Grado en Ingeniería Informática

Trabajo Final de Grado

Desarrollo de aplicación iOS utilizando webservices REST en formato JSON

Autor:
Ignacio Vicent Salvador

Supervisor:
Sergio Aguado González
Tutor académico:
Dolores María Llidó Escrivá

Fecha de lectura: 18 de julio de 2016
Curso académico 2015/2016

Resumen

Con la creación y la llegada de nuevos sistemas operativos para smartphones, emergen nuevas posibilidades que llevan a algunas empresas del sector informático a tomar ventaja de estas tendencias y generar ideas para abarcar nuevos nichos de mercado. Una de estas ideas es la creación de una aplicación para apuestas deportivas, enfocada especialmente a un sector joven de la sociedad. En este documento se describen los procesos de análisis, planificación y diseño que se han llevado a cabo para poner en marcha dicha aplicación. El desarrollo de la aplicación se ha realizado durante la estancia en prácticas en la empresa **Soluciones CuatroOchenta S.L.** [12], empresa especializada en el desarrollo de aplicaciones para smartphones. El proyecto general ha sido implementado por un equipo de desarrolladores con la metodología Scrum [9], de forma que hay un webservice restfull que provee todos los servicios necesarios. La presente memoria se centra en el trabajo desarrollado por el alumno durante su estancia en prácticas, en la creación de la aplicación iOS del sistema de apuestas, desarrollada nativamente en Swift [1] la cual consulta unos webservices. Se utiliza Jira para Scrum, Xcode es el entorno de programación y Swift el lenguaje.

Palabras clave

iOS, REST, JSON

Keywords

iOS, REST, JSON

Índice general

Índice de figuras

Índice de cuadros

Listings

Capítulo 1

Introducción

Este documento establece la memoria del proyecto **Desarrollo de aplicación iOS utilizando webservices REST [7] en formato JSON [4]** desarrollado durante la estancia en prácticas en la empresa **Soluciones CuatroOchenta S.L.** La presente memoria se desglosa en una pequeña introducción de la empresa, la descripción del proyecto realizado, la planificación seguida durante la estancia en prácticas, el análisis y diseño utilizado para llevar a cabo el proyecto, la implementación y pruebas realizadas y finalmente una conclusión por parte del alumno.

1.1. Contexto y motivación del proyecto

El proyecto se va a llevar a cabo en la empresa Soluciones Cuatroochenta S.L. ubicada en Castellón. El proyecto consta de tres partes (Interfaz y webservices al sistema de apuestas, Interfaz Android al sistema de apuestas y la Interfaz iOS al sistema de apuestas), en esta documentación se va a desarrollar la parte de Interfaz iOS al sistema de apuestas.

1.1.1. Contexto y visión general de la empresa

Este proyecto ha sido desarrollado en la empresa **Soluciones CuatroOchenta S.L.**, líder en el desarrollo de apps en España y Centroamérica, caracterizada por sus soluciones a medida para mejorar procesos de trabajo, inteligencia de negocio, movilidad, turismo, cultura, deporte y comunicación. También impulsa productos propios como **480interactive** [13], herramienta de referencia en el sector editorial para convertir publicaciones en apps interactivas, o el gestor de incidencias **Sefici** [14].

Cuatroochenta es una empresa especializada en el desarrollo integral de aplicaciones para smartphones y tabletas y programación avanzada a medida para mejorar procesos de trabajo. Cuenta con 50 empleados y más de 50 aplicaciones desarrolladas durante sus 5 años de experiencia, entre ellas se encuentran: la aplicación de las fiestas de la magdalena, la aplicación del Villarreal C.F., ASSA compañía de seguros y helados Nestlé.

Su metodología se basa en la **colaboración real** y una **altísima implicación tanto con sus clientes como con sus trabajadores**, con el objetivo de crear **apps que faciliten la vida, prácticas y eficientes**. Se basa en el trabajo y esfuerzo de su joven plantilla, consiguiendo clientes y ampliando mercados paso a paso, hasta lograr emplear a **50 personas** en dos países diferentes, entre ingenieros, informáticos, diseñadores, publicistas, comerciales y periodistas. Además de que cuenta con 6 premios gracias al éxito obtenido desde que empezó hasta la fecha actual.

El curioso nombre de la compañía está tomado de la resolución de pantalla del primer iPhone *480x320*, smartphone que abrió una nueva forma de difundir contenidos en la que después profundizaría Android.

1.1.2. Motivación del proyecto

En los últimos años el aumento del uso de *Smartphones* ha cambiado el mercado y el modo de acceso a Internet por parte de los usuarios. Internet ha pasado de usarse como una herramienta de consulta esporádica a una herramienta que interactúa en todo momento con nuestra vida diaria y nos facilita el aprendizaje, la comunicación y el acceso a servicios en Internet desde la palma de nuestra mano. Cada día son más los usuarios que disponen de un *Smartphone* con conexión a Internet, en la que cada usuario tiene de media 95 aplicaciones instaladas en su *Smartphone* y usa diariamente una media de 35 aplicaciones [11]. Según el periódico ABC [2], en España cada usuario tiene de media 39 aplicaciones instaladas y usa diariamente una media de 14 aplicaciones. Debido a este crecimiento, las empresas buscan programadores formados en el desarrollo de aplicaciones móviles, para cubrir las necesidades del mercado y de los usuarios. Por este motivo y la experiencia que tienen he elegido a la empresa **Soluciones CuatroOchenta S.L.** para la realización de mis prácticas y por el reto que supone la programación en iOS.

1.2. Objetivos del proyecto

El principal objetivo de este proyecto es realizar una aplicación nativa en iOS para el proyecto de apuestas deportivas. Para ello la aplicación de apuestas deportivas se ha desarrollado usando webservices RESTFULL, que son consultables tanto desde sistemas iOS como Android.

A los jugadores se les permitirá mediante una interfaz en el teléfono móvil realizar apuestas sin dinero sobre las distintas partidas de los juegos activos. En una partida pueden participar varios usuarios, si es una partida privada podrán participar los amigos del usuario que ha creado la partida y si es una partida pública podrán participar todos los usuarios del sistema. Los jugadores intentan predecir una serie de resultados para conseguir puntos. Una apuesta es una serie de resultados que envía cada usuario y gana la apuesta el usuario que acierte todos los resultados.

La aplicación iOS que se va a desarrollar permitirá a los usuarios ver una sección con los juegos que se juegan en los próximos días, dispondrá de una sección con todas las partidas que ya están en marcha y el estado en qué te encuentras para cada una de ellas, otra sección a modo de muro donde poder ver los últimos puntos conseguidos y una sección para poder consultar

todos tus contactos y poder invitarlos a jugar una partida. Este sistema debe permitir a los usuarios registrarse en la aplicación, modificar su perfil, añadir a sus amigos a la aplicación para poder jugar partidas con ellos y apostar en las partidas disponibles.

El objetivo principal se puede desglosar en los siguientes sub-objetivos:

- Estudio y análisis del diseño de la aplicación
- Programación de las distintas pantallas de la aplicación en iOS
- Integración con webservices para la recuperación de la información
- Programación del login y registro mediante Facebook
- Programación de las notificaciones

Capítulo 2

Descripción del proyecto

Se desea desarrollar un proyecto para realizar apuestas deportivas entre amigos, en las que no se apuesta dinero. La aplicación se va a llamar **MyBets**, que significa mis apuestas en ingles. La aplicación esta formada por competiciones, una competición tiene una o varias jornadas. Todas las jornadas tienen el mismo tipo de apuesta y una fecha tope para realizar las apuestas. Una apuesta son los resultados que ha enviado un usuario para una jornada concreta de una partida. De cada competición se pueden crear juegos, un juego son las reglas que tiene la competición, el número de jornadas, la forma de apostar y la norma de que jugador gana (por ejemplo que en caso de empate se repartan los puntos). Un juego puede tener varias partidas. Una partida puede ser pública (partidas creadas por el administrador solamente) o privada (partidas creadas por los usuarios de la aplicación contra sus amigos), si la partida es pública el usuario solo puede apostar y recibir los puntos si ha resultado ganador, si la partida es privada el usuario puede ver las apuestas realizadas por los otros usuarios para la jornada presente (siempre que el ya haya apostado) y el ranking general de la partida para ver el usuario con mas puntos (que será el que se lleve los puntos de la partida). Cada usuario puede ver el perfil de sus amigos, en el que puede ver el número total de puntos.

Las partidas están formadas por varias personas. En una primera fase del proyecto no habrán partidas públicas pero en una segunda fase se incorporarán. La forma de apostar en las partidas tiene que ser muy ágil y a poder ser sin necesidad del teclado del terminal. Al usuario le debería resultar muy rápido realizar una apuesta. Un usuario puede participar en varias partidas simultáneamente. Por ejemplo una partida con su familia a la liga española y otra con sus amigos de la pandilla al baloncesto.

Los juegos se podrán organizar: categorizados por deporte, populares, recientes, cine, videojuegos, etc. Así, cuando un usuario seleccione un juego no verá un simple listado sino que podrá encontrar un juego por sus características, ver los juegos que más se están utilizando y también los últimos juegos añadidos.

En las partidas privadas se debe poder consultar las apuestas que han realizado el resto de jugadores de la partida (una vez hayas apostado) y ver la clasificación general de esa partida. Para cada usuario que juega se debe poder ver su perfil donde poder ver cuántos puntos totales tiene y de forma rápida cómo los ha conseguido.

Los jugadores accederán al sistema mediante una interfaz iOS o Android, mientras el administrador dispondrá de una interfaz web. El proyecto que se describe en la propuesta se centra en el desarrollo de la interfaz iOS al sistema de apuestas utilizando webservices REST.

Para ello primero tendremos que realizar una análisis de requisitos conjuntamente con el cliente, después realizaremos un estudio del diseño de la aplicación y se lo entregaremos al departamento de Diseño para que nos genere el diseño de las interfaces. A continuación realizaremos toda la parte de programación iOS para llevar a cabo la aplicación. Finalmente integraremos la aplicación con webservices para poder recuperar la información.

Capítulo 3

Planificación del proyecto

3.1. Metodología de trabajo

La metodología que vamos a utilizar es la Ágil Scrum, para minimizar los riesgos durante la realización del proyecto que realizaremos de forma colaborativa. Entre las ventajas de aplicar la metodología Ágil Scrum se encuentra la productividad, calidad y que se realiza un seguimiento diario de los avances del proyecto, logrando que los participantes estén unidos, comunicados y que el cliente vaya viendo los avances.

El equipo de trabajo esta formado por:

- Scrum Manager
- Propietario del producto por homogeneizar
- Equipo Desarrollo iOS
- Equipo Desarrollo Web
- Equipo Desarrollo Android
- Equipo Diseño

En la pila de tareas del cuadro 3.1 podemos ver las historias de usuarios con sus respectivos pesos (ph), además podemos ver a que Sprint pertenece cada historia de usuario. Se han creado 7 sprints, para que cada Sprint tenga una carga de 34 horas, lo que supone una semana y media de trabajo (trabajando 5 horas diarias).

ID	PESO HISTORIA	DESCRIPCIÓN HISTORIA USUARIO	SPRINT
1	1	Registrarme en el sistema para realizar apuestas en una o varias partida	1
2	1	Acceder a mi cuenta para poder realizar apuestas en una o varias partidas	1
3	5	Ver una sección donde poder consultar todos tus contactos para poder invitarlos a jugar	1
4	3	Crear una partida de un juego con el objetivo de conseguir puntos	2
5	1	Elegir la modalidad de la competición que estoy creando para definir la partida	2
6	3	Elegir los amigos contra los que quiero jugar para poder competir contra ellos	2
7	2	Consultar los últimos juegos añadidos para poder crear partidas	2
8	5	Realizar una apuesta antes de cada jornada con el objetivo de ganar puntos o la competición	3
9	3	Ganar puntos dependiendo del resultado del juego para poder clasificarme	3
10	2	Modificar mi perfil para actualizar mi información	3
11	2	Participar en varias partidas simultáneamente para poder ganar todas las partidas posibles	4
12	3	Consultar los juegos populares para poder crear partidas	4
13	3	Consultar los juegos que se juegan en los próximos días para estar informado	4
14	2	Consultar las partidas en curso y su estado para poder saber qué partidas tengo pendientes de apostar	4
15	5	Ver las apuestas que han realizado los otros usuarios para poder ver como van en la clasificación	5
16	1	Ver un listado de los juegos filtrados por una categoría para crear partidas de la categoría que me interesen	5
17	3	Ver el perfil de mis amigos para ver como van en la partida	5
18	1	Consultar la clasificación de una partida para poder planear mi siguiente apuesta	5
19	5	Ver una sección a modo de muro donde poder ver los últimos puntos conseguidos por mi y mis amigos y los últimos juegos añadidos para tener una visión general	6
20	8	Invitar amigos a la aplicación para poder jugar contra ellos	6
21	8	Ver las notificaciones cuando acabe una partida o jornada para poder consultar mi resultado	7

Cuadro 3.1: Pila de tareas

3.2. Planificación temporal de las tareas

Respecto a la planificación, el proyecto cuenta con un total de 68 puntos de historia de usuario, dichos puntos se van a repartir a lo largo de 7 sprints que tendrán una duración total de 238 horas. Cada Sprint va a durar un total de 34 horas, por lo que cada Sprint contará con 10 puntos de historia de usuario. Como se puede ver en el cuadro 3.2, los sprints se han repartido a lo largo de varias semanas para obtener una planificación global del proyecto y obtener una fecha fin. Se supone que se va a trabajar 25 horas a la semana, por lo que cada Sprint tendrá una duración de 7 días laborables.

ID	FECHA INICIO	FECHA FIN	SPRINT
1	22/03/2016	04/04/2016	1
2	05/04/2016	13/04/2016	2
3	14/04/2016	22/04/2016	3
4	23/04/2016	03/05/2016	4
5	04/05/2016	12/05/2016	5
6	13/05/2016	23/05/2016	6
7	24/05/2016	01/06/2016	7

Cuadro 3.2: Planificación sprints

En el cuadro 3.3 podemos ver la planificación real del proyecto por quincenas.

ID	FECHA INICIO	FECHA FIN	FECHA INICIO REAL	FECHA FIN REAL	SPRINT
1	22/03/2016	04/04/2016	21/03/2016	02/04/2016	1
2	05/04/2016	13/04/2016	03/04/2016	11/04/2016	2
3	14/04/2016	22/04/2016	12/04/2016	20/04/2016	3
4	23/04/2016	03/05/2016	21/04/2016	01/05/2016	4
5	04/05/2016	12/05/2016	02/05/2016	08/05/2016	5
6	13/05/2016	23/05/2016	09/05/2016	23/05/2016	6
7	24/05/2016	01/06/2016	24/05/2016	01/06/2016	7

Cuadro 3.3: Planificación sprints

3.3. Estimación de recursos y costes del proyecto

En el proyecto va a trabajar un programador junior en la parte iOS durante 238 horas a lo largo de 3 meses, a la empresa le cuesta 21 € la hora, lo que supone un coste de 4998 €. La empresa quiere obtener un beneficio del 30 % sobre lo que le cuesta hacer el proyecto en la parte iOS, lo que supone una ganancia de 1500 €. Actualizar los conocimientos del programador junior mediante un curso cuesta 299 €. Para poder publicar la app en iOS debemos tener una licencia de desarrollador que cuesta 299 € al año. Cobramos íntegramente la licencia al cliente, porque se va a publicar con el nombre del cliente, si se utilizara la licencia de la empresa no se

cobraría íntegramente. Las herramientas y software utilizado es gratuito por lo que no supone un coste adicional.

Para la parte del servidor se necesita un programador junior durante 210 horas con un coste de 21 € la hora, lo que supone un coste de 4410 €. El servidor tiene un coste de 100 € anuales para un máximo de 10.000 usuarios, en caso de necesitar más usuarios se necesitaría un servidor más potente con un coste de 900 € anuales. Lo que supone un coste total del proyecto durante el primer año de 11606 €.

La empresa que nos contrata quiere la aplicación para obtener un valor añadido (no quiere monetizarla), por lo que su objetivo es obtener 5.000 usuarios que se descarguen y usen la aplicación. El coste de obtener cada usuario es de 1.8 €, teniendo en cuenta las campañas publicitarias de la aplicación, lo que supone un coste total de captación de clientes por 9000 €.

A continuación se muestra un pequeño resumen para aclarar los costes explicados anteriormente.

- Para la parte iOS
  - 21 € la hora durante 238 horas = 4998 €
  - Beneficio del 30 % respecto a la parte iOS = 1500 €
  - Actualizar conocimientos 299 €
  - Licencia para publicar 299 €
- Para el servidor
  - 21 € la hora durante 210 horas = 4410 €
  - 100 € coste del servidor
- Obtener 5000 usuarios
  - 1.8 € obtener cada usuario = 9000 €

Sumando el coste total del proyecto hace una inversión de 20606 € el primer año y luego 399 € al año de la licencia y el servidor, en el momento que superara los 10.000 usuarios pasaría a tener un coste anual de 1199 €. Si se cobrara la descarga de la aplicación a 5 €, con los 5.000 usuarios iniciales, la empresa cubriría los gastos del primer año y obtendría un beneficio de 4394 €.

3.4. Seguimiento del proyecto

En este apartado pondremos la tabla con la planificación real de cada Sprint e indicaremos si ha habido desviaciones o cambios en lo previsto. En caso de que hayan desviaciones o cambios explicaremos el motivo de los mismos.

3.4.1. Sprint 1

A continuación se muestra la pila del sprint 1, la cual consta de 3 historias de usuario, cada historia está descompuesta en tareas que hemos realizado para alcanzar los objetivos de la historia de usuario y nos proporcionará una estimación aproximada del tiempo que nos ha llevado implementarlo.

Código HU	Historia de usuario	Tareas	Código tarea
HU21	Yo como jugador quiero poder acceder a mi cuenta para poder realizar apuestas. (4PH)[1]	Crear la pantalla de inicio y su funcionalidad	T001
		Crear la pantalla de login con su funcionalidad	T002
HU20	Yo como jugador quiero poder registrarme en el sistema para poder realizar mis apuestas. (3PH)[2]	Diseñar y crear la pantalla de registro con su funcionalidad	T003
HU17	Yo como jugador quiero poder ver una sección donde poder consultar todos mis contactos. (5PH)	Diseñar y crear la pantalla de amigos con su funcionalidad	T004

Cuadro 3.4: Pila sprint 1

Este primer sprint ha costado 5 ph (17 horas) más de lo planificado, debido a que han habido cambios en los requisitos y se ha implementado una funcionalidad de la segunda versión de la aplicación.

3.4.2. Sprint 2

En la pila de abajo podemos ver el sprint 2, la cual consta de 4 historias de usuario, cada historia está descompuesta en tareas que hemos realizado para alcanzar los objetivos de la historia de usuario y nos proporcionará una estimación aproximada del tiempo que nos ha llevado implementarlo.

Este segundo sprint ha costado 1 ph (3 horas y 24 minutos) menos de lo planificado, debido a que la HU02 no se ha tenido que implementar. En la HU01 si se intenta jugar en una partida pública en la que ya participas, el sistema te muestra un aviso de que ya estás participando en dicha partida. De la HU01, si seleccionamos la opción de crear una partida privada con tus

[1]El valor de los puntos en la historia de usuario ha cambiado respecto a la planificación original. Esta historia de usuario ha costado más de lo planificado, ya que se ha implementado el inicio de sesión con *Facebook*.

[2]El valor de los puntos en la historia de usuario ha cambiado respecto a la planificación original por un cambio en los requisitos. En un principio no estaba contemplada la posibilidad de subir un fichero png como imagen de usuario, lo que ha conllevado a una investigación y una posterior implantación por parte de los 3 equipos para poder cumplir con este nuevo requisito

[3]Esta historia de usuario desaparece, debido a un cambio en los requisitos. El servidor nos devuelve las partidas con la modalidad implícita y ahora el usuario selecciona una partida u otra sabiendo la modalidad que lleva implícita cada partida

Código HU	Historia de usuario	Tareas	Código tarea
HU01	Yo como usuario quiero poder crear un juego de una competición, con el objetivo de conseguir puntos. (3PH)	Crear un pop-up donde elegir entre jugar a la partida pública o crear una partida privada con tus amigos	T005
		Comprobar que el usuario no pertenece ya a la partida (en caso de que sea pública)	T006
HU02	Yo como usuario quiero poder elegir la modalidad de la competición que estoy creando, para definir la partida. (0PH)[3]	Comprobar que las partidas devueltas por el servidor tienen una modalidad	T007
HU03	Yo como usuario quiero poder elegir los amigos contra los que poder jugar para poder competir contra ellos. (3PH)	Diseñar y crear la pantalla de elegir amigos con su funcionalidad	T008
HU09	Yo como usuario quiero poder ver las últimas competiciones añadidas para poder crear juegos con mis amigos de esas competiciones. (2PH)	Diseñar y crear la pantalla de últimas competiciones añadidas con su funcionalidad	T009

Cuadro 3.5: Pila sprint 2

amigos pasamos a la HU03 donde se pueden elegir los amigos contra los que jugar.

3.4.3. Sprint 3

En esta sección se puede ver la pila del sprint 3, la cual consta de 3 historias de usuario, cada historia está descompuesta en tareas que hemos realizado para alcanzar los objetivos de la historia de usuario y nos proporcionará una estimación aproximada del tiempo que nos ha llevado implementarlo.

Código HU	Historia de usuario	Tareas	Código tarea
HU04	Yo como usuario quiero poder realizar una apuesta antes de cada jornada con el objetivo de poder ganar puntos o la competición. (5PH)	Crear la pantalla para realizar una apuesta de tipo 1x2	T010
		Crear la pantalla para realizar una apuesta de tipo exacta	T011
		Crear el pop-up de apuesta realizada con éxito	T012
HU05	Yo como usuario quiero poder ganar puntos dependiendo del resultado de la competición para poder clasificarme. (1PH)[4]	Hacer las llamadas necesarias al webservice para tener los puntos de los usuarios actualizados	T013
HU06	Yo como usuario quiero poder modificar mi perfil para actualizar mi información. (2PH)	Crear la pantalla de perfil	T014
		Crear la pantalla de editar perfil	T015

Cuadro 3.6: Pila sprint 3

Este tercer sprint ha costado 2 ph (6 horas y 48 minutos) menos de lo planificado, debido a que la HU05 no se ha tenido que implementar, pero se ha tenido que implementar llamadas al webservice para mantener los puntos actualizados en todo momento.

Comentar que en la T015 cuando se edita el perfil, solo se manda la foto de perfil si el usuario la ha cambiado. Con esto conseguimos que el envío de los datos sea mucho más rápido y la aplicación vaya más fluida. En la T014 aparte de la información del usuario, se muestra en forma de tabla un resumen de los últimos puntos conseguidos, indicando en que partidas ha conseguido dichos puntos y la cantidad de puntos conseguidos.

Para futuras versiones, si se añaden nuevas formas de apostar, en la HU04 se deberán implementar las pantallas con los nuevos formatos de apuestas. Y a la hora de jugar en un juego, dependiendo de la modalidad, redirigir al usuario a la pantalla de apuesta correspondiente.

[4]Esta historia de usuario desaparece, debido a un cambio en los requisitos. El servidor asigna directamente los puntos a los usuarios una vez han terminado las partidas

3.4.4. Sprint 4

A continuación se muestra la pila del sprint 4, la cual consta de 4 historias de usuario, cada historia está descompuesta en tareas que hemos realizado para alcanzar los objetivos de la historia de usuario y nos proporcionará una estimación aproximada del tiempo que nos ha llevado implementarlo.

Código HU	Historia de usuario	Tareas	Código tarea
HU07	Yo como usuario quiero poder participar en varias competiciones simultáneamente para poder ganar todas las competiciones posibles. (0PH)[5]	Tratar los errores devueltos por el servidor, para dar un feedback al usuario	T016
HU08	Yo como usuario quiero poder ver las competiciones que más se están utilizando para poder crear juegos con mis amigos de esas competiciones. (3PH)	Crear la pantalla de partidas populares	T017
HU14	Yo como usuario quiero poder ver las competiciones que se juegan en los próximos días para estar informado sobre las competiciones que podría jugar. (3PH)	Crear la pantalla de apuestas, donde aparecen las competiciones que se juegan los próximos días	T018
HU15	Yo como usuario quiero poder ver las partidas que tengo en curso y su estado para poder saber qué competiciones tengo pendientes de apostar. (2PH)	Crear la pantalla mis apuestas	T019

Cuadro 3.7: Pila sprint 4

Este cuarto sprint ha costado 2 ph (6 horas y 48 minutos) menos de lo planificado, debido a que la HU07 no se ha tenido que implementar, ya que es el webservice quien controla que un usuario pueda participar en varias competiciones simultáneamente.

Comentar que en la HU08 se ha creado una pantalla de la competición *Fútbol*, debido a que es la competición en la que más usuarios participan. Así los usuarios pueden acceder a las partidas de la competición de *Fútbol* con el mínimo número de clicks. En la HU15, la pantalla mis apuestas se divide en dos bloques, el primero en el que se muestran las partidas pendientes y el segundo donde se muestran las partidas pasadas, en las que se pueden consultar las apuestas de los usuarios (siempre que hayas apostado en dicha partida) y ver el perfil de tus amigos.

[5]Esta historia de usuario desaparece, debido a un cambio en los requisitos. El servidor se encarga de que un usuario puede participar en varias competiciones simultáneamente y de devolver un error si un usuario se intenta unir a una partida que ya está jugando

3.4.5. Sprint 5

En el cuadro 3.8 podemos ver la pila del sprint 5, la cual consta de 3 historias de usuario, cada historia está descompuesta en tareas que hemos realizado para alcanzar los objetivos de la historia de usuario y nos proporcionará una estimación aproximada del tiempo que nos ha llevado implementarlo.

Código HU	Historia de usuario	Tareas	Código tarea
HU10	Yo como usuario quiero poder ver las apuestas que han realizado los otros usuarios para poder ver como van en la clasificación. (5PH)	Crear la pantalla ver apuestas	T020
HU11	Yo como usuario quiero poder ver un listado de las competiciones filtradas por una categoría para crear juegos de las categorías que me interesan. (4PH)[6]	Crear la pantalla de categorías	T021
		Crear la pantalla de subcategorías	T022
HU12	Yo como usuario quiero poder ver la clasificación de un juego para poder planear mis siguientes apuestas. (1PH)	Crear la pantalla ver clasificación	T023
HU13	Yo como usuario quiero poder ver el perfil de mis amigos para ver como van en el juego. (3PH)	Crear la pantalla ver perfil de mis amigos	T024

Cuadro 3.8: Pila sprint 5

Este quinto sprint ha costado 2 ph (6 horas y 48 minutos) más de lo planificado, debido a que la HU11 se ha tenido que implementar un filtrado de las partidas por subcategorías.

Cabe destacar que en la HU10 solo se podrán ver las apuestas que han realizado los otros usuarios si el usuario ha apostado en la partida, en caso de que no haya apostado le saldrá un mensaje de error pidiéndole que apueste primero. Además, solo se pueden ver las apuestas de los usuarios que tienes añadidos como amigo en la aplicación. En la HU13, para ver el perfil de mis amigos que están jugando en la misma partida que yo primero debemos apostar, ya que se accede desde desde la HU10 que tiene como requisito que el usuario haya apostado.

3.4.6. Sprint 6

En la pila de abajo podemos ver el sprint 6, la cual consta de 2 historias de usuario, cada historia está descompuesta en tareas que hemos realizado para alcanzar los objetivos de la historia de usuario y nos proporcionará una estimación aproximada del tiempo que nos ha llevado implementarlo.

[6]El valor de los puntos en la historia de usuario ha cambiado respecto a la planificación original por un cambio en los requisitos. En un principio no estaba contemplada la posibilidad de obtener las partidas filtrando por subcategoría, lo que ha conllevado a una investigación y una posterior implantación por parte de los 3 equipos para poder cumplir con este nuevo requisito

Código HU	Historia de usuario	Tareas	Código tarea
HU15	Yo como usuario quiero poder ver una sección a modo de muro donde poder ver los últimos puntos conseguidos por mi y mis amigos, y las últimas competiciones añadidas para tener una visión general. (5PH)	Crear la pantalla de consultar muro	T025
HU18	Yo como usuario quiero poder invitar amigos a la aplicación para poder jugar contra ellos. (8PH)	Crear la pantalla de invitar amigos	T026

Cuadro 3.9: Pila sprint 6

Cabe destacar que en esta primera versión, se muestran los amigos que tienen la aplicación instalada y los que no tienen la aplicación instalada. Pero solo se pueden añadir los amigos que tienen la aplicación instalada, para una futura versión, se podrá invitar a los amigos que no tienen la aplicación instalada enviando un email o whatsapp.

3.4.7. Sprint 7

Finalmente mostramos la pila del sprint 7, la cual consta de 1 historias de usuario, cada historia está descompuesta en tareas que hemos realizado para alcanzar los objetivos de la historia de usuario y nos proporcionará una estimación aproximada del tiempo que nos ha llevado implementarlo.

Código HU	Historia de usuario	Tareas	Código tarea
HU19	Yo como usuario quiero que se me notifique cuando acabe un juego para poder consultar mi resultado. (8PH)	Configurar el proyecto para recibir y tratar las notificaciones	T028

Cuadro 3.10: Pila sprint 7

Comentar que de momento las notificaciones se tienen que enviar de forma manual desde una página web, debido a que en esta primera versión el webservice no tiene esta funcionalidad implementada.

Capítulo 4

Diseño del sistema

Para la realización del proyecto se utiliza una arquitectura cliente/servidor [5] utilizando el patrón MVC [6], es un modelo de aplicación distribuida en el que las tareas se reparten entre los proveedores de recursos o servicios, llamados servidores, y los demandantes, llamados clientes. Un cliente realiza peticiones a otro programa, el servidor, quien le da respuesta. Esta idea también se puede aplicar a programas que se ejecutan sobre una sola computadora, aunque es más ventajosa en un sistema operativo multiusuario distribuido a través de una red de computadoras. En nuestro proyecto, los usuarios que se descarguen las aplicaciones iOS y Android serán los clientes, los cuales realizarán peticiones en formato JSON para obtener los datos necesarios de modo que la aplicación funcione correctamente, y el servidor dará la respuesta a los distintos clientes en formato JSON.

4.1. Arquitectura del sistema

Como podemos ver en la imagen 4.1, el cliente iOS hace peticiones a los distintos web services que administra el desarrollador web, en el cuerpo de las peticiones envía un JSON como el que podemos ver en la figura 4.2, el servidor una vez procesa los datos que le ha enviado el cliente iOS, genera una respuesta como la que se puede ver en la figura 4.3 para que el cliente la procese y obtenga los datos necesarios para el correcto funcionamiento de la aplicación.

La aplicación tiene un sistema de autenticación en el servidor, el cual valida un login y password (cifrada en md5) que le envía el cliente en una petición REST en formato JSON, para poder solicitar los datos necesarios para el correcto funcionamiento de la aplicación al servidor. *Representational State Transfer*, es un estilo de arquitectura software para sistemas hipermedia distribuidos como la World Wide Web.

Las sesiones en nuestro sistema se mantienen enviando al servidor siempre desde el cliente un JSON, con una estructura similar al de la figura 4.2, el cual consta de un usuario y contraseña. De tal manera que cada solicitud del cliente al servidor debe contener toda la información necesaria para comprender la solicitud, y no puede tomar ventaja de cualquier contexto almacenado en el servidor. Por lo tanto, el estado de sesión se mantiene por completo en el cliente. Con esto

Figura 4.1: Arquitectura general

nos aseguramos que el usuario que está solicitando un recurso al webservice está registrado en la aplicación y evitamos que puedan hacer un uso indebido de la misma.

En caso de que una petición resulte fallida, se le mostrará una alerta al usuario indicando que algo ha ido mal y no se puede acceder temporalmente al recurso que estaba solicitando. En caso de que el recurso no existiera, el usuario que intente acceder no se encuentre registrado, o no se tenga autorización, se mostrará un mensaje de error indicando el motivo del fallo.

Las peticiones REST en formato JSON tienen una estructura similar a la de la figura 4.2, cada petición se manda a una URL que nos proporciona el servidor con el método POST. Los parámetros de envío los encapsulamos en un objeto llamado *request*, el cual tiene como contenido un objeto JSON con dos miembros, el primer miembro es el email y el segundo la contraseña cifrada en md5 para hacer la aplicación más segura y evitar posibles escuchas.

Las respuestas del servidor tienen una estructura similar a la de la figura 4.3, la cual está compuesta por un objeto JSON que contiene tres miembros clave/valor, el primero de todos tiene como clave *data* y como valor otro objeto JSON, el cual encapsula los datos del usuario que ha hecho login, el segundo miembro clave/valor, tiene como clave *code* y como valor un entero que nos devuelve el servidor para saber si la operación se ha podido realizar correctamente o ha surgido algún error, el tercer miembro clave/valor, tiene como clave *message* y como valor un mensaje vacío si ha ido todo correcto o un mensaje explicativo del error si ha surgido algún error.

Login

URL: http://[BASE_URL]/services/mybets_login

Método: POST

Parámetros:

```
{
    "request": {
        "email": "email:String",
        "password": "password:String(md5)"
    }
}
```

Figura 4.2: Ejemplo petición JSON

Respuesta:

```
{
  "data": {
    "id": "id:Integer",
    "email": "email:String",
    "name": "nombre:String",
    "username": "nick:String",
    "phone": "teléfono:String"
  },
  "code": "SUCCESS (0) || ERROR (1)",
  "message": "STRING VACÍO || MENSAJE DE ERROR"
}
```

Figura 4.3: Ejemplo respuesta JSON

4.2. Diagrama de la arquitectura del sistema

Podemos ver el diagrama de la arquitectura de la información en la figura 4.4, lo primero que nos aparece al iniciar la aplicación es un Splash de la compañia (en este caso MyBets), el cual nos permite ir al registro o login. Si elegimos la rama del registro, deberemos introducir todos nuestros datos para poder acceder al contenido de la aplicación, o bien, acceder mediante Facebook y dar permisos a la aplicación para quedar registrado. Una vez registrado o logueado pasamos a la pantalla de Home, la cual muestra por defecto las últimas partidas públicas.

En la pantalla del Home además de cerrar sesión, podemos acceder al siguiente menú:

1. **Ver nuestro propio perfil**, nos aparecerá una pantalla con todos nuestros datos y una opción para poder ir a editar los datos. Si seleccionamos dicha opción, cambiaremos a la pantalla de modificar el perfil, donde podemos cambiar todos nuestros datos (menos el usuario y contraseña) e incluida la imagen de perfil.
2. **Ver los amigos que tenemos añadidos en la aplicación**, podremos ver todos los amigos que tenemos añadidos en la aplicación y por lo tanto estos serán contra los que podamos jugar partidas privadas. Si seleccionamos la opción de añadir amigos, iremos a dicha pantalla donde nos aparecerán primero los contactos del teléfono que tienen instalada la aplicación pero de los que aún no somos amigos y luego los contactos del teléfono que no tienen instalada la aplicación. En una futura versión, a los contactos que no tienen instalada la aplicación, se les permitirá enviar un whatsapp o email para que se descarguen la aplicación.
3. **Ver el muro**, veremos un resumen de los puntos conseguidos de nuestros amigos y los propios.
4. **Ver las últimas partidas públicas**, podremos participar en una partida pública, ver las partidas públicas por categorías y crear una partida privada, la cual nos llevará a la pantalla para elegir a los amigos contra los que queremos jugar dicha partida, una vez elegidos los amigos ya podremos apostar en la partida.
5. **Ver las partidas que tenemos pendientes y pasadas**, desde las partidas pendientes, podremos rechazar o participar en partidas que nos hayan invitado nuestros amigos, editar los nombres de las partidas para poder diferenciarlas y ver la clasificación de dicha partida. Desde las partidas pendientes y pasadas podremos ver la última apuesta que han hecho nuestros amigos para cada partida siempre y cuando hayamos apostado en esa partida, además, dentro de cada apuesta podremos ver el perfil del amigo que ha hecho la apuesta.

Si cerramos la sesión, volvemos a la pantalla del Splash, con lo que no podremos ver el contenido de la aplicación a menos que nos volvamos a registrar o loguear.

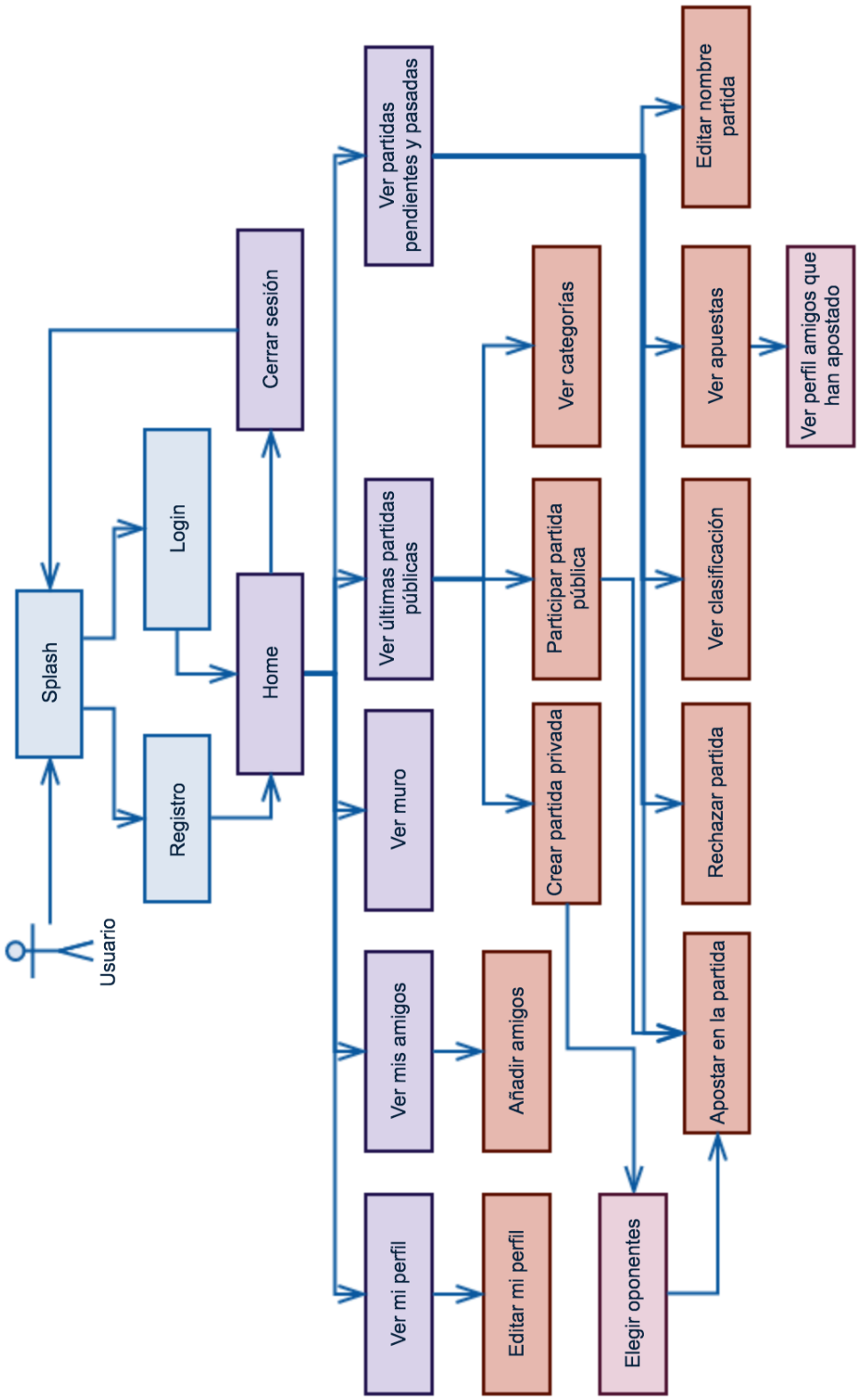

Figura 4.4: Diagrama de la arquitectura de la información

4.3. Diseño del sistema

A continuación se pueden ver los mockups realizados para llevar a cabo la aplicación.

Figura 4.5: Pantalla inicio (Splash)

Figura 4.6: Pantalla apuestas

En la figura 4.5 se puede ver la pantalla que aparece cuando inicias la aplicación por primera vez. Se puede acceder a la aplicación utilizando tu cuenta de usuario, como se puede ver en 1, también se puede acceder registrandote en la aplicación. Hay tres formas de registrarse en la aplicación, pero en esta primera versión solo se han implementado el registro mediante *Facebook* (como se puede ver en 2) y el registro mediante un formulario, como se puede ver en 4. El registro mediante sesión de correo, como se puede ver en 3, queda pendiente para futuras actualizaciones.

En la figura 4.6 se puede ver la pantalla que aparece cuando inicias la aplicación y ya estabas logueado, o la pantalla a la que te redirige la aplicación cuando haces login o te registras. En dicha pantalla podemos ver los puntos que tiene el usuario actualmente (como se puede ver en 1), también se puede ver el menú de Home (como se puede ver en 2) y que remarca en el apartado que te encuentras. Podemos visualizar las partidas mediante una serie de filtros (como se puede ver en 3), en el que se pueden ver todas las categorías, las partidas más populares independientemente de la categoría que sean y luego todas las partidas de fútbol. Como podemos ver en 4, tenemos las partidas que se juegan en los próximos días, en las cuales podemos jugar o crear una partida privada con los amigos. Finalmente en 5, tenemos un menú que persiste a lo largo de todo el Home, en el que podemos cerrar la sesión y volver a la pantalla de inicio o Splash (figura 4.5), ver nuestro perfil y configurar la aplicación, que en esta primera versión solo se podrá indicar si se desean recibir notificaciones o no.

Figura 4.7: Pantalla apostar 1x2

Figura 4.8: Pantalla perfil

En la figura 4.7 se puede ver la pantalla que aparece cuando acabas de hacer una apuesta del tipo 1x2. En 1 podemos ver el nombre de la partida a la que estamos apostando. En 2 se pueden ver los partidos a los que realizar la apuesta, marcando 1, x o 2. En 3 aparece un botón oculto que se activa una vez seleccionada una apuesta en todos los partidos. Finalmente, cuando pinchamos en el botón de OK (como se puede ver en 3), nos aparece un pop-up (como se ve en 4) con una duración de 2 segundos, indicando que la apuesta ha sido enviada y la cantidad de puntos que se pueden obtener.

En la figura 4.8 se puede ver la pantalla de perfil del usuario, la cual contiene la información del usuario y la cantidad total de puntos que tiene hasta el momento (como se puede ver en 1) y un pequeño resumen de los puntos que ha ido consiguiendo en las partidas que ha jugado.

Capítulo 5

Implementación y pruebas

5.1. Detalles de implementación

Se ha realizado una aplicación nativa iOS en el leguaje de programación Swift 2, que funciona en cualquier iPhone con una versión del sistema operativo 8.0 o superior y realiza peticiones REST.

Para la realización de la aplicación de apuestas se ha programado con el lenguaje **Swift 2**, que es un lenguaje de programación multiparadigma creado por Apple enfocado en el desarrollo de aplicaciones para iOS y Mac OS X. Se ha utilizado el entorno de programación **Xcode**, que permite simular el funcionamiento de la aplicación en los dispositivos iOS.

Para poder generar un *.ipa* de la aplicación y poder subirla al *App Store*, necesitamos registrarnos como desarrolladores en el iOS Developer Program (cuyo precio es de 99 dólares al año y que permite subir todas las aplicaciones que se quieran). El siguiente paso es la creación del certificado, el cual se tiene que crear desde el centro de desarrolladores, accediendo al iOS Provisioning Portal[8]. Una vez generado el certificado, debemos descargarlo e instalarlo, podemos comprobar en la aplicación *Llaveros* que aparece el certificado que acabamos de instalar. El último paso es el de registrar los dispositivos a los que vamos a instalar la aplicación, para ello accedemos de nuevo al portal iOS Provisioning y accedemos al apartado *Devices*, donde pulsando el botón *Add Device* podremos agregar el terminal.

Desde este momento se podrán probar las aplicaciones en el terminal directamente eligiendo el destino cuando pulsamos *Run* en el entorno de desarrollo: en vez de elegir *Simulador iPhone*, podremos elegir *iPhone de Nombre*.

5.1.1. Arquitectura REST

Para enviar y recibir un mensaje HTTP con arquitectura REST en formato JSON mediante código utilizando *Swift*, tal como aparecen en la imágen 4.2 y 4.3, es necesario la utilización del framework AFNetworking[10], el cual es una biblioteca de red para iOS que nos permite comunicarnos con el servidor. En el código 1 vemos lo necesario para poder enviar un JSON al webservice. Lo primero que tendremos que hacer, será importar el framework AFNetworking. Una vez importado el framework, lo siguiente que tenemos que crear es un gestor de sesiones, para gestionar todas las peticiones que vayamos a realizar en ese controlador al webservice. Una vez creado y configurado el gestor de sesiones, debemos crear el objeto JSON que enviaremos al webservice, el envío al webservice lo hará el gestor mediante una petición POST debido a que esperamos una respuesta del webservice. Una vez obtenemos la respuesta del webservice, debemos transformar cada objeto del JSON de respuesta en un diccionario para poder tratar los datos. Si todo ha ido correctamente, en este ejemplo se redirige al usuario a la pantalla de Home. Si ha habido un error, mostramos un mensaje en forma de alerta con el texto del error que nos devuelve el webservice (cabe destacar que el webservice devuelve los mensajes de error en lenguaje natural y de forma que cualquier usuario con conocimientos básicos puede entender). Finalmente, si no se ha podido hacer la petición POST al servidor, bien porque no disponemos de Internet o bien porque el servidor no se encuentra en funcionamiento, se imprime un mensaje avisando al usuario.

Listing 5.1: Código 1

```
1  //CÓDIGO 1
2  import AFNetworking
3
4  class LoginVC: UIViewController {
5
6    override func viewDidLoad() {
7        super.viewDidLoad()
8
9        //Creamos el gestor de sesiones
10       let manager = AFHTTPSessionManager()
11       manager.requestSerializer = AFJSONRequestSerializer()
12       manager.responseSerializer = AFJSONResponseSerializer()
13       manager.requestSerializer.cachePolicy = NSURLRequestCachePolicy.
             ReloadIgnoringCacheData
14
15       //Creamos el objeto JSON para enviarlo al servidor
16       let requestDict = NSMutableDictionary()
17       requestDict.setValue(emailTextField.text, forKey: "email")
18       requestDict.setValue(passwordTextField.text!.MD5(), forKey: "password")
19       let requestJSON = NSMutableDictionary()
20       requestJSON.setValue(requestDict, forKey: "request")
21
22       //Hacemos la petición al servidor enviando el JSON creado
23       manager.POST("http://mybetstest.cuatroochenta.com/services/mybets_login",
              parameters: requestJSON, success: { (NSURLSessionDataTask, data) ->
             Void in
24           //En data tengo el json que me ha devuelto del login y lo convertimos
                 a NSDictionary para poder leer los datos
25           let dataDict = data as! NSDictionary
26           if dataDict["code"] as! Int == 0 { //Ha ido todo bien
27               //Convertimos los datos a un NSDictionary
28               let datos = dataDict["data"] as! NSDictionary
29               //Guardamos los datos del usuario en la base de datos del
```

```
                    teléfono
30                  NSUserDefaults.standardUserDefaults().setObject(datos["id"],
                        forKey: "id")
31                  NSUserDefaults.standardUserDefaults().setObject(datos["username"
                        ], forKey: "username")
32                  NSUserDefaults.standardUserDefaults().setObject(datos["email"],
                        forKey: "email")
33                  NSUserDefaults.standardUserDefaults().synchronize()
34
35                  //Redirigimos al usuario a Home
36                  let home = self.storyboard?.
                        instantiateViewControllerWithIdentifier("
                        NavigationHomePaginator")
37                  self.presentViewController(home!, animated: true, completion: nil
                        )
38              } else {
39                  //No se ha podido iniciar la sesión. Mostramos el mensaje de
                        error que nos envía el servidor
40                  var alert = UIAlertController(title: dataDict["message"] as!
                        String, message: "", preferredStyle: UIAlertControllerStyle.
                        Alert)
41                  alert.addAction(UIAlertAction(title: "OK", style: .Default,
                        handler: nil))
42                  self.presentViewController(alert, animated: true, completion: nil
                        )
43              }
44              }, failure: { (NSURLSessionDataTask, NSError) -> Void in
45                  print(NSError)
46                  //No se ha podido enviar la petición, posible error de red.
47          })
48      }
49  }
```

5.1.2. Reglas de la arquitectura REST

REST define una serie de reglas que toda aplicación que pretenda llamarse REST debe cumplir. Estas reglas se nos dan ya dadas si vamos a usar el protocolo HTTP, como es el caso de nuestra aplicación.

- **Arquitectura cliente-servidor:** consiste en una separación clara y concisa entre los 2 agentes básicos en un intercambio de información: el cliente y el servidor. Estos 2 agentes deben ser independientes entre sí, lo que permite una flexibilidad muy alta en todos los sentidos.
- **Stateless:** esto significa que nuestro servidor no tiene porqué almacenar datos del cliente para mantener un estado del mismo. Esta limitación es sujeto de mucho debate en la industria, incluso ya empiezan a usarse tecnologías relacionadas que implementan el estado dentro de la arquitectura, como WebSockets. Como sabemos, HTTP también cumple esta norma, por lo que estamos acostumbrados ya a hacer uso de protocolos stateless.
- **Cacheable:** esta norma implica que el servidor que sirve las peticiones del cliente debe definir algún modo de cachear dichas peticiones, para aumentar el rendimiento, escalabili-

dad, etc. Una vez más, HTTP implementa esto con la cabecera *Cache-control*, que dispone de varios parámetros para controlar la cacheabilidad de las respuestas.

- **Sistema por capas:** nuestro sistema no debe forzar al cliente a saber por qué capas se tramita la información, lo que permite que el cliente conserve su independencia con respecto a dichas capas.
- **Interfaz uniforme:** esta regla simplifica el protocolo y aumenta la escalabilidad y rendimiento del sistema. No queremos que la interfaz de comunicación entre un cliente y el servidor dependa del servidor al que estamos haciendo las peticiones, ni mucho menos del cliente, por lo que esta regla nos garantiza que no importa quien haga las peticiones ni quien las reciba, siempre y cuando ambos cumplan una interfaz definida de antemano.

5.1.3. Transiciones

Debido a que la aplicación cuenta con distintas vistas, surge la necesidad de establecer una conexión entre las distintas vistas. Esta transición o navegación entre vistas recibe el nombre de segue[3] y se establece desde el fichero *StoryBoard* de nuestro proyecto. Existen distintos tipos de segue con los que podemos trabajar, algunos son válidos tanto para iPad como para iPhone, en cambio otros están pensados únicamente para las tabletas de Apple:

- **Show**: Empuja el controlador de vista de destino en la pila de navegación, desplazando el controlador de vista de origen fuera (el cambio se produce con un efecto visual de derecha a izquierda), se proporciona un botón de retroceso para navegar de vuelta a la fuente.
- **Show Detail**: Presenta el contenido en el área de detalle. Si la aplicación está mostrando una vista maestra y detalle, el nuevo contenido reemplaza el detalle actual. Si la aplicación solo está mostrando la vista maestra o detalle, el contenido reemplaza el controlador de vista de la parte superior de la pila actual.
- **Present Modally**: Se usa más comúnmente para presentar un controlador de vista que anima desde el fondo y cubre toda la pantalla en el iPhone, pero en el iPad es más común a presentarlo como una caja centrada que oscurece el controlador de vista subyacente y también anima desde el fondo. Se puede presentar el controlador de vista de varias formas diferentes, según lo que seleccionemos en la opción 'Presentation'.
- **Popup Presentation**: Cuando se usa en el iPad, el destino aparece en un pequeño popup, y tocando cualquier lugar fuera de este popup se cerraría. En el iPhone, por defecto, se presenta el controlador de vista ocupando toda la pantalla.
- **Custom**: Permite crear una transición segue personalizada entre controladores de vista.

5.1.4. Detalles implementación iOS

En este apartado voy a explicar el código utilizado en la aplicación para realizar los detalles de implementación en iOS.

Durante la aplicación necesitaremos mostrar alertas al usuario, indicando que algo ha ido mal o se ha dejado algo. Para poder mostrar una alerta en un iPhone, debemos utilizar el código 2, donde creamos una alerta con un título, mensaje y estilo de presentación. Para que el usuario pueda aceptar la alerta, creamos una acción que añadimos a la alerta, en la cual especificamos el nombre que va a tener, el estilo y de que se va a encargar o que se va a realizar una vez aceptada la alerta, si queremos que el usuario pueda rechazar la alerta, debemos crear una acción nueva. Finalmente, le indicamos al controlador actual que muestre la alerta que acabamos de crear, para que el usuario tenga una retroalimentación de lo que está ocurriendo en la aplicación.

Listing 5.2: Código 2

```
1 //CÓDIGO 2
2 //No se ha podido iniciar la sesión. Mostramos el mensaje de error que nos envía
      el servidor
3 var alert = UIAlertController(title: "Titulo de alerta", message: "Mensaje de la
     alerta", preferredStyle: UIAlertControllerStyle.Alert)
4 alert.addAction(UIAlertAction(title: "OK", style: .Default, handler: nil))
5 self.presentViewController(alert, animated: true, completion: nil)
```

Para cambiar de controlador de vista, si no hemos creado un segue, se puede realizar por código utilizando el código 3. En el ejemplo se puede ver como primero de todo debemos instanciar el storyboard, que es donde tenemos diseñadas todas las pantallas de nuestra aplicación. Una vez instanciado el storyboard, buscamos en él el controlador de vista de la pantalla que queremos mostrar, para ello antes le hemos tenido que asignar un nombre único a cada controlador de vista de nuestra aplicación. Finalmente, le decimos al controlador de vista actual que muestre la nueva pantalla y le pase el control al nuevo controlador de vista.

Listing 5.3: Código 3

```
1 //CÓDIGO 3
2 let storyboard = UIStoryboard(name: "Main", bundle: nil)
3 let perfil = self.storyboard?.instantiateViewControllerWithIdentifier("PerfilVC")
4 self.presentViewController(perfil!, animated: true, completion: nil)
```

Veamos ahora un ejemplo del código utilizado para validar los datos del login:

Listing 5.4: Código 4

```
1 //CÓDIGO 4
2  @IBAction func login(sender: AnyObject) {
3
4      if emailTextField.text == "" && passwordTextField.text == "" {
5          var alert = UIAlertController(title: "Introduce el email y la contraseña
               ", message: "", preferredStyle: UIAlertControllerStyle.Alert)
6          alert.addAction(UIAlertAction(title: "OK", style: .Default, handler: nil
               ))
7          self.presentViewController(alert, animated: true, completion: nil)
8       } else if emailTextField.text == "" {
9          var alert = UIAlertController(title: "Introduce el email", message: "",
               preferredStyle: UIAlertControllerStyle.Alert)
10         alert.addAction(UIAlertAction(title: "OK", style: .Default, handler: nil
               ))
11         self.presentViewController(alert, animated: true, completion: nil)
12     } else if passwordTextField.text == "" {
13         var alert = UIAlertController(title: "Introduce la contraseña", message:
               "", preferredStyle: UIAlertControllerStyle.Alert)
14         alert.addAction(UIAlertAction(title: "OK", style: .Default, handler: nil
               ))
15         self.presentViewController(alert, animated: true, completion: nil)
16     } else {
17         //Todos los datos son correctos, se mandan al servidor
18         let manager = AFHTTPSessionManager()
19         manager.requestSerializer = AFJSONRequestSerializer()
20         manager.responseSerializer = AFJSONResponseSerializer()
21         manager.requestSerializer.cachePolicy = NSURLRequestCachePolicy.
               ReloadIgnoringCacheData
22
23         //Creamos el objeto JSON para enviarlo al servidor
24         let requestDict = NSMutableDictionary()
25         requestDict.setValue(emailTextField.text, forKey: "email")
26         requestDict.setValue(passwordTextField.text!.MD5(), forKey: "password")
27         let requestJSON = NSMutableDictionary()
28         requestJSON.setValue(requestDict, forKey: "request")
29
30         //Guardamos la contraseña aquí, porque en JSON de respuesta no nos la
               manda
31         NSUserDefaults.standardUserDefaults().setObject(passwordTextField.text!.
               MD5(), forKey: "password")
32
33         //Hacemos la petición al servidor enviando el JSON
34         manager.POST("http://mybetstest.cuatroochenta.com/services/mybets_login"
               , parameters: requestJSON, success: { (NSURLSessionDataTask, data) ->
                Void in
35             //En data tengo el json que me ha devuelto del login y lo
                   convertimos a NSDictionary
36             let dataDict = data as! NSDictionary
37             if dataDict["code"] as! Int == 0 { //Ha ido todo bien
38                 //Convertimos los datos a un NSDictionary
39                 let datos = dataDict["data"] as! NSDictionary
40                 //Guardamos los datos del usuario
41                 NSUserDefaults.standardUserDefaults().setObject(datos["id"],
                       forKey: "id")
42                 NSUserDefaults.standardUserDefaults().setObject(datos["username"
                       ], forKey: "username")
43                 NSUserDefaults.standardUserDefaults().setObject(datos["email"],
```

```
                    forKey: "email")
44                  NSUserDefaults.standardUserDefaults().synchronize()
45                  //cambiamos de pantalla, para que el usuario pueda ver el home.
46                  let home = self.storyboard?.
                        instantiateViewControllerWithIdentifier("
                        NavigationHomePaginator")
47                  self.presentViewController(home!, animated: true, completion:
                        nil)
48              } else {
49                  //No se ha podido iniciar la sesión. Mostramos el mensaje de
                        error que nos envía el servidor
50                  var alert = UIAlertController(title: dataDict["message"] as!
                        String, message: "", preferredStyle: UIAlertControllerStyle.
                        Alert)
51                  alert.addAction(UIAlertAction(title: "OK", style: .Default,
                        handler: nil))
52                  self.presentViewController(alert, animated: true, completion:
                        nil)
53              }
54              }, failure: { (NSURLSessionDataTask, NSError) -> Void in
55                  print(NSError)
56                  //No se ha podido enviar la petición
57          })
58      } //fin else
59  } //fin botón
```

Primero realiza una comprobación para ver si el email y la contraseña están vacíos, en caso de que estén vacíos se le muestra una alerta al usuario notificandole *Introduce el email y la contraseña*. Luego comprobamos si el usuario se ha dejado el campo del email o el de la contraseña, en caso de que se haya dejado alguno de esos campos, se le muestra una alerta con el texto correspondiente para notificarle que debe introducirlo. Finalmente, si el usuario ha introducido todos los campos, creamos un objeto *JSON* que enviaremos al servidor para validar si el usuario se encuentra registrado y ha introducido correctamente los campos. Si no se encuentra registrado o algún campo no lo ha escrito correctamente, se le mostrará una aviso indicando que no está registrado o que algún campo está mal escrito. Si ha ido todo correctamente, nos guardamos los datos del usuario en la base de datos de la aplicación y lo redirigimos a la pantalla de Home.

5.2. Verificación y validación

Una vez desarrollada una pantalla, se hacen una serie de pruebas las cuales son:

- Probar la parte gráfica en el iPhone 6s con una resolución de pantalla de 1.334 por 750 pixeles.
- Probar la parte gráfica en el iPhone 4s con una resolución de pantalla de 960 x 640 pixeles.
- Probar la funcionalidad de los cambios de pantalla (en caso de que existan).
- Probar las peticiones al servidor.

- Probar el correcto funcionamiento de los botones.
- Probar que se actualizan los datos de la pantalla (en caso de que sea necesario).

Una vez la pantalla cumple con todas las pruebas descritas anteriormente, se puede considerar que la pantalla y por tanto la Historia de Usuario relacionada con esa pantalla están aceptadas. Una vez están aceptadas todas las Historias de Usuario de un Sprint, el Sprint se considera aceptado. Comentar que el equipo de Diseño se encarga de la validación de las pantallas con el cliente, por lo tanto se trabaja en todo momento sobre las pantallas validadas.

Capítulo 6

Conclusiones

El objetivo del trabajo era desarrollar una aplicación nativa en iOS para una empresa de apuestas deportivas. La aplicación debía contar con una interfaz de usuario que consultara los datos utilizando webservices REST en formato JSON, programación del login y registro mediante Facebook y la gestión de las notificaciones. Del objetivo del trabajo hemos realizado la interfaz del usuario con las consultas a los webservices REST en formato JSON, hemos programado el login y registro mediante Facebook y la gestión de las notificaciones. En esta primera versión, el login y registro con Facebook y la gestión de las notificaciones no están disponibles (pero sí implementadas), ya que por el momento, el servidor no da soporte a estas funcionalidades. Una vez finalizado el proyecto, los responsables de la aplicación han concluido que se ha llevado de forma correcta todo el trabajo planificado al principio de la estancia en prácticas, logrando los objetivos y metas marcadas.

Las principal clave de mi motivación durante el desarrollo del proyecto han sido el conocer un nuevo lenguaje de programación, el cual es cada día mas utilizado y popular en los entornos de programación. Personalmente, este proyecto me ha servido para aprender a trabajar como trabajan los programadores de una empresa real y conocer de forma más cercana las metodologías ágiles, las cuales me han permitido ser más flexible a los cambios, entregar software funcional desde el primer sprint, gestionar de forma más sencilla el proyecto y una auto organización del equipo de trabajo.

Me gustaría destacar las distintas dificultades que he tenido a la hora de programar para iOS en vez de programar para Web (como he aprendido durante la carrera). La programación para iOS es más difícil de aprender, ya que requiere unos conocimientos de programación previos y tiene una curva de aprendizaje mucho más pronunciada en comparación con los lenguajes que se utilizan para la Web (como PHP). Otra diferencia importante entre iOS y Web son los diseños, en la Web se tiene HTML, CSS y Javascript que hacen que sea bastante fácil el diseño, aplicar colores, animaciones o posicionar cualquier cosa en una página con un mínimo de esfuerzo, sin embargo, en iOS necesitamos diseñar la apariencia manualmente, indicando las posiciones de cada objeto que queramos situar en la pantalla, colores, tamaños, forma de visualización, etc., realizar un cambio simple en el diseño, a veces causa desperfectos en la aplicación sin ninguna razón aparente, teniendo que volver a ajustar todos los componentes que aparecen en la pantalla de nuevo.

Capítulo 7

Agradecimientos

La realización de este proyecto no hubiera sido posible sin el apoyo de mis familiares, que han sabido motivarme para que no dejara de trabajar día tras día. En segundo lugar, agradecer a mi tutor de estancia en prácticas (Dolores María Llidó Escrivá), por su ilusión, continuo respaldo e implicación. A CuatroOchenta S.L. por hacerme sentir uno más de la plantilla a lo largo de la estancia en prácticas. Al supervisor (Sergio Aguado González) por la oportunidad que me ha brindado, proporcionándome todo el material y soporte necesario para ampliar mis conocimientos.

Bibliografía

[1] *The Swift Programming Language*. Swift Programming Series. Apple Inc., 2014.

[2] ABC Tecnología, `http://www.abc.es/tecnologia/moviles-aplicaciones/20151020/abci-mejores-aplicaciones-espana-mundo-201510201429.html`. *Españoles: más selectivos, más fieles y demasiado confiados con las apps.* [Consulta: 4 de Julio de 2016].

[3] Apple Developer. *iOS Developer Library*. Apple Inc., `https://developer.apple.com/library/ios/recipes/xcode_help-IB_storyboard/Chapters/StoryboardSegue.html`. [Consulta: 27 de Junio de 2016].

[4] Dolores María Llidó Escrivá. Tema 4: Ajax y json. `https://aulavirtual.uji.es/pluginfile.php/3232191/mod_resource/content/1/E1036_1042_T4ajaxJson.pdf`.

[5] Fundación Wikimedia, Inc., `https://es.wikipedia.org/wiki/Cliente-servidor`. *Cliente Servidor*. [Consulta: 23 de Junio de 2016].

[6] Fundación Wikimedia, Inc., `https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador`. *Modelo-vista-controlador*. [Consulta: 18 de Julio de 2016].

[7] Fundación Wikimedia, Inc., `https://es.wikipedia.org/wiki/Representational_State_Transfer`. *Representational State Transfer*. [Consulta: 23 de Junio de 2016].

[8] Apple Inc. *Apple Developer*. `https://developer.apple.com/`. [Consulta: 8 de Marzo de 2016].

[9] proyectosagiles.org, `https://proyectosagiles.org/que-es-scrum/`. *Qué es SCRUM*. [Consulta: 18 de Julio de 2016].

[10] Scott Raymond and Mattt Thompson. *AFNetworking*. Alamofire Software Foundation, `https://github.com/AFNetworking/AFNetworking`. [Consulta: 27 de Junio de 2016].

[11] Paul Sawers. *Android Users Have an Average of 95 Apps Installed on Their Phones.* The Next Web, `http://thenextweb.com/apps/2014/08/26/android-users-average-95-apps-installed-phones-according-yahoo-aviate-data/`. [Consulta: 4 de Julio de 2016].

[12] Soluciones Cuatroochenta S.L., `http://www.cuatroochenta.com`. *CuatroOchenta WebSite*. [Consulta: 8 de Marzo de 2016].

[13] Soluciones Cuatroochenta S.L., `http://www.480interactive.com/`. *CuatroOchenta WebSite*. [Consulta: 23 de Junio de 2016].

[14] Soluciones Cuatroochenta S.L., `https://www.sefici.com/es/`. *Sefici WebSite.* [Consulta: 23 de Junio de 2016].

Glosario

HTTP Hypertext Transfer Protocol es el protocolo de comunicación que permite las transferencias de información en la World Wide Web.

iOS iOS es un sistema operativo móvil de la multinacional Apple Inc.

JSON JavaScript Object Notation, es un formato ligero de intercambio de datos. Leerlo y escribirlo es simple para humanos, mientras que para las máquinas es simple interpretarlo y generarlo.

Popup Un popup o ventana emergente, es una ventana nueva que aparece de repente en la pantalla de tu ordenador, tablet o móvil. Los popups se utilizan para dar un feedback al usuario de lo que está sucediendo en la aplicación.

REST Representational State Transfer, es un estilo de arquitectura software para sistemas hipermedia distribuidos como la World Wide Web.

Scrum Scrum es un proceso de la Metodología Ágil que se usa para minimizar los riesgos durante la realización de un proyecto, pero de manera colaborativa.

Segue Segue (pronunciado Seg-way) es la navegación/transición entre una pantalla y otra.

Splash Splash es la primera pantalla que el usuario ve cuando inicia la aplicación. La pantalla de Splash se utiliza para que el usuario pueda elegir entre distintas opciones, en este caso si acceder al login o registro.

Sprint Un Sprint es un conjunto de historias de usuario que se tienen que llevar a cabo.

Swift Swift es un lenguaje de programación creado por Apple que permite diseñar apps para iOS, Mac, el Apple TV y el Apple Watch.

Ágil Las metodologías ágiles son una serie de técnicas para la gestión de proyectos que han surgido como contraposición a los métodos clásicos de gestión.

Anexo A

Documentación peticiones JSON

A continuación se adjunta la documentación redactada para el envío y la recepción de las peticiones en formato JSON.

FORMATOS PARA FECHAS

“dd/MM/yyyy HH:mm ” = “12/03/2045 17:58”

Validar json http://www.freeformatter.com/json-validator.html

BaseURL = mybetstest.cuatroochenta.com

Login

URL: http://[BASE_URL]/services/mybets_login

Método: POST

Parámetros:

```
{
    "request": {
        "email": "email:String",
        "password": "password:String(md5)"
    }
}
```

Respuesta:

```
{
  "data": {
    "id": "id:Integer",
    "email": "email:String",
    "name": "nombre:String",
    "username": "nick:String",
    "phone": "teléfono:String"
  },
  "code": "SUCCESS (0) || ERROR (1)",
  "message": "STRING VACÍO || MENSAJE DE ERROR"
}
```

Registro

URL: http://[BASE_URL]/services/mybets_register

Método: POST

Parámetros:

```
{
    "request":
        "email": "email:String",
        "password": "contraseña:String(md5)",
        "name": "nombre:String",
        "username": "nick:String",
        "phone": "telefono:String",
        "image" : "imagen:String(Base64)"
    }
}
```

Respuesta:

```
{
  "data": {
    "id": "id:Integer",
    "email": "email:String",
    "name": "nombre:String",
    "username": "username:String",
    "phone": "telefono:String"
  },
  "code": "SUCCESS (0) || ERROR (1)",
  "message": "STRING VACÍO || MENSAJE DE ERROR"
}
```

Consultar contactos

URL: http://[BASE_URL]/services/mybets_consult_contacts

Método: POST

Parámetros:

```
{
    "request": {
        "email": "email:String",
        "password": "password:String(md5)"
    }
}
```

Respuesta:

```
{
  "data": [{
    "username": "username:String",
    "id": "id:Integer",
    "email": "email:String",
    "phone": "telefono:String",
    "name": "nombre:String",
    "imagen": "URLimagen:String"
  },]
  "code": "SUCCESS (0) || ERROR (1)",
  "message": "STRING VACÍO || MENSAJE DE ERROR"
}
```

Consultar últimas partidas

URL: http://[BASE_URL]/services/mybets_ultimas_partidas

Método: POST

Parámetros:

```
{
    "request": {
        "email": "email:String",
        "password": "password:String(md5)"
    }
}
```

Respuesta:

```
{
"data": [{
    "idpartida": "id:Integer",
    "nombrePartida": "nombre:String",
    "fecha": "fecha:String(dd/MM/yyyy HH:mm)",
    "bote": "bote:Integer",
    "numpersonas": "numpersonas:Integer",
    "urlicono": "urlicono:String",
    "coloricono": "colorrgb:String(#000000)",
    "tipopartida": "1:Integer ResultadoExacto || 0:Integer 1x2",
    "participa": "0:Integer NO || 1:Integer SI"
  }],
  "idfutbol": "id:String",
  "code": "SUCCESS (0) || ERROR (1)",
  "message": "STRING VACÍO || MENSAJE DE ERROR"
}
```

Consultar categorías

URL: http://[BASE_URL]/services/mybets_consult_categories

Método: POST

Parámetros:

```
{
    "request": {
        "email": "email:String",
        "password": "password:String(md5)"
    }
}
```

Respuesta:

```
{
"data": [{
    "icon": "URLimagen:String",
     "id": "id:Integer",
     "name": "nombre:String"
  },
{
    "icon": "URLimagen:String",
     "id": "id:Integer",
     "name": "nombre:String"
  },...],
  "code": "SUCCESS (0) || ERROR (1)",
  "message": "STRING VACÍO || MENSAJE DE ERROR"
}
```

Invitar a un amigo a la aplicación

URL: http://[BASE_URL]/services/mybets_peticion_amistad

Método: POST

Parámetros:

```
{
    "request": {
        "email": "email:String",
        "password": "password:String(md5)",
        "idamigo": "idamigo:Integer"
    }
}
```

Respuesta:

```
{
  "code": "SUCCESS (0) || ERROR (1)",
  "message": "STRING VACÍO || MENSAJE DE ERROR"
}
```

Crear una partida

URL: http://[BASE_URL]/services/mybets_crear_partida

Método: POST

Parámetros:

```
{
    "request": {
        "email": "email:String",
        "password": "password:String(md5)",
        "idpartida": "idpartida:Integer",
        "idamigos": ["idamigo:Integer", "idamigo:Integer", ...]
    }
}
```

Respuesta:

```
{
  "idjornada": "ID:Integer",
  "idpartida": "ID:Integer",
  "code": "SUCCESS (0) || ERROR (1)",
  "message": "STRING VACÍO || MENSAJE DE ERROR"
}
```

Aceptar partida

URL: http://[BASE_URL]/services/mybets_aceptar_partida

Método: POST

Parámetros:

```
{
    "request": {
        "email": "email:String",
        "password": "password:String(md5)",
        "idpartida": "idpartida:Integer"
    }
}
```

Respuesta:

```
{
  "code": "SUCCESS (0) || ERROR (1)",
  "message": "STRING VACÍO || MENSAJE DE ERROR"
}
```

Realizar apuesta (1x2, Exacta,...)

URL: http://[BASE_URL]/services/mybets_realizar_apuesta

Método: POST

Parámetros:

```
{
    "request": {
       "email": "email:String",
        "password": "password:String(md5)",
        "idpartida": "idpartida:Integer",
        "idjornada": "idjornada:Integer",
        "apuestas": ["apuesta1:Integer", "apuesta2:Integer",...]
    }
}
```

Respuesta:

```
{
  "code": "SUCCESS (0) || ERROR (1)",
  "message": "STRING VACÍO || MENSAJE DE ERROR"
}
```

Consultar Perfil

URL: http://[BASE_URL]/services/mybets_perfil

Método: POST

Parámetros:

```
{
    "request": {
        "email": "email:String",
        "password": "password:String(md5)"
    }
}
```

Respuesta:

```
{
"user": {
    "nombre":"nombre:String",
    "id": "id:Integer",
    "puntosTotales": "puntos:Integer",
    "imagen" : "URLImagen:String"
 },
"data": [
{
    "nombre": "nombrepartida:String",
    "puntos": "puntosganados:Integer",
    "fecha": "fecha:String"
  },
{
    "nombre": "nombrepartida:String",
    "puntos": "puntosganados:Integer",
    "fecha": "fecha:String"
  }]
  "code": "SUCCESS (0) || ERROR (1)",
  "message": "STRING VACÍO || MENSAJE DE ERROR"
}
```

Consultar Partidas de una categoría

URL: http://[BASE_URL]/services/mybets_partidas_categoria

Método: POST

Parámetros:

```
{
    "request": {
        "email": "email:String",
        "password": "password:String(md5)",
        "idcategoria": "idcategoria:Integer"
    }
}
```

Respuesta:

```
{
"subcategorias": [{
     "idsubcategoria": "id:Integer",
     "nombresubcategoria": "nombre:String"},
     {...}],
"data": [
{
    "idpartida": "id:Integer",
    "nombrePartida": "nombre:String",
    "fecha": "fecha:String",
    "bote": "bote:Integer",
    "numpersonas": "numpersonas:Integer",
    "urlicono": "URLIcono:String",
    "tipopartida": "0:INT 1x2 | 1:INT result ex",
    "participa": "0:INT NO | 1:INT SI"
  },
{
    "idpartida": "id:Integer",
    "nombrePartida": "nombre:String",
    "fecha": "fecha:String",
```

```
    "bote": "bote:Int",
    "numpersonas": "numpersonas:Integer",
    "urlicono": "URLIcono:String",
    "tipopartida": "0:INT 1x2 | 1:INT result ex",
    "participa": "0:INT NO | 1:INT SI"
  }...],
  "color": "color:String(#000000)",
  "code": "SUCCESS (0) || ERROR (1)",
  "message": "STRING VACÍO || MENSAJE DE ERROR"
}
```

Consultar Partidas Pendientes

URL: http://[BASE_URL]/services/mybets_partidas_pendientes

Método: POST

Parámetros:

```
{
    "request": {
        "email": "email:String",
        "password": "password:String(md5)"
    }
}
```

Respuesta:

```
{
"data": [
{
    "idpartida": "id:Integer",
    "nombrepartida": "nombre:String",
    "fecha": "fecha:String",
    "bote": "bote:Integer",
    "numpersonas": "numpersonas:Integer",
    "urlicono": "URLIcono:String",
    "coloricono": "color:String(#454545)",
    "tipopartida": "0:Integer 1x2 | 1:Integer result ex",
"estadoapuesta": "0:Integer = rechazada, 1:Integer = esperando
resultado, 2:Integer = jugar ya",
 "participa": "0:Integer NO | 1:Integer SI"
  },
{
    "idpartida": "id:Integer",
    "nombrepartida": "nombre:String",
    "fecha": "fecha:String",
    "bote": "bote:Integer",
    "numpersonas": "numpersonas:Integer",
```

```
    "urlicono": "URLIcono:String",
    "coloricono": "color:String(#454545)",
    "tipopartida": "0:Integer 1x2 | 1:Integer result ex",
"estadoapuesta": "0:Integer = rechazada, 1:Integer = esperando
resultado, 2:Integer = jugar ya",
 "participa": "0:Integer NO | 1:Integer SI"
  }...]
  "code": "SUCCESS (0) || ERROR (1)",
  "message": "STRING VACÍO || MENSAJE DE ERROR"
}
```

Consultar Partidas Pasadas

URL: http://[BASE_URL]/services/mybets_partidas_pasadas

Método: POST

Parámetros:

```
{
    "request": {
        "email": "email:String",
        "password": "password:String(md5)"
    }
}
```

Respuesta:

```
{
"data": [
{
    "idpartida": "id:Integer",
    "nombrepartida": "nombre:String",
    "puntosganados": "puntos:Integer",
    "bote": "bote:Integer",
    "numpersonas": "numpersonas:Integer",
    "urlicono": "URLIcono:String",
    "coloricono": "color:String(#454545)",
    "tipopartida": "0:Integer 1x2 | 1:Integer result ex"
  },
{
    "idpartida": "id:Integer",
    "nombrepartida": "nombre:String",
    "puntosganados": "puntos:Integer",
    "bote": "bote:Integer",
    "numpersonas": "numpersonas:Integer",
    "urlicono": "URLIcono:String",
    "coloricono": "color:String(#454545)",
```

```
    "tipopartida": "0:Integer 1x2 | 1:Integer result ex"
  },...]
  "code": "SUCCESS (0) || ERROR (1)",
  "message": "STRING VACÍO || MENSAJE DE ERROR"
}
```

Consultar Muro

URL: http://[BASE_URL]/services/mybets_consultar_muro

Método: POST

Parámetros:

```
{
    "request": {
        "email": "email:String",
        "password": "password:String(md5)"
    }
}
```

Respuesta:

```
{
"data": [
{
    "nombreamigo": "nombre:String",
    "puntosganados": "puntos:Integer",
    "nombrejuego": "nombrejuego:String",
    "fecha": "fecha:String",
    "urlfoto": "URLFoto:String"
  },
{
    "nombreamigo": "nombre:String",
    "puntosganados": "puntos:Integer",
    "nombrejuego": "nombrejuego:String",
    "fecha": "fecha:String",
    "urlfoto": "URLFoto:String"
  },...]
  "code": "SUCCESS (0) || ERROR (1)",
  "message": "STRING VACÍO || MENSAJE DE ERROR"
}
```

Cambiar Nombre Partida

URL: http://[BASE_URL]/services/mybets_cambiar_nombre_partida

Método: POST

Parámetros:

```
{
    "request": {
        "email": "email:String",
        "password": "password:String(md5)",
        "idpartida": "id:Integer",
        "nuevonombre": "nombrenuevo:String"
    }
}
```

Respuesta:

```
{
"data": [
{
   "idpartida": "id:Integer",
    "nombrepartida": "nombre:String",
    "fecha": "fecha:String",
    "bote": "bote:Integer",
    "numpersonas": "numpersonas:Integer”,
    "urlicono": "URLIcono:String",
    "coloricono": "color:String(#454545)",
    "tipopartida": "0:INT 1x2 | 1:INT result ex",
    "estadoapuesta": "0 = rechazada, 1 = esperando resultado, 2 =
jugar ya",
    "participa": "0:INT NO | 1:INT SI"

  },
{
    "idpartida": "id:Integer",
```

```
    "nombrepartida": "nombre:String",
    "fecha": "fecha:String",
    "bote": "bote:Integer",
    "numpersonas": "numpersonas:Integer”,
    "urlicono": "URLIcono:String",
    "coloricono": "color:String(#454545)",
    "tipopartida": "0:INT 1x2 | 1:INT result ex",
    "estadoapuesta": "0 = rechazada, 1 = esperando resultado, 2 =
jugar ya",
    "participa": "0:INT NO | 1:INT SI"
  },...],
  "code": "SUCCESS (0) || ERROR (1)",
  "message": "STRING VACÍO || MENSAJE DE ERROR"
}
```

Rechazar Partida

URL: http://[BASE_URL]/services/mybets_rechazar_partida

Método: POST

Parámetros:

```
{
    "request": {
        "email": "email:String",
        "password": "password:String(md5)",
        "idpartida": "id:Integer"
    }
}
```

Respuesta:

```
{
  "code": "SUCCESS (0) || ERROR (1)",
  "message": "STRING VACÍO || MENSAJE DE ERROR"
}
```

Consultar Partidas de una subcategoría

URL: http://[BASE_URL]/services/mybets_partidas_subcategoria

Método: POST

Parámetros:

```
{
    "request": {
        "email": "email:String",
        "password": "password:String(md5)",
        "idsubcategoria": "id:Integer"
    }
}
```

Respuesta:

```
{
"data": [
{
    "idpartida": "id:Integer",
    "nombrepartida": "nombre:String",
    "fecha": "fecha:String",
    "bote": "bote:Integer",
    "numpersonas": "numpersonas:Integer”,
    "urlicono": "URLIcono:String",
    "coloricono": "color:String(#454545)",
    "tipopartida": "0:INT 1x2 | 1:INT result ex",
    "participa": "0:INT NO | 1:INT SI"  },
{
    "idpartida": "id:Integer",
    "nombrepartida": "nombre:String",
    "fecha": "fecha:String",
    "bote": "bote:Integer",
    "numpersonas": "numpersonas:Int”,
    "urlicono": "URLIcono:String",
    "coloricono": "color:String(#454545)",
```

```
    "tipopartida": "0:INT 1x2 | 1:INT result ex",
    "participa": "0:INT NO | 1:INT SI"
  },...],
  "code": "SUCCESS (0) || ERROR (1)",
  "message": "STRING VACÍO || MENSAJE DE ERROR"
}
```

Consultar Datos Usuario

URL: http://[BASE_URL]/services/mybets_consultar_datos_usuario

Método: POST

Parámetros:

```
{
    "request": {
        "email": "email:String",
        "password": "password:String(md5)"
    }
}
```

Respuesta:

```
{
"data": {
   "username": "username:String",
   "name": "nombre:String",
   "email": "email:String",
   "imagen": "URLImagen:String"
},
  "code": "SUCCESS (0) || ERROR (1)",
  "message": "STRING VACÍO || MENSAJE DE ERROR"
}
```

Modificar Datos Usuario

URL: http://[BASE_URL]/services/mybets_modificar_datos_usuario

Método: POST

Parámetros:

```
{
    "request": {
        "email": "email:String",
        "password": "password:String(md5)",
        "username": "username:String",
        "name": "nombre:String",
        "image" : "imagen:Base64String"
    }
}
```

Respuesta:

```
{
  "code": "SUCCESS (0) || ERROR (1)",
  "message": "STRING VACÍO || MENSAJE DE ERROR"
}
```

Pedir Datos Jornada

URL: http://[BASE_URL]/services/mybets_pedir_datos_jornada

Método: POST

Parámetros:

```
{
    "request": {
        "email": "email:String",
        "password": "password:String(md5)",
        "idpartida": "id:Integer"
    }
}
```

Respuesta:

```
{
  "data" : {
      "idjornada": "id:Integer",
      "listapartidos": [
         { "idpartido" : "id:Integer",
           "listaequipos" : [{"id" : "id:Integer",
                        "nombre" : "nombre:String",
                        "iconurl" : "URLIcono:String"},...]
        },...]
      },
  "code": "SUCCESS (0) || ERROR (1)",
  "message": "STRING VACÍO || MENSAJE DE ERROR"
}
```

Pedir Clasificación de una partida

URL: http://[BASE_URL]/services/mybets_clasificacion

Método: POST

Parámetros:

```
{
    "request": {
        "email": "email:String",
        "password": "password:String(md5)",
        "idpartida": "id:Integer"

    }
}
```

Respuesta:

```
{
  "partida" : {"nombre": "nombre:String",
               "fecha": "fecha:String",
               "bote": "bote:Integer"
               "urlicono": "URLIcono:String",
               "colorfondo": "color:String(#454545)"},
  "data" : [
{
    "idusuario": "id:Integer",
    "nombre": "nombre:String",
    "puntos": "puntos:Integer",
    "urlicono": "URLIcono:String",
    "posicion": "numero:Integer"
  },
{
    "idusuario": "id:Integer",
    "nombre": "nombre:String",
    "puntos": "puntos:Integer",
    "urlicono": "URLIcono:String",
```

```
    "posicion": "numero:Integer"
  }...],
  "code": "SUCCESS (0) || ERROR (1)",
  "message": "STRING VACÍO || MENSAJE DE ERROR"
}
```

Consultar Partidas Populares

URL: http://[BASE_URL]/services/mybets_partidas_populares

Método: POST

Parámetros:

```
{
    "request": {
        "email": "email:String",
        "password": "password:String(md5)"
    }
}
```

Respuesta:

```
{
"data": [
{
    "idpartida": "id:Integer",
    "nombrepartida": "nombre:String",
    "fecha": "fecha:String",
    "bote": "bote:Integer",
    "numpersonas": "numeropersonas:Integer”,
    "urlicono": "URLIcono:String",
    "coloricono": "color:String(#454545)",
    "tipopartida": "0:INT 1x2 | 1:INT result ex",
    "participa": "0:INT NO | 1:INT SI"
  },
{
    "idpartida": "id:Integer",
    "nombrepartida": "nombre:String",
    "fecha": "fecha:String",
    "bote": "bote:Integer",
    "numpersonas": "numeropersonas:Integer”,
    "urlicono": "URLIcono:String",
    "coloricono": "color:String(#454545)",
```

```
    "tipopartida": "0:INT 1x2 | 1:INT result ex",
    "participa": "0:INT NO | 1:INT SI"
  },...],
  "code": "SUCCESS (0) || ERROR (1)",
  "message": "STRING VACÍO || MENSAJE DE ERROR"
}
```

Obtener Puntuación Usuario

URL: http://[BASE_URL]/services/mybets_puntuacion

Parámetros:

```
{
    "request": {
        "email": "email:String",
        "password": "password:String(md5)"
    }
}
```

Respuesta:

```
{
  "data": {
      "puntuacion": "puntuacion:Integer"
  },
  "code": "SUCCESS (0) || ERROR (1)",
  "message": "STRING VACÍO || MENSAJE DE ERROR"
}
```

Obtener subcategoría Más Usada

URL: http://[BASE_URL]/services/mybets_subcategoria_mas_usada

Parámetros:

```
{
    "request": {
        "email": "email:String",
        "password": "password:String(md5)"
    }
}
```

Respuesta:

```
{
  "data": {
      "nombresubcategoria": "nombre:String",
      "idsubcategoria": "id:Integer"
  },
  "code": "SUCCESS (0) || ERROR (1)",
  "message": "STRING VACÍO || MENSAJE DE ERROR"
}
```

Obtener Apuestas Realizadas (1x2)

URL: http://[BASE_URL]/services/mybets_apuestas_realizadas

Parámetros:

```
{
    "request": {
        "email": "email:String",
        "password": "password:String(md5)",
        "idpartida": "id:Integer"
    }
}
```

Respuesta:

```
{
 "partida" : { "fecha": "fecha:String",
               "bote": "bote:Integer"
               "urlicono": "URLIcono:String",
               "colorfondo": "colorf:String(#454545)"},
"data": [
{
    "urlperfil": "URLImagenPerfil:String",
    "idusuario": "idusuario:Integer",
    "apuestas": [{
    "nombrelocal": "nombre:String",
    "urllocal": "URLImagen:String",
    "nombrevisitante": "nombre:String",
    "urlvisitante": "URLImagen:String”,
    "apuesta": "apuesta:Integer"},
    {"nombrelocal": "nombre:String",
    "urllocal": "URLImagen:String",
    "nombrevisitante": "nombre:String",
    "urlvisitante": "URLImagen:String”,
    "apuesta": "apuesta:Integer" },...]
```

```
},
{
     "urlperfil": "URLImagenPerfil:String",
    "idusuario": "idusuario:Integer",
    "apuestas": [{
    "nombrelocal": "nombre:String",
    "urllocal": "URLImagen:String",
    "nombrevisitante": "nombre:String",
    "urlvisitante": "URLImagen:String”,
    "apuesta": "apuesta:Integer"},
    {"nombrelocal": "nombre:String",
    "urllocal": "URLImagen:String",
    "nombrevisitante": "nombre:String",
    "urlvisitante": "URLImagen:String”,
    "apuesta": "apuesta:Integer" },...]

},...],
  "code": "SUCCESS (0) || ERROR (1)",
  "message": "STRING VACÍO || MENSAJE DE ERROR"
}
```

Obtener Apuestas Realizadas (Exacto)

URL: http://[BASE_URL]/services/mybets_apuestas_realizadas

Parámetros:

```
{
    "request": {
        "email": "email:String",
        "password": "password:String(md5)",
        "idpartida": "id:Integer"
    }
}
```

Respuesta:

```
{
  "partida" : { "fecha": "fecha:String",
                "bote": "bote:Integer"
                "urlicono": "URLIcono:String",
                "colorfondo": "color:String(#454545)"},
"data": [
{
    "urlperfil": "URLImagenPerfil:String",
    "idusuario": "id:Integer",
    "apuestas": [{
    "nombrelocal": "nombre:String",
    "urllocal": "URLImagen:String",
    "nombrevisitante": "nombre:String",
    "urlvisitante": "URLImagen:String",
    "apuesta": [ScoreEquipo1:Integer, ScoreEquipo2:Integer]},
    {"nombrelocal": "nombre:String",
    "urllocal": "URLImagen:String",
    "nombrevisitante": "nombre:String",
    "urlvisitante": "URLImagen:String",
    "apuesta": [ScoreEquipo1:Integer, ScoreEquipo2:Integer] },...]
},
```

```
{
   "urlperfil": "URLImagenPerfil:String",
    "idusuario": "id:Integer",
    "apuestas": [{
    "nombrelocal": "nombre:String",
    "urllocal": "URLImagen:String",
    "nombrevisitante": "nombre:String",
    "urlvisitante": "URLImagen:String",
    "apuesta": [ScoreEquipo1:Integer, ScoreEquipo2:Integer]},
    {"nombrelocal": "nombre:String",
    "urllocal": "URLImagen:String",
    "nombrevisitante": "nombre:String",
    "urlvisitante": "URLImagen:String",
    "apuesta": [ScoreEquipo1:Integer, ScoreEquipo2:Integer] },...]
},...],
  "code": "SUCCESS (0) || ERROR (1)",
  "message": "STRING VACÍO || MENSAJE DE ERROR"
}
```

Consultar Perfil amigo

URL: http://[BASE_URL]/services/mybets_perfil_amigo

Método: POST

Parámetros:

```
{
    "request": {
        "email": "email:String",
        "password": "password:String(md5)",
        "idamigo": "id:Integer"

    }
}
```

Respuesta:

```
{
"user": {
    "nombre": "nombre:String",
    "id": "id:Integer",
    "puntosTotales": "puntos:Integer",
    "imagen" : "URLImagen:String"
},
"data": [
{
    "nombre": "nombre:String",
    "puntos": "puntos:Integer",
    "fecha": "fecha:String"
}],
  "code": "SUCCESS (0) || ERROR (1)",
  "message": "STRING VACÍO || MENSAJE DE ERROR"
}
```

Comprobar lista amigos

URL: http://[BASE_URL]/services/mybets_lista_amigos

Método: POST

Parámetros:

```
{
    "request": {
        "email": "email:String",
        "password": "password:String(md5)",
        "emails": [email:String, ... ],
        "telefonos": [telefono:String, ...]
       }
}
```

Respuesta:

```
{
"data": [
{
    "nombre": "nombre:String",
    "urlicono": "URLIcono:String",
    "idusuario": "id:Integer",
    "telefono": "telefono:String",
    "email": "email:String"
  },
{
     "nombre": "nombre:String",
    "urlicono": "URLIcono:String",
    "idusuario": "id:Integer",
    "telefono": "telefono:String",
    "email": "email:String"
  },...],
  "code": "SUCCESS (0) || ERROR (1)",
  "message": "STRING VACÍO || MENSAJE DE ERROR"
}
```

Anexo B

Manual para crear un proyecto en Xcode 7.0

A continuación vamos a explicar los pasos para crear un proyecto en Xcode 7.0, desde el inicio del proyecto hasta la creación de las pantallas y scripts. Lo primero que debemos hacer es ejecutar el Xcode y nos aparecerá una pantalla como la de la figura B.1. Seleccionamos la opción de *Create a new Xcode Project*. En la siguiente pantalla, figura B.2, Xcode nos pedirá

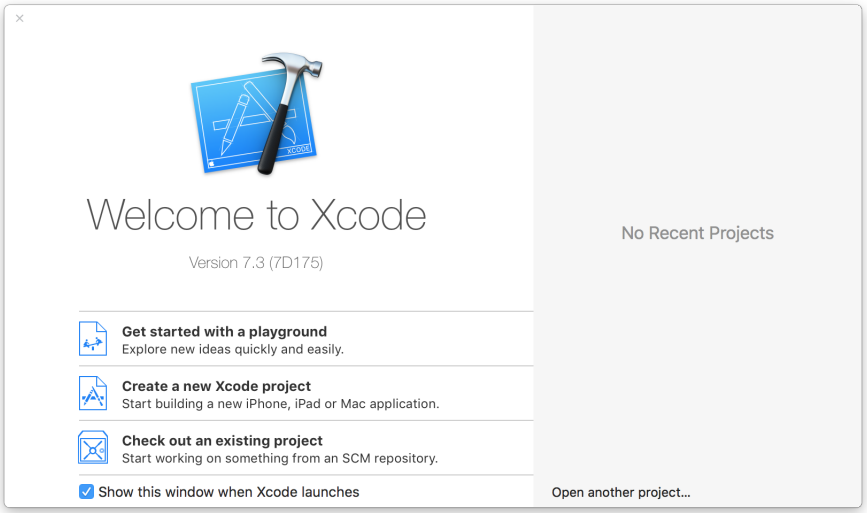

Figura B.1: Pantalla inicio Xcode

que seleccionemos una de las plantillas que existen por defecto, en nuestro caso seleccionaremos la plantilla *Single View Application*.

Xcode nos ofrece varias plantillas para empezar una aplicación, es posible cambiar la aplicación después de haber seleccionado una plantilla:

- **Master-Detail:** Puede ser usada cuando la aplicación va a tener una pantalla que tiene contenido en forma de listas o tablas y cada elemento abre una pantalla nueva con el

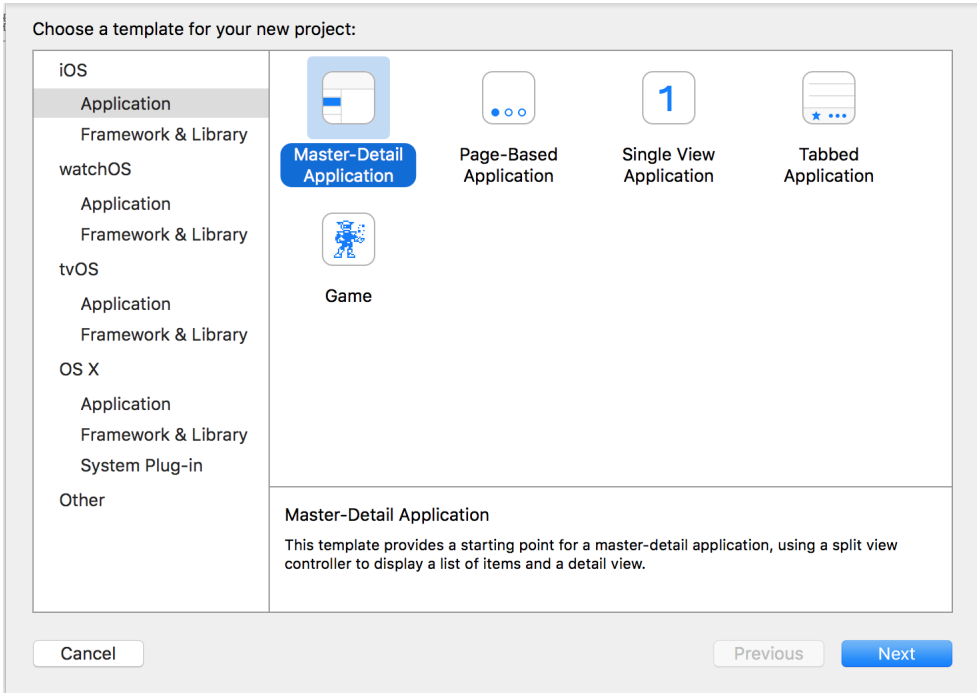

Figura B.2: Selección plantillas

detalle pertinente. Xcode crea una aplicación haciendo uso de un Split View Controller, el cual contiene la tabla (master) en la parte izquierda y los detalles (Detail) en la parte derecha. Este tipo de plantilla es muy útil cuando se quiere soportar iPhone y iPad porque el Split View Controller se encarga de mostrar todo al mismo tiempo en el caso de iPad o solo la tabla en una pantalla y el detalle en una nueva en el caso de iPhone.

- **Page-Based:** Esta plantilla crea una serie de archivos para demostrar cómo utilizar un UIPageViewController que es una clase nativa que se encarga de presentar diferentes View Controllers en forma de páginas. Como podemos ver, Xcode crea un Root View Controller, que contiene una instancia del UIPageViewController, el cual utiliza un patrón de diseño llamado data-source, éste patrón tiene un objeto llamado ModelController, que se encarga de manejar los diferentes View Controllers que se ubican en cada página.
- **Single View:** Crea un proyecto con un sólo View Controller en blanco, sin ninguna navegación o detalles.
- **Tabbed:** Crea una plantilla con dos View Controllers presentados por medio de una barra en la parte de abajo de la pantalla, tal vez la estructura más usada, como la aplicación de Facebook, Instagram, Twitter, etc.
- **Game:** Es una plantilla para crear juegos, presenta varias opciones como SceneKit, SpriteKit, OpenGL ES y Metal.

A continuación debemos configurar algunos parámetros generales del proyecto, como se puede observar en la figura B.3:

- **Product Name:** Es el nombre que va a aparecer en el iPhone debajo del ícono de la

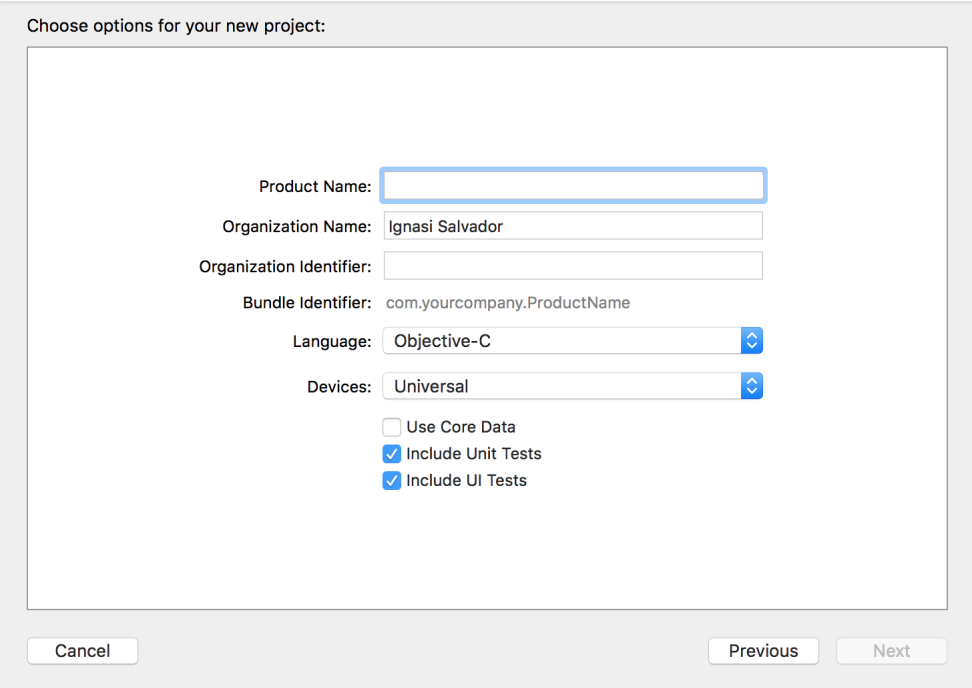

Figura B.3: Selección parámetros generales del proyecto

aplicación, puede ser modificado en el futuro.

- **Organization Name:** El nombre de la organización, empresa o persona que ?escribió? el código fuente de la aplicación (puede ser diferente del nombre que aparece en el App Store), éste aparecerá repetido en varias partes pero puede ser modificado en el futuro.
- **Organization Identifier:** Es una cadena para identificar a la organización, Apple recomienda usar una notación en reversa de nombres de dominio, (por ejemplo: si mi página web es www.mydominio.com, la notación en reversa sería ?com.mydominio?), éste identificador puede ser modificado en el futuro.
- **Bundle Identifier:** Esta es una cadena de caracteres automáticamente conformada por el identificador de la organización y el nombre de la aplicación, este identificador es utilizado para identificar la aplicación y tiene que ser único (no puede existir otra aplicación en la plataforma Apple con el mismo identificador) y va a ser utilizado en un futuro por Apple para cosas, por ejemplo para enviar notificaciones push, éste identificador no puede ser modificado después de haber publicado la aplicación, aunque inicialmente es creado concatenando el nombre de la aplicación y el identificador de la organización, éstos sí pueden ser modificados después de publicada.
- **Language:** Es el lenguaje de programación que vas a utilizar para programar la aplicación, Xcode te deja elegir entre Objective-C o Swift.
- **Devices:** Podemos seleccionar para que dispositivos vamos a desarrollar. Iphone, Ipad o Universal (ambos). La elección de uno no impide agregar el otro después, sin embargo se recomienda tomar esta decisión en este momento.
- **Use Core Data** nos crea automáticamente código plantilla para la utilización de Core Data, al igual que Unit Tests y UI Tests, por ahora los dejamos sin seleccionar, estas

herramientas pueden ser adicionadas a la aplicación en el futuro si es necesario.

Por último, debemos seleccionar un sitio dónde guardar nuestro proyecto, como se puede ver en la figura B.4.

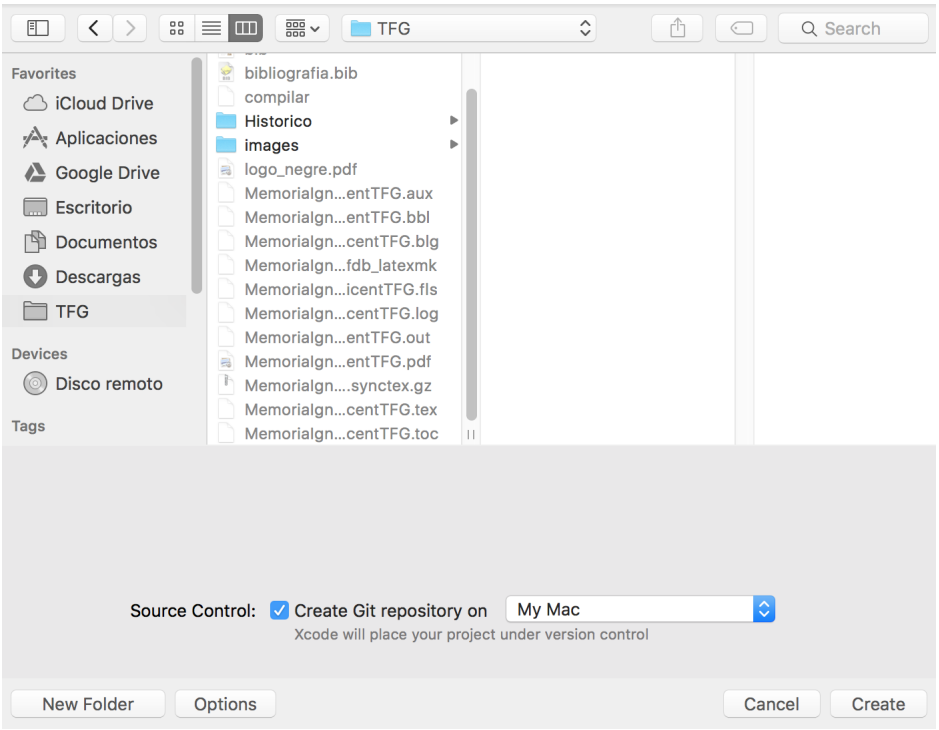

Figura B.4: Seleccionar sitio dónde guardar nuestro proyecto

Después de confirmar estos pasos tenemos nuestra primera aplicación en blanco, si la ejecutamos en este momento sólo veremos una pantalla en blanco.

Una vez dentro de nuestro proyecto en Xcode. Cómo se puede ver en la figura B.5, en la parte izquierda tenemos lo que yo llamo el grupo de carpeta y archivos. Aquí estarán todos los objetos, clases, y demás componentes que contendrá nuestra aplicación.

Hay tres archivos que son muy importantes:

- **AppDelegate.swift** Este fichero es el que delega el comportamiento básico de la app y donde están las funciones que se lanzan cuando se arranca esta, cuando se cierra, cuando se entra en background o se vuelve de dicho estado, etc.

  Aquí podemos programar el comportamiento de nuestra app controlando cosas como qué necesitamos que arranque cuando se inicie la misma, si recibimos alguna notificación o si se ha ido a un proceso en segundo plano.

  El fichero viene pre-cargado con las funciones básicas y si entramos, veremos una clara explicación de cuándo se ejecuta cada una de estas funciones.

- **Main.storyboard** Aquí es donde están todos los ficheros de la interfaz de nuestra app. Al

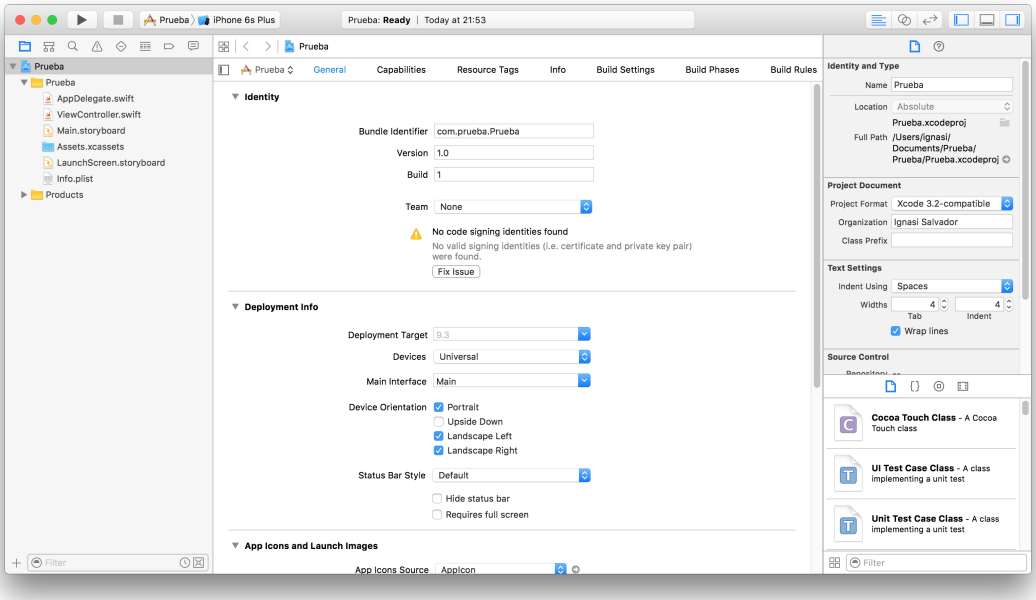

Figura B.5: Pantalla general de nuestro proyecto

pulsar en él veremos que se nos muestra el editor de interfaces y los elementos necesarios para construir esta como etiquetas, campos, botones, contenedores de imágenes, etc.

También tenemos todas las herramientas que nos permiten crear las *constraints* o restricciones que permitirán definir interfaces que se adapten automáticamente a cualquier tamaño, resolución y/o orientación de un dispositivo.

El *storyboard* también permite crear lo que se denominan escenas, que son las diferentes pantallas que mostrará nuestra app. En el *storyboard* podremos crear toda la estructura de navegación de la misma, cómo se interconectan o invocan los diferentes elementos y cuál es la secuencia de navegación que queramos darle.

- **ViewController.swift** Este fichero es la parte de modelo de la vista que representa el *storyboard* y la vista principal: su correspondiente controlador de vista. Es donde le diremos a un botón de la interfaz que, al pulsarse, ejecute un procedimiento que está en este fichero y quién se encargará de almacenar los resultados.

  Viene pre-cargado con dos funciones vacías: *viewDidLoad* que se lanza cuando se termina de cargar la vista correspondiente, y *didReceiveMemoryWarning* cuando hay una advertencia de memoria en el dispositivo.

  Otras posibles que podemos usar son *viewWillAppear* cuando la vista va a aparecer (antes de empezar con su dibujado, por lo que podemos personalizar su resultado), *viewDidAppear* cuando ha aparecido (y se han terminado de dibujar sus elementos), *viewWillDismiss* cuando se llama a una vista para que desaparezca y *viewDidDismiss* cuando una vista ha desaparecido.