pantallas de la aplicación con la de «Favoritos» de sistema operativo del *iPhone*.

Esta opción no era funcionalmente necesaria, ya que existe una alternativa más directa con la propia pestaña y que ofrece el mismo resultado. Sin embargo, desde el punto de vista de hábito de uso tiene un mayor sentido.

Figura 4.13 Comparación entre diseño de pantalla de favoritos de iPhone (izda.) y diseño genérico de las pantallas principales de HiUp (dcha.)

Con esta disposición de la interfaz se pretende que el usuario, de un modo intuitivo, sea capaz de usar la aplicación de una manera fluida desde la primera vez que entre en contacto con la ella. Además, es interesante que tenga a mano una alternativa más para crear un evento, siempre y cuando sea coherente con el contexto, ya que se trata del principal medio de ingresos de la aplicación.

4.3.4. *Modificación/Consulta*

Lo siguiente que necesitará el usuario, una vez que tenga eventos ya creados, es poder consultar toda la información de cualquiera de ellos y modificarla si fuera pertinente.

Para gestionar los permisos de edición de la información de los eventos se ha recurrido al caso más sencillo. Si el usuario es el creador del evento, podrá modificar toda la información del evento, mientras que si se trata de un invitado solo podrá consultar la información. Para conocer esta información, se va a recurrir al campo *«creatorOfEvent»* que poseen todos los objetos de la clase «*Event*». Para consultar un dato de cualquier objeto del tipo *PFObject*, se procederá prácticamente igual que si se tratara de un *NSDictionary*. Una vez obtenido el evento del modo explicado en el comienzo del apartado 4.3, se puede recuperar el valor asociado a la clave «*creatorOfEvent*» a través de un método que devolverá un objeto de la clase que se indique en el parámetro *key*. En este caso el parámetro será *«creatorOfEvent»* y el objeto será del tipo *PFUser*. Comparándolo con el usuario registrado en la aplicación, puede comprobarse si se trata del creador o no.

En este proyecto no se ha abordado el tema de los permisos para usuarios que no sean el creador, los cuales podrían tratarse en una versión posterior. En ambos casos, la interfaz que se mostrará será prácticamente la misma, como se muestra en la figura 4.15.

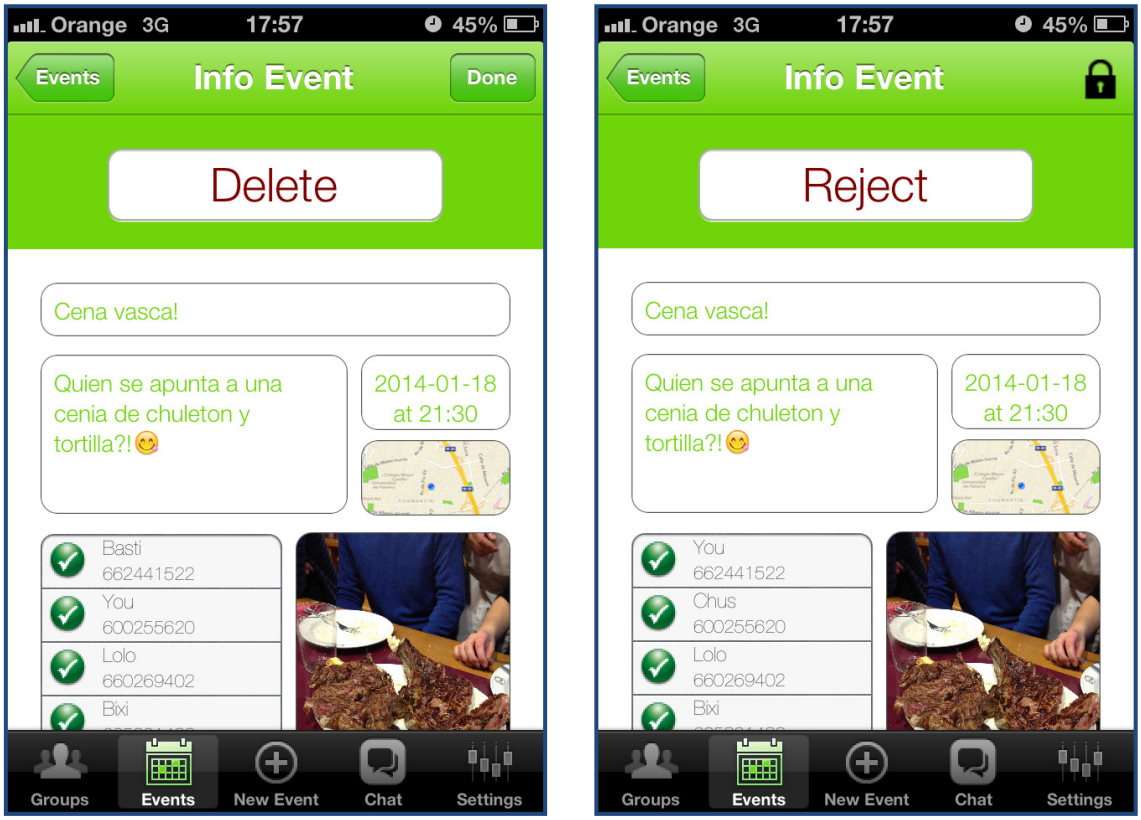


Figura 4.14 Pantalla Info Event con permisos de edición (izda.) y sin permisos (dcha.)

Como se puede observar, la pantalla de modificación/consulta de los eventos es la misma que la de creación de eventos, pero con todos los campos ya rellenos con los datos almacenados del evento. Todos los datos se obtendrán del mismo modo que se ha conseguido el creador del evento. Ésta es una forma sencilla de conseguir que el usuario se familiarice con la aplicación. Introducir apariencias similares para funcionalidades parecidas.

La principal diferencia con respecto a la interfaz de un nuevo evento es que en este caso, se le debe permitir al usuario aceptar o rechazar la invitación. En este proyecto se ha simplificado esta posibilidad con la siguiente decisión. Se considerará que cualquier usuario que ha sido invitado a un evento asistirá al mismo hasta que lo rechace. Para rechazarlo, deberá pulsar el botón que aparece detrás de la vista principal, el cual aparecerá cuando el usuario haga *scroll* en la pantalla, como se muestra en la figura anterior.

Si el usuario fuera un invitado del evento, se eliminará la relación que existe entre ambos y el evento no volverá a aparecer en su lista. Sin embargo, en el caso de que fuera el creador, la aplicación eliminará el evento por completo y ninguno de los invitados podrá verlo a partir de ese momento. Este comportamiento podrá modificarse si se aborda la gestión de permisos en futuras actualizaciones.

Para el caso en el que el usuario no sea el creador, hace falta explicar cómo eliminar elementos de una relación ya creada. Como ya se explicó en el apartado 4.2.4, todos los usuarios invitados al evento deben añadirse a un objeto del tipo *PFRelation*, el cual gestionará internamente la plataforma para crear una relación entre los usuarios y el evento. Cuando se quiera eliminar a un usuario del evento se recuperará esa relación del objeto «*event*» a través de la clave «*invitedUserToTheEvent*». Una vez almacenada la relación de la que se quiere eliminar un usuario —«*relationToRemove*»—, se utilizará un método que borrará al usuario que rechaza la invitación, que en este caso será el usuario registrado. Puesto que «*relationToRemove*» es directamente la relación almacenada en el objeto «*event*», con registrar de nuevo el objeto en *Parse*, la relación se actualizará y desaparecerá el usuario de la misma.

Este último paso de registrar los cambios efectuados también se aplica en el caso que el usuario sí que tenga permisos de edición y haya realizado algún cambio. Cuando presione el botón «*Done*», la aplicación guardará las modificaciones a través de este método y notificará a todos los usuarios registrados de que ha habido un cambio en el evento. El proceso de notificaciones será explicado en un apartado independiente al final del capítulo.

4.3.5. *Eliminación*

Aparte de eliminarlo desde la propia ventana del evento, como se ha comentado en el anterior apartado, el usuario puede hacerlo más rápidamente desde la vista general de la pestaña «*Events*». Puesto que la pantalla principal de esta pestaña ha sido diseñada sobre una *UITableView*, se va a aprovechar los métodos de eliminación asociados a la misma.

Si se desea eliminar algún evento —siempre que tenga permisos para hacerlo—, pulsando el botón *«Edit»,* aparecerá un símbolo de prohibido en cada una de las celdas. Pulsando sobre éste, se mostrará un botón «*Delete*» con el que poder eliminar el evento. Este botón también será visible si el usuario desliza sobre la propia celda. Ambas posibilidades son implementadas por la *API* del entorno de desarrollo.

Una vez visible el botón, únicamente deberá pulsar en la celda y confirmar la acción. Así, el evento y todos sus campos serán eliminados tanto del sistema como visualmente de la pantalla principal. Si el usuario no tuviera los permisos, la aplicación ni siquiera mostraría los botones de eliminación mencionados. Para eliminar un evento, la aplicación únicamente debe notificar esta acción a *Parse* con el método «*deleteInBackground*». Este método indica a *Parse* que debe eliminar el objeto que está realizando esta llamada.

La figura 4.16 muestra ambos métodos de eliminación explicados.

Figura 4.15 Pantallas para eliminación de un evento

4.4. Groups

Resulta bastante normal que, en muchas ocasiones, el usuario desee añadir a varios eventos a las mismas personas. Igual de comprensible sería que un usuario quiera incluir a distintos grupos de amigos a un mismo evento sin tener que marcar uno a uno a cada uno de ellos. Es por esto que se ha creado la opción *Groups*.

En la pestaña de grupos, el usuario podrá crear, modificar o eliminar un grupo de personas, aparte de poder consultar de una manera clara y ordenada todos los grupos que ya tiene, utilizando de nuevo el elemento *UITableView*. La figura 4.17 muestra la pantalla principal de esta pestaña.

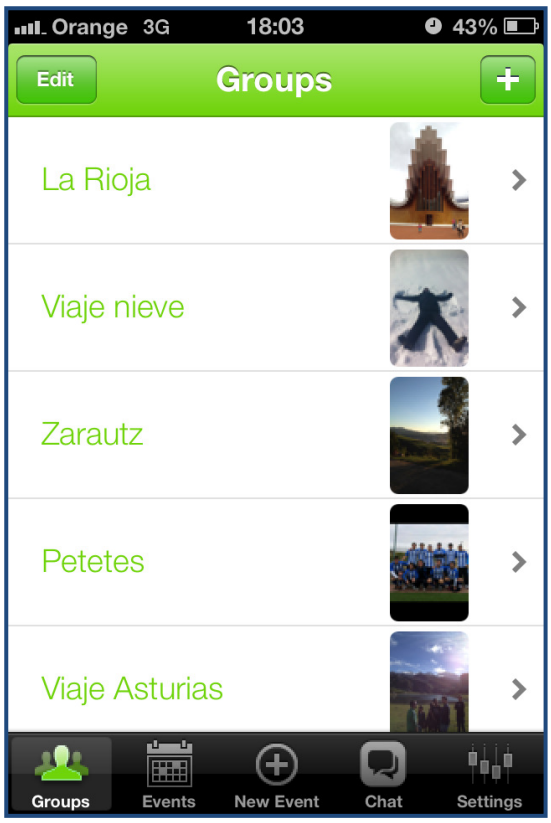


Figura 4.16 Vista principal de Groups

Como ya se ha comentado, la organización de los grupos es independiente para cada usuario, por lo que se van a almacenar todos ellos en una base de datos creada para este proyecto y gestionada con una pequeña *API* interna. Para ello, durante la inicialización de la aplicación se llamará al método de esta *API* que crea la base de datos o la abre en caso de que ya existiera. Una vez abierta, la aplicación podrá hacer distintas consultas para realizar las transacciones de datos que sean oportunas a través de esta misma *API*.

Puesto que la funcionalidad de la vista principal de *Groups* es muy similar a la de *Events*, sus interfaces no deberían diferir mucho. Es por esto que se va a permitir al usuario hacer las mismas acciones por medio de los mismos botones.

4.4.1. *Creación*

Para poder beneficiarse de los grupos, es necesario crear al menos uno de ellos. De nuevo con el botón con el símbolo más (+) que se encuentra en la parte superior derecha de la vista principal, se abrirá la ventana para la creación de un nuevo grupo.

Ya que la información de un grupo es bastante reducida y simple, se ha tratado de transmitir esta simplicidad a la ventana. Bajo esta premisa, la aplicación muestra únicamente una pantalla lisa con un fondo gris, dónde se mostrará la imagen del grupo, dos botones que permitirán la navegación entre vistas y una barra desplegable en la parte inferior de la pantalla que muestra el título del grupo, tal como se muestra en la figura 4.18.

Figura 4.17 Pantalla creación de grupo con barra sin desplegar y desplegada

Lo primero que puede definir el usuario es la foto con la cual va a identificar al grupo. Al igual que en el apartado *«New Event»*, se va a hacer uso de la *API* del entorno de desarrollo para la selección de imágenes. Para acceder a la galería, el usuario únicamente deberá pulsar el fondo gris. Una vez el usuario haya escogido cualquier foto que tenga almacenada en la galería del terminal, ésta aparecerá cómo fondo en la pantalla, ayudando al usuario a pre visualizar en todo momento la imagen escogida.

Lo siguiente por concretar en el grupo es el nombre del mismo. Pulsando sobre la barra desplegable, el teclado aparecerá automáticamente y el usuario podrá modificar el

nombre del grupo. La longitud del nombre se encuentra limitada entre 1 y 20 caracteres para asegurar un comportamiento gráfico y funcional óptimo. La figura 4.18 muestra también como quedaría la vista con la barra desplegada.

Una vez que ha terminado de escribir, si el usuario pulsa en cualquier lado de la pantalla o el botón «*Intro*» del teclado, éste desparecerá dejando ver el espacio desplegado por la barra que se encontraba en el inferior de la vista. En este espacio aparecerán los usuarios del grupo cuando sean añadidos. Hay dos motivos por los que la barra se despliega para escribir el nombre:

- El primero es un motivo meramente estético. Al desplegar la barra, se evita que el propio teclado solape el campo de texto e impida que el usuario pueda ver lo que está escribiendo.

- El segundo se trata más de un motivo de intuición. La barra se despliega deslizando un dedo sobre ella y es posible que el usuario no sepa esto a priori. De este modo, se muestra automáticamente el lugar donde van a ir los usuarios, descubriendo una zona no visible que podría ser desconocida para el usuario.

Una vez elegido el nombre del grupo, el último paso es añadir a los contactos que van a pertenecer a dicho grupo. Puesto que en un principio la barra ya se encuentra desplegada, cuando se oculte el teclado aparecerá un espacio con un fondo translúcido y un botón con el símbolo más (+) donde aparecerán los contactos añadidos, como muestra la figura 4.19.

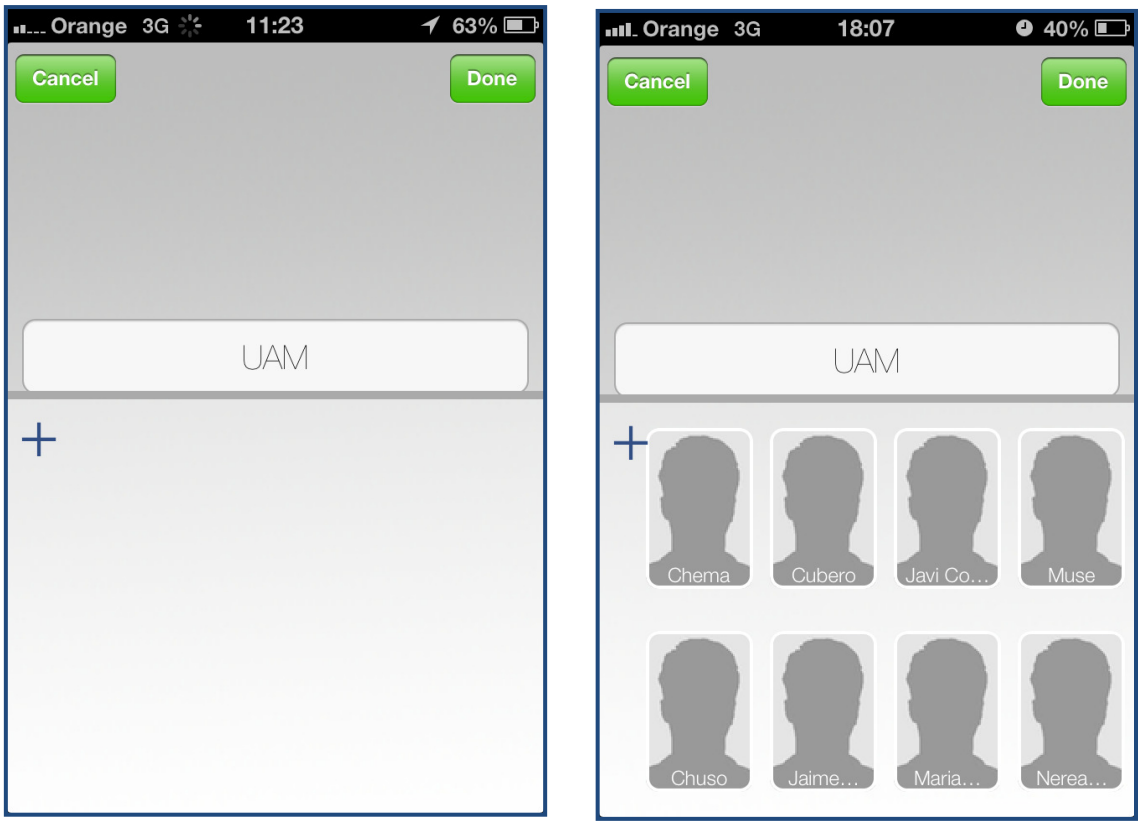


Figura 4.18 Vista para contactos antes y después de añadir nuevos usuarios

Cuando el botón para añadir usuarios sea pulsado, la aplicación mostrará todos los contactos que el usuario tiene almacenados en el dispositivo. De nuevo se utilizará el método multiselección antes mencionado y que se explicará en el apartado 4.7.

Con todos los usuarios añadidos, la aplicación los mostrará ordenados en la zona definida para ello. Para evitar sobrecargar la pantalla y mantener la limpieza en la interfaz, únicamente se mostrarán las fotos que el usuario tenga asignadas a cada contacto en la agenda y un nombre indicativo al pie de cada una de las imágenes. En caso de que algún contacto no tenga foto se mostrará una imagen por defecto. Esta situación se muestra también en la figura 4.19.

Aunque se quiera mantener una simplicidad, es importante que el usuario pueda conocer la información básica de los contactos agregados, como el nombre o el teléfono. Por ello, cuando el usuario pulse sobre alguno de los contactos, la aplicación desplegará una pequeña tabla en la parte inferior de la celda en la que se mostrará esta información. Si el usuario pulsa varias celdas, la tabla irá moviéndose dinámicamente a la celda que corresponda, pero sólo aparecerá una única tabla para evitar saturar gráficamente la interfaz.

Al mismo tiempo que aparece la tabla, se mostrará un aspa (✕) por si el usuario deseara ocultarla. Ésta desaparecerá automáticamente en el caso que el usuario haga *scrolling* en la zona de usuarios. En la figura 4.20 se muestra esta tabla con la información básica.

Figura 4.19 Pantalla de nuevo grupo con tabla de información de usuario

Una vez que los contactos están visibles, es posible que el usuario se haya equivocado al añadir a alguno de los contactos o quiera borrar a varios de ellos. Si se diera el caso, sólo debería arrastrar las celdas que desee eliminar fuera de la zona delimitada para los contactos y la aplicación los borraría automáticamente del grupo, recolocando el resto de celdas al instante.

Por último, es necesario que la aplicación almacene todos los datos en la base de datos creada para ello. Por eso, el usuario debe confirmar la creación del grupo a través del botón «*Done*». De este modo, el grupo quedará registrado con toda la información introducida.

Una vez pulsado el botón, la aplicación recogerá toda la información incluida en el grupo y creará nuevas entradas en las tablas correspondientes en la base de datos.

La primera entrada será en la tabla «*Groups*». En esta entrada se recogerá el nombre del grupo, la foto, la fecha de creación y se creará un *ID* único para esa entrada, ya que será necesario para luego relacionar ese grupo con los usuarios incluidos en él.

Aparte de esta primera entrada, la aplicación comprobará si todos los usuarios agregados al grupo se encontraban ya registrados en la tabla «*Users*». En caso de que alguno no estuviera registrado, primero se creará una entrada en la tabla, guardando el nombre, el número de teléfono y su identificador único. Con todos los contactos del grupo almacenados, la aplicación recogerá los *IDs* de cada uno de ellos para crear la relación con el nuevo grupo.

Por último, la aplicación va a solventar la relación entre usuarios y grupos a partir de una tercera tabla, ya que se trata de una relación muchos-a-muchos. Este tercera tabla se llamará «*Group-Users*» y en ella se van a almacenar en una entrada el *ID* de un usuario con el *ID* de un grupo. De este modo, cuando se quiera realizar una consulta, se podrá obtener los datos escogidos seleccionando adecuadamente las entradas de esta última tabla.

Si durante cualquier momento de todo este proceso el usuario se diera cuenta de que no es necesario el grupo, sólo deberá pulsar el botón «*Cancel*», y la aplicación volverá a mostrar la ventana principal de la pestaña.

4.4.2. Modificación

Una vez creado un grupo, aparecerá como una nueva entrada de la tabla principal. Si el usuario quisiera añadir o eliminar usuarios, cambiar la foto asociada al grupo o editar el nombre, deberá pulsar la celda correspondiente para poder ver toda la información de ese grupo.

De nuevo se vuelven a tener funcionalidades similares con la creación del grupo, por lo que la interfaz de edición va a ser muy similar a la del anterior apartado. Puesto que se trata de un grupo ya creado, la única diferencia es que todos los campos van a encontrarse

rellenos en cuanto se inicialice la ventana. Para ello, aunque cada grupo posee un identificador único para acceder a él en la base de datos, su nombre también es identificativo, ya que no se va a permitir crear dos grupos con el mismo nombre. Por esto, cuando el usuario seleccione un grupo, se recuperará toda la información seleccionando la entrada de la tabla «*Groups*» cuyo nombre corresponda con el grupo y se mostrará en el campo correspondiente.

Cabría destacar una ligera diferencia o peculiaridad en cuanto a los tiempos en los que se almacena la información. En el apartado anterior se ha comentado que toda la información queda guardada en la base de datos una vez que el usuario ha pulsado el botón «*Done*». Este comportamiento se mantiene en la modificación de un grupo a excepción de los contactos borrados.

Si el usuario borra un contacto, su celda desaparece y la aplicación informa de que ha sido eliminado. Que el contacto volviera a aparecer una vez descartado podría dar cierta sensación de incoherencia. Por lo tanto en caso que la celda sea eliminada, el contacto será borrado del grupo instantáneamente, sin esperar a la pulsación de confirmación por parte del usuario. Este proceso de borrado queda descrito con en la figura 4.21.

Figura 4.20 Proceso de eliminación de contacto en la creación o modificación de un grupo

Cuando el usuario pulse el botón «*Done*», todos los datos se actualizarán a la vez a través de una sentencia *UPDATE* en cada una de las tablas correspondientes. Aparte, si el usuario eliminara algún usuario, únicamente habría que eliminar la entrada que relaciona al grupo con ese contacto en la tabla «*Group-Users*».

4.4.3. *Eliminación*

Una vez creados los grupos y modificados según las necesidades del usuario, la aplicación debe permitir eliminar aquellos que el usuario considere ya innecesarios. De nuevo se va a utilizar la funcionalidad propia del elemento *UITableView*.

Cuando el usuario desee eliminar un grupo, pulsando el botón *«Edit»* o deslizando sobre la celda, aparecerá un símbolo junto a cada evento para que pueda ser eliminado, como se muestra en la figura 4.22. Pulsando sobre el que se desee eliminar, se borrará de la base de datos gracias al método correspondiente de la *API* implementada. Aunque tanto el grupo como la relación con los usuarios desaparecerán por completo de ambas tablas, los usuarios no deben ser eliminados, ya que podrían pertenecer a otros grupos y provocaría un comportamiento erróneo. Incluso si el usuario no estuviera en ningún otro grupo, la aplicación mantendrá el contacto en la tabla «*Users*». Esto se ha escogido debido a que, en un comportamiento normal, esta tabla debería tener entre 100 y 200 entradas como máximo, cantidad que se pueden gestionar muy asequiblemente. Además, de este modo, no será necesario comprobar todos los grupos cada vez que se elimine uno, ya que podría ser excesivamente costoso computacionalmente si hubiera un número elevado de grupos.

Figura 4.21 Pantallas para eliminación de un grupo

4.5. Chat

Una aplicación de carácter social como *HiUp* quedaría bastante incompleta si los usuarios no pudieran hablar entre ellos. Hasta el momento, si dos usuarios quisieran comunicarse de alguna forma, debería ser a través de la creación de «eventos ficticios». Este método es complejo y muy incómodo, por lo que los usuarios recurrirían a alguna aplicación externa para poder comentar todo lo referente a cualquier evento. Esto sería muy perjudicial para la aplicación.

Es por ello que se ha implementado un chat como herramienta complementaria a la funcionalidad principal de la aplicación, la gestión de eventos. Este chat se encuentra en una versión *beta* y dista de la funcionalidad que ofrecen los chats comerciales ampliamente extendidos en la actualidad. Aun así, cumple con su función principal, que es la conexión entre los usuarios de la aplicación y es una buena base para posibles mejoras en futuras actualizaciones.

El chat permite, aparte de la transmisión de mensajes entre dos usuarios, la creación de salas de chat tanto para grupos creados en el momento como que estuvieran ya almacenados dentro de la pestaña *Groups*. Además, todas las conversaciones poseen un servicio de notificaciones para mensajes nuevos, al igual que las aplicaciones comerciales actuales. Este servicio de notificaciones y su uso en la aplicación será desarrollado en un apartado propio más adelante.

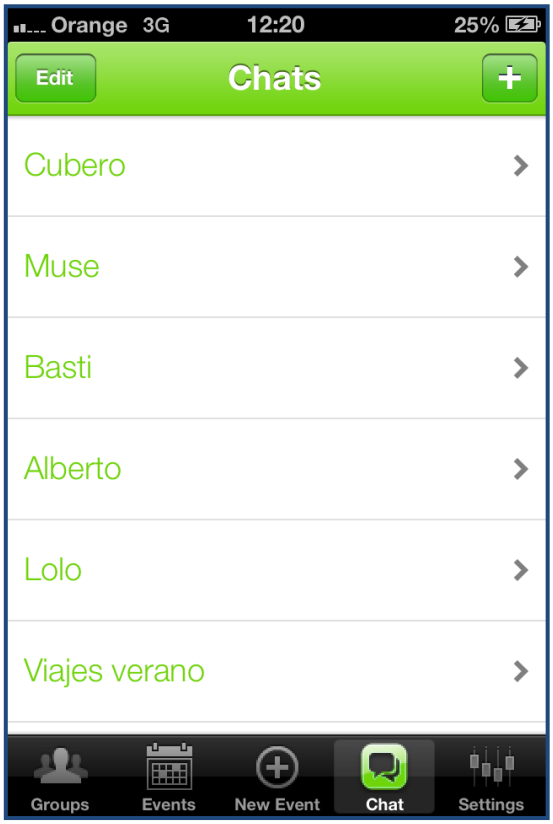


Figura 4.22 Vista principal de la pestaña de Chat

Como se puede observar en la figura 4.23, de nuevo se ha recurrido a una interfaz con el mismo aspecto para homogeneizar todas las vistas con una funcionalidad parecida.

4.5.1. Creación

Para crear una sala de chat con otros usuarios se va a recurrir al mismo planteamiento que el de otras pestañas. Pulsando el botón más (+) de la esquina superior derecha, se mostrará un mensaje al usuario dando a escoger si quiere agregar a varios contactos de la agenda o a todos los usuarios de un grupo, como muestra la figura 4.24. De nuevo, esta funcionalidad agiliza mucho el uso habitual de la aplicación.

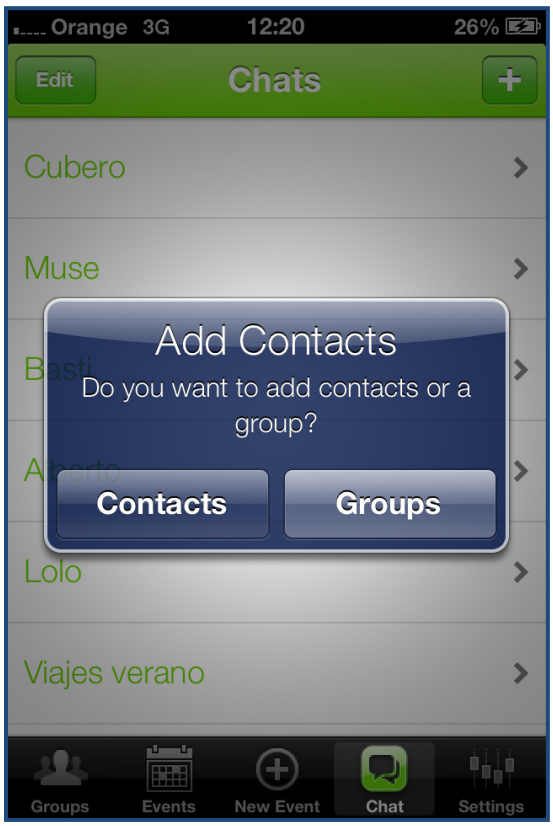


Figura 4.23 Mensaje para creación nuevo chat

Cuando el usuario pulse uno de los dos botones, la aplicación se comportará exactamente igual que lo explicado en el apartado 4.2.1 para añadir usuarios a un evento. La única diferencia es que en este caso, en vez de aparecer todos los contactos en una tabla, se creará una sala de chat con todos los usuarios invitados que ya estén registrados.

Puesto que el usuario puede tener muchos contactos en la agenda que no tengan la aplicación, antes de crear la sala, mostrará aquellos que no están registrados y un mensaje informando que no recibirán los mensajes que se manden dentro de la sala que se va a crear. Si ninguno de los contactos invitados estuviera registrado, la aplicación volverá a

informar del error y no permitirá la creación, puesto que se trataría de un chat con el creador como único participante. Ambos mensajes se muestra en la figura 4.25.

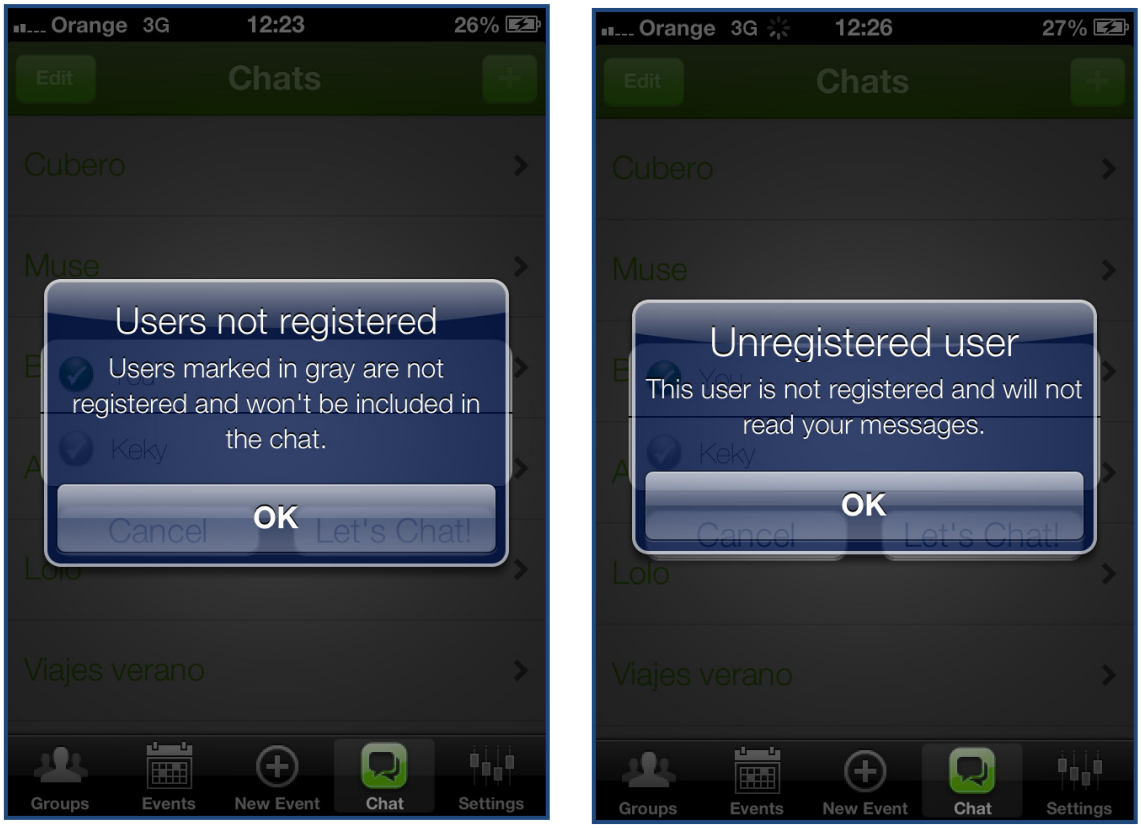


Figura 4.24 Mensajes de error para usuarios no registrados (izda.) y cuando para chats con un único usuario (dcha.)

Aparte de estos errores, la aplicación está preparada para evitar crear más de un chat con el mismo usuario, crear salas de grupo sin ningún nombre…

Con todos los datos introducidos correctamente, la aplicación creará un objeto de la clase *PFObject* y se añadirá a este objeto toda la información necesaria como se explica a continuación.

El primer paso es crear un objeto del tipo *PFObject* cuya clase va a ser *«Chatroom»*. A este objeto se le añadirán algunas variables para contener toda la información necesaria. Esta información añadida va a ser el creador de la sala de chat, que se gestionará como una relación uno-a-muchos, al igual que en «*New Event*», el nombre del grupo en caso de que la conversación tuviera más de dos participantes y un *array* con el teléfono de todos los participantes. Este *array* va a permitir a la aplicación discernir en cuáles de todas las conversaciones está incluido el usuario registrado.

Aparte de estas variables, también hace falta crear una relación muchos-a-muchos entre los participantes de la conversación y la sala de chat. Para eso se crea el objeto *PFRelation*, al cual se le añaden luego todos los participantes registrados.

Por último, la aplicación almacenará este objeto del mismo modo que se explicó en el apartado 4.2.4.

4.5.2. Chatrooms

Una vez creada la sala de chat, ésta se mostrará junto con todas las demás en la tabla de la pantalla. Esta tabla se irá reordenando automáticamente según se vaya hablando por unas u otras salas, mostrando en la parte superior aquellas que han estado activas más recientemente.

Aparte, al igual que en el apartado «*Events*», el usuario podrá refrescar todas las salas de chat para ver si hay alguna novedad, aunque no debería ser necesario puesto que las conversaciones se actualizan automáticamente según se van recibiendo los mensajes. Tanto si lo hace manualmente como si recibiera una nueva invitación a una conversación, la aplicación va a recuperar todas las salas de chat del mismo modo que se explicó para recuperar eventos en el apartado 4.3, salvo que ahora la consulta se hará sobre los objetos de tipo *Chatroom.*

Desde la pantalla principal, la cual se puede ver más adelante en la figura 4.27, en la imagen de la izquierda, se van a poder eliminar las conversaciones de chat con el botón «*Edit*», al igual que en el resto de vistas explicadas anteriormente. El proceso de borrado se va a explicar en un apartado independiente más adelante.

4.5.3. Custom transition

Desde la ventana anterior, el usuario podrá acceder a cualquier conversación que tenga pulsando la celda correspondiente. Antes de explicar el funcionamiento de las salas de chat, cabe pararse un momento en la transición entre ambas vistas.

Una gran cantidad de usuarios con *smartphones* hace uso de alguna aplicación de chat comercial, por lo que la mayoría están muy familiarizados con ellas. Debido a esto, si se consiguiera en la parte del chat alguna característica que diferenciara mínimamente a *HiUp* del resto, se conseguiría un gran beneficio para la aplicación.

Observando dos de las aplicaciones de chat más extendidas en España como son *Whatsapp* y *Line*, se puede comprobar que el paso entre la pantalla con todas las salas y la vista de cada sala privada es una transición estándar de *iOS*. Es en esta transición dónde se va a intentar obtener un rasgo mínimamente diferenciador, aunque bastante atractivo visualmente.

En este proyecto se han implementado tres efectos que servirán de transición entre ambas pantallas y que serán configurables desde la pestaña de *Settings*, que se explicará más adelante. Los tres efectos son *Flip*, *Explossion* y *Shrink.* De esta manera, de un modo muy visual, puede conseguirse una respuesta positiva por parte del usuario.

4.5.4. Chatroom

Una vez dentro de la sala de chat, el comportamiento es bastante intuitivo. Existe un campo donde se puede escribir los mensajes y al pulsar la tecla «*Intro*», el mensaje se mandará al o los receptores, dependiendo de si se trata de un chat privado o una sala de grupo. Para esto es necesario volver a recurrir a nuevas clases del tipo *PFObject*, en este caso una clase del tipo «*Message*».

Con el objeto *Chatroom* ya identificado, la relación que va a existir entre los mensajes y las salas de chat va a ser del tipo uno-a-muchos. Un mensaje va a pertenecer únicamente a una sala de chat, mientras que una sala de chat podrá contener muchos mensajes.

Con este planteamiento claro, la implementación de esta relación es exactamente igual a la que la aplicación realiza para guardar el creador del evento, por ejemplo. Se añade el objeto «*newMessage*» al objeto «*chatRoom*».

Con la relación ya creada, el resto de campos que van a conformar un mensaje van a ser planos, sin ninguna relación con otra clase almacenada en *Parse*. Estos campos son:

- El teléfono del usuario que mandó el mensaje, almacenando en una clase del tipo *NSString* con la clave «*senderName*».
- El texto del mensaje, almacenado en una clase del tipo *NSString* con la clave «*text*».
- El nombre del grupo, en el caso de que fuera una conversación con varios usuarios, almacenada en una clase del tipo *NSString* con la clave «*groupName*».
- Los números de los usuarios que deberían recibir una notificación del mensaje, almacenados en una clase del tipo *NSArray* con la clave «*targetUsers*».
- Información acerca de si un usuario ha leído o no el mensaje, almacenada en una clase del tipo *NSArray* con la clave «*isNew*».

De todos estos campos, es interesante dar un poco más de información de los dos últimos, puesto que ambos tienen relación con la lógica que se ha llevado a cabo para poder gestionar las notificaciones de los mensajes.

El primero de los dos almacena los teléfonos de los usuarios que van a recibir el mensaje. Así, cuando se haga la consulta, se podrá filtrar por aquellos mensajes que vayan dirigidos al propio usuario. El segundo es necesario cuando se trata de una conversación con varios integrantes. Para poder mantener el icono que indica al usuario cuantos mensajes tiene sin leer en cada conversación, era necesario permitir que un mismo mensaje tuviera varios estados, en función del usuario que lo hubiera leído. Para ello, se almacena una matriz en la que cada entrada tiene el número de un usuario que va a recibir y adjunto un *booleano* que indica si lo ha leído o no.

Una vez que el mensaje esté compuesto por completo, la aplicación realizará tres acciones. La primera es guardar el mensaje en la plataforma *Parse*, almacenándolo como cualquier otro objeto del tipo *PFObject*. La segunda es enviar una notificación con la información adecuada a los destinatarios. El método de implementación de estas notificaciones será explicado en el capítulo 4.8. Por último, refresca el espacio reservado para los mensajes, añadiendo el que acaba de enviarse el último para que el usuario pueda verlo al instante.

La aplicación está preparada para admitir cualquier longitud de mensajes y adaptará el tamaño de la celda al mismo. Aun así, puesto que todos los mensajes van a ser almacenados en la plataforma *Parse*, cuanto mayor sea la longitud del mensaje, mayor será el tiempo necesario para enviarlo.

El terminal receptor, gracias al sistema de notificaciones antes mencionado junto con una lógica de implementación, será informado de que hay un nuevo mensaje y automáticamente refrescará la tabla realizando una consulta a *Parse* con el identificador del chat como parámetro. En la figura 4.26 se puede observar una imagen del envío de un mensaje y su posterior recepción en el otro terminal.

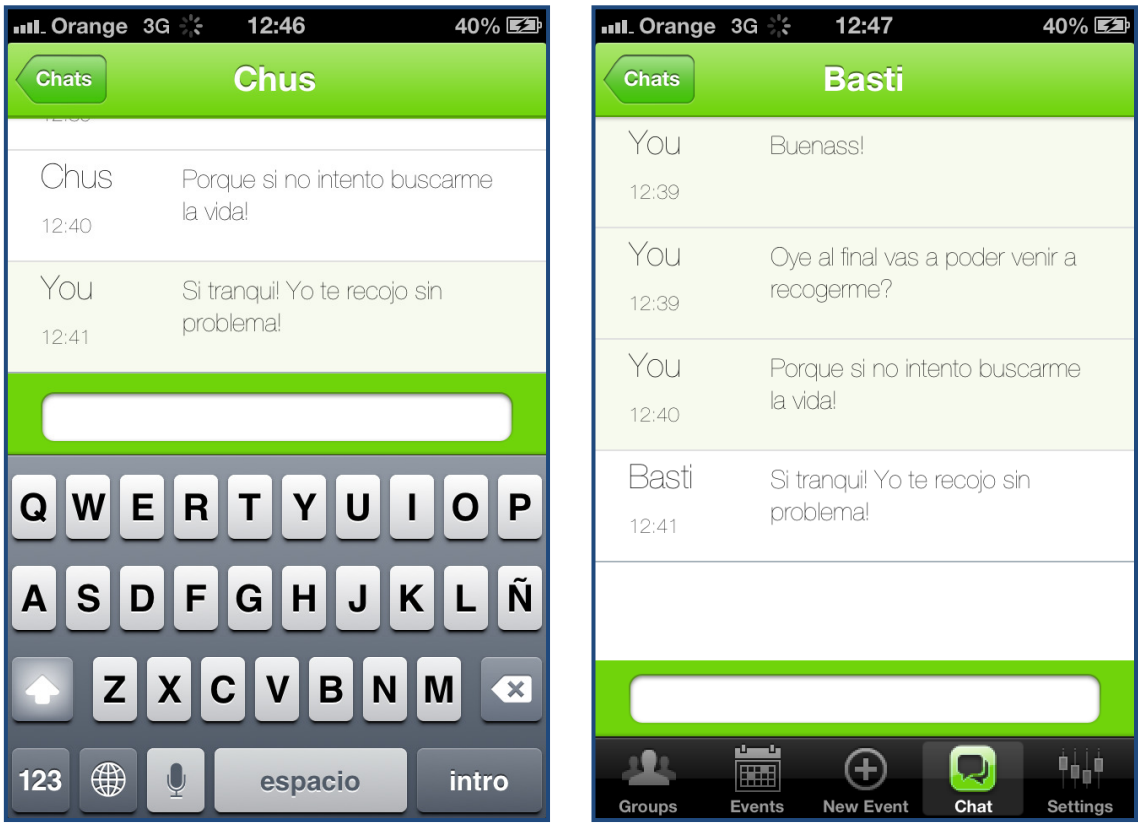


Figura 4.25 Envío (izda.) y recepción (dcha.) de un mensaje de chat

Al igual que en la ventana de «*Events*» y «*Chatrooms*», el usuario podrá refrescar la conversación y comprobar si hay algún nuevo mensaje deslizando la pantalla hacia abajo cuando se encuentre en la parte superior de la misma.

Cuando el usuario reciba un mensaje, existen tres posibilidades. Que el usuario se encuentre dentro de la sala desde donde se ha enviado el mensaje, que esté dentro de la aplicación pero no dentro de la sala, o que ni siquiera esté usando *HiUp* en el momento de recibir el mensaje. Dependiendo de una u otra situación, la aplicación se comportará de un modo u otro.

- En el primer caso, cuando el usuario está dentro de la sala de chat, la aplicación detectará que se ha enviado un mensaje y automáticamente refrescará la sala de chat para mostrar el o los mensajes enviados. Cabe mencionar que, puesto que toda la implementación se basa en la plataforma *Parse*, la latencia que puede haber entre el envío y la recepción del mensaje puede variar entre cinco y quince segundos de media, dependiendo de si ya se ha establecido una conexión. Este tiempo se basa considerando que existe una buena cobertura de datos del terminal. Este es el caso ya mostrado en la figura 4.26.

- Si el usuario estuviera usando la aplicación pero no se encontrara en la sala de chat, la aplicación notificará que hay un mensaje nuevo a través de un indicador sobre la pestaña de Chat. Este indicador se irá incrementando a medida que el usuario vaya recibiendo mensajes y se reseteará cuando el usuario entre en dicha pestaña. Una vez que lo haga, cada una de las salas mostrará cuántos mensajes hay sin leer a partir de un indicador dentro de la propia celda. Esta celda en concreto no ha sido desarrollada en este proyecto, sólo se ha adaptado a partir de un código libre obtenido en Internet. En la figura 4.27 se muestra un ejemplo de ambas situaciones descritas.

Figura 4.26 Pantallas de gestión de nuevos mensajes desde dentro de la aplicación

- Por último, si el usuario no estuviera usando *HiUp*, la aplicación mostrará una notificación estándar del sistema operativo. Esta notificación será distinta dependiendo de si el terminal está activo o si por el contrario, está en un estado de reposo —*background*—. Ambas situaciones quedan reflejadas en la figura 4.28.

Figura 4.27 Notificaciones con el móvil activo fuera de la aplicación (izda.) y en background (dcha.)

4.5.4. Eliminación

A diferencia de cómo ocurre con los eventos, en el caso de que un usuario elimine un chat, lo único que se va a eliminar es la relación entre él y la sala de chat. Incluso cuando sólo quede un usuario, si éste lo elimina, se eliminará la sala de chat, pero los mensajes asociados a ella se mantendrán para el caso de que más adelante se quisieran recuperar. De este modo, aunque los usuarios abandonen la conversación, si en algún momento vuelven a recuperarla, los mensajes se mantendrán.

4.6. Settings

La última de las pestañas es la pestaña de *Settings.* En esta pestaña se van a modificar todos los parámetros configurables de la aplicación, al igual que se va a dar información del programador para ofrecer al usuario un medio de comunicación.

Esta última ventana, debido a su simplicidad, no requiere más que una tabla con varias secciones, cada una de las cuales modificara uno de los parámetros. La figura 4.29 muestra en varias imágenes toda la pantalla de esta última pestaña.

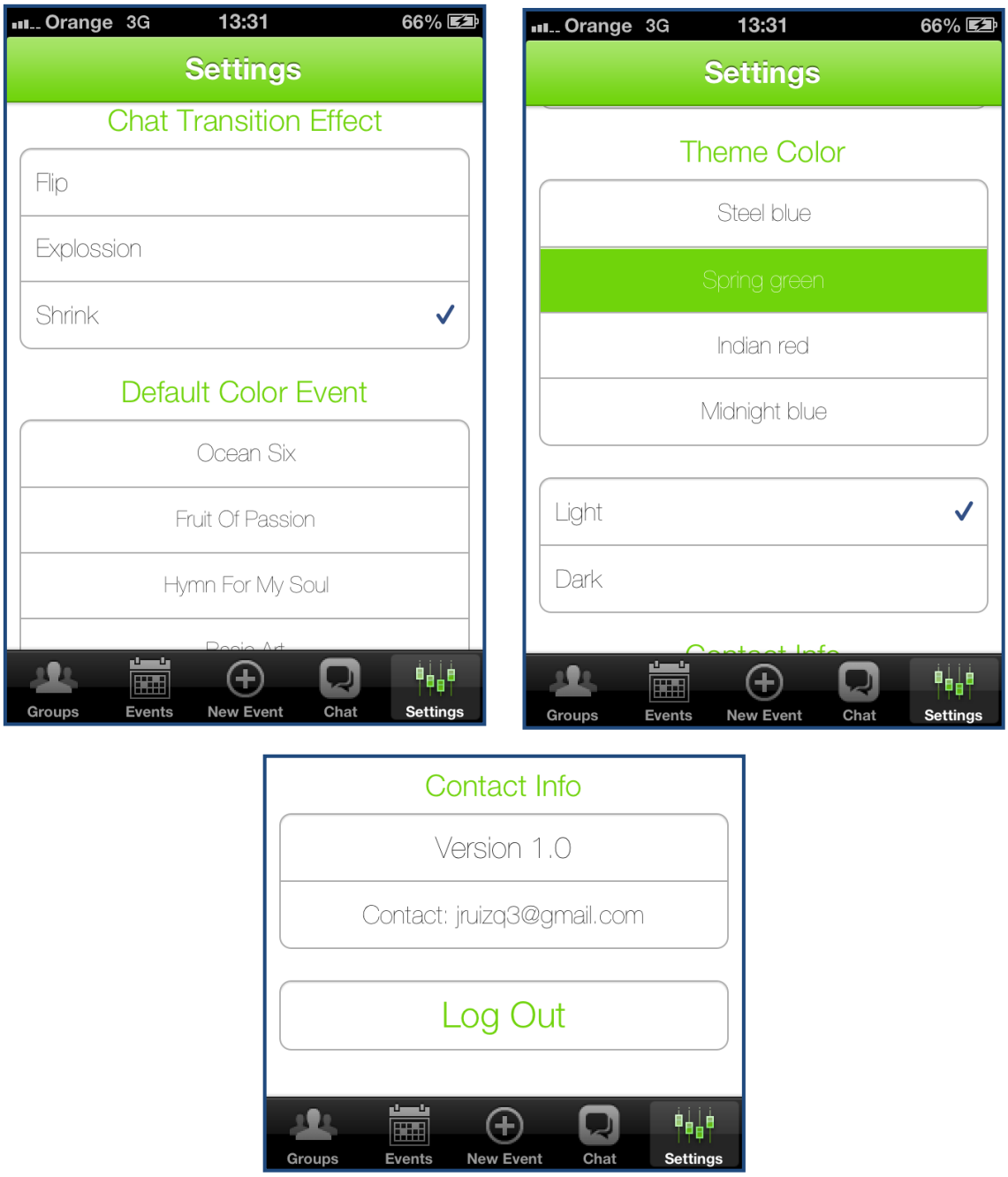


Figura 4.28 Composición de pestaña Settings

4.6.1. Chat Transition

El primer parámetro que se puede configurar es la transición entre la ventana con las salas de chat y cada una de las conversaciones. Como ya se ha comentado, uno de los rasgos con los que se pretende diferenciar el chat de la aplicación es la transición entre estas dos vistas. Para ello, se permite al usuario escoger entre tres transiciones personalizadas: *Flip, Explossion* y *Shrink.*

- *Flip:* En esta transición, las celdas se van desplazando alternativamente a derecha e izquierda fuera de la pantalla, dejando únicamente la celda elegida. A continuación, la celda se ensancha hasta ocupar la pantalla y aparece la conversación.
- *Explossion:* En la segunda, la celda elegida simula acercarse al usuario, cayendo después y provocando que el resto de celdas se precipiten a la parte inferior de la pantalla. Una vez que la celda se queda sola, se ensancha hasta ocupar la pantalla y muestra la conversación.
- *Shrink:* En la tercera y última transición, todas las celdas se encogen hacia el fondo de la vista, dejando sólo la celda seleccionada. Cuando se queda sola, se ensancha dejando ver la conversación correspondiente.

4.6.2. Default Color Event

En la segunda sección, el usuario podrá elegir el color por defecto con el que van a inicializarse todos los eventos que cree o a los que le inviten. La aplicación va a permitir seleccionar uno entre los 24 colores posibles que puede escoger el usuario, ya mencionados en el apartado 4.3.1. Una vez marcado, todos los eventos que no tengan un color definido por el usuario, serán modificados a este color.

4.6.3. Theme Color

Otro parámetro variable es el color predominante en toda la aplicación. Como se muestra de la figura 3.5 hasta la 3.8, la aplicación puede tener cuatro colores diferentes. Este color va a marcar todos los detalles genéricos en toda la aplicación, como son el color de la barra superior, el fondo de los iconos de la barra inferior, el color de los textos… De este modo, el usuario podrá personalizar la aplicación a su gusto. Los cuatro colores ofrecidos son *Steel Blue, Spring Green, Indian Red* y *Midnight Blue.*

Junto con esta opción existe una segunda alternativa, también configurable desde esta pestaña. El usuario puede escoger que el fondo de la mayoría de las vistas de la aplicación sea blanco, o por el contrario, que sean del color escogido como tema. Para ello, dentro de este apartado, existe una segunda sección en la que escoger entre *«Light»* o «*Dark»*, dependiendo si escoge el fondo blanco o coloreado respectivamente.

4.6.4. *Contact Info*

En esta sección se ofrece el mail del desarrollador y la versión en la que se encuentra la aplicación. Este apartado es muy importante, puesto que ofrece un camino al usuario para poder proponer futuras mejoras, al igual que posibles errores que existan en la aplicación —*bugs*—.

4.6.5. *Log Out*

Por último, si el usuario quisiera salir de la aplicación podrá hacerlo desde el botón preparado para ello.

4.6.6. NSUserDefaults

Todas estas variables, son en sí valores predeterminados del usuario. El entorno de desarrollo de *iOS* permite gestionar estos valores a través de la clase *NSUserDefaults,* como ya se explicó en el apartado 4.3.1.

En este caso se van a almacenar en esta clase dedicada, tanto el tipo de transición que el usuario escoja como el tema que más le guste para personalizar la aplicación. Igualmente, también es necesario guardar el color por defecto para los nuevos eventos que vayan apareciendo. Para poder almacenar cualquier dato compatible con la clase *NSUserDefaults* únicamente hace falta asociarlo a una clave y lanzar un método que guardará la información en la memoria.

Todos los datos almacenados serán recuperados cuando se lance la aplicación para mantener la consistencia con los valores escogidos por el usuario.

4.7. Agenda Multiselección

En varios puntos de la memoria se ha comentado que para añadir usuarios tanto a grupos como a los eventos se va a recurrir a una agenda de contactos que permitiera la multiselección.

El sistema operativo *iOS* provee una *API* sencilla con la que poder acceder a la agenda de contactos nativa del *iPhone* y seleccionar desde ahí un contacto. Sin embargo, no permite seleccionar varios a la vez, por lo que si se quisiera añadir a más de uno, sería necesario abrir la agenda tantas veces como contactos se quisieran.

Como se ha especificado varias veces, la aplicación pretende basarse en acciones rápidas y dinámicas, y este comportamiento de la agenda de *iOS* contradice completamente estas bases. Es por ello que en el desarrollo de este proyecto se ha integrado un sistema para poder seleccionar varios contactos a la vez.

Es importante destacar que esta parte del proyecto no ha sido desarrollada por el estudiante, si no que únicamente se ha adaptado a las necesidades que tenía este proyecto e incorporado posteriormente a la aplicación. Aun así, no ha sido tarea fácil, ya que se trataba de un código bastante extenso y desarrollado para versiones anteriores a las que se ha programado toda la aplicación, por lo que también ha sido necesario actualizarlo.

En sí, la agenda multiselección se trata de una agenda muy parecida a la nativa de *iOS*, cuya principal diferencia es que al seleccionar un contacto, en vez de volver a la pantalla principal, se queda seleccionado a la espera de que el usuario pulse el botón «*Done*», en la parte inferior de la pantalla.

Para poder obtener a los contactos del terminal, la aplicación debe mostrar un mensaje al usuario para que permita que la aplicación acceda a la agenda. Una vez permitido, la aplicación recorrerá toda la agenda gracias a los métodos que proporciona la *API* e irá almacenándolos para luego mostrarlos como muestra la figura 4.30. Para poder obtener los contactos únicamente hay que emplear un objeto del tipo *ABAddressBook*, el cual se encarga de gestionar toda la agenda. Con el objeto creado, la aplicación realizará una copia de todos los contactos de la agenda. Desde esta copia se pueden recuperar todos los datos de cada contacto. En este caso, el módulo que gestiona la agenda multiselección únicamente necesitará el nombre completo y el teléfono, por lo que serán los únicos datos que se recojan.

Una vez obtenidos toda la información de los contactos, la aplicación la mostrará al usuario. Es importante destacar que, puesto que el terminal utilizado en el proyecto tiene un gran número de contactos y se encuentra algo limitado en potencia, este proceso puede tardar unos segundos. Para evitar confusiones al usuario, durante este tiempo se mostrará un icono que indicará que la aplicación está cargando los usuarios.

En el caso de que el contacto seleccionado tuviera más de un número, la agenda multiselección mostrará una segunda vista con todas las posibilidades, para que el usuario pueda escoger aquel que más le interese. Además, permite la navegación por toda la agenda de un modo rápido gracias a atajos en la barra lateral derecha y la barra de búsqueda en la parte superior. La figura 4.30 muestra la interfaz de la agenda y sus funcionalidades.

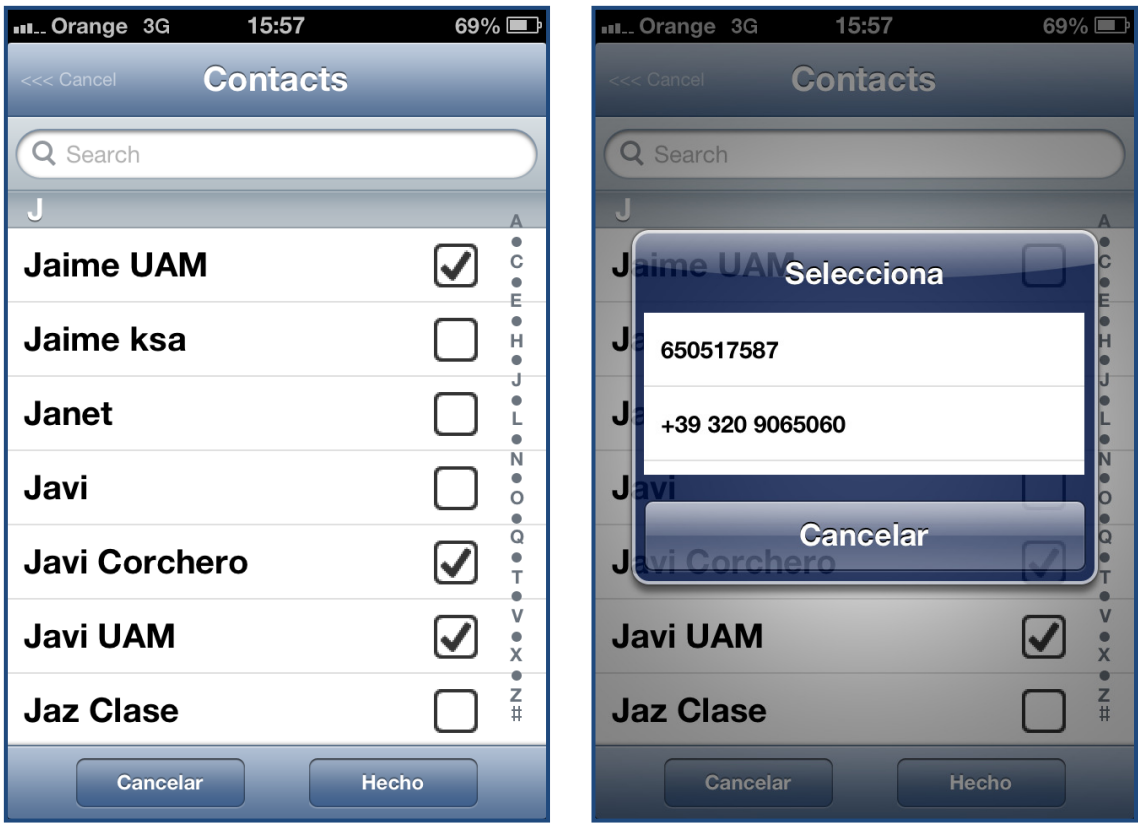


Figura 4.29 Interfaz de agenda multiselección (izda.) y vista auxiliar para elección de múltiples números (dcha.)

4.8. Servicio de notificaciones

En el apartado de chat se ha explicado que cuando un usuario recibía un mensaje, era informado a través de una notificación. También cuando es invitado a un nuevo evento o había alguna modificación, el usuario recibirá un aviso para advertirle. Todos estos mensajes se reciben gracias a que existe un servicio de notificaciones *push* que permite avisar al usuario cuando ha ocurrido algo referente a la aplicación.

La tecnología de notificaciones *push* es un tipo de comunicación en la que es el servidor el que inicia la petición al cliente —el dispositivo móvil en este caso— cuando tiene una información nueva, permitiendo un importante ahorro de recursos y tiempo respecto a la tecnología convencional *pull*, donde es el cliente quien accede constantemente a comprobar si existen nuevos datos.

Las notificaciones *push* en el sistema operativo de *Apple* siguen un ciclo de vida reflejado en el diagrama de la figura 4.31:

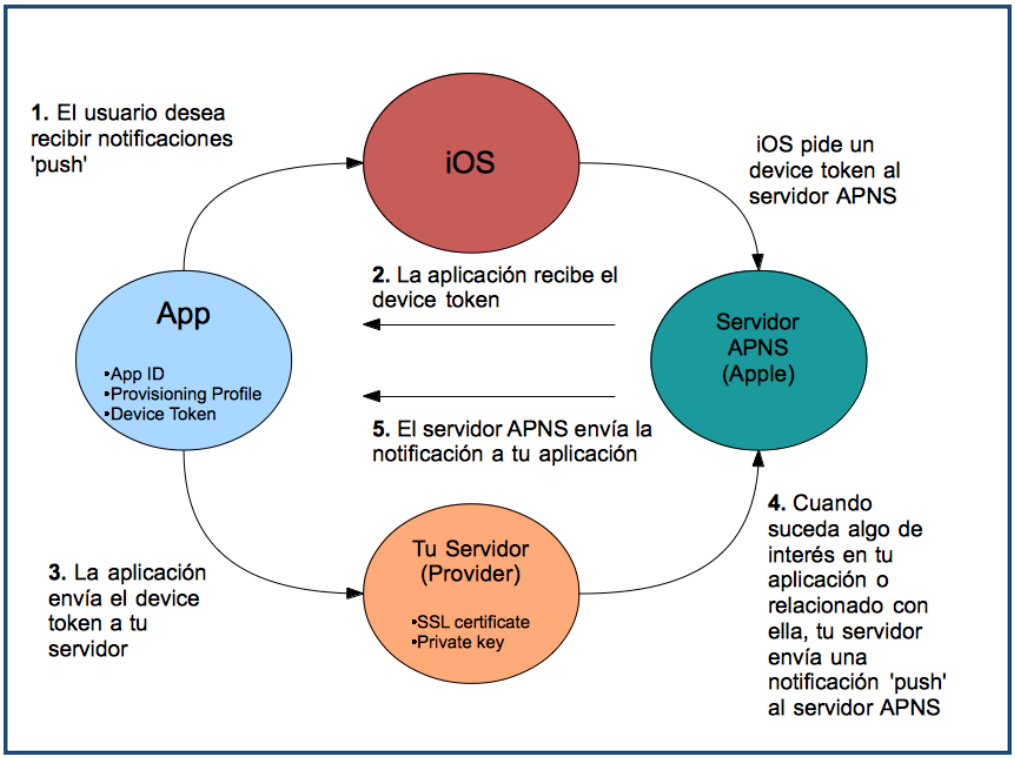


Figura 4.31 Ciclo de vida de notificaciones push

Siguiendo lo reflejado en el diagrama, lo primero que debe hacer el usuario es acceder a recibir notificaciones *push*. Una vez acceda, cada vez que un evento se cree o modifique o se mande un mensaje en el chat, el dispositivo informará a lo que en el diagrama se denomina «Tu servidor» y que en el caso de este proyecto es la plataforma *Parse.*

Aparte del almacenamiento en la Nube, *Parse* provee una *API* muy completa para la gestión de las notificaciones *push*, permitiendo incluso la integración de distintos sistemas operativos gracias a su entorno multiplataforma. Para el desarrollo de este proyecto este aspecto no es demasiado relevante, pero si se considera la posibilidad de continuar con el desarrollo en distintos sistemas operativos, el uso de la plataforma *Parse* facilita enormemente la integración de todos los usuarios de la aplicación.

Comprobando las prestaciones de esta plataforma dentro de la propia aplicación, sin llegar a ser óptimas, se adaptan bastante bien a las necesidades que tiene el proyecto. La latencia de las notificaciones *push* se encuentra normalmente entre cinco y quince/veinte segundos, aunque este dato es bastante variable y depende de muchos factores. En ocasiones cuando el servidor parecía especialmente sobrecargado, se ha llegado a valores cercanos a un minuto de latencia. Aún así, esto último se trata de casos aislados.

Para evitar este comportamiento, que sería intolerable en el caso que la aplicación llegara a un volumen importante de usuarios, *Parse* ofrece servicios con mejores prestaciones, los cuales conllevan un coste asociado. Para el desarrollo de este proyecto no es necesario contratar estos servicios, pero siempre es bueno tener la alternativa para futuras actualizaciones.

4.8.1. Implementación

Para poder enviar una notificación *push* correctamente, es necesario seguir una serie de pasos.

- El primero de todos es el registro. Es necesario obtener un certificado que otorga *Apple* y permite unirse al servidor *APNS* antes presentado. Este proceso requiere de varios pasos que no tienen especial importancia para este proyecto. Una vez registrado en el servidor *APNS*, también es necesario configurar la plataforma *Parse* y la propia aplicación para permitir mandar notificaciones.

- Con toda la configuración realizada, es necesario indicar dentro del propio código de la aplicación una serie de valores para interrelacionar todos los permisos y perfiles configurados anteriormente. Al igual que los pasos anteriores, este proceso merece una importancia más allá que ser mencionado.

  Una vez que la aplicación se encuentra preparada para mandar y recibir notificaciones *push*, el siguiente paso es introducir en el método adecuado la lógica para la gestión. En este método se identificará el motivo por el que se ha recibido esta notificación —nuevo evento, nuevo mensaje de chat...— y se actuará en consecuencia. En el caso de que el usuario reciba un nuevo mensaje, la aplicación refrescará las salas de chat, poniendo en primer lugar aquella donde se ha recibido el mensaje. Sin embargo si detecta que el usuario ya está hablando en esa sala, lo que refrescará será la conversación para que pueda leer el mensaje en cuanto lo reciba. Y así con todos los casos posibles ya explicados en apartados anteriores.

- Una vez preparada la aplicación para recibir notificaciones *push,* el siguiente paso es mandar una notificación desde la aplicación. Para esto, es necesario introducir un nuevo objeto, *PFPush*. Esta clase se encarga de gestionar todo lo relacionado con las notificaciones, permitiendo desde indicar el mensaje o el sonido que va a mostrarse cuando se reciba la aplicación, hasta seleccionar cuál o cuáles son las plataformas móviles que deberían recibir esa notificación.

  El objeto *PFPush* requiere que se le introduzca como parámetro un objeto del tipo *NSDictionary* con una estructura precisa. En este *NSDictionary*, se va a indicar al menos el mensaje, el sonido y si se debe o no incrementar el icono que indica cuantas notificaciones tiene la aplicación. Todos estos parámetros se introducirán asociados a una clave, que en este caso ya está predefinida. El mensaje irá asociado a la clave «*alert*», el sonido con la clave «*sound*» y si se debe o no incrementar el icono de notificaciones irá definido con la clave «*badge*».

  Una vez rellenado el objeto con la información de la notificación, sólo es necesario crear un objeto del tipo *PFPush*, asociarlo al *NSDictionary* creado e indicar los usuarios a los que se quiere enviar la notificación. Ya instanciado, ejecutando el método adecuado de la clase *PFPush*, se mandará la notificación.

5.Viabilidad empresarial

Al tratarse de un proyecto con un carácter empresarial, es de obligado cumplimiento realizar un estudio acerca de la viabilidad económica inicial del proyecto.

En este capítulo se pretende, sin entrar en excesivos detalles, dar una visión global de las posibilidades financieras que tiene la empresa para comenzar su actividad y las alternativas que existen para las nuevas empresas —*startups*— de carácter tecnológico.

5.1. Financiación en aplicaciones móviles

Según un estudio realizado por *Gartner*, el mercado de las aplicaciones móviles generó durante el 2013, una cantidad cercana a los 25.000 millones de dólares, un 62% más que el mismo dato tomado en el 2012. Y este dato se queda pequeño si se compara con las previsiones para los próximos años, dónde se calcula que llegara a 46.000 millones en 2016. [16]

Estas cifras son generadas a partir de cada una de las aplicaciones móviles existentes, aunque existen distintos métodos que permiten conseguir ingresos con ellas.

5.1.1. Pago por descarga

Este primer método es el modelo más rápido y convencional. Se pone un precio a la aplicación y todo aquel que quiera usarla debe pagar el precio para poder hacerlo.

La ventaja principal de este sistema es la rapidez y seguridad del cobro. Cada usuario nuevo va a repercutir positivamente en los ingresos de la aplicación, independientemente del uso que haga o el nivel de satisfacción que tenga.

Sin embargo, puesto que se le obliga a pagar «por adelantado», siempre se va a mostrar mucho más reticente a adquirir este tipo de aplicaciones. Para evitar esta incertidumbre del usuario, las empresas desarrolladoras suelen ofrecer una versión con funcionalidades limitadas para poder probar la aplicación antes de adquirirla.

Este sistema suele tener éxito en aplicaciones desarrolladas por empresas ya reconocidas en el sector, con un gran volumen de clientes y ni tan siquiera en estos casos garantiza un resultado económicamente exitoso. Existen multitud de aplicaciones de todas las categorías que tienen este modo de ingresos. Algunas de las más conocidas son *WhatsApp*, *TomTom Iberia* o *Pou*, entre otras.

5.1.2. *In-App Purchases*

También conocidas como aplicaciones *freemium*, estas aplicaciones no tienen ningún tipo de coste por descarga. Sin embargo, una vez dentro de la aplicación, se ofrecen servicios o contenidos «extra», por los cuáles sí que será necesario pagar. Dentro de este tipo de aplicaciones existen varias modalidades:

- *Item selling*: La aplicación contiene un paquete básico y si el usuario quiere aumentar las funcionalidades, debe comprar esos servicios dentro de la propia aplicación. En algunas ocasiones, el usuario puede obtenerlos invirtiendo mucho más tiempo y lo que se permite es reducirlo pagando la consecuente cantidad por ello. En este grupo entrarían aplicaciones como *CSR Racing* o *Clash of Clans*.

- *Pago por suscripción*: La aplicación ofrece servicios durante un período de tiempo a cambio de una cantidad fija. En esta modalidad, es la aplicación la que suele ser un complemento del servicio principal, que es la suscripción. De este modo, las empresas proveedoras incrementa el número de clientes al dar un servicio añadido, permitiendo a los suscriptores acceder desde su terminal móvil. Es el caso de aplicaciones como *Spotify Premium* u *Orbyt*, la cual ofrece ejemplares digitales de los principales periódicos nacionales.

5.1.3. *Publicidad en la aplicación*

Esta tercera modalidad es la más amplia y extendida, ya que abarca una gran variedad de alternativas. En este caso, la aplicación es totalmente gratuita y el usuario no tiene que abonar nada para poder usarla al completo. A cambio, la aplicación mostrará periódicamente anuncios o reservará un espacio de la pantalla para introducir *banners* —espacios fijos donde mostrar la publicidad—.

El desarrollador es libre de introducir en estos espacios cualquier tipo de publicidad, siempre que respete los códigos de ética de la plataforma de desarrollo correspondiente. Sin embargo, estas plataformas proveen módulos ya programados para mostrar la publicidad que ellas mismas se encargan de gestionar. La publicidad a través de estos módulos es la más extendida en las aplicaciones gratuitas, aunque suele ser bastante molesta para el usuario.

Por norma general, los *banners* no suelen tener ninguna relación directa con la propia aplicación. No obstante, existe otro tipo de publicidad dentro de la aplicación que se encuentra totalmente integrada con la misma. Es el caso de aplicaciones como «El Tenedor».

Esta aplicación se dedica a la gestión de reservas en restaurantes de muchos lugares tanto dentro como fuera de España. Y puesto que debe ofrecer información de éstos al usuario, la aplicación publicita de un modo mucho menos intrusivo, todos los restaurantes que gestiona. Se trata de otra manera de obtener ingresos por publicidad sin crear en el usuario una mala impresión con publicidad «agresiva». Ambos tipos de publicidad quedan reflejados en la figura 5.1.

Figura 5.1 Interfaz de aplicación con banners de publicidad (izda.) y publicidad menos intrusiva de la aplicación El Tenedor (dcha.)

5.1.4. Conclusiones

Aunque es posible que sea necesario tener un apoyo del resto de opciones, la publicidad que pretende implantarse en la aplicación es la publicidad dentro de la aplicación. Y más concretamente la publicidad que resultaba menos intrusiva, integrada con la propia aplicación.

Los usuarios no tendrán que abonar nada por descargarse la aplicación y podrán gestionar los eventos y utilizar toda la funcionalidad del chat sin ningún tipo de recargo. De este modo se fomentará que los usuarios hagan un uso frecuente de la aplicación y obtenga más usuarios a medida que pase el tiempo. Es muy probable que para conseguir un volumen de usuarios considerablemente grande, sea necesaria una campaña inicial de publicidad, la cual requerirá una financiación en los primeros pasos de la empresa. En el siguiente apartado se abordará como poder obtener esa financiación para captar usuarios.

Una vez conseguido un volumen de usuarios interesante, la intención es ofrecer los servicios de publicidad a comercios y locales en los cuáles pudieran organizarse los eventos. Esta publicidad podría tratarse desde un mero espacio dentro de la aplicación, una entrada fija en la tabla de resultados o publicidad orientada a la búsqueda realizada por el usuario. Todo esto se concretaría más adelante dependiendo de las necesidades de los clientes y los patrones de uso de los usuarios.

Del mismo modo, las opciones de pago podrán ser desde una cuota fija durante un intervalo de tiempo hasta una cantidad fija dependiendo del número de usuarios que acudan gracias a la aplicación.

Como se ha comentado, aunque es importante conseguir un volumen grande de usuarios para poder obtener una alta rentabilidad de la aplicación, gracias a la versatilidad en el pago, la aplicación podría ser rentable prácticamente desde el momento en el que se empezara a conseguir usuarios.

En los primeros pasos, cuando existen pocos usuarios registrados y la aplicación no es muy conocida, puede ofrecerse al cliente la posibilidad de que únicamente retribuyera una cantidad a la empresa cuando algún usuario hiciera una reserva desde *HiUp*. De este modo, el cliente únicamente debería devolver un pequeño porcentaje del beneficio obtenido, por lo que ambas partes resultarían beneficiadas.

Si la aplicación progresara y se obtuvieran niveles más altos de usuarios, la empresa podría aprovechar la posición de influencia que le otorga ese número de visitas para cobrar una cantidad fija a los clientes en concepto de publicidad, aparte del porcentaje inicial. Al igual que antes, este sistema de cobro dependería de circunstancias a largo plazo.

5.2. Financiación para s*tartups*

Uno de los puntos más importantes en la creación de una empresa es el capital inicial que se aporta a la misma. Durante los primeros meses de producción es muy poco probable que la empresa dé la rentabilidad esperada y un buen apoyo económico es fundamental para poder alcanzar el umbral temporal a partir del cual la empresa comienza a dar resultados.

Aparte de la financiación a través de la aplicación móvil explicada en el apartado anterior, existen distintos organismos públicos y privados que gestionan ayudas económicas para empresas nuevas —*startups*— de carácter tecnológico. Cualquier de estas ayudas podrían usarse para poder abordar el gasto inicial de la empresa. A continuación van a presentarse algunos de estos organismos, dando la información más relevante de las ayudas más adecuadas a este proyecto que ofrece cada uno de ellos.

5.2.1. Avalmadrid

Avalmadrid es una Sociedad de Garantía Recíproca cuya labor es facilitar y ayudar a la financiación de las PYMES, autónomos y emprendedores de la Comunidad de Madrid, mejorando las condiciones de financiación y/o subvención a las que pueden acceder las empresas madrileñas tanto en coste como en plazo. [17]

Dentro de las posibilidades que ofrece, existen dos que se adaptan a las características de la empresa que se pretende fundar:

- *Emprendedores*: El objetivo de esta ayuda es potenciar la creación de empresas en la Comunidad de Madrid, sobre proyectos empresariales viables y rentables en términos de generación de riqueza y empleo. Avalmadrid cuenta para ello con una línea financiera que apoya a los emprendedores promoviendo de esta manera la financiación y mejorando el acceso a préstamos de hasta 60.000€ para la ayuda en la puesta en marcha de nuevos proyectos empresariales en nuestra región. El plazo de devolución máximo son 5 años, con un tipo de interés del Euribor+3.00 %
- *Plan Impulsa Pyme*: El Plan Impulsa Pyme de Avalmadrid cuenta con productos financieros ajustados a las posibles necesidades de circulante y/o liquidez de las PYMES de la Comunidad de Madrid con unos mayores plazos y mejores tipos de interés con la correspondiente ayuda a su financiación. Las cuatro opciones más interesantes son póliza de crédito, préstamo tesorería, refinanciación y anticipo de subvenciones.

5.2.2. Ministerio de Economía y Competitividad

Además de las ayudas a nivel de la Comunidad Autónoma de Madrid, el Ministerio de Economía ofrece unas líneas de crédito de más alto nivel para casos donde se requiriera más capital. Este es el caso de las Líneas ICO Empresas y Emprendedores 2014, donde el importe máximo del crédito alcanza los 10.000.000€ [18]. En un principio esta opción podría ser demasiado ostentosa para las pretensiones de este proyecto, pero es bueno tener en cuenta alternativas para futuras posibilidades.

Las Líneas ICO Empresas y Emprendedores 2014 están orientadas a autónomos, empresas y entidades públicas y privadas, tanto españolas como extranjeras, que realicen inversiones productivas en territorio nacional. La tramitación de las operaciones se realiza directamente a través de las Entidades de Crédito.

En la figura 5.2 se observa las diferentes alternativas de financiación del Plan ICO.

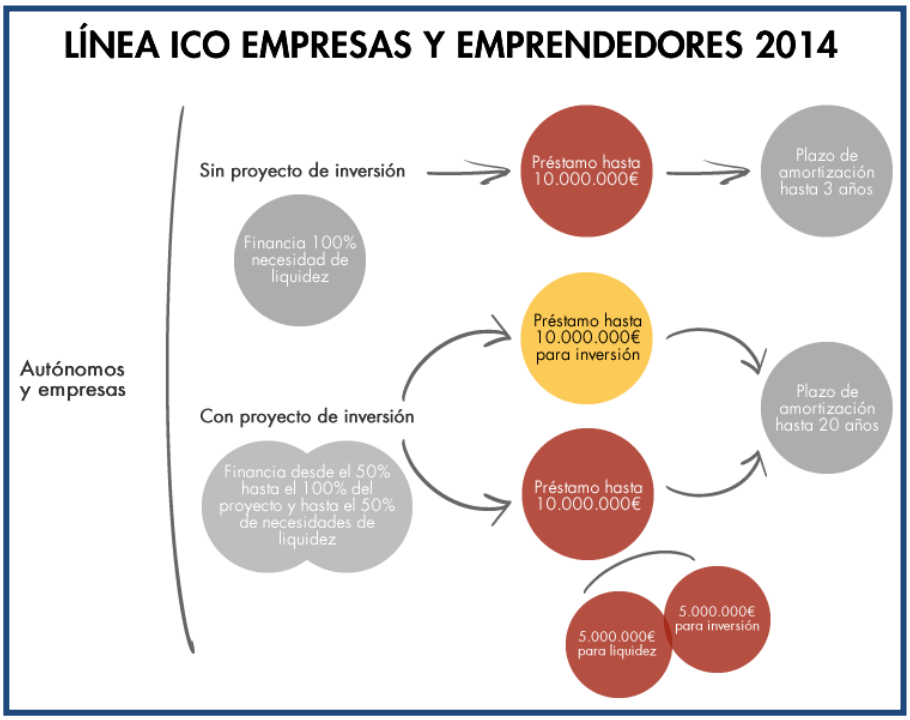


Figura 5.2 Plan de financiación Líneas ICO

5.2.3. Impulso al nuevo empleo

Más allá de la financiación inicial, existen otro tipo de ayudas que fomentan el empleo y la creación de empresas con carácter emprendedor. Es el caso de la ayuda que ofrece la Comunidad de Madrid para la «Compensación de cuotas a la Seguridad Social a emprendedores que contraten a trabajadores» [19]. Esta ayuda consiste en subvencionar las cuotas de trabajadores que, residiendo en la Comunidad de Madrid, sean contratados con empresas de carácter emprendedor.

De media, una empresa debe pagar aproximadamente un 6,4% del salario bruto de un trabajador por la cotización en la Seguridad Social. Si consideramos que el salario medio bruto de un Ingeniero de Telecomunicación se encuentra entre los 25.000 y los 30.000 euros en los primeros años de carrera profesional, esta subvención estaría reduciendo entre 1.600 y 1.920 euros anuales el coste por trabajador a la futura empresa.

De nuevo, esta ayuda no es extremadamente útil en los comienzos de una empresa, pero siempre es bueno tener en cuenta posibilidades por si el proyecto llegara a buen puerto.

6. Conclusiones y líneas futuras

6.1. Conclusiones

En este proyecto se ha presentado *HiUp*, una aplicación desarrollada para la plataforma *iOS* que permite gestionar eventos entres usuarios registrados en la aplicación. Además, se ha implementado un chat interno con el que estos usuarios podrán comunicarse sin necesidad de recurrir a otras aplicaciones. Diseñada sobre una interfaz sencilla e intuitiva, *HiUp* se adapta a cualquier usuario familiarizado con un *smartphone*.

En el capítulo 2 se hizo un estudio comparando las distintas plataformas de desarrollo y cómo la plataforma de Apple era una de las que tenía mayor potencial. Aparte se vieron un conjunto de aplicaciones que, si bien aportaron ideas para el desarrollo de *HiUp*, no conseguían abarcar todas las funciones que se pretendía.

La aplicación desarrollada cumple la estructura de diseño propuesta en el capítulo 3: permite crear grupos y editarlos para agilizar el uso de contactos dentro de la aplicación, ver y gestionar los eventos desde una pantalla independiente, personalizar la apariencia de la aplicación, invitar a usuarios no registrados a través de un mail al igual que comunicarse por un chat interno con los registrados, localizar lugares para realizar los eventos de distintos y por supuesto, crear un evento aprovechando todas las funcionalidades antes mencionadas.

Para implementar todas estas tareas, la aplicación ha basado en la plataforma *Parse* la gestión de la información en la Nube. Esta plataforma se encarga de almacenar de un modo similar a una base de datos, todos los datos que recibe de los terminales, al igual que de relacionar las distintas entidades para entrelazar la información que guardan cada una de ellas. Del mismo modo, *Parse* tiene un papel muy importante en la generación de notificaciones *push* y la gestión de usuarios.

Una vez desarrollada, en el capítulo 5 se ha planteado un modelo de negocio en torno a ella que permita monetizar los beneficios y crear, a medio plazo, una empresa de carácter innovador para el desarrollo de aplicaciones móviles.

Este modelo pretende obtener beneficios gracias a los locales y negocios dónde se celebrarían las reuniones. Para ello se plantean dos escenarios, en función del número de usuarios de la aplicación.

- El primer escenario se plantea para los comienzos, cuando no se trata de una aplicación conocida y el número de usuarios es bajo. En este caso, el modelo pretende obtener beneficio a través de un porcentaje de las reservas realizadas por los usuarios.

- En el segundo escenario, el número de usuarios es considerablemente grande, aumentando así el número de reservas. En este caso, aparte de ese porcentaje por reserva, se puede plantear ofrecer a los negocios y locales una tarifa en función de la publicidad ofrecida por la aplicación. En este caso se recurriría a una publicidad poco intrusiva para evitar una percepción negativa del usuario.

Una vez planteado el modelo de negocio, se han presentado algunos métodos de financiación para nuevas empresas que permitirían cargar con los costes iniciales que requiere la creación de una empresa dedicada a la invención y el desarrollo de aplicaciones para dispositivos móviles.

6.2. Líneas de futuro

HiUp es una aplicación base para futuras actualizaciones y con una gran cantidad de posibilidades de mejora.

Una de las más necesarias es la actualización del chat. Aunque se trata de un chat funcional que permite la conexión entre usuarios, dista mucho de los chats comerciales distribuidos actualmente. No permite la transmisión de archivos, comprobar cuando el usuario ha recibido el mensaje ni personalizar los perfiles de usuario, entre otras. Además la latencia de envío de mensajes puede llegar a ser muy alta, aunque esto podría tener fácil solución mejorando el servicio con la plataforma *Parse.*

Otra actualización posible dentro de la aplicación es la integración de los eventos con las redes sociales. Existen *APIs* de *Facebook* o *Twitter*, entre otras, que permiten publicar en los perfiles que tenga el usuario en estas redes. Dado el carácter social que posee *HiUp*, poder publicar en redes sociales mundialmente extendidas daría un gran impulso y repercusión a la aplicación.

Fuera de la aplicación, la principal mejora que requiere *HiUp* es el desarrollo multiplataforma. De nuevo, debido a su carácter social, resulta fundamental que la aplicación esté desarrollada al menos en las tres plataformas que copan el mercado actualmente. *iOS*, *Android* y *WindowsPhone*. En este proyecto sólo se ha abordado la implementación en *iOS*, por lo que haría falta migrar esta misma aplicación a los otros dos sistemas operativos.

Bibliografía

[1] Plan de Emprendedores Comunidad de Madrid [En línea]. - Mayo de 2013. - http::://www.madrid.org

[2] Madrid Emprende [En línea]. - Mayo de 2013. - http::://www.ipyme.org

[3] Línea ICO Empresas y Emprendedores 2013 [En línea]. - Mayo de 2013. - http::://www.ipyme.org

[4] **Rodríguez Delia** Conectados. La era de las redes sociales. - 25 de Abril de 2010.

[5] Doodle [En línea]. - Octubre de 2013. - http::://www.doodle.com

[6] Schedule Once [En línea]. - Octubre de 2013. - http::://www.scheduleonce.com

[7] **Global Web Index** Social Platforms. - Octubre 2013.

[8] Facebook [En línea]. - Octubre de 2013. - http::://www.facebook.com

[9] Mitmi [En línea]. - Octubre de 2013. - http::://www.mitmiapp.com

[10] Flurry Analytics [En línea]. - Septiembre de 2013. - http::://www.flurry.com

[11] Forbes [En línea]. - Septiembre de 2013. – https::://www.forbes.com

[12] Gartner [En línea]. - Septiembre de 2013. - http::://www.gartner.com

[13] **Mestras Juan Pavón** Facultad de Informática, Universidad Complutense de Madrid [En línea]. - Junio de 2013. - https://www.fdi.ucm.es/

[14] Fundamentos de Bases de Datos [En línea]. - Junio de 2013. - http::://fundamentosdebasededatosmaribelduenasroa.wikispaces.com

[15] Parse [En línea]. - 2012-2014. - http::://www.parse.com

[16] Madri+d [En línea]. - Abril de 2013. - http://www.madrimasd.org/

[17] AvalMadrid [En línea]. - Enero de 2014. - http://www.avalmadrid.es/

[18] ICO [En línea]. - Enero de 2014. - http://www.ico.es/web/ico/ico-empresas-y-emprendedores

[19] Madrid [En línea]. - Enero de 2014. - http://www.madrid.org/

[20] Número de aplicaciones disponibles por sistema operativo [En línea]. - http://andro4all.com/2013/01/google-play-supera-app-store

[21] División de países según uso de aplicaciones [En línea]. - http://blog.flurry.com/bid/94447/The-New-Global-App-Market

[22] Uso de aplicaciones de estilo de vida [En línea]. - http://blog.flurry.com/bid/94447/The-New-Global-App-Market

[23] Uso de aplicaciones de productividad [En línea]. - http://blog.flurry.com/bid/94447/The-New-Global-App-Market

[24] Predicción Market Share de los sistemas operativos [En línea]. - http://www.forbes.com/

[25] Modelo Vista Controlador [En línea]. - http://alexcytek.blogspot.com/2013/05/como-crear-una-aplicacion-crud-con.html

[26] **Apple Developer** Guía de diseño de navegación [Informe].

[27] **Apple Developer** Guía de diseño de navegación [Informe].

[28] **Apple Developer** Guía de diseño de navegación [Informe].

APÉNDICE A

Presupuesto

1. Ejecución Material
   - Compra de ordenador personal 1.400 €
   - Total de ejecución material 1.400 €
2. Gastos generales
   - 21 % sobre Ejecución Material 294 €
3. Beneficio Industrial
   - 6 % sobre Ejecución Material 84 €
4. Honorarios Proyecto
   - 1.300 horas a 15 € / hora 19.500 €
5. Material fungible
   - Gastos de impresión 60 €
   - Encuadernación 20 €
6. Subtotal del presupuesto
   - Subtotal Presupuesto 21.358 €
7. I.V.A. aplicable
   - 21 % Subtotal Presupuesto 4.485.18 €
8. Total presupuesto
   - Total Presupuesto 25.843,18 €

Madrid, Marzo de 2014

El Ingeniero Jefe de Proyecto

Fdo.: Jesús Ruiz Quijano

Ingeniero Superior de Telecomunicación

APÉNDICE B

Pliego de condiciones

Este documento contiene las condiciones legales que guiarán la realización, en este proyecto, de la «Creación de empresa dedicada a la invención y desarrollo de dispositivos móviles». En lo que sigue, se supondrá que el proyecto ha sido encargado por una empresa cliente a una empresa consultora con la finalidad de realizar dicho sistema. Dicha empresa ha debido desarrollar una línea de investigación con objeto de elaborar el proyecto. Esta línea de investigación, junto con el posterior desarrollo de los programas está amparada por las condiciones particulares del siguiente pliego.

Supuesto que la utilización industrial de los métodos recogidos en el presente proyecto ha sido decidida por parte de la empresa cliente o de otras, la obra a realizar se regulará por las siguientes:

Condiciones generales:

1. La modalidad de contratación será el concurso. La adjudicación se hará, por tanto, a la proposición más favorable sin atender exclusivamente al valor económico, dependiendo de las mayores garantías ofrecidas. La empresa que somete el proyecto a concurso se reserva el derecho a declararlo desierto.

2. El montaje y mecanización completa de los equipos que intervengan será realizado totalmente por la empresa licitadora.

3. En la oferta, se hará constar el precio total por el que se compromete a realizar la obra y el tanto por ciento de baja que supone este precio en relación con un importe límite si este se hubiera fijado.

4. La obra se realizará bajo la dirección técnica de un Ingeniero Superior de Telecomunicación, auxiliado por el número de Ingenieros Técnicos y Programadores que se estime preciso para el desarrollo de la misma.

5. Aparte del Ingeniero Director, el contratista tendrá derecho a contratar al resto del personal, pudiendo ceder esta prerrogativa a favor del Ingeniero Director, quien no estará obligado a aceptarla.

6. El contratista tiene derecho a sacar copias a su costa de los planos, pliego de condiciones y presupuestos. El Ingeniero autor del proyecto autorizará con su firma las copias solicitadas por el contratista después de confrontarlas.

7. Se abonará al contratista la obra que realmente ejecute con sujeción al proyecto que sirvió de base para la contratación, a las modificaciones autorizadas por la superioridad o a las órdenes que con arreglo a sus facultades le hayan comunicado por escrito al Ingeniero Director de obras siempre que dicha obra se haya ajustado a los preceptos de los pliegos de condiciones, con arreglo a los cuales, se harán las modificaciones y la valoración de las diversas unidades sin que el importe total pueda exceder de los presupuestos aprobados. Por consiguiente, el número de unidades que se consignan en el proyecto o en el presupuesto, no podrá servirle de fundamento para entablar reclamaciones de ninguna clase, salvo en los casos de rescisión.

8. Tanto en las certificaciones de obras como en la liquidación final, se abonarán los trabajos realizados por el contratista a los precios de ejecución material que figuran en el presupuesto para cada unidad de la obra.

9. Si excepcionalmente se hubiera ejecutado algún trabajo que no se ajustase a las condiciones de la contrata pero que sin embargo es admisible a juicio del Ingeniero Director de obras, se dará conocimiento a la Dirección, proponiendo a la vez la rebaja de precios que el Ingeniero estime justa y si la Dirección resolviera aceptar la obra, quedará el contratista obligado a conformarse con la rebaja acordada.

10. Cuando se juzgue necesario emplear materiales o ejecutar obras que no figuren en el presupuesto de la contrata, se evaluará su importe a los precios asignados a otras obras o materiales análogos si los hubiere y cuando no, se discutirán entre el Ingeniero Director y el contratista, sometiéndolos a la aprobación de la Dirección. Los nuevos precios convenidos por uno u otro procedimiento, se sujetarán siempre al establecido en el punto anterior.

11. Cuando el contratista, con autorización del Ingeniero Director de obras, emplee materiales de calidad más elevada o de mayores dimensiones de lo estipulado en el proyecto, o sustituya una clase de fabricación por otra que tenga asignado mayor

precio o ejecute con mayores dimensiones cualquier otra parte de las obras, o en general, introduzca en ellas cualquier modificación que sea beneficiosa a juicio del Ingeniero Director de obras, no tendrá derecho sin embargo, sino a lo que le correspondería si hubiera realizado la obra con estricta sujeción a lo proyectado y contratado.

12. Las cantidades calculadas para obras accesorias, aunque figuren por partida alzada en el presupuesto final —general—, no serán abonadas sino a los precios de la contrata, según las condiciones de la misma y los proyectos particulares que para ellas se formen, o en su defecto, por lo que resulte de su medición final.

13. El contratista queda obligado a abonar al Ingeniero autor del proyecto y director de obras así como a los Ingenieros Técnicos, el importe de sus respectivos honorarios facultativos por formación del proyecto, dirección técnica y administración en su caso, con arreglo a las tarifas y honorarios vigentes.

14. Concluida la ejecución de la obra, será reconocida por el Ingeniero Director que a tal efecto designe la empresa.

15. La garantía definitiva será del 4% del presupuesto y la provisional del 2%.

16. La forma de pago será por certificaciones mensuales de la obra ejecutada, de acuerdo con los precios del presupuesto, deducida la baja si la hubiera.

17. La fecha de comienzo de las obras será a partir de los 15 días naturales del replanteo oficial de las mismas y la definitiva, al año de haber ejecutado la provisional, procediéndose si no existe reclamación alguna, a la reclamación de la fianza.

18. Si el contratista al efectuar el replanteo, observase algún error en el proyecto, deberá comunicarlo en el plazo de quince días al Ingeniero Director de obras, pues transcurrido ese plazo será responsable de la exactitud del proyecto.

19. El contratista está obligado a designar una persona responsable que se entenderá con el Ingeniero Director de obras, o con el delegado que éste designe, para todo

relacionado con ella. Al ser el Ingeniero Director de obras el que interpreta el proyecto, el contratista deberá consultarle cualquier duda que surja en su realización.

20. Durante la realización de la obra, se girarán visitas de inspección por personal facultativo de la empresa cliente, para hacer las comprobaciones que se crean oportunas. Es obligación del contratista, la conservación de la obra ya ejecutada hasta la recepción de la misma, por lo que el deterioro parcial o total de ella, aunque sea por agentes atmosféricos u otras causas, deberá ser reparado o reconstruido por su cuenta.

21. El contratista, deberá realizar la obra en el plazo mencionado a partir de la fecha del contrato, incurriendo en multa, por retraso de la ejecución siempre que éste no sea debido a causas de fuerza mayor. A la terminación de la obra, se hará una recepción provisional previo reconocimiento y examen por la dirección técnica, el depositario de efectos, el interventor y el jefe de servicio o un representante, estampando su conformidad el contratista.

22. Hecha la recepción provisional, se certificará al contratista el resto de la obra, reservándose la administración el importe de los gastos de conservación de la misma hasta su recepción definitiva y la fianza durante el tiempo señalado como plazo de garantía. La recepción definitiva se hará en las mismas condiciones que la provisional, extendiéndose el acta correspondiente. El Director Técnico propondrá a la Junta Económica la devolución de la fianza al contratista de acuerdo con las condiciones económicas legales establecidas.

23. Las tarifas para la determinación de honorarios, reguladas por orden de la Presidencia del Gobierno el 19 de Octubre de 1961, se aplicarán sobre el denominado en la actualidad “Presupuesto de Ejecución de Contrata” y anteriormente llamado ”Presupuesto de Ejecución Material” que hoy designa otro concepto.

Condiciones particulares:

1. La empresa consultora, que ha desarrollado el presente proyecto, lo entregará a la empresa cliente bajo las condiciones generales ya formuladas, debiendo añadirse las siguientes condiciones particulares:

2. La propiedad intelectual de los procesos descritos y analizados en el presente trabajo, pertenece por entero a la empresa consultora representada por el Ingeniero Director del Proyecto.

3. La empresa consultora se reserva el derecho a la utilización total o parcial de los resultados de la investigación realizada para desarrollar el siguiente proyecto, bien para su publicación o bien para su uso en trabajos o proyectos posteriores, para la misma empresa cliente o para otra.

4. Cualquier tipo de reproducción aparte de las reseñadas en las condiciones generales, bien sea para uso particular de la empresa cliente, o para cualquier otra aplicación, contará con autorización expresa y por escrito del Ingeniero Director del Proyecto, que actuará en representación de la empresa consultora.

5. En la autorización se ha de hacer constar la aplicación a que se destinan sus reproducciones así como su cantidad.

6. En todas las reproducciones se indicará su procedencia, explicitando el nombre del proyecto, nombre del Ingeniero Director y de la empresa consultora.

7. Si el proyecto pasa la etapa de desarrollo, cualquier modificación que se realice sobre él, deberá ser notificada al Ingeniero Director del Proyecto y a criterio de éste, la empresa consultora decidirá aceptar o no la modificación propuesta.

8. Si la modificación se acepta, la empresa consultora se hará responsable al mismo nivel que el proyecto inicial del que resulta el añadirla.

9. Si la modificación no es aceptada, por el contrario, la empresa consultora declinará toda responsabilidad que se derive de la aplicación o influencia de la misma.

10. Si la empresa cliente decide desarrollar industrialmente uno o varios productos en los que resulte parcial o totalmente aplicable el estudio de este proyecto, deberá comunicarlo a la empresa consultora.

11. La empresa consultora no se responsabiliza de los efectos laterales que se puedan producir en el momento en que se utilice la herramienta objeto del presente proyecto para la realización de otras aplicaciones.

12. La empresa consultora tendrá prioridad respecto a otras en la elaboración de los proyectos auxiliares que fuese necesario desarrollar para dicha aplicación industrial, siempre que no haga explícita renuncia a este hecho. En este caso, deberá autorizar expresamente los proyectos presentados por otros.

13. El Ingeniero Director del presente proyecto, será el responsable de la dirección de la aplicación industrial siempre que la empresa consultora lo estime oportuno. En caso contrario, la persona designada deberá contar con la autorización del mismo, quien delegará en él las responsabilidades que ostente